Encryption in SAS® 9.4, Sixth Edition

SAS® Documentation
January 31, 2024

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *Encryption in SAS® 9.4, Sixth Edition*. Cary, NC: SAS Institute Inc.

Encryption in SAS® 9.4, Sixth Edition

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

January 2024

9.4-P11:secref

Contents

About This Book

Syntax Conventions for the SAS Language

Overview of Syntax Conventions for the SAS Language

Overview of Syntax Conventions for the SAS Language

SAS uses standard conventions in the documentation of syntax for SAS language elements. These conventions enable you to easily identify the components of SAS syntax. The conventions can be divided into these parts:

- syntax components
- style conventions
- special characters
- references to SAS libraries and external files

Syntax Components

Syntax Components

The components of the syntax for most language elements include a keyword and arguments. For some language elements, only a keyword is necessary. For other

language elements, the keyword is followed by an equal sign (=). The syntax for arguments has multiple forms in order to demonstrate the syntax of multiple arguments, with and without punctuation.

keyword
specifies the name of the SAS language element that you use when you write your program. Keyword is a literal that is usually the first word in the syntax. In a CALL routine, the first two words are keywords.

In these examples of SAS syntax, the keywords are bold:

CHAR (*string, position*)

CALL RANBIN (*seed, n, p, x*);

ALTER (*alter-password*)

BEST *w*.

REMOVE <*data-set-name*>

In this example, the first two words of the CALL routine are the keywords:

CALL RANBIN(*seed*, *n*, *p*, *x*)

The syntax of some SAS statements consists of a single keyword without arguments:

DO;
... *SAS code* ...
END;

Some system options require that one of two keyword values be specified:

DUPLEX | **NODUPLEX**

Some procedure statements have multiple keywords throughout the statement syntax:

CREATE <UNIQUE> **INDEX** *index-name* **ON** *table-name* (*column-1* <, *column-2*, …>)

argument
specifies a numeric or character constant, variable, or expression. Arguments follow the keyword or an equal sign after the keyword. The arguments are used by SAS to process the language element. Arguments can be required or optional. In the syntax, optional arguments are enclosed in angle brackets (< >).

In this example, *string* and *position* follow the keyword CHAR. These arguments are required arguments for the CHAR function:

CHAR (*string, position*)

Each argument has a value. In this example of SAS code, the argument *string* has a value of 'summer', and the argument *position* has a value of 4:

```
x=char('summer', 4);
```

In this example, *string* and *substring* are required arguments, whereas *modifiers* and *startpos* are optional.

FIND(*string, substring* <, *modifiers*> <, *startpos*>

argument(s)
specifies that one argument is required and that multiple arguments are allowed. Separate arguments with a space. Punctuation, such as a comma (,) is not required between arguments.

The MISSING statement is an example of this form of multiple arguments:

MISSING *character(s)*;

<LITERAL_ARGUMENT> *argument-1* <<LITERAL_ARGUMENT> *argument-2* ... >
specifies that one argument is required and that a literal argument can be associated with the argument. You can specify multiple literals and argument pairs. No punctuation is required between the literal and argument pairs. The ellipsis (...) indicates that additional literals and arguments are allowed.

The BY statement is an example of this argument:

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* …>;

argument-1 <*options*> <*argument-2* <*options*> ...>
specifies that one argument is required and that one or more options can be associated with the argument. You can specify multiple arguments and associated options. No punctuation is required between the argument and the option. The ellipsis (...) indicates that additional arguments with an associated option are allowed.

The FORMAT procedure PICTURE statement is an example of this form of multiple arguments:

PICTURE name <(*format-options*)>
<*value-range-set-1* <(*picture-1-options*)>
<*value-range-set-2* <(*picture-2-options*)> …>>;

argument-1=value-1 <*argument-2=value-2* ...>
specifies that the argument must be assigned a value and that you can specify multiple arguments. The ellipsis (...) indicates that additional arguments are allowed. No punctuation is required between arguments.

The LABEL statement is an example of this form of multiple arguments:

LABEL *variable-1=label-1* <*variable-2=label-2* …>;

argument-1 <, *argument-2*, ...>
specifies that one argument is required and that you can specify multiple arguments that are separated by a comma or other punctuation. The ellipsis (...) indicates a continuation of the arguments, separated by a comma. Both forms are used in the SAS documentation.

Here are examples of this form of multiple arguments:

AUTHPROVIDERDOMAIN (*provider-1:domain-1* <, *provider-2:domain-2*, …>

INTO :*macro-variable-specification-1* <, :*macro-variable-specification-2*, …>

Note: In most cases, example code in SAS documentation is written in lowercase with a monospace font. You can use uppercase, lowercase, or mixed case in the code that you write.

Style Conventions

Style Conventions

The style conventions that are used in documenting SAS syntax include uppercase bold, uppercase, and italic:

UPPERCASE BOLD
identifies SAS keywords such as the names of functions or statements. In this example, the keyword ERROR is written in uppercase bold:

ERROR <*message*>;

UPPERCASE
identifies arguments that are literals.

In this example of the CMPMODEL= system option, the literals include BOTH, CATALOG, and XML:

CMPMODEL=BOTH | CATALOG | XML |

italic
identifies arguments or values that you supply. Items in italic represent user-supplied values that are either one of the following:

- nonliteral arguments. In this example of the LINK statement, the argument *label* is a user-supplied value and therefore appears in italic:

  LINK *label*;

- nonliteral values that are assigned to an argument.

  In this example of the FORMAT statement, the argument DEFAULT is assigned the variable *default-format*:

  FORMAT *variable(s)* <*format* > <DEFAULT = *default-format*>;

Special Characters

Special Characters

The syntax of SAS language elements can contain the following special characters:

=
an equal sign identifies a value for a literal in some language elements such as system options.

In this example of the MAPS system option, the equal sign sets the value of MAPS:

MAPS=*location-of-maps*

< >
angle brackets identify optional arguments. A required argument is not enclosed in angle brackets.

In this example of the CAT function, at least one item is required:

CAT (*item-1* <, *item-2*, …>)

|
a vertical bar indicates that you can choose one value from a group of values. Values that are separated by the vertical bar are mutually exclusive.

In this example of the CMPMODEL= system option, you can choose only one of the arguments:

CMPMODEL=BOTH | CATALOG | XML

...
an ellipsis indicates that the argument can be repeated. If an argument and the ellipsis are enclosed in angle brackets, then the argument is optional. The repeated argument must contain punctuation if it appears before or after the argument.

In this example of the CAT function, multiple *item* arguments are allowed, and they must be separated by a comma:

CAT (*item-1* <*, item-2*, …>)

'*value*' or "*value*"
indicates that an argument that is enclosed in single or double quotation marks must have a value that is also enclosed in single or double quotation marks.

In this example of the FOOTNOTE statement, the argument *text* is enclosed in quotation marks:

FOOTNOTE <*n*> <*ods-format-options* '*text*' | "*text*">;

;
a semicolon indicates the end of a statement or CALL routine.

In this example, each statement ends with a semicolon:

```
data namegame;
   length color name $8;
   color = 'black';
   name = 'jack';
   game = trim(color) || name;
run;
```

References to SAS Libraries and External Files

References to SAS Libraries and External Files

Many SAS statements and other language elements refer to SAS libraries and external files. You can choose whether to make the reference through a logical name (a libref or fileref) or use the physical filename enclosed in quotation marks.

If you use a logical name, you typically have a choice of using a SAS statement (LIBNAME or FILENAME) or the operating environment's control language to make the reference. Several methods of referring to SAS libraries and external files are available, and some of these methods depend on your operating environment.

In the examples that use external files, SAS documentation uses the italicized phrase *file-specification*. In the examples that use SAS libraries, SAS documentation uses the italicized phrase *SAS-library* enclosed in quotation marks:

```
infile file-specification obs = 100;
libname libref 'SAS-library';
```

What's New in Encryption with SAS 9.4

Overview

Starting with SAS 9.4, encryption in SAS is affected by the following changes and enhancements:

- TLS version 1.2 is the protocol that is supported in SAS 9.4. Earlier versions of TLS and SSL are unsecure. Starting with SAS 9.4M8, TLS 1.3 is also supported
- For SAS 9.4 and maintenance releases of SAS 9.4 prior to SAS 9.4M8, updated versions of OpenSSL are provided and updated through hot fixes for Linux, UNIX, and z/OS.
- Starting with SAS 9.4, SAS/SECURE is included with Base SAS. Prior to SAS 9.4, SAS/SECURE was licensed and ordered separately. Other changes have been made to “SAS/SECURE with SAS 9.4M8”.
- SAS complies with current U.S. export laws and is shipped to most countries, with the exception of those embargoed or sanctioned by the U.S. Department of Treasury without restriction.
- Encoding type SAS004 is added to (uses AES encryption with 64-bit salt) provide increased security for stored passwords.
- Increased security is provided for SAS data at rest.
- Enhanced logging features are introduced for encryption. These enhancements include additional loggers and better debugging and traceback features that are part of the SAS Logging Facility.
- Digital certificates can be imported to a central location on a Windows client or server.
- Starting with SAS 9.4, environment variable SSLREQCERT= enables you to specify checks to perform on server certificates in a TLS session. For the TLS session to continue, you can demand that certificates be provided for the session to continue. You can also allow the session to continue when invalid certificates are provided or when no certificates are provided.

  Note: The SSLREQCERT= environment variable is used only for Linux and UNIX operating systems.

Starting with SAS 9.4M1, SAS supports the following:

- The default location for the Certificate Authority (CA) trust list has changed for the UNIX and z/OS foundation servers. This default location is specified by the SSLCALISTLOC= option.
- Environment variables SSL_CERT_DIR and SSLCACERTDIR are introduced. They are used to point to the location of certificates. These environment variables are supported on UNIX and support logging. The default location is specified by the SSLCALISTLOC= system option.

  Note: These environment variables were also made available in maintenance releases prior to SAS 9.4M1 through hot fixes. However, in SAS 9.4M5, these environment variables have been replaced by similarly named system options.

- Subject Alternative Names (SAN) in TLS certificates are supported. Server Name Indications (SNI) in the TLS handshake between clients and servers are also supported. These are supported on UNIX and z/OS clients and servers.
- CA certificates are located in the trustedcerts.pem file for UNIX and z/OS. The SSLCALISTLOC= option on UNIX and z/OS points to the trustedcerts.pem file by default.

Starting with SAS 9.4M3, SAS supports the following:

- The SAS_SSL_MIN_PROTOCOL= environment variable is added and is supported on UNIX, Windows, and z/OS. This environment variable should be set to support TLS 1.2. Earlier versions of TLS are unsecure.

  Note: This environment variable is available in prior SAS releases by applying security hot fixes.

- The SAS_SSL_CIPHER_LIST environment variable supported on UNIX and z/OS was added.

  Note: This environment variable was also made available in maintenance releases prior to SAS 9.4M3 through hot fixes. However, in SAS 9.4M5, this environment variable has been replaced by a similarly named system option.

- The SAS Deployment Manager is used to automate the process of updating the CA certificates on all hosts at SAS installation. The SAS Deployment Manager is used to manage the truststore for all hosts. The Mozilla bundle of trusted CA certificates is included in the truststore at installation. After SAS installation, you can use the SAS Deployment Manager to add your own trusted certificates to the truststore.

  CAUTION
  Expired Sectigo Certificate in the Mozilla Bundle of Trusted CA Certificates Causes Errors The Mozilla CA trusted bundle of certificates contains a CA certificate from Sectigo with an expiration date of May 30, 2020. This expired certificate generates errors with SAS 9.4M3 - SAS 9.4M6. A SAS hot fix has been created to resolve these issues.

- Information about setting up a FIPS 140-2 environment has been updated in the SAS Deployment Wizard.

Starting with SAS 9.4M5, SAS supports the following:

- System options SSLCACERTDIR=, SSLCIPHERLIST=, and SSLSNIHOSTNAME= have been added for LINUX, UNIX, and z/OS. The SSLMODE= system option has been added for LINUX, UNIX, z/OS, and Windows.
- By default the Server Name Indication (SNI) is sent to the web servers in the TLS handshake. The environment variable, SSL_USE_SNI is now used only to disable SNI.
- When encrypting data at rest, you can now specify data set option ENCRYPT=AES2. AES2 is another key generation algorithm for AES encryption.
- Encoding type SAS005 uses AES encryption with a 256-bit fixed key and a 64-bit random salt value. SAS005 increases security for stored passwords by using the SHA256 hashing algorithm and is hashed for additional iterations.
- For more security, you can use SHA256-10000 for internal account passwords used in metadata. SHA256-10000 is the same as SHA256, but is hashed for additional iterations.
- Environment variable SAS_VIYA_TOKEN is added and provides the OAuth token that is needed for a user to access SAS Viya services.
- The CAS_CLIENT_SSL_CA_LIST environment variable is added to allow SAS 9.4 acting as a client to use the same TLS certificates as those used by SAS Viya.

Starting with SAS 9.4M6, SAS supports the following:

- TLS is supported on all IOM servers.
- TLS environment variables and system options can be applied locally using the PROC HTTP SSLPARMS statement.

Starting with SAS 9.4M7, SAS supports the following:

- When lockdown is in effect, paths to SAS system options (or equivalent environment variables) SSLCALISTLOC, SSLCACERTDIR, SSLCERTLOC, SSLPVTKEYLOC, SSLPKCS12LOC, and SSLCRLLOC are added by default to the LOCKDOWN allowlist.
- SAS encoded passwords (SAS001–SAS005) are supported for SAS system options SSLPVTKEYPASS= and SSLPKCS12PASS=.
- When hot fixes are applied, the RSA Crypto-J library used with Java for encryption is updated. Hot fix L1K001 is applied to the SAS 9.4M7 server and provides the ability to use 4096-bit RSA keys. Hot fix K9Z001 is applied to the SAS 9.4M7 client by SAS Technical Support upon request.

Starting with SAS 9.4M8, SAS supports the following:

- SAS supports IBM z/OS Pervasive Encryption for data sets.
- IBM System SSL is used to provide TLS encryption on z/OS.

  FIPS 140-2 support is provided as part of IBM System SSL.

  SAS supports IBM System SSL on z/OS 64-bit SAS Metadata Server through hot fixes. The z/OS 64-bit SAS Metadata Server is used in a SAS Business Intelligence platform deployment.

  Because SAS 9.4M8 uses System SSL, some SAS System Options have been deprecated for use with z/OS and new system options have been added.

- SSLMINPROTOCOL= system option supports TLS 1.3.
- SSLMODE=SSLMODESP800131A provides the most secure ciphers for TLS 1.2 and TLS 1.3. SSLMODESP800131A is the default value.
- For SAS Foundation servers, SAS uses the cryptographic libraries that are installed on the operating system where SAS 9.4M8 is installed to provide encryption for data in motion and data at rest. SAS no longer delivers the cryptographic libraries that are used by SAS Foundation servers for SAS/SECURE or OpenSSL.
- When hot fixes are applied, AES 256-bit encryption is used to encrypt PDF files generated by ODS when system option PDFSECURITY=HIGH.
- Encrypted communication between SAS/ACCESS to Teradata and SAS Viya using TLS 1.2 is supported.

General Enhancements

Starting with SAS 9.4, the following changes and enhancements were made to encryption:

- TLS version 1.2 is the minimum protocol supported with SAS 9.4. Earlier versions of TLS and SSL are unsecure. Starting with SAS 9.4M5, the default version of TLS is 1.2. For SAS software releases prior to SAS 9.4M5, you must configure SAS to use TLS 1.2 by setting the SAS_SSL_MIN_PROTOCOL= environment variable.

  Note: The SAS_SSL_MIN_PROTOCOL= environment variable is available with SAS releases prior to SAS 9.4M3 by applying security hot fixes.

- For SAS 9.4 and all maintenance releases of SAS 9.4 prior to SAS 9.4M8, updated versions of OpenSSL for UNIX and z/OS are provided and updated through hot fixes. See the SAS Security Bulletin on OpenSSL for the most current information about the versions of OpenSSL used with SAS products and about the advisories under consideration.

  Starting with SAS 9.4M8, SAS requires that supported cryptographic libraries be installed on the operating system for SAS Foundation. SAS no longer delivers the cryptographic libraries that are used with OpenSSL.

- For software delivery purposes, SAS/SECURE is a feature within SAS. SAS/SECURE is included with the Base SAS software. With SAS releases prior to SAS 9.4, SAS/SECURE was an add-on product that was licensed separately. This change makes strong encryption available in all deployments.

  Note: Changes have been made to “SAS/SECURE with SAS 9.4M8”.

- If you use SAS/SECURE, you can use encoding type SAS0004 for stored passwords. SAS004 uses AES encryption with 64-bit salt. The salt size was increased to 64 bits to comply with the minimum recommended salt size for PKCS #5 v2.0: Password-Based Cryptography Standard, http://www.rsa.com/

rsalabs/node.asp?id=2127. See "SAS/SECURE " on page 20 and Chapter 13, "PWENCODE Procedure," on page 185.

- If you use SAS/SECURE, you can use an industry standard algorithm (AES) to encrypt SAS data on disk. For more information, see "ENCRYPT= Data Set Option" in *SAS Data Set Options: Reference* and "SAS Data File Encryption" in *SAS Programmer's Guide: Essentials*.
- The SAS Logging Facility now supports full logging and debugging of the SAS/CONNECT spawner operations. See "LOGCONFIGLOC= System Option" in *SAS Logging: Configuration and Programming Reference* for detailed information.
- The SAS Logging Facility supports full logging and debugging of encryption activity. See "LOGCONFIGLOC= System Option" in *SAS Logging: Configuration and Programming Reference* for system option information. For information about security loggers, see "Encryption: SAS Logging Facility" on page 36.
- You can use environment variable SSLREQCERT= to specify checks to perform on server certificates in a TLS session. You can specify settings that allow a session to proceed normally when no TLS certificate is provided or an invalid certificate is provided. You can demand that when a server certificate is requested, the session terminates if no valid certificate is provided. See "SSLREQCERT=" on page 182.

  Note: The SSLREQCERT= environment variable is used only for Linux and UNIX operating systems.

The following changes and enhancements were made to encryption with SAS maintenance releases.

Starting with SAS 9.4M1, SAS supports the following:

- For certificates used with TLS, SAS sets the default location of the Certificate Authority (CA) trust list to `SAS-configuration-directory/levn/certs/cacert.pem` for UNIX and z/OS on SAS Foundation servers. This default location is specified by the SSLCALISTLOC= option in configuration files. For more information, see "SSLCALISTLOC=" on page 145.
- Environment variables SSL_CERT_DIR and SSLCACERTDIR can also be used to point to the location of certificates. These environment variables are supported on UNIX and z/OS and support logging.

  Note: These environment variables are available through hot fixes in some maintenance releases.

  For information, see "SSLCACERTDIR" on page 209 and "SSL_CERT_DIR" on page 211.
- Starting with SAS 9.4M1, UNIX and z/OS clients and servers support Server Name Indication (SNI) and Subject Alternative Names (SAN) in TLS. The client uses SNI in the TLS handshake to tell the server which server name it is trying to connect to. SANs are used in TLS certificates. For information, see "SSL_USE_SNI" on page 183.

Starting with SAS 9.4M3, SAS supports the following:

- Trusted certificates are located in the trustedcerts.pem file. The SSLCALISTLOC= system option points to the trustedcerts.pem file by default.

This file is located in `<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/`. The SSLCALISTLOC= system option and new location are automatically added at SAS installation.

- The SAS_SSL_MIN_PROTOCOL= environment variable is added and is supported on UNIX, Windows, and z/OS. This environment variable should be set to support TLS 1.2. Earlier versions of TLS are unsecure. For more information, see "SAS_SSL_MIN_PROTOCOL=" on page 179.

  Note: The SAS_SSL_MIN_PROTOCOL= environment variable is available with SAS releases prior to SAS 9.4M3 by applying security hot fixes.

- Environment variable SAS_SSL_CIPHER_LIST is supported on UNIX and z/OS. For more information, see "SAS_SSL_CIPHER_LIST" on page 207.
- On a Windows server or client, the user can import digital certificates to a Machine Store as well as to a Personal Store. See "Configure TLS and Request Digital Certificates on Windows" on page 109.
- The SAS Deployment Manager can be used to automate the process of updating the list of trusted CA Certificates. At installation, a list of trusted CA certificates that are distributed by Mozilla is installed and SAS products are automatically configured to use this. The SAS Deployment Manager is used to manage the truststore for all hosts. The trustedcerts.pem and trustedcerts.jks files are both updated. On Windows, the SAS Deployment Manager tasks manage the Java version of the truststore, on UNIX, the SAS Deployment Manager task updates the truststore (trustedcerts.pem and the trustedcerts.jks files), and on z/OS, the SAS Deployment Manager tasks update the truststore (trustedcerts.pem file).

  CAUTION
  Expired Sectigo Certificate in the Mozilla Bundle of Trusted CA Certificates Causes Errors The Mozilla bundle of trusted CA certificates contains a CA certificate from Sectigo with an expiration date of May 30, 2020. This expired certificate generates errors with SAS 9.4M3 - SAS 9.4M6. A SAS hot fix has been created to resolve these issues. See "TLS Certificate Verification: ERROR, Certificate Has Expired" on page 203.

  See "Add Your Certificates to the Windows CA Store" on page 112 and "Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager" on page 92. For the specific details about these SAS Deployment Manager tasks, see the *SAS® Deployment Wizard and SAS® Deployment Manager 9.4: User's Guide*.

- Information has been added about setting the FIPS security settings on a Windows server. See "Configure FIPS 140-2 Capable TLS on Windows" on page 117.
- Information about setting up a FIPS-2 environment on UNIX has been updated in the SAS Deployment Wizard. For specific information, see *SAS® Deployment Wizard and SAS® Deployment Manager 9.4: User's Guide*. For more information about FIPS, see "FIPS 140-2 Standards Compliance" on page 15, "Configure TLS to Run in FIPS 140-2 Compliant Mode " on page 28, and "Process to Build FIPS 140-2 Capable OpenSSL on UNIX and Linux" on page 104.

Starting with SAS 9.4M5, SAS supports the following:

- The following system options have been added for LINUX, UNIX, and z/OS. These system options can also be used as environment variables. See "SAS System Options for Encryption" on page 133.
  - SSLCIPHERLIST= specifies a list of cipher suites to use.

    Note: This system option replaces the SAS_SSL_CIPHER_LIST= environment variable.

  - SSLSNIHOSTNAME= enables the client to use Server Name Indication (SNI) in the TLS handshake to identify the server name that it is trying to connect to. SSLSNIHOSTNAME= is used when you want to change the name of the host.
  - SSLCACERTDIR= specifies the location of the trusted certificate authorities (CA) found in OpenSSL format.

    Note: This system option replaces the SAS_SSL_CERT_DIR= environment variable.

  - SSLMODE= specifies the SAS data-in-motion standard being used for all data-in-transit across network boundaries.
- The SSLMODE= system option has been added for Windows.
- By default the SNI is sent to the web servers in the TLS handshake. The environment variable, SSL_USE_SNI is now used only to disable SNI.
- When encrypting data at rest, you can specify data set option ENCRYPT=AES2. AES2 is another key generation algorithm for AES encryption. See "ENCRYPT= Data Set Option" in *SAS Data Set Options: Reference* and "SAS Data File Encryption" in *SAS Programmer's Guide: Essentials*.
- Encoding type SAS005 uses AES encryption with a 256-bit fixed key and a 64-bit random salt value. SAS005 increases security for stored passwords by using the SHA256256 hashing algorithm and is hashed for additional iterations.

  You can specify SAS005 as a method to encode passwords using PROC PWENCODE. You can also use this encoding for passwords stored in metadata and in configuration files. See Chapter 13, "PWENCODE Procedure," on page 185.
- For more security, you can use SHA256-10000 for internal account passwords used in metadata. SHA256-10000 is the same as SHA256, but is hashed for additional iterations.
- Environment variable SAS_VIYA_TOKEN= is added and provides the OAuth token that is needed for a user to access SAS Viya services. This environment variable enables you to pre-load your token before starting your SAS session. See "SAS_VIYA_TOKEN" on page 180.
- The CAS_CLIENT_SSL_CA_LIST= environment variable can be specified to allow a SAS 9.4 client to use the same TLS certificates as those used by SAS Viya. The environment variable points to the path and file name of the file that contains the list of trusted certificate authority (CA) certificates.

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

For more information, see CAS TLS Environment Variables

- The OpenSSL libraries provided by SAS are updated on an ongoing basis. For maintenance releases starting with SAS 9.4 - SAS 9.4M7, updated versions of OpenSSL are provided and updated through hot fixes for UNIX and z/OS. For a quick reference, see Table 3.2 on page 26.

  Note: Starting with SAS 9.4M8, SAS Foundation servers use supported cryptographic libraries that are installed on the operating system to provide encryption for data at rest and data in motion.

Starting with SAS 9.4M6, SAS supports the following:

- TLS is supported on Integrated Object Model (IOM) servers and server processes that provide IOM Bridge access. The following servers and processes support TLS:
  - SAS Metadata Server
  - SAS OLAP Server
  - workspace server
  - SAS Stored Process Server
  - pooled workspace server
  - object spawner

  To configure TLS on IOM servers, see information in TLS support for IOM Servers.
- TLS environment variables and system options can be applied locally using the PROC HTTP SSLPARMS statement. For more information, see "SSLPARMS" in *Base SAS Procedures Guide*.

Starting with SAS 9.4M7, SAS supports the following:

- When lockdown is in effect, paths to SAS system options (or equivalent environment variables) SSLCALISTLOC=, SSLCACERTDIR=, SSLCERTLOC=, SSLPVTKEYLOC=, SSLPKCS12LOC=, and SSLCRLLOC= are added by default to the LOCKDOWN allowlist. These SAS system options can be specified in a SAS configuration file or as a SAS start-up command line option. For information about LOCKDOWN, see "SAS Processing Restrictions for Servers in a Locked-Down State" in *SAS Programmer's Guide: Essentials*.
- SAS encoded passwords (SAS001–SAS005) are supported for SAS system options SSLPVTKEYPASS= and SSLPKCS12PASS=. See "SAS System Options for Encryption" on page 133.

Starting with SAS 9.4M8, SAS supports the following:

- IBM z/OS Pervasive Encryption on data sets. SAS direct access bound libraries and sequential access libraries can use Pervasive Encryption. See "IBM z/OS Pervasive Encryption for Data Sets with SAS 9.4M8".
- SAS uses IBM System SSL to provide TLS encryption for z/OS. Prior to SAS 9.4M8, SAS delivered OpenSSL libraries for use with z/OS. See "IBM System SSL Provides OpenSSL Capabilities for z/OS in SAS 9.4M8" on page 10.

  FIPS 140-2 support is provided as part of IBM System SSL. System SSL provides FIPS 140-2 certified cryptographic support for z/OS.

SAS supports IBM System SSL on z/OS 64-bit SAS Metadata Server through hot fixes. The z/OS 64-bit SAS Metadata Server is used in a SAS Business Intelligence platform deployment. For more information, see Usage Note 70504: Support for the IBM "System SSL" for 64–bit z/OS. For configuration information, see *Configuration Guide for SAS 9.4 Foundation for z/OS*.

Because SAS 9.4M8 uses System SSL, some SAS System Options have been deprecated for use with z/OS and new system options have been added.

Because of these changes, the following SAS System Options have been deprecated for use with z/OS and others have been added.

- SAS system options that are deprecated for use with z/OS are: SSLCACERTDIR, SSLCALISTLOC, SSLCERTLOC, SSLCIPHERLIST, SSLCRLLOC, SSLPKCS12LOC, SSLPKCS12PASS, SSLPVTKEYLOC, SSLPVTKEYPASS, and SSLSNIHOSTNAME . See “SAS System Options Deprecated for Use with SAS 9.4M8 on z/OS”.
- SAS System Options that are new and are used with z/OS are: SSLGSKTRACE, SSLHWDETECTMESSAGE, SSLICSFERRORMESSAGE, SSLGSKTRACEFILE, SSLKEYRINGFILE, SSLKEYRINGLABEL, SSLKEYRINGPW, and SSLKEYRINGSTASHFILE. See “SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8”.

- SSLMINPROTOCOL= system option supports TLS 1.3. See “SSLMINPROTOCOL=” on page 159.
- SSLMODE= system option supports the most secure ciphers for TLS 1.2 and TLS 1.3. See “SSLMODE=” on page 156.
- SAS Foundation servers use supported cryptographic libraries that are installed on the operating system to provide encryption for data at rest and data in motion. Because of this change, it is no longer necessary to deliver the cryptographic libraries that were used by SAS/SECURE or OpenSSL libraries as part of the SAS installation for SAS Foundation.

  Because of the cryptographic library changes, the following changes have been made:

  - It is no longer necessary to deliver the cryptographic libraries that were used by SAS/SECURE or OpenSSL as part of the SAS installation for SAS Foundation.
  - SAS requires that supported cryptographic libraries be installed on the operating system.
  - Because operating system’s OpenSSL libraries might not provide all of the encryption algorithms that were available with SAS releases prior to SAS 9.4M8 (mainly because they are deemed unsecure), a warning might be generated. See “WARNING: The OpenSSL 3 "legacy" provider could not be loaded. Certain functions may not work without ciphers from the "legacy" provider”.

- Encrypted communication between SAS/ACCESS to Teradata and SAS Viya using TLS 1.2 is supported. For more information, see “Configure TLS for SAS/ACCESS Connection to Teradata” on page 73.
- When hot fixes are applied, AES 256-bit encryption is used to encrypt PDF files generated by ODS when system option PDFSECURITY=HIGH. Prior to SAS 9.4M8, SAS encrypts PDF files using RC4 128-bit encryption. For more information, see “Encrypting ODS Generated PDF Files” on page 38.

Documentation Enhancements

Starting with SAS 9.4M4, information about certificate management has been moved into this document and into the *SAS 9.4 Intelligence Platform: Security Administration Guide*. The following topic information previously existed in the *SAS 9.4 Intelligence Platform Installation and Configuration Guide*.

- See "Add Your Certificates to the SAS Private JRE" on page 99.
- See "Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager" on page 92.

PART 1

Overview

1

Overview of Encryption

Two Classes of Encryption Strength

Two classes of encryption strength are available:

- For compatibility with legacy systems, SASProprietary encoding is supported. These methods are available in all deployments and are appropriate for preventing accidental exposure of information. They have minimal impact on performance.
- For a higher level of security, it is highly recommended that you use industry-standard encryption and hashing algorithms. For the highest security for data in motion, use TLS. For data at rest, use AES encryption.

  Starting with SAS 9.4M8, SAS uses the cryptographic libraries that are installed on the operating system to provide encryption for data in motion and data at rest for SAS Foundation servers. Prior to SAS 9.4M8, SAS provided the cryptographic libraries that were used with SAS/SECURE and OpenSSL to provide industry-standard encryption.

SAS recommends that you use the strongest security standards available for your environment.

Two Contexts for Encryption Coverage

SAS 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. TLS is

used as the preferred mechanism to provide encryption for data in motion. Starting with SAS 9.4M8, SAS Foundation servers use supported cryptographic libraries that are provided and installed on the operating system to provide encryption for data in motion.

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

- 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. The emphasis is on protection of passwords in configuration files, in the metadata repository, and on encryption of SAS data sets.

  Starting with SAS 9.4M8, SAS Foundation servers use the cryptographic libraries that are provided and installed on the operating system to provide encryption for data at rest.

  See "IBM z/OS Pervasive Encryption for Data Sets with SAS 9.4M8" and "SAS Data File Encryption" in *SAS Programmer's Guide: Essentials*.

PART 2

Encryption Starting with SAS 9.4M8

2

Encryption Starting with SAS 9.4M8

SAS/SECURE with SAS 9.4M8

Prior to SAS 9.4M8, the cryptographic libraries that were used by SAS/SECURE were delivered as part of the SAS installation. Starting with SAS 9.4M8, SAS requires that supported cryptographic libraries be installed on the SAS Foundation operating system where SAS 9.4M8 is installed. SAS uses the cryptographic libraries that are installed and configured on the operating system to provide encryption for data in motion and data at rest.

Because of the cryptographic library changes starting with SAS 9.4M8, best practices when upgrading SAS software are to uninstall retired product features after you upgrade.

Cryptographic Library Support Starting with SAS 9.4M8

Staring with SAS 9.4M8, SAS Foundation servers use the cryptographic libraries that are provided and installed on the operating system to provide encryption for data at rest and data in motion. SAS requires that supported cryptographic libraries be installed on the SAS Foundation operating system where SAS 9.4M8 is installed.

> **IMPORTANT** SAS 9.4M8 no longer supports OpenSSL versions 0.9.8 and 1.0.0.
>
> SAS 9.4M8 no longer uses Crypto-C ME libraries.

Starting with SAS 9.4M8, the following cryptographic libraries are supported on SAS Foundation servers.

- For UNIX and Linux servers, the OpenSSL libraries that are installed on the SAS Foundation operating system are used by the SAS deployment. Many versions of OpenSSL are tested with SAS 9.4M8 to ensure that SAS can run with the OpenSSL libraries and FIPS 140-2 certified libraries that are provided. Some of the OpenSSL library versions that are supported and tested with SAS software include OpenSSL 3, OpenSSL 1.1.1, and OpenSSL 1.0.2.
- For z/OS, SAS uses IBM System SSL to provide FIPS 140-2 certified cryptographic library support. Prior to SAS 9.4M8, SAS delivered OpenSSL libraries for use with z/OS.

  Beginning with SAS 9.4M8, because z/OS uses the cryptographic libraries that System SSL provides, each SAS supported cipher suite might not be available. When this issue is encountered, use a cipher suite that is supported by both SAS and IBM System SSL.

  You can compare the list of SAS supported cipher suites to those supported by IBM System SSL cipher suites .
- On Windows, SAS uses SChannel SSP (Security Support Provider) that provides TLS internet standard authentication protocols and BCrypt that provides other encryption algorithms. All Windows system libraries are FIPS certified.

  Because SAS uses the cryptographic libraries that are provided by the Windows operating system, each SAS supported cipher suite might not be available in the version of Windows operating system that is being used. When this issue is encountered, use a cipher suite that is supported by both SAS and Windows.

  You can compare the list of SAS supported cipher suites to those supported by Microsoft Schannel SSP. These are found in Protocols in TLS/SSL (Schannel SSP) and Cipher Suites in TLS/SSL (Schannel SSP).

TLS Versions and Cipher Suites Supported Starting with SAS 9.4M8

Starting with SAS 9.4M8, SAS supports TLS 1.3. SAS also supports TLS 1.2. It is highly recommended that TLS is the encryption protocol used to provide encryption for data in motion.

The default minimum protocol recommended by SAS is TLS 1.2. By default, SAS first tries to use TLS 1.3 ciphers to provide the highest level of security. If your TLS libraries do not support the TLS 1.3 ciphers, SAS tries to use the default TLS 1.2 ciphers.

The default cipher suites supported for TLS 1.3 are as follows:

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_AES_128_CCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_CHACHA20_POLY1305_SHA256

The default cipher suites supported for TLS 1.2 are as follows:

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

On UNIX and Linux, the SSLMODE SAS system option can be used to change the default cipher suites.See SSLMODE= system option.

FIPS 140-2 Support with SAS 9.4M8 on UNIX and Linux

For an overview of FIPS 140-2, see “FIPS 140-2 Standards Compliance”.

Here are some differences between how FIPS works in SAS 9.4M8 and in software releases prior to SAS 9.4M8.

- Prior to SAS 9.4M8 , SAS delivers OpenSSL libraries for use by SAS and documents how to build FIPS enabled libraries and install them. Setting the ENCRYPTFIPS system option enables FIPS and places the SAS software in FIPS mode.

- Starting with SAS 9.4M8, the cryptographic libraries that support FIPS must be installed on the operating system and FIPS must be enabled on the operating system. The ENCRYPTFIPS system option is then set to ensure that the SAS software is placed in FIPS mode.

Each operating system provider enables FIPS differently according to their security policies. SAS validates that when the operating system cryptographic libraries are FIPS enabled, the SAS software operates as designed. Refer to the operating system documentation for information about FIPS 140-2 certification and enablement.

IBM System SSL Provides OpenSSL Capabilities for z/OS in SAS 9.4M8

Starting with SAS 9.4M8, SAS supports the use of IBM System SSL to provide TLS for data in motion for z/OS. SAS no longer provides OpenSSL libraries to support TLS on z/OS.

IBM System SSL provides all the capabilities of OpenSSL and is fully integrated with the z/OS security services (SAF). The encryption keys can be stored using SAF. These services include RACF, ACF/2, and Top-Secret.

GSK (Global Security Kit) is the API that SAS uses.

Here are key features for using System SSL with SAS 9.4M8.

- SAS no longer provides OpenSSL libraries to support TLS on z/OS. SAS uses IBM System SSL to provide TLS for data in motion for z/OS. For information about configuring and using System SSL with SAS 9.4M8 on 31-bit z/OS, see Chapter 9, “Configure TLS Certificates and FIPS on z/OS”.
- SAS supports IBM System SSL on z/OS 64-bit SAS Metadata Server through hot fixes. For more information, see Usage Note 70504: Support for the IBM "System SSL" for 64–bit z/OS. For configuration information, see *Configuration Guide for SAS 9.4 Foundation for z/OS*.
- System SSL supports FIPS 140-2 on z/OS. See “FIPS 140-2 Support Is Provided with System SSL on z/OS with SAS 9.4M8”.
- The cryptographic libraries used by SAS are those supported by IBM System SSL. TLS versions and ciphers that are supported by IBM System SSL are shown in Cipher suite definitions. SAS supports TLS versions and cipher suites described in “Cryptographic Library Support Starting with SAS 9.4M8” and “TLS Versions and Cipher Suites Supported Starting with SAS 9.4M8”. You need to compare the cipher suites and TLS versions supported by SAS 9.4M8 to determine compatibility with System SSL.
- In SAS 9.4M8, new SAS system options are used to manage certificates and to trace and detect error messages. See “SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8”. Also see “SAS System Options Deprecated for Use with SAS 9.4M8 on z/OS”.
- To use the SAS provided truststore and add your CA certificates to the Mozilla trusted bundle of CA certificates, see SAS Usage Note 69954. These instructions provide a way to download the truststore in a format that System

SSL can use (trustedcerts.kdb and trustedcerts.sth) and provide instructions for adding additional CA certificates to the truststore.

For additional configuration and programming information about IBM System SSL, see z/OS Cryptographic Services System SSL Programming and Using cryptographic features with System SSL.

SAS Support of IBM z/OS Pervasive Encryption with SAS 9.4M8

Pervasive Encryption for IBM Z platforms enables extensive encryption of data at rest. Starting with SAS 9.4M8, support for z/OS Pervasive Encryption is available for SAS direct access bound libraries and sequential access libraries. The access method that SAS uses to read and write data sets for z/OS is Execute Channel Program (EXCP).

Pervasive Encryption support is available only in the SAS 31bit (MVS) offering. z/OS data set encryption currently supports only data-encrypting keys that are using the AES algorithm with a 256-bit key length.

For information about using z/OS Pervasive Encryption with SAS, see “IBM z/OS Pervasive Encryption for Data Sets with SAS 9.4M8”.

PART 3

Concepts

3

Encryption Technologies

Providers of Encryption

FIPS 140-2 Standards Compliance

Overview

FIPS 140-2 is not a technology, but a definition of what security mechanisms should do. FIPS 140-2 is the current version of the Federal Information Processing Standardization 140 (FIPS 140) publication. FIPS 140-2 is a standard that describes US Federal government requirements that IT products should meet for Sensitive, but Unclassified (SBU) use.

The standard defines the security requirements that must be satisfied by a cryptographic module used in a security system protecting unclassified information within IT systems. FIPS 140-2 requires organizations that do business with a government agency or department that requires the exchange of sensitive information, to ensure that they meet the FIPS 140-2 security standards. In addition, the financial community increasingly specifies FIPS 140-2 as a procurement requirement.

The National Institute of Standards and Technology (NIST) issued the FIPS 140 Publication Series to coordinate the requirements and standards for cryptography modules that include both hardware and software components. Federal agencies and departments can validate that the module in use is covered by an existing FIPS 140-2 certificate. The certificate specifies the exact module name, hardware, software, firmware, and applet version numbers. For more information about FIPS 140-2, see publications at FIPS PUB 140-2 Security Requirements for Cryptographic Modules.

For configuration details about FIPS 140-2, see the following:

- "SAS/SECURE FIPS 140-2 Compliant Installation and Configuration"
- "Process to Build FIPS 140-2 Capable OpenSSL on UNIX and Linux"
- "How to Use FIPS 140-2 Capable OpenSSL Libraries with SAS 9.4M8 on UNIX and Linux"
- "FIPS 140-2 Support Is Provided with System SSL on z/OS with SAS 9.4M8"
- "Configure FIPS 140-2 Capable TLS on Windows"
- "ENCRYPTFIPS"

How SAS Implements FIPS

The ENCRYPTFIPS option is provided by SAS primarily as a mechanism to help ensure that SAS is configured to leverage the encryption algorithms and cipher suites specified by the FIPS 140-2 standard and that libraries are validated for compliance when loaded. With the ENCRYPTFIPS option enabled, SAS verifies that all of your SAS servers have been configured to use the FIPS approved Advanced Encryption Standard (AES) libraries or the TLS protocol.

However, turning off the SAS system option ENCRYPTFIPS does not impact the ability of SAS to use FIPS approved encryption algorithms such as the Advanced Encryption Standard (AES), nor does it prevent SAS from leveraging strong FIPS approved cipher suites when acting as a TLS client.

If the ENCRYPTFIPS option is turned on, then SAS server-based TLS clients attempt to load a special subset of OpenSSL libraries, contained as part of the OpenSSL FIPS Object Module. With SAS 9.4 versions prior to SAS 9.4M8, these libraries are not present by default and need to be downloaded and compiled in accordance with the specific instructions specified by the FIPS standard. Therefore, turning on this option is not generally recommended, unless absolutely required by a customer's policy.

Standard OpenSSL libraries are capable of providing strong AES level encryption. However, only the FIPS OpenSSL Object Module libraries meet the FIPS standard. If the ENCRYPTFIPS system option is enabled and the libraries are not present, then SAS produces an error when a SAS server needs to act as a TLS client and communicate over HTTPS protocol.

Note: Prior to SAS 9.4M6, only AES encryption was used to secure IOM servers when ENCRYPTFIPS was set.

Note: When the ENCRYPTFIPS option is turned on, SAS Internal Passwords are stored using the SHA256 hashing algorithm.

SAS Proprietary Encryption

SAS Proprietary Encryption Overview

SAS Proprietary Encryption is licensed with Base SAS software and is available in all deployments. It requires no additional SAS product licenses. The SAS Proprietary algorithm is strong enough to protect your data from casual viewing. However, SAS/SECURE and TLS provide higher levels of encryption.

IMPORTANT Prior to SAS 9.4M8, the cryptographic libraries that were used by SAS/SECURE were delivered as part of the SAS deployment. Starting with SAS 9.4M8, the supported cryptographic libraries must be installed on the SAS Foundation operating system to provide the higher levels of encryption for data in motion and data at rest.

There are two types of SAS Proprietary Encryption algorithms:

- A 32-bit rolling-key encryption algorithm that is used for SAS data set encryption with passwords.

  This encryption technique 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. Users must supply the appropriate passwords to authorize their access to the data, but 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.

  Note: Using AES encryption for data at rest provides a higher level of security.

  For detailed information, see "SAS Data File Encryption" in *SAS Programmer's Guide: Essentials*.

- A 32-bit fixed-key encoding used to protect passwords used for communications in configuration files, passwords for login objects, login passwords, internal account passwords, and so on.

  This SASProprietary algorithm is strong enough to protect your data from casual viewing. It provides a medium level of security.

Note: SAS recommends that you use the highest levels of security possible. Using the AES algorithm or the TLS protocol provides the highest level of encryption for data in motion.

Data-in-motion passwords that use SASProprietary encoding are passwords in transit in a logon attempt and general traffic between clients and servers. Depending on the type of client or server, higher levels of password security can be used.

Data at rest passwords are secured in the following ways:

- Login passwords on disk in the metadata can use SASProprietary (SAS002) encoding and AES (SAS003-SAS005). By default, metadata stores passwords on login objects with SAS003, but returns passwords using SAS002 by default.
- Internal account passwords on disk in the metadata can use SHA256, SHA256-10000, and MD5 hashing. By default, SHA256 is used. If MD5 hashing is specified in the configuration file, it overrides the default SHA256 hashing.

  Note: If the highest level of encryption being used is SASProprietary, MD5 hashing is used.

- Passwords on disk in configuration files can use SASProprietary (SAS002) encoding or AES (SAS003-SAS005). By default, SASProprietary (SAS002) is used.

Note: Configuration file passwords can be upgraded to AES (SAS003-SAS005). SAS003 uses a 256-bit key plus 16-bit salt value to encode passwords. SAS004 uses a 256-bit key plus 64-bit salt value to encode passwords. SAS005 uses 256-bit key plus 64-bit salt value and more iterations than SAS005.

SAS Proprietary Encryption System Requirements

SAS supports SAS Proprietary Encryption for these operating environments:

- UNIX and Linux
- Windows
- z/OS

SAS Proprietary Encryption Software Availability

SAS Proprietary Encryption is licensed with Base SAS software and is available in all deployments. It requires no additional SAS product licenses.

SAS Proprietary Encryption Configuration

SAS Proprietary Encryption is part of Base SAS. Separate installation is not required.

Configure SAS Proprietary Encryption or a higher level of encryption as follows:

- SAS Proprietary Encryption for SAS data sets is implemented with the ENCRYPT= data set option set to YES. For more information, see "SAS Proprietary Encryption" in *SAS Programmer's Guide: Essentials*.

  AES (Advanced Encryption Standard) encryption is implemented with the ENCRYPT=AES or ENCRYPT=AES2 data set option. For more information, see "AES Encryption" in *SAS Programmer's Guide: Essentials*.

  Note: Beginning with SAS 9.4M1, a metadata-bound library administrator can require that all data files in the bound library be encrypted with either AES or SAS Proprietary encryption. For more information, see "Requiring Encryption for Metadata-Bound Data Sets" in *Base SAS Procedures Guide*.

- SAS Proprietary Encryption for communications and networking is implemented by setting system option NETENCRYPTALGORITHM=SASPROPRIETARY. The NETENCRYPTALGORITHM= option must be set before the LIBNAME statement establishes the connection to the server. On the server, you set the NETENCRYPT option to specify that encryption is required by any client that accesses this server. The NETENCRYPTALGORITHM= option specifies that the algorithm or protocol set be used for encryption of all data that is exchanged with connecting clients.

  For detailed information, see "NETENCRYPT" on page 137 and "NETENCRYPTALGORITHM=" on page 138.

For an example of configuring and using SAS Proprietary Encryption in your environment, see "Using SAS Proprietary for Encryption of SAS/SHARE: Example " on page 40. For an example of configuring SAS Data File encryption using the SASProprietary algorithm, see "SAS Data File Encryption" in *SAS Programmer's Guide: Essentials*.

SAS/SECURE

IMPORTANT Prior to SAS 9.4M8, the cryptographic libraries that were used by SAS/SECURE were delivered as part of the SAS deployment. Starting with SAS 9.4M8, SAS uses the cryptographic libraries that are installed and configured on SAS Foundation operating systems to provide encryption for data in motion and data at rest. Ensure that supported cryptographic libraries are installed on the SAS Foundation operating system.

SAS/SECURE Overview

SAS/SECURE software provides industry standard encryption capabilities in addition to the SASProprietary algorithm. SAS/SECURE is provided as part of Base SAS. Prior to SAS 9.4M8, the cryptographic libraries that supported encryption were provided as part of the SAS installation.

Starting with SAS 9.4M8, SAS uses supported cryptographic libraries that are installed on the SAS Foundation operating system to provide encryption for data in motion and data at rest. Cryptographic libraries that are used for encryption are no longer delivered as part of the SAS installation.

On UNIX, Linux, Windows, and z/OS, the following encryption algorithms and the TLS protocol are supported.

- SASProprietary
- RC2
- RC4
- DES
- TripleDES
- AES
- SSL

Note: SSL is the value that is specified for SSL and TLS when setting system option NETENCRYPTALGORITIM.

Refer to “Encryption Algorithms” for more details about encryption algorithms supported by SAS.

Encryption is configured for the following:

- data in motion

  System options NETENCRYPT on page 137 and NETENCRYPTALGORITHIM on page 138 are set to configure encryption.

> **IMPORTANT** It is highly recommended that TLS is configured and used for encryption of data in motion.

- stored login passwords

  By default, the stored login passwords are stored using SAS002 (SASProprietary) encoding. You can use stronger protection for stored login passwords as long as supported cryptographic libraries are installed. The SAS003 encoding method uses AES with 16-bit salt, and SAS004 and SAS005 encoding methods use AES with 64-bit salt. However, SAS005 uses more iterations for more security.

  > **IMPORTANT** In SAS 9 releases prior to SAS 9.4M8, SAS/SECURE is the prerequisite for using SAS003-SAS005 for stored login passwords.
  >
  > Starting with SAS 9.4M8, the prerequisite for using SAS003-SAS005 for stored login passwords is having the supported cryptographic libraries on the operating systems of SAS Foundation servers..

  You can use the PWENCODE procedure (specify the METHOD= option) to upgrade passwords that use AES (SAS003-SAS005). Refer to Chapter 13, “PWENCODE Procedure,” on page 185 for details.

- internal account passwords stored in the metadata repository

  You can provide stronger protection for internal account passwords stored in the metadata repository by using a minimum of SHA256 hashing.

  > **IMPORTANT** In SAS 9 releases prior to SAS 9.4M8, SAS/SECURE is the prerequisite for using SAS003-SAS005 to encode passwords or to use SHA256 hashing and higher. If SAS/SECURE is not available, SAS002 is used for encoding and MD5 is used for password hashing.
  >
  > Starting with SAS 9.4M8, the prerequisite for using SAS003-SAS005 for password encoding and SHA256 and higher hashing is having the supported cryptographic libraries on the operating systems of SAS Foundation servers..

  CAUTION
  If you discontinue conformance with an applicable prerequisite, stored passwords are unusable and inaccessible. Do not uninstall SAS/SECURE or required cryptographic libraries without first converting all stored passwords to SAS002 format or use MD5 for hashing.

- services that are part of the Federal Information Processing Standard (FIPS) 140-2 standard

  SAS provides the ability to use the services that are part of the Federal Information Processing Standard (FIPS) 140-2 standard. When SAS system option ENCRYPTFIPS is configured, only FIPS 140-2 validated encryption and hashing algorithms from encryption libraries that are validated when loaded are used. AES is the encryption algorithm and SAS003 is the encoding format (for stored passwords) used with FIPS 140-2 enabled software. The SHA256

hashing algorithm is used with FIPS 140-2 enabled software for stored internal account passwords in the metadata server.

> **IMPORTANT** Starting with SAS 9.4M8, supported cryptographic libraries must be installed on the SAS Foundation operating systems to support FIPS 140-2.

See "FIPS 140-2 Standards Compliance" on page 15.

- AES Encryption of SAS Data Sets

  AES encryption of SAS Data Files is available with SAS 9.4. AES produces stronger encryption by using a key value that can be up to 64 characters long. Starting with SAS 9.4M5, a stronger AES key generation algorithm is available. You use ENCRYPT=AES2 data set option. Instead of passwords that are stored in the data set (SAS Proprietary encryption), AES and AES2 use a key value that is not stored in the data set. The key value is created using the ENCRYPTKEY= data set option when the data set is created. You cannot change the ENCRYPTKEY= key value on an AES encrypted data set without re-creating the data set or using PROC AUTHLIB MODIFY to change the recorded key of a metadata-bound library.

  For more information, see "SAS Data File Encryption" in *SAS Programmer's Guide: Essentials* and "Requiring Encryption for Metadata-Bound Data Sets" in *Base SAS Procedures Guide*..

SAS/SECURE System Requirements

SAS supports SAS/SECURE under these operating environments:

- UNIX and Linux
- Windows
- z/OS

SAS/SECURE Software Availability

SAS/SECURE is included with the Base SAS software. In releases prior to SAS 9.4, SAS/SECURE was an add-on product that was licensed separately.

> **IMPORTANT** Prior to SAS 9.4M8, the cryptographic libraries that were used by SAS/SECURE were delivered as part of the SAS deployment. Starting with SAS 9.4M8, SAS/SECURE uses supported cryptographic libraries that are installed and configured on SAS Foundation operating systems to provide encryption for data in motion and data at rest.

SAS/SECURE Configuration

SAS/SECURE is delivered on every SAS installation. SAS/SECURE is installed with the Base SAS software. However, the default is SAS Proprietary Encryption.

To use a higher form of encryption for communications and networking (data in motion), specify system option NETENCRYPT on page 137 and set NETENCRALG on page 138to a value of RC2, RC4, DES, TRIPLEDES, AES, or SSL.

SAS/SECURE FIPS 140-2 Compliant Installation and Configuration

To configure a FIPS 140-2 compliant system, you must use cryptographic libraries that are FIPS certified and set ENCRYPTFIPS system option to AES or SSL. When ENCRYPTFIPS is specified, an INFO message is written at server start-up to indicate that FIPS encryption is enabled.

When FIPS is properly enabled, you can connect only to servers and clients that are configured with AES or SSL in system option NETENCRYPTALG. Errors are generated when other encryption algorithms are specified.

Prior to SAS 9.4M8, the following applies to the cryptographic libraries that were shipped for use with SAS/SECURE for AES encryption.

- For AES encryption on UNIX and Windows, the cryptographic libraries are FIPS compliant.

  Prior to SAS 9.4M8, on Microsoft Windows and UNIX platforms SAS uses RSA Crypto-C ME version 4.01 that has a certificate number of 2056. For more information, see Cryptographic Module Validation Program.

- For AES encryption on z/OS, the cryptographic libraries are FIPS compliant. SAS uses the z/OS system libraries.

In the FIPS 140-2 compliant mode, the SHA256 hashing algorithm is used for stored password protection. You can connect only to servers and clients that are also enabled for FIPS 140-2.

CAUTION
With SAS 9.2, the password hash list was created using the MD5 hash algorithm. If you are moving from SAS 9.2 to a higher version of SAS and configuring your system to be FIPS 140-2 compliant, you need to clear all previously stored passwords. When you reset the passwords, they use the SHA256 hashing algorithm.

See the following information for details about FIPS:

- “ENCRYPTFIPS” on page 135
- “NETENCRYPTALGORITHM=” on page 138
- “FIPS 140-2 Standards Compliance” on page 15

Transport Layer Security (TLS)

IMPORTANT Prior to SAS 9.4M8, the OpenSSL cryptographic libraries that were used for TLS were delivered as part of the SAS deployment. Starting with SAS 9.4M8, SAS uses the cryptographic libraries that are installed and configured on SAS Foundation operating systems to provide encryption for data in motion and data at rest. Ensure that supported cryptographic libraries are installed on the SAS Foundation operating system.

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 is a protocol that provides 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 more information about client and server negotiations using certificates, refer to "Certificate Implementation: How TLS Client and Servers Negotiate" on page 72.

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 under these operating environments:

- UNIX and Linux
- Windows

- z/OS

TLS and OpenSSL Version Support with SAS 9.4M8

SAS deployments on Windows, UNIX, Linux, and z/OS platforms can be configured to use TLS. The implementation and file extensions, however, vary based on the operating system.

TLS 1.2 is the minimum TLS version supported for SAS 9.4. Earlier versions of TLS and SSL are unsecure. Contact Technical Support if you need an earlier version of TLS.

Starting with SAS 9.4M8, you can set environment variable SSLMINPROTOCOL to TLS 1.2 or TLS 1.3. SSLMODE= is set by default to SSLMODESP800131A to support the highest level of TLS 1.2 and TLS 1.3 ciphers.

- On Windows, SAS uses the Windows Secure Channel (Schannel) library that comes with the Windows operating system for TLS encryption.
- On UNIX and Linux, starting with SAS 9.4M8, SAS uses the OpenSSL libraries that are provided on the operating system. Prior to SAS 9.4M8, SAS provided the OpenSSL libraries that are needed to provide TLS on UNIX and Linux. SAS tests the software with the following versions of OpenSSL libraries.

  Table 3.1 *OpenSSL Version Support for TLS 1.2 and TLS 1.3 with SAS 9.4M8 for Linux*

OpenSSL Library Supported	TLS Version Supported
OpenSSL 3	TLS 1.2, TLS 1.3
OpenSSL 1.1.1	TLS 1.2, TLS 1.3
OpenSSL 1.1.0	TLS 1.2
OpenSSL 1.0.2	TLS 1.2

- Prior to SAS 9.4M8, SAS provided the OpenSSL libraries that were used to provide TLS for z/OS. Starting with SAS 9.4M8, SAS uses the GSK libraries that are part of IBM System SSL to provide TLS support on z/OS. See “IBM System SSL Provides OpenSSL Capabilities for z/OS in SAS 9.4M8”.

TLS and OpenSSL Version Support Prior to SAS 9.4M8

TLS 1.2 is the minimum TLS version supported for SAS 9.4. Earlier versions of TLS and SSL are unsecure. Contact Technical Support if you need an earlier version of TLS.

Starting with SAS 9.4M5, TLS version 1.2 is the default TLS protocol supported. Prior to SAS 9.4M5, to configure SAS to use TLS 1.2, set environment variable SAS_SSL_MIN_PROTOCOL= to TLS 1.2. See "SAS_SSL_MIN_PROTOCOL=" on page 179.

Note: For SAS 9 releases prior to SAS 9.4M3, apply security hot fixes and set the SAS_SSL_MIN_PROTOCOL environment variable to TLS 1.2.

For SAS 9.4 and all maintenance releases of SAS 9.4 prior to SAS 9.4M8, updated versions of OpenSSL are provided and are updated as needed through hot fixes. See the SAS Security Bulletin, OpenSSL Security Advisories for the most current information about the versions of OpenSSL used with SAS products and about the advisories under consideration for software fixes.

The following table provides a quick reference of the OpenSSL versions supported for UNIX and Linux for each version of SAS Foundation prior to SAS 9.4M8.

Table 3.2 *OpenSSL Version and Library Files Provided Prior to SAS 9.4M8*

SAS Version	OpenSSL Version Library	Library Files
9.4M6 hot fix	1.0.2r	libcrypto.so.1.0.0, libssl.so.1.0.0
9.4 - 9.4M5b hot fixes	1.0.2n	libcrypto.so.1.0.0, libssl.so.1.0.0
9.3 - 9.3M2 hot fixes	1.0.2n	libcrypto.so.1.0.0, libssl.so.1.0.0
9.2M3 hot fix	0.9.8zh	not applicable

The OpenSSL libraries that are supplied by SAS on UNIX and Linux are external libraries. You can use OpenSSL to compile your own libraries.

Note: Different operating systems require the use of different library file extensions. For example, HP-UX, Linux, and Solaris use libcrypto.so.1.0.0 and libssl.so.1.0.0. AIX uses libcrypto.so and libssl.so. Refer to your operating system vendor documentation when using the vendor's OpenSSL libraries. There might be additional procedures that need to be followed to make the libraries work properly in your environment.

The OpenSSL libraries that are shipped with SAS 9.4 for UNIX and Linux are not FIPS 140-2 compliant. However, you can compile a FIPS 140-2 compliant version of OpenSSL and install it. See "Process to Build FIPS 140-2 Capable OpenSSL on UNIX and Linux" on page 104.

TLS Configuration

Prior to SAS 9.4M8, encryption libraries for UNIX, Linux, z/OS, and Windows are shipped with SAS Foundation. No additional software installation is required. For SAS 9.4 and all maintenance releases of SAS 9.4 prior to SAS 9.4M8, updated versions of OpenSSL are provided and updated through hot fixes. See OpenSSL

Security Advisories for the latest information about OpenSSL security advisories under consideration for software fixes for SAS components.

Starting with SAS 9.4M8, SAS uses the supported cryptographic libraries that are provided on the operating system to provide encryption for data in motion.

Starting with SAS 9.4M3, the SAS Deployment Manager is used as the interface to add and remove intermediate, root, or self-signed CA certificates. At installation, a list of CA certificates that are distributed by Mozilla software products is installed in the cacerts.pem and trustedcerts.pem files. The SAS Deployment Manager can then be used to add certificates to the truststore and to remove digital certificates from the truststore.

The trusted certificate list is a file in X.509 Base-64 encoded format and is named trustedcerts.pem. System option SSLCALISTLOC= points to the location of this file. The path to the certificates is `<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem`.

> **IMPORTANT** Starting with SAS 9.4M3, the Mozilla bundle of CA certificates that is part of the truststore contains an expired certificate from Sectigo. There is a hot fix that can be applied to SAS 9.4m3 - SAS 9.4M6 to resolve this issue. See "TLS Certificate Verification: ERROR, Certificate Has Expired" on page 203.

Starting with SAS 9.4M6, TLS is supported on Integrated Object Model (IOM) servers and server processes that provide IOM Bridge access. These servers and processes are as follows:

- SAS Metadata Server
- SAS OLAP Server
- workspace server
- SAS Stored Process Server
- pooled workspace server
- object spawner

To configure TLS on IOM servers, see information in TLS support for IOM Servers.

The instructions that you use to configure certificates for TLS at your site depend on whether you use UNIX or Linux, Windows, or z/OS. See the appropriate details:

- Chapter 7, "Configure TLS Certificates and FIPS on UNIX and Linux," on page 77
- Chapter 8, "Configure TLS Certificates and FIPS on Windows," on page 107
- Chapter 9, "Configure TLS Certificates and FIPS on z/OS," on page 119

For examples of configuring and using TLS in your environment prior to SAS 9.4M8, see Chapter 4, "Encryption Examples," on page 39.

Configure TLS to Run in FIPS 140-2 Compliant Mode

You can configure TLS to run in FIPS compliant mode with the following guidance.

- Prior to SAS 9.4M8, the libraries that are supplied by Windows for use with SAS are FIPS 140-2 compliant. "Configure FIPS 140-2 Capable TLS on Windows" on page 117.
- Prior to SAS 9.4M8, the OpenSSL libraries that were shipped with SAS for UNIX and Linux are not FIPS 140-2 compliant. However, you can compile a FIPS 140-2 compliant version of OpenSSL and install it. For more information, see "Process to Build FIPS 140-2 Capable OpenSSL on UNIX and Linux" on page 104.
- Prior to SAS 9.4M8, for TLS encryption on z/OS, SAS uses OpenSSL libraries and these libraries are not FIPS compliant. OpenSSL libraries for use with z/OS cannot be replaced and are therefore not FIPScompliant libraries.

  Note: However, you can continue to use SAS/SECURE to provide FIPS on z/OS by setting NETENCRYPTALG to AES.

- Starting with SAS 9.4M8, SAS uses the supported cryptographic libraries that are provided on the operating system.For more information, see "FIPS 140-2 Support with SAS 9.4M8 on UNIX and Linux".
- Starting with SAS 9.4M8, SAS uses System SSL on z/OS. The cryptographic libraries that support System SSL are installed on the operating system. System SSL also provides FIPS 140-2 support . See "FIPS 140-2 Support Is Provided with System SSL on z/OS with SAS 9.4M8".

After compiling and installing the FIPS compliant libraries, set the NETENCRYPTALGORITHIM system option to AES or SSL and set the ENCRYPTFIPS system option. You can then run in FIPS compliant mode and connect to other servers and clients that are also configured with FIPS.

CAUTION

Use ENCRYPTFIPS with caution. Turning on the ENCRYPTFIPS option is not generally recommended for TLS, unless absolutely required by your site's policy. If the ENCRYPTFIPS option is turned on, the SAS server-based TLS clients attempt to load a special subset of OpenSSL libraries, contained as part of the OpenSSL FIPS Object Module. Because these libraries are not present by default, you must follow the process described in "Process to Build FIPS 140-2 Capable OpenSSL on UNIX and Linux" on page 104.

When ENCRYPTFIPS is specified, an INFO message is written at server start-up to indicate that FIPS encryption is enabled.

IBM z/OS Pervasive Encryption for Data Sets with SAS 9.4M8

Pervasive encryption for IBM Z platforms enables extensive encryption of data-at-rest. Starting with SAS 9.4M8, support for z/OS Pervasive Encryption is available for SAS direct access bound libraries and sequential access libraries. The access method that SAS uses to read and write data sets for z/OS is Execute Channel Program (EXCP).

Pervasive Encryption support is available only in the SAS 31bit (MVS) offering. z/OS data set encryption currently supports only data-encrypting keys that are using the AES algorithm with a 256-bit key length.

See the following documents for information needed to use and configure z/OS Pervasive Encryption:

- To use Pervasive Encryption on SAS bound libraries, you must first set up the feature using information in IBM Data Set Encryption.
- Additional information can be found in Getting Started with z/OS Data Set Encryption.
- Additional information is also described in Using the z/OS data set encryption enhancements.
- To use z/OS Pervasive Encryption with SAS bound libraries, an encryption key label must be specified through either the DSKEYLBL= LIBNAME option or the DLDSKEYLBL system option.

PROC DATASETS and PROC CONTENTS include fields that denote that a library is operating system (OS) encrypted.

SAS libraries can also be encrypted using data set encryption that is included in Base SAS. Some differences between SAS data set encryption and Pervasive Encryption are as follows:

- Specifying ENCRYPT= and ENCRYPTKEY= data set options encrypts the underlying stored member data, but not the directory information needed to keep track of library layouts. Pervasive Encryption encrypts 100% of the data in the library.

  SAS data set encryption also requires that users know and provide encryption keys and passwords to unlock encrypted SAS data sets.

  For more information, see “SAS Data File Encryption” in *SAS Programmer’s Guide: Essentials*.
- IBM z/OS Pervasive Encryption is a single source solution to encryption and the required key and password management. Pervasive encryption allows the system administrator to control and track encryption of data sets that are stored and not stored with SAS and libraries using the same approach and system tools. Pervasive Encryption keys are system protected resources (in RACF, ICSF, and CPACF), and users are given access to the associated label names for access. The actual key values are stored in electromagnetically protected hardware. Users do not have to provide encryption keys and passwords to unlock encrypted data.

- Data sets that are protected using Pervasive Encryption are tied to the current host machine. These encrypted data sets are not physically transferable (by removing disk drives) without the accompanying encryption keys.

For more information, see Configuration Guide for SAS® 9.4 Foundation for z/OS and System Requirements for SAS® 9.4 Foundation for z/OS.

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.

Only Windows and UNIX operating systems can access an OpenSSH server on another UNIX system. To access an OpenSSH server, UNIX systems require OpenSSH software.

Windows systems require PuTTY software.

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

SAS also supports SSH on z/OS. The IBM Ported Tools for z/OS Program Product must be installed for OpenSSH support. See IBM Ported Tools for z/OS - OpenSSH.

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 these operating environments:

- UNIX
- Windows
- z/OS

SSH Software Availability

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

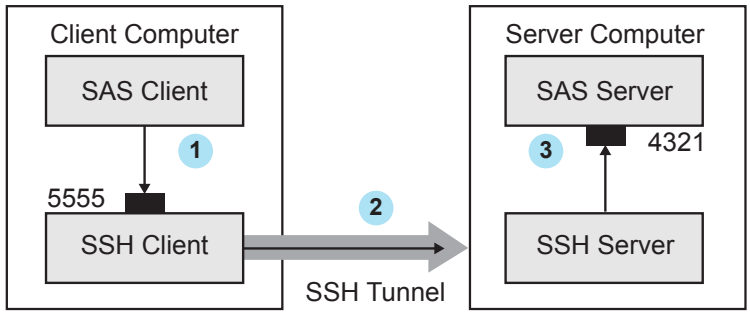
To build the OpenSSL software, refer to the following resources:

- www.openssh.com
- www.ssh.com
- SSH UNIX manual page
- IBM Ported Tools for z/OS - OpenSSH
- PuTTY Download Page
- Barrett, Daniel J., Richard E. Silverman, and Robert G. Byrnes. 2005. *SSH, the Secure Shell: The Definitive Guide.* Sebastopol, CA: O'Reilly
- Configuring SSH Client Software in UNIX and Windows Environments for Use with the SFTP Access Method in SAS 9.2, SAS 9.3, and SAS 9.4

SSH Tunneling Process

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

Figure 3.1 *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 UNIX, you use OpenSSH software to access your UNIX OpenSSH server.
   - On Windows, you use PuTTY software to access your UNIX OpenSSH server.
   - On z/OS, the IBM Ported Tools for z/OS Program Product must be installed for OpenSSH support.
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.

For examples of setting up and using a tunnel, see “Using SSH Tunneling for SAS/CONNECT: Example ” on page 60 and “Using SSH Tunneling for SAS/SHARE: Example ” on page 61.

Encryption Algorithms

The following encryption algorithms are supported with Foundation SAS.

Note: Starting with SAS 9.4M8, the following algorithms are supported on Foundation SAS servers using the cryptographic libraries that are provided on the operating system.

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.

Note: This algorithm is not FIPS 140-2 compliant.

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.

Note: This algorithm is not FIPS 140-2 compliant.

RC2
is a *block cipher* encryption algorithm that divides a message into blocks of 64 bits and encrypts each block. The RC2 key size ranges from 8 to 128 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.

Note: This algorithm is not FIPS 140-2 compliant.

RC4
is a stream cipher and symmetric key algorithm. A *stream cipher* is an encryption algorithm that encrypts data one byte at a time. RC4 supports variable key sizes that range from 40 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.

Note: This algorithm is not FIPS 140-2 compliant.

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).

Note: This algorithm is not FIPS 140-2 compliant.

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 three separate 56-bit keys. 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.

Note: TripleDES is a FIPS 140-2 compliant encryption algorithm.

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).

Note: AES is a FIPS 140-2 compliant encryption algorithm.

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 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.

Note: RSA is a FIPS 140-2 compliant signing algorithm.

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 algorithm 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.

Note: DSA is a FIPS 140-2 compliant signing algorithm.

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.

SHA1 (Secure Hash Algorithm)
produces a 160-bit (20-byte) hash value. An SHA1 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.

Note: SHA1 is a FIPS 140-2 compliant hashing algorithm.

SHA256 (Secure Hash Algorithm)
is essentially a 256-bit block cipher algorithm that encrypts the intermediate hash value using the message block as key. SHA stands for Secure Hash Algorithm. 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.

Note: SHA256 is a FIPS 140-2 compliant hashing algorithm.

Comparison of Encryption Technologies

The following table compares the features of encryption technologies. Note that starting with SAS 9.4M8, for Foundation SAS servers SAS uses the cryptographic libraries that are provided on the operating system to provide encryption. SAS no longer provides the cryptographic libraries that are used by SAS/SECURE and TLS as part of the SAS 9.4M8 installation.

Table 3.3 *Summary of SAS Proprietary, SAS/SECURE, TLS, and SSH Features*

Features	SAS Proprietary	SAS/SECURE	TLS	SSH
Encryption and authentication	Encryption only	Encryption only	Encryption and authenticatio n	Encryption only
Encryption level	Medium	High	High	High
Algorithms supported	SAS Proprietary fixed encoding SAS Proprietary 32-bit rolling key encoding	RC2, RC4, DES, TripleDES, AES. Default is SAS Proprietary	RC2, RC4, DES, TripleDES, AES, and the TLS protocol	Product dependent
Installation required	No (part of Base SAS)	No (delivered with Base SAS prior to SAS 9.4M8)	Delivered with Base SAS prior to SAS 9.4M8.	Yes
Operating environments supported	UNIX/Linux Windows z/OS	UNIX/Linux Windows z/OS	UNIX/Linux Windows z/OS	UNIX/Linux Windows z/OS
SAS version support	8 and later	8 and later	9 and later	8.2 and later

Encryption: Implementation

The implementation of the installed encryption technology depends on the environment that you work in. If you work in a SAS enterprise intelligence infrastructure, encryption might be transparent to you because it has already been configured into your site's overall security plan. After the encryption technology has been installed, the site system administrator configures the encryption method (level of encryption) to be used in all client/server data exchanges. All enterprise activity uses the chosen level of encryption, by default.

If you work in a SAS session on a client computer that exchanges data with a SAS server, specify SAS system options that implement encryption for the duration of the SAS session. If you connect a SAS/CONNECT client to a spawner, specify encryption options in the spawner start-up command. For details about SAS system options, see "SAS System Options for Encryption" on page 133. For examples, see Chapter 4, "Encryption Examples," on page 39.

Encryption: SAS Logging Facility

Security-related events are now logged as part of the system-wide logging facility. If the LOGCONFIGLOC= system option is specified when SAS starts, logging is performed by the SAS logging facility. The following table lists security-related loggers.

Note: When the SAS Deployment Manager add and remove certificate tasks are used to manage the truststore, the logs generated are located at `<SASHOME>/InstallMisc/InstallLogs/certframe*`.

Table 3.4 *Selected Security-Related Loggers*

Logger	SAS/SECURE Information
App.tk.eam	Logs security information.
App.tk.eam.ssl	Logs TLS encryption information including the OpenSSL protocol and cipher suites being used.
App.tk.eam.sas	Logs SAS Proprietary encryption information.
App.tk.eam.rsa	Logs RC2, RC4, DES, DES3, and AES encryption information.

Logger	SAS/SECURE Information
App.tk.eam.rsa.pbe	Enables or disables the password-based encryption processing that creates a key.
App.tk.eam.rsa.capi	Logs RC2, RC4, DES, and DES3 encryption information for Windows C API.
App.tk.eam.rsa.cc	Logs RC2, RC4, DES, DES3, and AES encryption information for BSAFE Crypto-C.
App.tk.eam.rsa.ccme	Logs AES encryption information for BSAFE Crypto-C ME. This log is for FIPS.
App.tk.eam.rsa.icsf	Logs AES encryption information for IBM Integrated Cryptographic Service Facility (ICSF). This log is for FIPS.
App.tk.eam.ssl.gsk	Logs TLS information specific to using System SSL on z/OS.

Note: Prior to SAS 9.4M8, on z/OS if the SAS Logging Facility loggers App.tk.eam.ssl or App.tk.eam.rsa are in DEBUG or TRACE levels, SAS writes the debug file to the location specified by the TKELBOX_CRYPTO_DEBUG_LOG variable in the TKMVSENV file. If the specified file name is not found in TKMVSENV, then SAS saves the file in either `/tmp/sas.rsabxdbg.<process_id>.log` for RC2, RC4, DES, and TRIPLEDES, or in `/tmp/sas.sslbxdbg<process_id>.log` for TLS.

Starting with SAS 9.4M8, on z/OS log output is directed to the location specified by LOGCONFIGLOC=.

Note: Starting with SAS 9.4M8, in addition to the SAS logging facility, you can also set additional SAS system options to trace and debug z/OS System SSL. Specify SAS SSLGSKTRACE, SSLHWDETECTMESSAGE, SSLICSFERRORMESSAGE, and SSLGSKTRACEFILE. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8".

See Also

For more information, see "Overview of the SAS Logging Facility" in *SAS Logging: Configuration and Programming Reference*

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= system option are NONE or HIGH.

Starting with SAS 9.4M8, when hot fixes are applied, AES 256-bit encryption is used to encrypt ODS generated PDF files when system option PDFSECURITY=HIGH. For more information, see SAS Note 70305.

Note: Adobe Acrobat 9 or later or Adobe Reader 9 or later is required to open the AES encrypted document.

Prior to SAS 9.4M8, ODS encrypts PDF files using RC4 128-bit encryption.

IMPORTANT A warning might be generated when older encryption algorithms are not available to the operating system.

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.

For more information, see "PDF Security" in *SAS Output Delivery System: User's Guide*, "Securing ODS-Generated PDF Files" in *SAS Output Delivery System: User's Guide*, and "PDFSECURITY=" in *SAS System Options: Reference*.

4

Encryption Examples

Overview

The examples in this section work with SAS 9.4 and all maintenance releases unless specified otherwise. Starting with SAS 9.4M8, SAS uses IBM System SSL for z/OS data in motion. There are new SAS system options for use with System SSL, and deprecated system options that were used with z/OS prior to SAS 9.4M8. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8", "SAS System Options Deprecated for Use with SAS 9.4M8 on z/OS", and "IBM System SSL Provides OpenSSL Capabilities for z/OS in SAS 9.4M8".

Using SAS Proprietary for Encryption of SAS/SHARE: Example

SAS/SHARE Client

In this example, the NETENCRYPTALGORITHM= option is set to sasproprietary to specify the use of the proprietary algorithm to encrypt the data between the client and the server. The NETENCRYPTALGORITHM= option must be set before the LIBNAME statement establishes the connection to the server.

```
options netencryptalgorithm=sasproprietary;
options comamid=tcp;
libname sasdata 'edc.prog2.sasdata' server=rmthost.share1;
```

SAS/SHARE Server

This example shows how to set the options for encryption services on a SAS/SHARE server. The NETENCRYPT option specifies that encryption is required by any client that accesses this server. The NETENCRYPTALGORITHM= option specifies that the SAS Proprietary Encryption algorithm (value is sasproprietary) be used for encryption of all data that is exchanged with connecting clients.

```
options netencrypt netencryptalgorithm=sasproprietary;
options comamid=tcp;
proc server id=share1;
run;
```

Using the RC4 Algorithim for Encryption of SAS/CONNECT: Example

SAS/CONNECT Client on UNIX

The following statements configure the client. The NETENCRYPTALGORITHM= option specifies the use of the RC4 algorithm.

```
options netencryptalgorithm=rc4;
options remote=unxnode comamid=tcp;
signon;
```

SAS/CONNECT Server on UNIX

The following command starts a spawner on the computer that runs the server. The -NETENCRYPT option specifies that encryption is required for all clients that connect to the spawner. The -NETENCRYPTALGORITHM option specifies the use of the RC4 algorithm for encrypting all network data. The -SASCMD option specifies the SAS start-up command.

```
cntspawn -service spawner -netencrypt -netencryptalgorithm rc4 -sascmd mystartup
```

The spawner executes a UNIX shell script that executes the commands to start SAS.

```
#!/bin/ksh
#_________________
# mystartup
#_________________
. ~/.profile
```

```
sas dmr -noterminal -comamid tcp $*
```

Using TLS for Encryption of a SAS/CONNECT UNIX Spawner: Example

Start-up of a UNIX Spawner on a SAS/CONNECT Server

After digital certificates are generated for the CA, the server, and the client, and a CA trust list for the client is created, you can start a UNIX spawner program that runs on a server that SAS/CONNECT clients connect to. The spawner acts both as a TLS server to CONNECT clients and a TLS client to the spawned CONNECT server.

The following example code starts the spawner using TLS encryption and specifies a private password that must be provided either through prompting or within a file:

```
% cntspawn -service unxspawn -netencryptalgorithm ssl
-sslcertloc /users/server/certificates/server.pem
-sslpvtkeyloc /users/server/certificates/serverkey.pem
-sslpvtkeypass starbuck1
-sslcalistloc /users/server/certificates/sas.pem
-sascmd /users/server/command.ksh
```

Note: Starting in the third maintenance release of SAS, this option might not be needed if you are managing certificates using the SDM.

The following table explains the SAS commands that are used to start a spawner on a SAS/CONNECT single-user server.

Table 4.1 *SAS Commands and Arguments for Spawner Start-Up Tasks*

SAS Commands and Arguments	Function
CNTSPAWN	Starts the spawner
-SERVICE *unxspawn*	Specifies the spawner service (configured in the services file)
-NETENCRYPTALGORITM *SSL*	Specifies the TLS encryption algorithm

SAS Commands and Arguments	Function
-SSLCERTLOC */users/server/certificates/server.pem*	Specifies the file path for the location of the server's public certificate
-SSLPVTKEYLOC */users/server/certificates/serverkey.pem*	Specifies the file path for the location of the server's private key
-SSLPVTKEYPASS *password*	Specifies the password to access the server's private key if the private key is encrypted with a password
-SSLCALISTLOC */users/server/certificates/sas.pem*	Specifies the CA trust list **Note:** Starting in the third maintenance release of SAS, this option might not be needed if you are managing certificates using the SDM.
-SASCMD */users/server/command.ksh*	Specifies the name of an executable file that starts a SAS session when you sign on without a script file

In order for the UNIX CONNECT server to locate the appropriate server digital certificate, you must specify either the -SSLCERTLOC, -SSLPVTKEYLOC, and -SSLPVTKEYPASS options or the -SSLPKCS12LOC and -SSLPKCS12PASS options in the script that is specified by the -SASCMD option.

Here is an example of an executable file:

```
#!/bin/ksh
#----------------------------------
# mystartup
#----------------------------------

. ~/.profile
sas -noterminal -sslcertloc /users/server/certificates/server.pem
-sslpvtkeyloc /users/server/certificates/serverkey.pem $*
#-----------------------------
```

For complete information about starting a UNIX spawner, see *SAS/CONNECT User's Guide*.

Connection of a SAS/CONNECT Client to a UNIX Spawner

After a UNIX 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;
```

```
options sslcalistloc="/users/johndoe/certificates/sas.pem";
%let machine=unxspawn;
signon machine.spawner user=_prompt_;
```

The following table explains the SAS options that are used to connect to a SAS/CONNECT server.

Table 4.2 SAS Options, Statements, and Arguments for Client Access to a SAS/CONNECT Server

SAS Options, Statements, and Arguments	Client Access Tasks
NETENCRYPTALGORITHM=*SSL*	Specifies the encryption algorithm
SSLCALISTLOC=*sas.pem*	Specifies the CA trust list
SIGNON=*unxspawn*	Specifies the server and service to connect to
USER=_PROMPT_	Prompts for the user ID and password to be used for authenticating the client to the server

The server-ID and the server's Common Name, which was specified in the server's digital certificate, must be identical.

For complete information about connecting to a UNIX spawner, see *SAS/CONNECT User's Guide*.

Using TLS for Encryption of a SAS/CONNECT Windows Spawner: Example

Start-Up of a Windows Spawner on a Single-User SAS/CONNECT Server

After digital certificates for the CA, the server, and the client have been generated and imported into the appropriate Certificate Store, you can start a spawner program that runs on a server that SAS/CONNECT clients connect to.

Here is an example of how to start a Windows spawner on a SAS/CONNECT server. From **<SASHome>\SASFoundation\9.4**, execute the following command:

```
cntspawn -install -netencryptalgorithm ssl -sslcertsubj "apex.pc.com"
```

```
-sascmd mysas.bat -servuser userid -servpass password
```

The following table shows the SAS commands that are used to start a spawner on a SAS/CONNECT single-user server.

Table 4.3 *SAS Commands and Arguments for Spawner Start-Up Tasks*

SAS Command and Arguments	Function
CNTSPAWN	Starts the spawner.
-INSTALL	Causes an instance of a spawner to be installed as a Windows service. For information about the -INSTALL option, see "Spawner Options" in *SAS/CONNECT User's Guide*.
-NETENCRYPTALGORITHM *SSL*	Specifies the TLS encryption algorithm.
-SSLCERTSUBJ "*apex.pc.com*"	Specifies the subject name that is used to search for a certificate from the Microsoft Certificate Store.
-SASCMD *mysas.bat*	Specifies the name of an executable file that starts a SAS session when you sign on without a script file.
-SERVUSER *user-ID*	Specifies the *user-ID* to be used to start the spawner and to obtain a digital certificate. The -SERVUSER and -SERVPASS options are used together and must be specified when the spawner is installed as a service (the -INSTALL option is specified). For information about the -SERVUSER option, see "Spawner Options" in *SAS/CONNECT User's Guide*.
-SERVPASS *password*	Specifies the *password* to be used to start the spawner and to obtain a digital certificate. The -SERVUSER and -SERVPASS options are used together and must be specified when the spawner is installed as a service (the -INSTALL option is specified). For information about the -SERVPASS option, see "Spawner Options" in *SAS/CONNECT User's Guide*.

In order for the Windows spawner to locate the appropriate server digital certificate in the Microsoft Certificate Store, you must specify the -SSLCERTSUBJ system option in the script that is specified by the -SASCMD option. -SSLCERTSUBJ specifies the subject name of the digital certificate that TLS should use. The subject

that is assigned to the -SSLCERTSUBJ option and the computer that is specified in the client sign-on must be identical.

Note: You can also use the SSLCERTISS= and SSLCERTSERIAL= options instead of the SSLCERTSUBJ= option to identify a digital certificate.

If the Windows spawner is started as a service, the -SERVPASS and -SERVUSER options must also be specified in the Windows spawner start-up command in order for TLS to locate the appropriate CA digital certificate.

For complete information about starting a Windows spawner, see *SAS/CONNECT User's Guide*.

Connection of a SAS/CONNECT Client to a Windows Spawner on a SAS/CONNECT Server

After a spawner has been started on a SAS/CONNECT server, a SAS/CONNECT client can connect to it.

Here is an example of how to make a client connection to a Windows spawner that is running on a SAS/CONNECT server:

```
options netencryptalgorithm=ssl;
%let machine=apex.pc.com;
signon machine.unxspawn user=_prompt_;
```

The computer that is specified in the client sign-on and the subject (the -SSLCERTSUBJ option) that is specified at the server must be identical.

The following table shows the SAS options that are used to connect to a Windows spawner that runs on a SAS/CONNECT server.

Table 4.4 *SAS Options, Statements, and Arguments for Client Access to a SAS/CONNECT Server*

SAS Options, Statements, and Arguments	Function
NETENCRYPTALGORITHM=*SSL*	Specifies the encryption algorithm
SIGNON=server-ID	Specifies which server to connect to
USER=_PROMPT_	Prompts for the user ID and password to be used for authenticating the client to the server

The server-ID and the server's Common Name, which was specified in the server's digital certificate, must be identical.

Using TLS for Encryption of a z/OS Spawner on a SAS/CONNECT Server: Example

Note: This example works prior to SAS 9.4M8. Starting with SAS 9.4M8, SAS used IBM System SSL for z/OS data in motion. There are new SAS system options for use with System SSL, and deprecated system options that were used with z/OS prior to SAS 9.4M8. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8", "SAS System Options Deprecated for Use with SAS 9.4M8 on z/OS", and "IBM System SSL Provides OpenSSL Capabilities for z/OS in SAS 9.4M8".

Start-Up of a z/OS Spawner on a SAS/CONNECT Server

After digital certificates are generated for the CA, the server, and the client, and a CA trust list for the client is created, you can start a z/OS spawner program that runs on a server that SAS/CONNECT clients connect to.

Note: Starting in the third maintenance release of SAS 9.4, you can use the SDM to manage your certificates. The SSLCALSTLOC defaults to **<SASRoot>/SASHome/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem** and is set at SAS installation in the z/OS common options template. Therefore, you no longer need to specify the -SSLCALISTLOC option.

For example:

```
//SPAWNER  EXEC PGM=CNTSPAWN,
//         PARM='-service 4321 =<//DDN:SYSIN'
//STEPLIB  DD   DISP=SHR,DSN=<customer.high.level.pfx>.LIBRARY
//STEPLIB  DD   DISP=SHR,DSN=<customer.high.level.pfx>.LIBE
//SYSPRINT DD   SYSOUT=*
//SYSTERM  DD   SYSOUT=*
//TKMVSJNL DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//SYSIN    DD   *
-netencryptalgorithm ssl
-sslpkcs12loc /users/server/certificates/server.p12
-sslpkcs12pass starbuck1
-sslcalistloc /users/server/certificates/sas.pem
-sascmd /users/server/command.sh
```

The following table explains the SAS commands that are used to start a spawner on a SAS/CONNECT server.

Table 4.5 SAS Commands and Arguments for Spawner Start-Up Tasks

SAS Commands and Arguments	Function
CNTSPAWN	Starts the spawner
-SERVICE *4321*	Specifies the spawner service that is listening on port 4321
- NETENCRYPTALGORITHM *SSL*	Specifies the TLS encryption algorithm
-SSLPKCS12LOC */users/server/certificates/serverkey.p12*	Specifies the file path for the location of the server's PKCS#12 DER encoding package
-SSLPKCS12PASS *password*	Specifies the password to access the server's private key in the PKCS#12 package
-SSLCALISTLOC */users/server/certificates/sas.pem*	Specifies the CA trust list. **Note:** Starting in the third maintenance release of SAS 9.4, if you are using the SDM to manage your certificates, you no longer need to specify this command.
-SASCMD */users/server/command.sh*	Specifies the name of an executable file that starts a SAS session when you sign on without a script file

In order for the z/OS spawner to locate the appropriate server digital certificate, you must specify either the -SSLCERTLOC, -SSLPVTKEYLOC, and -SSLPVTKEYPASS options or the -SSLPKCS12LOC and -SSLPKCS12PASS options in the script that is specified by the -SASCMD option.

Here is an example of an executable file, **`command.sh`**:

```
#!/bin/sh
args=$*
if [ -n "$NETENCRALG" ] ; then
   args="$args -netencralg $NETENCRALG"
fi
if [ -n "$SASDAEMONPORT" ] ; then
   args="$args -sasdaemonport $SASDAEMONPORT"
fi
if [ -n "$SASCLIENTPORT" ] ; then
   args="$args -sasclientport $SASCLIENTPORT"
fi
export TSOOUT=
export SYSPROC=SAS.CLIST
/bin/tso -t %sas -dmr -noterminal
-sslpkcs12loc /users/server/certificates/serverkey.p12
```

```
-sslpkcs12pass password $args
```

For complete information about starting a z/OS spawner, see *SAS/CONNECT User's Guide*.

Connection of a SAS/CONNECT Client to a z/OS Spawner

After a z/OS 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 comamid=tcp netencryptalgorithm=ssl;
options sslcalistloc="/users/johndoe/certificates/sas.pem";
%let machine=apex.server.com;
signon machine.4321 user=_prompt_;
```

The following table explains the SAS options that are used to connect to a SAS/CONNECT server.

Table 4.6 *SAS Options and Arguments for Client Access to a SAS/CONNECT Server*

SAS Options and Arguments	Client Access Tasks
COMAMID=TCP	Specifies the TCP/IP access method
NETENCRYPTALGORITHM=*SSL*	Specifies the encryption algorithm
SSLCALISTLOC=*sas.pem*	Specifies the CA trust list
SIGNON=*server-ID.service*	Specifies the server and service to connect to
USER=_PROMPT_	Prompts for the user ID and password to be used for authenticating the client to the server

The server ID and the server's Common Name, which was specified in the server's digital certificate, must be identical.

For complete information about connecting to a z/OS spawner, see *SAS/CONNECT User's Guide*.

Using System SSL for Encryption of SAS/CONNECT with SAS 9.4M8: Example

Starting with SAS 9.4M8, SAS uses IBM System SSL for z/OS data in motion. There are new SAS system options for use with System SSL and, deprecated system options that were used with z/OS prior to SAS 9.4M8. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8", "SAS System Options Deprecated for Use with SAS 9.4M8 on z/OS", and "IBM System SSL Provides OpenSSL Capabilities for z/OS in SAS 9.4M8".

SAS 9.4M8 Start-Up of a z/OS Spawner on a z/OS SAS/CONNECT Server

After digital certificates are generated for the CA, the server, and the client, and a CA trust list for the client is created, you can start a z/OS spawner program that runs on a server that SAS/CONNECT clients connect to.

For example:

```
//SPAWNER  EXEC PGM=CNTSPAWN,
//         PARM='-service 4321 =<//DDN:SYSIN'
//STEPLIB  DD   DISP=SHR,DSN=<customer.high.level.pfx>.LIBRARY
//STEPLIB  DD   DISP=SHR,DSN=<customer.high.level.pfx>.LIBE
//SYSPRINT DD   SYSOUT=*
//SYSTERM  DD   SYSOUT=*
//TKMVSJNL DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//SYSIN    DD   *
-netencryptalgorithm ssl
-sslkeyringfile /users/derver/certs/server.p12
-sslkeyringpw password
-sascmd /users/server/command.sh
```

The following table explains the SAS commands that are used to start a z/OS spawner on a SAS/CONNECT server with SAS 9.4M8.

Table 4.7 *SAS Commands and Arguments for z/OS Spawner Start-Up Tasks with SAS 9.4M8*

SAS Commands and Arguments	Function
CNTSPAWN	Starts the spawner

SAS Commands and Arguments	Function
-SERVICE *4321*	Specifies the spawner service that is listening on port 4321
-MGMTPORT *1234*	Port number
- NETENCRYPTALGORITHM *SSL*	Specifies the TLS encryption algorithm
-SSLKEYRINGFILE */users/derver/certs/server.p12*	Specifies the file path for the location of the keyring file in PKCS#12 format. This file contains a certifcate and key.
SSLKEYRINGPW *password*	Specifies the password to access the server's private key in the PKCS#12 package.
-SASCMD */users/server/command.sh*	Specifies the name of an executable file that starts a SAS session when you sign on without a script file

In order for the z/OS spawner to locate the appropriate server digital certificate, you must specify either the SSLKEYRINGFILE=, SSLKEYRINGLABEL=, and SSLKEYRINGPW SAS system options or the SSLKEYRINGSTASHFILE system option (if SSLKEYRINGFILE= system option references a key database and a stash file was created using the gskkyman utility.)

Here is an example of an executable file, **command.sh**:

```
#!/bin/sh
args=$*
if [ -n "$NETENCRALG" ] ; then
   args="$args -netencralg $NETENCRALG"
fi
if [ -n "$SASDAEMONPORT" ] ; then
   args="$args -sasdaemonport $SASDAEMONPORT"
fi
if [ -n "$SASCLIENTPORT" ] ; then
   args="$args -sasclientport $SASCLIENTPORT"
fi
export TSOOUT=
export SYSPROC=SAS.CLIST
/bin/tso -t %sas -dmr -noterminal
-sslkeyringfile /users/server/certs/serverkey.p12
-sslkeyringpw password $args
```

For complete information about starting a z/OS spawner, see *SAS/CONNECT User's Guide*.

Connection of a SAS/CONNECT Client to a z/OS Spawner

After a z/OS spawner is started on a SAS/CONNECT server, a SAS/CONNECT client can connect to it.

The following example shows how to connect a SAS 9.4M8 z/OS client to a spawner that is running on a SAS/CONNECT server:

```
options comamid=tcp netencryptalgorithm=ssl;
options sslkeyringfile="/users/johndoe/certs/sas.p12";
options sslkeyringpw="password";
%let machine=apex.server.com;
signon machine.4321 user=_prompt_;
```

The following table explains the SAS system options that are used to connect to a SAS 9.4M8 z/OS SAS/CONNECT server.

Table 4.8 *SAS Options and Arguments for Client Access to a SAS/CONNECT Server*

SAS Options and Arguments	Client Access Tasks
COMAMID=TCP	Specifies the TCP/IP access method
NETENCRYPTALGORITHM=*SSL*	Specifies the encryption algorithm
SSLKEYRINGFILE="*sas.p12*	Specifies the location of the PKCS#12 file containing CA trusted certificates. You can also specify a key database. The key database is specified with file type extension .kdb
SSLKEYRINGPW=*"password"*	The password for the PKCS#12 file.
SIGNON=*server-ID.service*	Specifies the server and service to connect to
USER=_PROMPT_	Prompts for the user ID and password to be used for authenticating the client to the server

The server ID and the server's Common Name, which was specified in the server's digital certificate, must be identical.

For complete information about connecting to a z/OS spawner, see *SAS/CONNECT User's Guide*.

Using TLS for Encryption of SAS/SHARE on UNIX: Example

Start-Up of a Multi-User SAS/SHARE Server

After certificates for the CA, the server, and the client have been generated, and a CA trust list for the client has been created, you can start a SAS/SHARE server.

Here is an example of starting a secured SAS/SHARE server:

```
%let tcpsec=_secure_;
options netencryptalgorithm=ssl;
options sslcertloc="/users/johndoe/certificates/server.pem";
options sslpvtkeyloc="/users/johndoe/certificates/serverkey.pem";
options sslpvtkeypass="password";
proc server id=shrserv;
run;
```

The following table lists the SAS option or statement that is used for each task to start a server.

Table 4.9 *SAS Options and Statements for Server Start-Up Tasks*

SAS Options and Statements	Server Start-Up Tasks
TCPSEC= _SECURE_	Secures the server
NETENCRALG=*SSL*	Specifies TLS as the encryption algorithm
SSLCERTLOC=*server.pem*	Specifies the filepath for the location of the server's certificate
SSLPVTKEYLOC=*serverkey.pem*	Specifies the filepath for the location of the server's private key
SSLPVTKEYPASS="*password*"	Specifies the password to access server's private key
PROC SERVERID=*shrserv*	Starts the server

Note: As an alternative to using the SSLPVTKEYPASS= option to protect the private key, you might prefer that the private key remain unencrypted, and use the file system permissions to prevent Read and Write access to the file that contains

the private key. To store the private key without encrypting it, use the-NODES option when requesting the certificate.

SAS/SHARE Client Access of a SAS/SHARE Server

After a SAS/SHARE server has been started, the client can access it.

Here is an example of how to make a client connection to a secured SAS/SHARE server:

```
options sslcalistloc="/users/johndoe/certificates/cacerts.pem";
%let machine=apex.server.com;
libname a '.' server=machine.shrserv user=_prompt_;
```

The following table lists the SAS options that are used to access a SAS/SHARE server from a client.

Table 4.10 *SAS Options and Arguments Tasks for Accessing a SAS/SHARE Server from a Client*

SAS Options and Arguments	Client Access Tasks
SSLCALISTLOC=*cacerts.pem*	Specifies the CA trust list
SERVER=*machine.shrserv*	Specifies the machine and server to connect to
USER=_PROMPT_	Prompts for the user ID and password to be used for authenticating the client to the server

The server-ID and the server's Common Name, which was specified in the server's certificate, must be identical.

Using TLS for Encryption of SAS/SHARE on Windows: Examples

Start-Up of a Multi-User SAS/SHARE Server

After certificates for the CA, the server, and the client have been generated and imported into the appropriate certificate store, you can start a SAS/SHARE server. Here is an example of how to start a secured SAS/SHARE server:

```
%let tcpsec=_secure_;
options comamid=tcp netencryptalgorithm=ssl;
options sslcertiss="Glenn's CA";
options sslcertserial="0a1dcfa3000000000015";
proc server id=shrserv;
run;
```

The following table lists the SAS option or statement that is used for each task to start a server.

Table 4.11 *SAS Options, Statements, and Arguments for Server Start-Up Tasks*

SAS Options, Statements, and Arguments	Server Start-Up Tasks
TCPSEC= _SECURE_	Secures the server
COMAMID=*TCP*	Specifies the TCP/IP access method
NETENCRALG=*SSL*	Specifies TLS as the encryption algorithm
SSLCERTISS="*Glenn's CA*"	Specifies the name of the issuer of the digital certificate that TLS should use
SSLCERTSERIAL="*0a1dcfa3000000000015*"	Specifies the serial number of the digital certificate that TLS should use
PROC SERVERID=*shrserv*;	Starts the server

SAS/SHARE Client Access of a SAS/SHARE Server

After a SAS/SHARE server has been started, the client can access it.

Here is an example of how to make a client connection to a secured SAS/SHARE server:

```
options comamid=tcp;
%let machine=apex.server.com;
libname a '.' server=machine.shrserv user=_prompt_;
```

The following table lists the SAS options that are used for accessing a server from a client.

Table 4.12 *SAS Options and Arguments for Accessing a SAS/SHARE Server from a Client*

SAS Options and Arguments	Client Access Tasks
COMAMID=*TCP*	Specifies the TCP/IP access method
SERVER=*machine.shrserv*	Specifies the machine and server to connect to
USER=_PROMPT_	Prompts for the user ID and password to be used for authenticating the client to the server

The server-ID and the server's Common Name, which was specified in the server's certificate, must be identical.

Using TLS for Encryption of SAS/SHARE on z/OS Prior to SAS 9.4M8: Example

Note: This example works prior to SAS 9.4M8. Starting with SAS 9.4M8, SAS used IBM System SSL for z/OS data in motion. There are new SAS system options for use with System SSL, and deprecated system options that were used with z/OS prior to SAS 9.4M8. See “SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8”, “SAS System Options Deprecated for Use with SAS 9.4M8 on z/OS”, and “IBM System SSL Provides OpenSSL Capabilities for z/OS in SAS 9.4M8”.

Start-Up of a Multi-User SAS/SHARE Server

After certificates for the CA, the server, and the client have been generated, and a CA trust list for the client has been created, you can start a SAS/SHARE server.

Here is an example of starting a secured SAS/SHARE server:

```
%let tcpsec=_secure_;
options netencryptalgorithm=ssl;
options sslpkcs12loc="/users/johndoe/certs/server.p12";
options sslpkcs12pass="password";
proc server id=shrserv;
run;
```

The following table lists the SAS option or statement that is used for each task to start a server.

Table 4.13 *SAS Options, Statements, and Arguments for Server Start-Up Tasks*

SAS Options, Statements, and Arguments	Server Start-Up Tasks
TCPSEC= _SECURE_	Secures the server
NETENCRALG=*SSL*	Specifies TLS as the encryption algorithm
SSLPKCS12LOC=*server.p12*	Specifies the filepath for the location of the server's private key
SSLPKCS12PASS="*password*"	Specifies the password to access server's private key
PROC SERVERID=*shrserv*	Starts the server

SAS/SHARE UNIX Client Access of a SAS/SHARE Server

After a SAS/SHARE server has been started, the client can access it.

Here is an example of how to make a client connection to a secured SAS/SHARE server:

```
options sslcalistloc="/users/johndoe/certs/cacerts.pem";
%let machine=apex.server.com;
libname a '.' server=machine.shrserv user=_prompt_;
```

The following table lists the SAS options that are used to access a SAS/SHARE server from a client.

Table 4.14 SAS Options and Arguments for Accessing a SAS/SHARE Server from a Client

SAS Options and Arguments	Client Access Tasks
SSLCALISTLOC=*cacerts.pem*	Specifies the CA trust list
SERVER=*machine.shrserv*	Specifies the machine and server to connect to
USER=_PROMPT_	Prompts for the user ID and password to be used for authenticating the client to the server

The server-ID and the server's Common Name, which was specified in the server's certificate, must be identical.

Using System SSL for Encryption of SAS/SHARE z/OS Client with SAS 9.4M8: Example

Start-Up of a Multi-User SAS/SHARE Server

After certificates for the CA, the server, and the client have been generated, and a CA trust list for the client has been created, you can start a SAS/SHARE server.

Here is an example of starting a secured SAS/SHARE server:

```
%let tcpsec=_secure_;
options netencryptalgorithm=ssl;
options sslpkcs12loc="/users/johndoe/certs/server.p12";
options sslpkcs12pass="password";
proc server id=shrserv;
run;
```

The following table lists the SAS option or statement that is used for each task to start a server.

Table 4.15 *SAS Options, Statements, and Arguments for Server Start-Up Tasks*

SAS Options, Statements, and Arguments	Server Start-Up Tasks
TCPSEC= _SECURE_	Secures the server
NETENCRALG=*SSL*	Specifies TLS as the encryption algorithm
SSLPKCS12LOC=*server.p12*	Specifies the filepath for the location of the server's private key
SSLPKCS12PASS="*password*"	Specifies the password to access server's private key
PROC SERVERID=*shrserv*	Starts the server

SAS 9.4M8 SAS/SHARE z/OS Client Access of a SAS/SHARE Server

After a SAS/SHARE server has been started, the z/OS client can access it.

Here is an example of how to make a z/OS client connection to a secured SAS/SHARE server:

```
options sslkeyringfile="/users/johndoe/certs/cacerts.p12";
options sskkeyringpw="password";
%let machine=apex.server.com;
libname a '.' server=machine.shrserv user=_prompt_;
```

The following table lists the SAS system options that are used to access a SAS/SHARE server from a z/OS client starting in SAS 9.4M8.

Table 4.16 *SAS Options and Arguments for Accessing a SAS/SHARE Server from a z/OS Client in SAS 9.4M8*

SAS Options and Arguments	Client Access Tasks
SSLKEYRINGFILE=*cacerts.p12*	Specifies the path the PKCS#12 file or key database containing the CA trusted certs.
SSLKEYRINGPW=*"password"*;	Password for PKCS#12 (.p12) file.
SERVER=*machine.shrserv*	Specifies the machine and server to connect to

SAS Options and Arguments	Client Access Tasks
USER=_PROMPT_	Prompts for the user ID and password to be used for authenticating the client to the server

The server-ID and the server's Common Name, which was specified in the server's certificate, must be identical.

Using SSH Tunneling for SAS/CONNECT: Example

Start-Up of a UNIX Spawner on a Single-User SAS/CONNECT Server

Here is an example of code for starting a UNIX spawner program that runs on a server that SAS/CONNECT clients connect to.

```
cntspawn -service 4321
```

The UNIX spawner is started and is listening on destination port 4321. For complete details about starting a UNIX spawner, see *SAS/CONNECT User's Guide*.

Connection of a SAS/CONNECT Client to a UNIX Spawner on a SAS/CONNECT Server

After the UNIX spawner has been started on a SAS/CONNECT server, a SAS/CONNECT client can connect to it.

Here is an example of code for setting up an SSH tunnel using OpenSSH and making a client connection to the UNIX spawner that is running on a SAS/CONNECT server:

```
ssh -N -L
5555:SSH-client-computer:4321
SSH-server-computer
```

The SSH command is entered in the command line. The SSH software is started on the computer on which the SSH client runs. The SSH client's listen port is defined as 5555. The SAS/CONNECT client accesses the SSH client's listen port that is tunneled to the UNIX spawner, which runs on destination port 4321.

```
%let sshhost=SSH-client-computer
5555;
```

```
signon sshhost;
```

In SAS, the macro variable SSHHOST is assigned to the SSH client computer and its listen port 5555. A sign-on is specified to a SAS/CONNECT client at listen port 5555. The SSH client forwards the request from port 5555 through an encrypted tunnel to the SSH server, which forwards the request to the UNIX spawner that is listening on destination port 4321.

Using SSH Tunneling for SAS/SHARE: Example

Start-Up of a Multi-User SAS/SHARE Server

Here is an example of code for starting a SAS/SHARE server:

```
proc server id=_4321; run;
```

A SAS/SHARE server is started and is ready to receive requests on destination port 4321.

SAS/SHARE Client Access of a SAS/SHARE Server

Here is an example of code for setting up an SSH tunnel and making a client connection to a SAS/SHARE server:

```
ssh -N -L
5555:SSH-client-computer:4321
SSH-server-computer
```

The SSH command is entered in the command line. The SSH software is started on the computer on which the SSH client runs. The SSH client's listen port is defined as 5555. The SAS/SHARE client accesses the SSH client's listen port that is tunneled to the SAS/SHARE server, which runs on destination port 4321.

```
%let sshhost=SSH-client-computer
5555;
libname orion '.' server=sshhost;
```

In SAS, the macro variable SSHHOST is assigned to the SSH client computer and its listen port 5555. A LIBNAME statement is specified to access the library that is located on the SAS/SHARE server. The SSH client forwards the request from port 5555 through an encrypted tunnel to the SSH server, which forwards the request to destination port 4321 on the SAS/SHARE server.

PART 4

Configure TLS, FIPS, and Certificates

5

Certificates Explained

About Certificates

Certificates are used to authenticate a server process or a human user. A certificate authority (CA) is an authority in a network that issues and manages security credentials and public keys for message encryption. As part of a public key infrastructure (PKI), a CA checks with a registration authority to verify information that is provided by the requestor of a digital certificate. If the registration authority verifies the requestor's information, then the CA can issue a certificate.

A certificate authority (CA) is a third-party organization that verifies the information or the identity of computers on a network and issues digital certificates of authenticity. Digital certificates are used in a network security system to guarantee that the two parties exchanging information are really who they claim to be. Depending on how a network's security system is configured, the certificate can include its owner's public key and name, the expiration date of the certificate, or other information.

Authenticating entities is accomplished through three types of certificates:

- third-party-signed

  Obtain a certificate from a commercial third-party certificate authority such as Comodo, DigiCert, and etc.

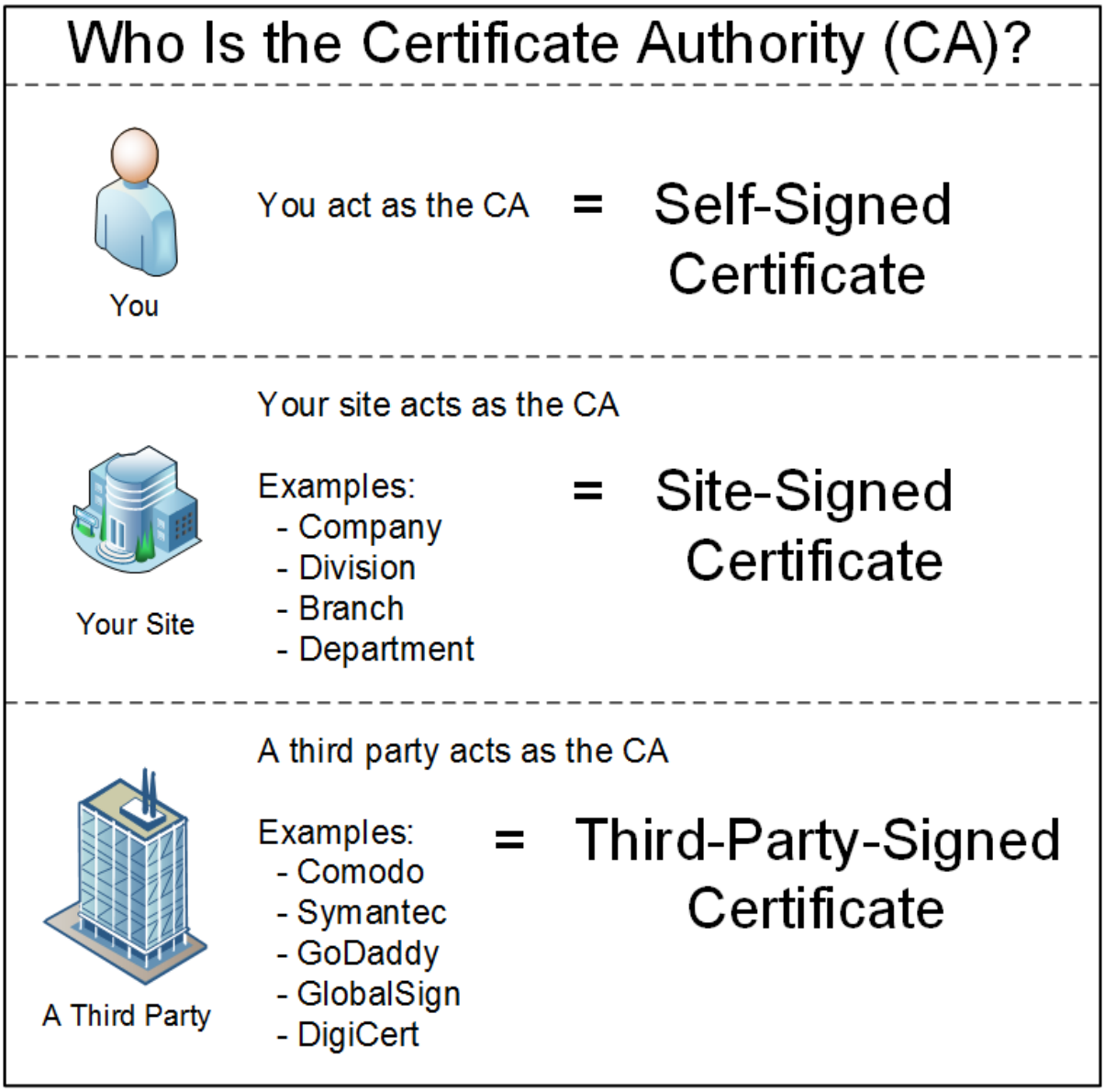
- site-signed

  Obtain a certificate from the IT department at your site.

- self-signed

Serve as your own certificate authority.

Figure 5.1 Types of Certificates



Certificate Encoding Formats and Extensions

There are many file formats used to identify certificates. Here are some of the X.509 certificate encoding formats and extensions:

Note: When requesting certificates from a CA, know the type of certificate file that the server was designed to accept. There are also ways to convert certificates to the file format needed using OpenSSL.

- Encodings (also used as extensions)

  PEM
  : Privacy Enhanced mail (.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 that includes a public key, a private key, 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). Refer to "Convert between PEM and DER File Formats Using OpenSSL" on page 91, "Convert between PEM and DER File Formats for TLS" on page 115.

  DER
  : Distinguished Encoding Rules (.der) is used for binary DER encoded certificates. A file with .der extension contains a single certificate and does not contain a private key. A PEM file is just a Base64 encoded DER file. OpenSSL can convert these to PEM. Windows sees these as certificate files. These files can also bear the .cer extension or the .crt extension. Refer to "Convert between PEM and DER File Formats Using OpenSSL" on page 91, "Convert between PEM and DER File Formats for TLS" on page 115.

  PKCS#12 .p12, .pfx
  : Public-Key Cryptography Standards (.pkcs12) is a file format that has both public and private keys in the file. Private keys are password protected. These files are also known as *.pfx format on Windows. Unlike PEM files, this container is fully encrypted.

  PKCS#7, P7B
  : PKCS#7 is a standard from the Public-Key Cryptography Standard series. The PKCS#7 files are Base64 (ASCII) encoded files and they usually have .p7b or .p7c as the file extension. PKCS#7 files contain only certificates and chain certificates. These files do not contain private keys.

  KDB file, .kdb
  : A key database stores key pairs and digital certificates, and enables secure network connections between clients and servers. This file type is used with IBM System SSL for z/OS starting with SAS 9.4M8.

- Common Extensions

  CRT
  : The .crt extension is used for storage of a single certificate. No private key is included. 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.

CER
: A file with extension .cer is recognized by Windows as a certificate. It is an alternate form of CRT (Microsoft Convention). This file contains a single certificate. It does not contain a private key.

  Use the Windows Certificate Export Wizard to convert a file in CRT to CER format. These can be converted to DER-encoded CER or to Base64 PEM-encoded CER.

  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 and uses file extension .csr. 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 PKCS#8 keys. The keys can be encoded as binary DER or as ASCII PEM.

Overview of CA Certificate Management Using the SAS Deployment Manager

IMPORTANT Starting with SAS 9.4M8, the ability to use the SAS Deployment Manager to manage certificates is not supported on z/OS. SAS uses IBM System SSL for TLS encryption, which requires the use of a key database file (.kdb) or PKCS#12 file (.p12). SAS Deployment Manager is using the trustedcerts.pem file, which cannot be used with System SSL. You can still use the SAS provided bundle of trusted CA certificates and manage those certificates for use with z/OS by following the instructions in SAS Usage Note 69954.

Starting with SAS 9.4M3, a bundle of root digital certificates is provided to get TLS up and working at SAS installation. SAS provides a bundle of certificates from Mozilla that can be used as the default trust provider for setting up protocols such as TLS.

This trusted bundle of CA certificates are located in files trustedcerts.pem and trustedcerts.jks and are maintained using the SAS Deployment Manager. Certificates can be added and removed from the trusted certificates bundle.

Starting with SAS 9.4M3, SAS adds trusted CA certificates for HTTPS usage to the Java Runtime Environment (JRE) as follows:

- Use the SAS Deployment Manager to update the default Java truststore (trustedcerts.jks). This default Java truststore is located at **`SASHome/SASPrivateJavaRuntimeEnvironment/9.4/jre/lib/security/cacerts`**. Java uses the CA certificates at this location to verify HTTPS connections and for Java code signing.
- The SAS Deployment Manager makes a copy of the trustedcerts.jks file and adds it to **`SASHome/SASPrivateJavaRuntimeEnvironment/9.4/jre/lib/security/jssecacerts`**. Java uses the CA certificates at this location only to verify HTTPS connections.

When providing signed certificates, use the SAS Deployment Manager to add the CA root and intermediate certificates to the trusted CA bundle.See "Add a Certificate to the Trusted CA Bundle" on page 94.

Also add self-signed certificates to the trusted CA bundle if needed.

Note: With SAS 9.4M2 and earlier, when signed certificates are provided, add the CA root and intermediate certificates to the SAS Private JRE using the Java `keytool -importcert` command. See "Add Your Certificates to the SAS Private JRE" on page 99.

Note: On Windows, you must add the signed CA root and intermediate certificates to the Windows certificates stores using the Windows Certificates Snap-in regardless of the SAS 9.4 software release.See "Add Your Certificates to the Windows CA Store" on page 112.

The Mozilla bundle of CA certificates (root certificates) is used to create two new files, the trustedcerts.pem file and the trustedcerts.jks file (used by Java apps). Initially, these files contain only a list of root certificates that have been approved by Mozilla for inclusion in Network Security Services (NSS). These files are updated each time the SAS Deployment Manager add and remove certificates tasks are performed.

For more information, see Mozilla's CA Certificate Program. The current list of included root certificates can be found at Mozilla Included CA Certificates.

When the SAS Deployment Manager is used to add custom CA certificates, the certificates are added to the trustedcerts.pem and trustedcerts.jks files. The trustedcerts.jks is copied to the jssecacerts file in the SAS Private JRE on Windows, UNIX, and Linux hosts. The three truststore files then contain the CA certificates redistributed by SAS from Mozilla as well as the certificates that were just added. The truststore files are regenerated each time the SAS Deployment Manager task is used to remove or add CA certificates. All three files (trustedcerts.pem, trustedcerts.jks, and jssecacerts) are kept in sync using the SAS Deployment Manager tasks. Refer to the *SAS Deployment Wizard and SAS Deployment Manager 9.4: User's Guide* for more information.

When the initial installation of SAS software is complete on UNIX and Linux platforms and on z/OS platforms prior to SAS 9.4M8, the SSLCALISTLOC= SAS system option is set by default to point to the trustedcerts.pem file. The SSLCALISTLOC= system option is not used for z/OS starting with SAS 9.4M8.

Note: THE SSLCALISTLOC= system option should not be overridden or changed unless directed by SAS technical support. In addition, the trustedcerts.pem file should not be altered by any means other than by using the SAS Deployment Manager add and remove certificate tasks. If the file is changed by another means,

the provided truststore might not be supported, and maintenance of those changes is not guaranteed.

IMPORTANT Do not remove any of the CA certificates that were initially included as part of the Mozilla bundle of trusted CA certificates.

Best Practices When Generating Certificates and Keys

SAS recommends following best practices when creating certificates, managing certificates, or securing private keys. The following best practices are recommended for managing certificates with a SAS 9.4 deployment.

- When generating new certificates, provide the following information for the certificate signing request:

  Note: For more information, see Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates at https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-v2.0.1.pdf.

  - Provide fixed host names.
  - Provide fully qualified domain names (FQDN).
  - Provide at least one subject alternative name (SAN) extension. This extension must contain at least one entry of dNSName or iPAddress type:
    - The dNSName must be either a fully qualified domain name (FQDN) or a wildcard domain name. A dNSName at a minimum should include the dNSName used to access the environment.

      When requesting server certificates, include "localhost", a *short-host-name*, and fully qualified domain names.

      Note: When SAN extensions are used, the subject Common Name might be ignored.

    - Optionally, you can include iPAddress. If an X.509 v.3 certificate contains iPAddress, it must be included in the SAN extension as an iPAddress name form (not dNSName). Multiple IP addresses can be included in the same certificate.

      IMPORTANT The iPAddress value should not be a Reserved IP Address.

When the SAN extension contains iPAddress, the address must be stored in the octet string in network byte order. For IPv4, the octet string must contain exactly four octets. For IPv6, the octet string must contain exactly sixteen octets.

For more information about SAN extensions, see RFC 5280, RFC 6125, and RFC 2818.

- When using SAS 9.4 to start a SAS/CONNECT session to a SAS Viya CAS server, the server certificate needs subject alternative name (SAN) extension entries in the server certificate for each name the host can be known by. The certificate needs the physical host name and DNS alias listed in the SAN extension.
- For Windows, when providing a certificate to enable TLS on a JAVA client, provide a certificate in PFX format. The PFX file contains the following information:
  - The PFX file has a private key embedded within it.
  - The private key within the PFX file is protected with a password.
  - The PFX file contains all certificates in the certification path (the PFX file contains the certificates that comprise the CA chain).
- Starting with SAS 9.4M8, when providing certificates for z/OS, provide a certificate file type that is specified by SAS system option SSLKEYRINGFILE=.
- Place the server identity certificate chained to any intermediate certificates in a location pointed to by system option SSLCERTLOC= for UNIX, Linux, and z/OS (prior to SAS 9.4M8) operating systems. 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.

  For z/OS, the SSLCERTLOC = system option and SSLCALISTLOC= system options are used prior to SAS 9.4M8. Starting with SAS 9.4M8, the SSLKEYRINGFILE= system option is used for z/OS instead. For more information, see “SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8” on page 167.
- If the custom root certificate is site-signed or is not already included in the Mozilla bundle of trusted CA certificates, add the root certificate to the trustedcerts files.

  On UNIX and Linux, place a copy of the root certificate in the same location as the trustedcerts files. The root certificate should have a .crt file extension.

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

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

  Starting with SAS 9.4M8 on z/OS, place a copy of the root certificate in the same location as the trustedcerts.kdb and trustedcerts.sth files.
- Encrypt the private key when possible.
- Password-protect the private key file.

Certificate Implementation: How TLS Client and Servers Negotiate

Public and private key pairs are used to negotiate algorithms between the TLS client and the TLS enabled server. Here are a few key points:

- TLS needs public and private key pairs. The server sends its public key to the client. The client can then send its public key to the server. However, the private key is never sent anywhere.
- Public keys are stored in files commonly called certificates and private keys are stored in files commonly called keys. TLS uses certificates to describe the public and private key pairs to use. TLS uses certificates defined by the X.509 standard. These certificates contain information that includes the subject (usually the host name) and the Public Key Signature signed by a Certificate Authority (CA).

  Certificates come in many file formats. File formats PEM, DER, and PKCS12 are the most used formats by SAS. Starting with SAS 9.4M8, z/OS uses IBM System SSL for encryption and requires that PKCS#12 and KDB file types be used.

  For more details, see "Certificate Encoding Formats and Extensions" on page 66.
- To send a certificate, the sender indicates which public certificate to send and has access to its private key associated with that public certificate. If the private key uses a password, the sender must know that password to use the private key.
- Secure servers always send their certificates to the client.
- Clients are required to send their certificates to the server only if they are asked.
- The receiver verifies the certificates in the following ways:
  - Makes sure the certificate has not expired.
  - Makes sure the certificate authority (CA) listed in the certificate is known and is valid. If the CA in a certificate is signed by another CA certificate, it is known as an intermediate CA. The signer CA's certificate must also be verified. This creates a CA certificate *chain*.
  - Makes sure that the certificate's "Subject" common name (CN) is for the host that the certificate was sent from. Wildcards such as "*.mydomain.com" can be used in the certificate.
  - Makes sure the certificate has not been revoked.

6

Configure TLS Communication Between SAS/ACCESS Databases and SAS

Configure TLS for SAS/ACCESS Connection to Teradata

In SAS 9.4M8, TLS 1.2 is supported when using SAS/ACCESS Interface to Teradata. You need Teradata Tools and Utilities (TTU) version 17.10 and later to support TLS 1.2. Configuring TLS enables you to have encrypted communication between SAS/ACCESS to Teradata and a SAS client using TLS 1.2. TLS encryption is supported on all operating systems. For information about the required versions of Teradata software, see the *SAS 9.4 Foundation System Requirements* documentation for your operating system.

The Teradata administrator configures TLS using the instructions in How to Secure Connections using TLS.

To enable TLS on Teradata, specify the SSLMODE connection parameter and the value REQUIRE in the clispb.dat configuration file. The location of the clispb.dat file is configurable, and the location can be pointed to by the COPLIB variable. The clisp.dat file on the client is usually located where TTU software is installed. In the clisp.dat file, add the following line of code:

```
sslmode=require
```

All of the Teradata SSLMODE connection parameter values that are documented are supported. However, here is important information about the support of TLS communication for SAS/ACCESS Interface to Teradata.

- Only TLS 1.2 is currently supported to encrypt communication between Teradata and SAS. You cannot configure falling back to non-TLS connection types when connecting to servers that do not support TLS, or where TLS is disabled.
- When the Teradata SSLMODE connection parameter values verify-ca and verify-full are specified, to ensure that there is a successful connection to a server, the server's root certificate and any other intermediate certificates must be imported to the operating system's truststores.

The three Teradata SSLMODE connection parameter values that are recommended to support TLS between Teradata and SAS are shown in Table 6.1.

Table 6.1 *Teradata SSLMODE Connection Parameter Values*

SSLMODE Value	Description
require	Encryption is required when connecting to the database. If TLS encryption is not supported, the connection fails. Establish an encrypted (TLS) connection if the server supports TLS connections. The connection attempt fails if the TLS connection cannot be established. Do not fallback to the non-TLS port. Using this value does not protect against man-in-the-middle (MITM) attacks.
verify-ca	Setting verify-ca establishes an encrypted (TLS) connection similar to the SSLMODE require value and also verifies the server Certificate Authority (CA) certificate against the configured trusted CA certificates. Requiring verification of a CA certificate depends on your database policy. The connection attempt fails if no valid matching CA certificates are found. CA certificate verification protects against MITM attacks.
verify-full	Setting verify-full establishes an encrypted (TLS) connection and CA certificate verification is required. In addition, the Teradata client verifies that the host name that SAS uses to connect matches either the Subject Alternative Name value or the Common Name value in the server certificate. The connection fails if there is a mismatch. Using this value protects against MITM attacks. In verify-full mode, the host name is matched against the certificate's Subject Alternative Name attribute(s), or against the Common Name attribute if no Subject Alternative Name of type "DNS Name" is present. If the certificate's name attribute starts with an asterisk (*), the asterisk is treated as a wildcard, which matches

SSLMODE Value	Description
	all characters except a dot (.). This means that the certificate will not match subdomains. If the connection is made using an IP address instead of a host name, the IP address is matched without using DNS lookups.

7

Configure TLS Certificates and FIPS on UNIX and Linux

Using Operating System OpenSSL Libraries with SAS 9.4M8

Starting with SAS 9.4M8, SAS uses the OpenSSL versions that are installed on UNIX or Linux operating systems to provide encryption for data in motion for SAS Foundation servers. Prior to SAS 9.4M8, SAS provided OpenSSL libraries as part of the SAS installation.

SAS supports various OpenSSL versions and tests SAS 9.4 software with these OpenSSL versions. For more information, see the following:

- "Cryptographic Library Support Starting with SAS 9.4M8"
- "TLS Versions and Cipher Suites Supported Starting with SAS 9.4M8"
- "Set Library Path Variables to OpenSSL Libraries in SAS 9.4M8"

Another difference with SAS 9.4M8 is that each operating system vendor has different requirements for how FIPS 140-2 is enabled. Refer to the operating system provider documentation and enable FIPS on the operating system before using SAS system option ENCRYPTFIPS. For more information, see the following:

- "FIPS 140-2 Support with SAS 9.4M8 on UNIX and Linux"
- "How to Use FIPS 140-2 Capable OpenSSL Libraries with SAS 9.4M8 on UNIX and Linux"
- "Set Library Path Variables to OpenSSL Libraries in SAS 9.4M8"

Set Library Path Variables to OpenSSL Libraries in SAS 9.4M8

Starting with SAS 9.4M8, the operating system cryptographic libraries are being used for SAS Foundation. For more information about the see supported cyrptographic libraries .

On UNIX and Linux operating systems where SAS 9.4M8 is installed and running, you need to specify the path to the OpenSSL cryptographic libraries. Some of the variables that need to be set are as follows:

- TKERSA2_OPENSSL_LIB= and TKESSL_OPENSSL_LIB= set to the same path where the libssl library is located.
- TKECERT_CRYPTO_LIB= set to the path of where the libcrypto cryptographic library is located.
- LD_LIBRARY_PATH= set to the same paths as the variables above.

Before starting SAS on SAS Foundation servers, the SAS administrator needs to set the environment path variables to point to the OpenSSL libraries (FIPS 140-2 capable libraries if needed) that are being used.

You can set the library paths to the OpenSSL library in the sasenv_local.cfg file. You can also specify the library paths on the command line before starting a SAS/CONNECT spawner, before executing SAS Programming software, and more.

1 To set the OpenSSL libraries in the sasenv_local.cgf file, go to the **`<SASHome>/SASFoundation/9.4/bin`** directory. This directory contains the sasenv script that sets the environment variables that are required by SAS. Modify the sasenv_local file to point to the OpenSSL version that you want to use.

   Note: Prepend the customized library path in the script that is run before invoking SAS.

   For more information, see "Contents of the !SASROOT Directory" in *SAS Companion for UNIX Environments*.

2 Set the library paths to the same OpenSSL version. Following are various examples of setting the library paths for various Operating Systems.

   Here is an example where the OpenSSL library paths are set for IBM Advanced Interactive eXecutive (AIX) UNIX operating system.

```
export TKERSA2_OPENSSL_LIB="/usr/lib/libssl.a(libssl.so.1.0.2)" export
TKESSL_OPENSSL_LIB="/usr/lib/libssl.a(libssl.so.1.0.2)" export
LD_LIBRARY_PATH=/usr/lib:$LD_LIBRARY_PATH
```

   Here is an example of setting the library path variables to pick up a FIPS 140-2 enabled OpenSSL library.

```
export TKERSA2_OPENSSL_LIB=/lib/openssl/fips-140/64/libssl.so.1.0.0 export
TKESSL_OPENSSL_LIB=/lib/openssl/fips-140/64/libssl.so.1.0.0 export
LD_LIBRARY_PATH=/lib/openssl/fips-140/64/:$LD_LIBRARY_PATH
```

   Here is an example of setting the library path variable on a Linux Operating System.

```
export TKERSA2_OPENSSL_LIB=/usr/lib/libssl.so.10 export
TKESSL_OPENSSL_LIB=/usr/lib/libssl.so.10 export LD_LIBRARY_PATH=/usr/
lib:$LD_LIBRARY_PATH
```

   Here is an example where a 64-bit version of the OpenSSL library is needed and specifically specified.

```
export TKERSA2_OPENSSL_LIB=/usr/lib/libssl.a(libssl64.so.1.0.2) export
TKESSL_OPENSSL_LIB=/usr/lib/libssl.a(libssl64.so.1.0.2) export
LD_LIBRARY_PATH=/usr/lib:$LD_LIBRARY_PATH
```

   Here are two examples of setting the OpenSSL library paths for the IBM Storage Networking SAN64B-7 (machine type-models 8960-P74, R64) UNIX operating system.

```
export TKECERT_CRYPTO_LIB='/usr/lib/libcrypto.a(libcrypto64.so.1.0.0)'
```

```
-set TKERSA2_OPENSSL_LIB /lib/openssl/default/64/libssl.so.1.0.0 -set
TKESSL_OPENSSL_LIB /lib/openssl/default/64/libssl.so.1.0.0
```

Note: Different operating systems require the use of different library file extensions. For example, HP-UX, Linux, and Solaris use libcrypto.so.1.0.0 and libssl.so.1.0.0. AIX uses libcrypto.so and libssl.so. Refer to your operating system vendor

documentation when using the vendor's OpenSSL libraries. There might be additional procedures that need to be followed to make the libraries work properly in your environment.

Depending on your operating system, you might need to set other library path variables in addition to or instead of LD_LIBRARY_PATH. For example, the AIX operating system uses LIBPATH for older operating systems and LD_LIBRARY_PATH for newer versions. A solution is to set both variables to the same shared library path. HP-UX uses the SHLIB_PATH environment variable. Refer to the operating system vendor documentation for additional information.

Following are more advanced examples and possible use cases for how you might specify the paths to the OpenSSL libraries:

- A user wants to use OpenSSL version 1.0.2. They have an OpenSSL FIPS 140-2 capable version located in`/lib/libssl.1.0.0` and a non-FIPS capable OpenSSL version located in `/usr/lib/libssl.1.0.2`. You can create symlinks in `SASFoundation/9.4/sasexe` to point libssl.so.1.0.2 to `/lib/libssl.so.1.0.0` and libcrypto.so.1.0.2 to `/lib/libcrypto.so.1.0.0`. This process instructs the SAS software to find the OpenSSL libraries in `SASFoundation/9.4/sasexe` before the OpenSSL libraries located in `/usr/lib`.
- A user has FIPS 140-2 capable OpenSSL library 1.0.2 in `/usr/lib/libssl.so.1.0.0` and non-FIPS OpenSSL 1.1.1 in `/usr/lib/libssl.so.1.1.1`. SAS first tries to load the most recent version of OpenSSL, OpenSSL 1.1.1. To direct the load process to load libssl.so.1.0.2, set path variables TKESSL_OPENSSL_LIB= and TKERSA2_OPENSSL_LIB= to libssl.so.1.0.2 to ensure that OpenSSL library is used.
- A user has a FIPS 140-2 capable OpenSSL 1.0.2 library and a non-FIPS capable OpenSSL 1.0.2 library in `/lib/libssl.so.1.0.2`. They also have a non-FIPS capable OpenSSL 1.1.1 version in `/usr/lib/libssl.so.1.1.1`.

  In order to ensure that the libssl.so.1.1.1 library is not picked up, but that the correct libssl.so.1.0.2 in the `/lib` directory is picked up, set TKESSL_OPENSSL_LIB and TKERSA2_OPENSSL_LIB to `/lib/libssl.so.1.0.2`. Then, create symlinks in `SASFoundation/9.4/sasexe` for both libssl.so.1.0.2 and libcrypto.so.1.0.2 pointing to `/lib/libssl.so.1.0.2` and `/lib/libcrypto.so.1.0.2`.

System and Software Requirements

The system and software requirements for using TLS on UNIX and Linux operating environments are as follows:

> **IMPORTANT** Starting with SAS 9.4M8, SAS no longer delivers OpenSSL libraries for SAS Foundation servers. SAS uses supported cryptographic libraries that are installed on the operating system to provide encryption for data in motion.

- a computer that runs UNIX or Linux.

- Internet access and a web browser.
- the TCP/IP communications access method when connecting to SAS/CONNECT or SAS/SHARE servers. See "Access Methods Supported by SAS/CONNECT" in *SAS/CONNECT User's Guide* and Access Methods in *SAS/SHARE User's Guide* for your operating environment.
- knowledge of your site's security policy, practices, and technology. The properties of the digital certificates that you request are based on the security policies that have been adopted at your site.
- access to the SAS Deployment Manager if you plan to add digital certificates when you install SAS 9.4M3 and later. See "Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager".
- access to the SAS Deployment Wizard. You can enable FIPS 140-2 when you install SAS 9.4.
- access to the OpenSSL utility at OpenSSL Source. You need access if you plan to use OpenSSL for the following actions:
  - You plan to apply to become a CA.
  - Your site administrator plans to generate a site-signed certificate and private key.
  - You plan to build your own OpenSSL libraries.
  - You plan to build FIPS 140-2 capable OpenSSL libraries.

Certificate Locations

SSLCALISTLOC= points to one file that contains a list of root certificates. System option SSLCACERTDIR= points to a directory that contains all of the public certificate files of all CAs in the trust chain. One file exists for each CA in the trust chain.

Certificates must be in one of the following locations:

- All certificates must be in one file in PEM format that is referenced by the SSLCALISTLOC= system option. The system options are specified in the server's invocation command.

  Starting with SAS 9.4M3, the sasv9.cfg file in UNIX and Linux deployments includes the SSLCALISTLOC= option for server side processes to use for certificate validation. In the `<SASHome>/SASFoundation/9.4/sasv9.cfg` file, the SSLCALISTLOC= option points to `<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem`. The file where the trusted certificates reside is named trustedcerts.pem. For syntax, see "SSLCALISTLOC=" on page 145.
- On UNIX and Linux, all certificates must be in an OpenSSL CA certificates directory that is pointed to by the SSLCACERTDIR= system option. The layout of this directory is specified by OpenSSL where the certificates are in PEM format.

  If you use the SSLCACERTDIR= system option, create a link to the file named after the certificate's hash value:

```
$ ln -s SAS-configuration-directory/certs/cacert1.pem 'openssl x509 -noout -
hash -in SAS-configuration-directory/certs/cacert.pem'.0
```

Note: You must add ".0" to the hash value.

For more information, see “SSLCACERTDIR=” on page 143.

Preparation for Setting Up Digital Certificates

The process of setting up TLS on UNIX and Linux involves setting up digital certificates. Following are steps that you need to take and information that you need to know to request digital certificates and to add the certificates to a CA truststore of certificates.

Starting with SAS 9.4M3, you can use the SAS Deployment Manager to add your existing digital certificates to the truststore (trustedcerts.pem). The truststore at installation includes the Mozilla bundle of trusted CA certificates. For more information, see “Add a Certificate to the Trusted CA Bundle” on page 94.

Here is the process to set up digital certificates:

- If your server comes with an instance of OpenSSL, locate that directory. You need that information to set UNIX or Linux environment variable OPENSSL_CONF=.
- Create a system (database or other) to keep track of your signed certificates.
- Create an openssl.cnf file. This is optional. This file stores the locations of your CA keys. For a partial example of this file, see Figure 7.1 on page 84.
- To prepare the certificate(s) to add to the trusted CA bundle, you can, as needed, create a root certificate. See “ Generate Digital Certificates Using OpenSSL ” on page 83.
- To prepare the certificate(s) to add to the trusted CA bundle, you can, as needed, create a certificate trust list and verify it using the instructions provided. See “Step 5 Create a Certificate Chain in PEM Format Using OpenSSL” on page 89 and “Step 6 Verify Certificates in the Chain of Trust Are Using OpenSSL” on page 91.
- Use the SAS Deployment Manager after SAS installation to add your root and intermediate certificates to the truststore. See “Add a Certificate to the Trusted CA Bundle” on page 94.
- Use best practices when generating certificates on page 70.

Generate Digital Certificates Using OpenSSL

Step 1 Create a Certificate Authority

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 the following example, a CA is being created.

There are many different options that you can use with OpenSSL to generate certificates and private keys. SAS recommends using the highest encryption standards with access controls to secure your deployment.

In the OpenSSL.cnf file that is shown below in one of the OpenSSL examples, 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 CA's role is to accept certificate applications, authenticate applications, issue certificates, and maintain status information about certificates issued. The certificates generated by the Proton, Inc. CA are considered site-signed certificates.

Perform the following tasks to generate a certificate in PEM format:

1. In this example, an OpenSSL configuration file is being used.

   Note: You do not have to use this file. You can submit your options on the command line using the OpenSSL command or allow OpenSSL to prompt you for options.

   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 a good practice to add `-config ./openssl.cnf` to the command 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 following OpenSSL command:

   ```
   openssl version -d
   ```

   Here is a partial example of some of the information that can be specified in the openssl.cnf file. There is a lot more information about certificates that can be specified.

Figure 7.1 Example of an OpenSSL.cnf File

```
#
# 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      = sha256                        # which sha to use.
preserve        = no                    # keep passed DN ordering
policy          = policy_match

# For the CA policy
[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
```

2 Select the directory where OpenSSL was built.

3 Initialize OpenSSL:

```
$ openssl
```

4 Issue the appropriate command to request a digital certificate. If you are using the openssl.cnf file, specify

```
-config ./openssl.cnf
```

in the OpenSSL command.

Here are a few examples of generating a CSR (Certificate Signing Request):

- In this example, a self-signed CA certificate with subject alternative names is being generated. The request creates a private key, from which it generates a CSR and signs it with the private key.

```
openssl req -x509 -nodes -days 365 -newkey rsa:4096 -keyout <SASHome>/
SASSecurityCertificateFramework/1.1/cacerts/private/customer.key -out
<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/customer.csr -
addext "subjectAltName=DNS:fully-qualified-host-name, DNS:short-host-
name, DNS:'localhost' " -addext "subjectAltName=IP:ip-address" -addext
"basicConstraints=CA:true"
```

- In this example, a CSR and 4096-bit RSA key file are being generated. These files are sent to a certificate signing authority to be signed. This example is a server certificate:

```
openssl req -newkey rsa:4096 -keyout <SASHome>/
SASSecurityCertificateFramework/1.1/cacerts/private/server.key -nodes -
sha256 -new -out <SASHome>/SASSecurityCertificateFramework/1.1/cacerts/
server.csr -addext "subjectAltName=DNS:fully-qualified-host-name,
DNS:short-host-name, DNS:'localhost'"
```

- In this example, a CSR and 4096-bit RSA key file is being generated. This example uses an openssl.conf file:

```
openssl req -config ./openssl.cnf -newkey rsa:4096 -keyout <SASHome>/
SASSecurityCertificateFramework/1.1/cacerts/private/server.key -nodes -
sha256 -new -out <SASHome>/SASSecurityCertificateFramework/1.1/cacerts/
server.csr
```

Note: For FIPS 140-2 compliant TLS, specify -sha256. SHA256 and above is highly recommended when creating your private key.

Table 7.1 *Arguments and Values Used in OpenSSL Commands*

OpenSSL Arguments and Values	Functions
req	Requests a certificate.
-config ./openssl.cnf	Specifies the name and location of the OpenSSL configuration file. You can use it to specify extensions, basic constraints, and many other options.
-new	Identifies the request as new.
-out customer.csr	Specifies the location of the certificate signing request.
-newkey rsa:4096	Generates a new private key along with the certificate request that is 4096 bits in length using the RSA algorithm.
-keyout customer.key	Specifies the storage location for the private key.
-nodes	Prevents the private key from being encrypted. **Note:** SAS recommends that you encrypt the private key.
-sha256	Specifies that the SHA256 hash algorithm be used. Use SHA256 for FIPS 140-2. Without this option, the default is SHA1.
-addext	Used to add extensions to the OpenSSL request. Some of the extensions can be Subject Alternative Names (SAN), basic constraints, and

OpenSSL Arguments and Values	Functions
	certificate policies. For more information, see openssl req.

5 When you submit your CSR, 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. It is highly recommended that you supply a password to help protect the private key.

Here is an example of a request for a digital certificate that is using information from the openssl.cnf file.

Note: For the server, the Common Name must be the name of the computer that the server runs on. In the following example, proton.com is the name of the computer.

```
openssl req -config ./openssl.cnf -new -out customer.req -newkey rsa:4096 -
keyout customer.key -nodes


Using configuration from ./openssl.cnf
Generating a 4096 bit RSA private key
.............................++++++
.............................................++++++
writing new private key to 'customer.key'
-----
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):
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>
```

Note: Starting in September of 2022, the Organizational Unit Name (OU) field is deprecated for publicly trusted certificates. All new or re-issued publicly trusted TLS certificates no longer contain OU information. Existing certificates are not affected.

The request for a digital certificate request is complete.

Step 2 (Optional) Generate a Public Certificate from an Existing Certificate

Perform the following tasks to generate a digital certificate for a CA, a server, and a client based on an existing certificate.

1 Issue the appropriate command to generate a public certificate from the certificate signing request.

Table 7.2 OpenSSL Commands for Generating Digital Certificates on UNIX

Generate Certificate for	OpenSSL Command
CA	x509 -req -in customer.csr -signkey customer.key -out customer.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.

Note: The -md sha256 option is the minimum value that should be specified when using FIPS 140-2 compliant TLS.

Table 7.3 Arguments and Values Used in OpenSSL Commands to Generate a Certificate

OpenSSL Arguments and Values	Functions
x509	Identifies the certificate display and signing utility.

OpenSSL Arguments and Values	Functions
-req	Specifies that a certificate be generated from the request.
ca	Identifies the Certificate Authority utility.
-config ./openssl.cnf	Specifies the location of the openssl.cnf file and where the OpenSSL utility is located.
-in filename.csr	Specifies the location of the input file for the certificate request.
-out filename.pem	Specifies the location of the certificate file in pem format.
-signkey cakey.pem	Specifies the private key that is used to sign the certificate that is generated by the certificate request.
-md sha256	Specifies that the SHA256 hash algorithm be used. Use SHA256 for FIPS 140-2. Without this option, the default is SHA1.

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 for creating 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.'
commonName            :PRINTABLE:'proton.com'
Certificate is to be certified until Oct 16 17:48:27 2023 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.

A root CA digital certificate is self-signed, which means that the digital certificate is signed with the private key that corresponds to the public key that is in the digital certificate. Except for root CAs, digital certificates are usually signed with a private key that corresponds to a public key that belongs to someone else, usually the CA.

Step 3 Secure Your Private Key File

To help secure access to the private key, use a password to restrict access to the private key file. This can either be done when the private key is generated or it can be performed afterward. For example, to use OpenSSL to add a password to a private key file, use the following command:

```
openssl rsa -aes256 -in /tmp/customer.pem -out /tmp/customer.key
```

OpenSSL prompts for the password to use on the private key file.

Step 4 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.

Step 5 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*.

Note: Starting with SAS 9.4M3, you can use the SAS Deployment Manager after installation to add to the trusted CA bundle of certificates. For more information, see "Add a Certificate to the Trusted CA Bundle" on page 94.

Note: Starting with SAS 9.4M6, IOM servers support TLS.

If there is only one CA to trust, in the client application specify 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 trust list, use the following template:

```
(Your Server Certificate - server.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 and CA's 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 primary certificate file and concatenate them into a single PEM file. An example of concatenating certificates is as follows:

```
cat server.pem > certchain.pem
cat intermediateCA.pem >> certchain.pem
cat rootCA.pem >> certchain.pem
```

Note: You can place these files in any order.

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 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 cert.der -inform der -text -noout
```

Note: If you are including a digital certificate that is stored in DER format in your certificate chain, you must first convert it to PEM format. For more information, see "Convert between PEM and DER File Formats Using OpenSSL" on page 91.

Step 6 Verify Certificates in the Chain of Trust Are 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. The certificates are either combined into one file pointed to by the SSLCALISTLOC= option or are located as individual files in an OpenSSL directory pointed to by the SSLCACERTDIR= system option.

For more information, see "SSLCACERTDIR=" on page 143 and "SSLCALISTLOC=" on page 145.

You can use the following OpenSSL command to verify certificates 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.

Note: Starting with SAS 9.4M3, you can use the SAS Deployment Manager after installation to add your trust chain. The SAS Deployment Manager also validates those certificates. For more information, see "Add a Certificate to the Trusted CA Bundle" on page 94.

Step 7 End OpenSSL

To end OpenSSL, type `quit` at the prompt.

Convert between PEM and DER File Formats Using OpenSSL

By default, OpenSSL files are created in PEM (Privacy Enhanced Mail) format. TLS files that are created in Windows operating environments are created in DER (Distinguished Encoding Rules) format.

On Windows, you can import a file that is created in either PEM or DER format. However, a digital certificate that is created in DER format must be converted to PEM format before it can be included in a trust list on UNIX.

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

```
openssl x509 -inform PEM -outform DER -in <SASHome>/
SASSecurityCertificateFramework/1.1/cacerts/server.pem -out <SASHome>/
SASSecurityCertificateFramework/1.1/cacerts/server.der
```

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

```
openssl x509 -inform DER -outform PEM -in <SASHome>/
SASSecurityCertificateFramework/1.1/cacerts/server.der -out <SASHome>/
SASSecurityCertificateFramework/1.1/cacerts/server.pem
```

Convert PEM Files to a PFX File Using OpenSSL

If you need to use a certificate with a Java application or with any other application that accepts only PKCS#12 formatted files, you can create a single PFX file that contains both the certificate and the key file:

```
openssl pkcs12 -export -out <SASHome>/SASSecurityCertificateFramework/1.1/
cacerts/customer.pfx -inkey <SASHome>/SASSecurityCertificateFramework/1.1/
cacerts/private/customer.key -in <SASHome>/SASSecurityCertificateFramework/1.1/
cacerts/customer.csr
```

Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager

Overview

Starting with SAS 9.4M3, after installation, the SAS Deployment Manager can be used to add certificates to the truststore or to remove certificates from the truststore for SAS Foundation servers. Starting with SAS 9.4M6, IOM servers support TLS. The SAS Deployment Manager can also be used to manage certificates for the IOM servers.

Note: Starting with SAS 9.4M2 and earlier, when providing your own signed certificates, you must add the CA root and intermediate certificates to the SAS Private JRE using the Java `keytool -importcert` command. See "Add Your Certificates to the SAS Private JRE" on page 99.

The SAS Deployment Manager installation process provides the following:

- a Mozilla bundle of trusted CA certificates. It is provided in the cacerts.pem file that is located in this directory: **`<SASHome>/SASSecurityCertificateFramework/1.1/cacerts`**.
- the ability to manage the trusted CA bundle of certificates by adding or removing certificates. The process also validates the certificates.
- the SSLCALISTLOC= system option set to the default certificate path: **`<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem`**

Files placed in the new directory and updated at installation are described below. How the cacerts directory looks is shown in Figure 7.2 on page 93.

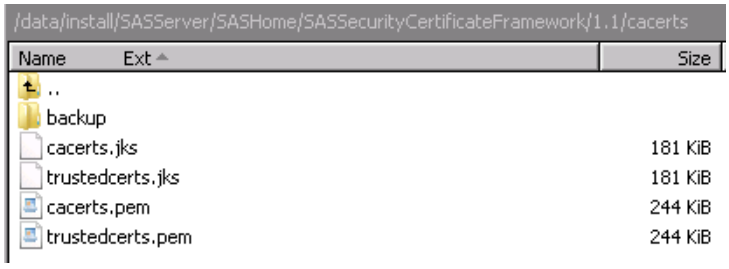
cacerts.pem
: contains the Mozilla bundle of trusted CA certificates that is provided at SAS installation. This file is in PEM format.

cacerts.jks
: contains the Mozilla bundle of trusted CA certificates that is provided at SAS installation. This file is in JKS (Java keystore) format.

trustedcerts.pem
: contains a merged list of trusted CA certificates, including both CA certificates in the cacerts.pem file and CA certificates that are added using the SAS Deployment Manager. Trusted CA certificates can be added to and removed from this file using the SAS Deployment Manager during the deployment process. This file is in PEM format.

trustedcerts.jks
: contains a merged list of trusted CA certificates, including both CA certificates in the cacerts.jks file and CA certificates that are added using the SAS Deployment Manager. Trusted CA certificates can be added to and removed from this file using the SAS Deployment Manager during the deployment process. This file is in JKS (Java keystore) format.

Figure 7.2 *SAS Security Certificate Framework Directory at Install*



Add a Certificate to the Trusted CA Bundle

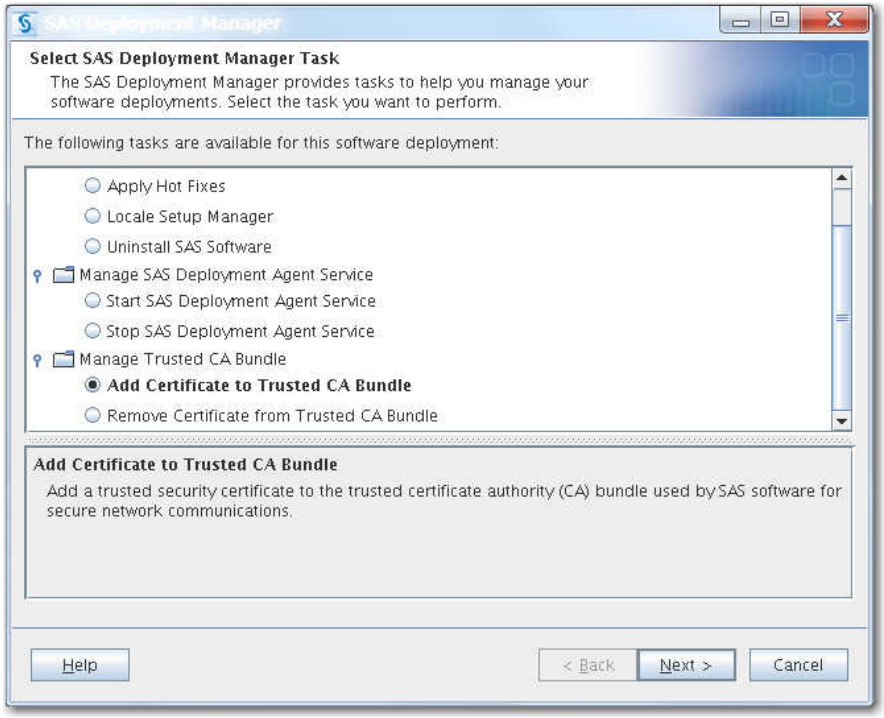
Starting with SAS 9.4M3, if you are providing your own site-signed certificates, then you must add the CA root certificate and all of its intermediate certificates to the trusted CA bundle of certificates. If you are using self-signed certificates, the self-signed certificate needs to be added to the trusted CA bundle as well. Use the SAS Deployment Manager to perform this function.

Before adding your certificates to the truststore, consider the following information:

- You can add only one certificate at a time using SAS Deployment Manager. You must rerun SAS Deployment Manager each time you add a certificate to the trusted CA bundle of certificates.
- If you have any Windows machines, you must also add the CA root and intermediate certificates to the Windows Certificate stores. For more information, see “Add Your Certificates to the Windows CA Store” on page 112.
- If you are importing certificates from SAS Viya, see Configure SAS 9.4 Clients to Work with SAS Viya.
- These steps are to be performed for sites that are running SAS 9.4M3 and later. If you are running SAS 9.4M2 or earlier, see “Add Your Certificates to the SAS Private JRE” on page 99.

To add CA root and intermediate certificates or to add self-signed certificates that are being used in the deployment, perform these steps:

1. Log on to the primary machine as the SAS Installer user.
2. Start SAS Deployment Manager by navigating to `SAS-installation-directory/SASDeploymentManager/9.4` and launching `sasdm.exe` (Windows) or `sasdm.sh` (UNIX). On Windows, you can use the shortcut on the **Start** menu.
3. When prompted for the task, select **Add Certificate to Trusted CA Bundle**, and click **Next**.

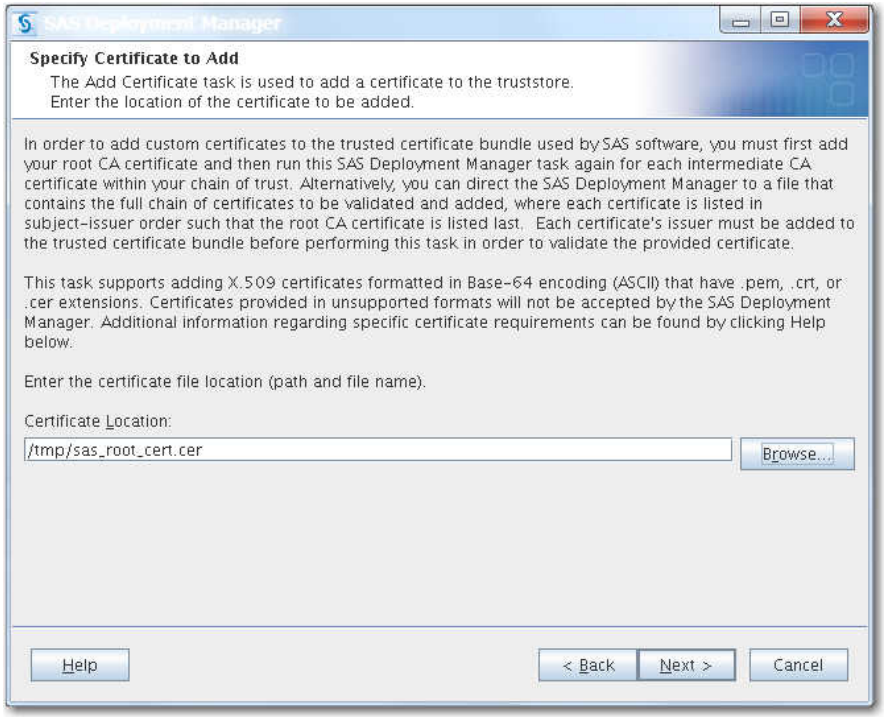


4 Specify the path to your CA root certificate, and click **Next**.

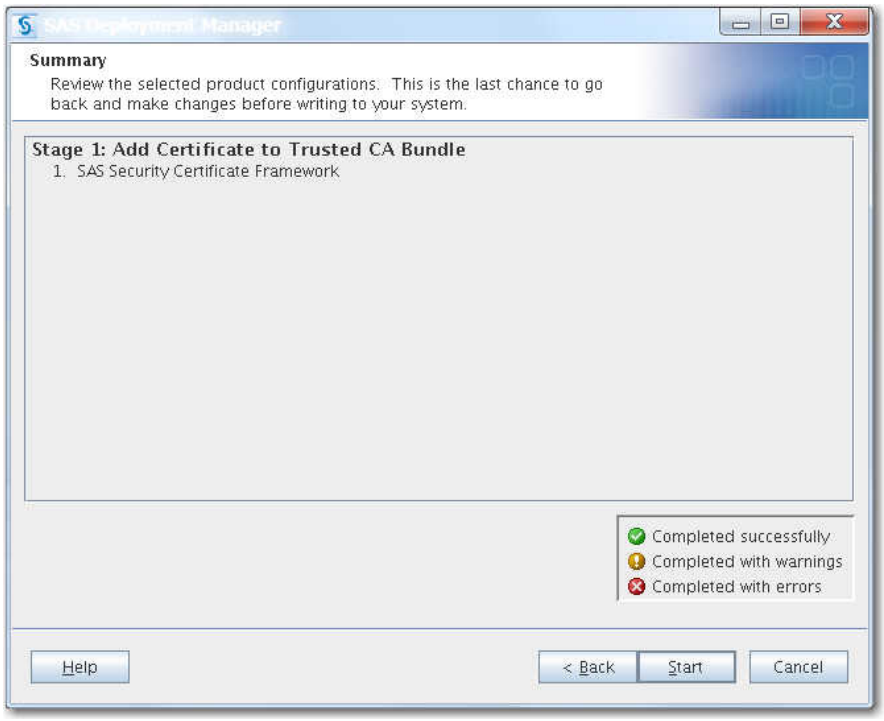
Note: **Certificate Location** is the location that you established in "Certificate Locations" on page 81.

The CA root certificate must be in base64 encoding (ASCII) and have a PEM, CRT, or CER file extension. For more information, see "Certificate Encoding Formats and Extensions" on page 66.

Note: Add your CA root certificate before adding your CA intermediate certificates.

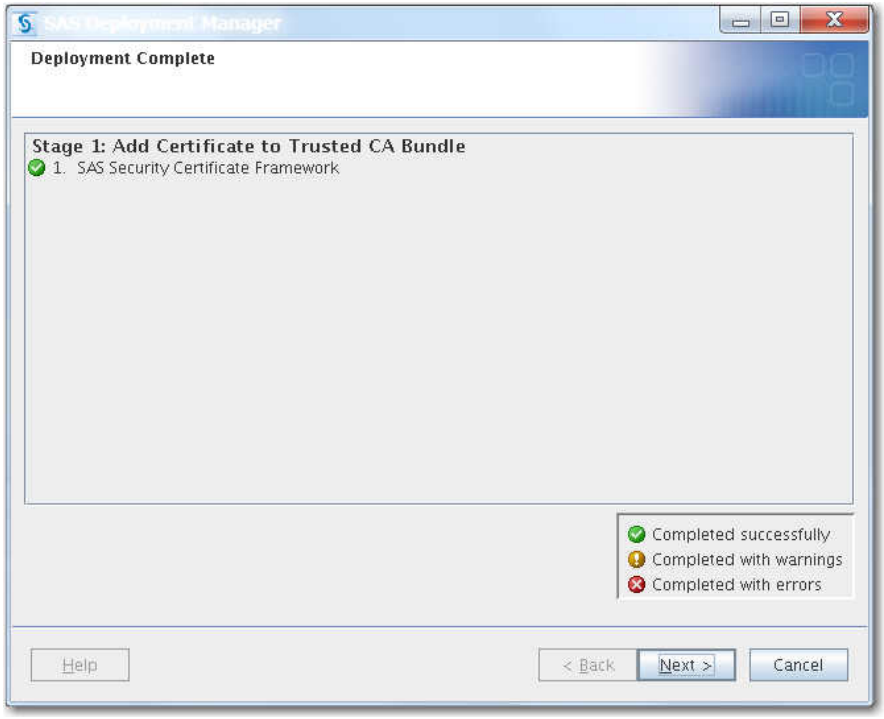


5 On the Summary page, click **Start**.



6 When you see a green checkmark on the Deployment Complete page, this means that you added your certificate successfully to ***SAS-installation-directory*/SASSecurityCertificateFramework/1.1/cacerts**.

Click **Next**.



> **TIP** The log files that are created by the add certificate task are located at **`<SASHOME>/InstallMisc/InstallLogs/certframe*`**.

7 On the Additional Resources page, click **Finish** to close SAS Deployment Manager.

8 Repeat steps 2 through 7 to add your CA intermediate certificates.

9 To verify that your CA root and intermediate certificates were successfully added, enter the following command:

> **Note:** The name of the root CA certificate is rootca and the intermediate CA certificate is named intca.

```
path-to-keytool-command/keytool -list -keystore /SAS-installation-directory/
SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.jks
```

You should see output similar to the following:

```
intca, Oct 15, 2022, trustedCertEntry,
Certificate fingerprint (SHA1): 12:4D:C9:88:CD:D5:F3:E9:9E:29:D8:AB:F1:00:AD:93
:60:61:11:45 cn=twca root certification authority,ou=root ca,o=taiwan-ca,c=tw,
Jun 2, 2020, trustedCertEntry, Certificate fingerprint (SHA1): CF:9E:87:6D:D3:EB
:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
rootca, Oct 15, 2022, trustedCertEntry,
Certificate fingerprint (SHA1): 57:7B:1E:D7:16:5D:5F:44:EB:79:AB:40:FA:98:49:DD:
DF:4F:B5:F7 cn=microsec e-szigno root ca,ou=e-szigno ca,o=microsec ltd.,l=budapest,
c=hu, Jun 2, 2020, trustedCertEntry,
```

10 Repeat steps 1 through 9 on each machine.

Note: Repeating these steps is required on client machines with Java clients. For example, such as SAS Management Console.

11 If you have any Windows machines in your SAS deployment, proceed to "Add Your Certificates to the Windows CA Store".

Remove a Certificate from the Trusted CA Bundle

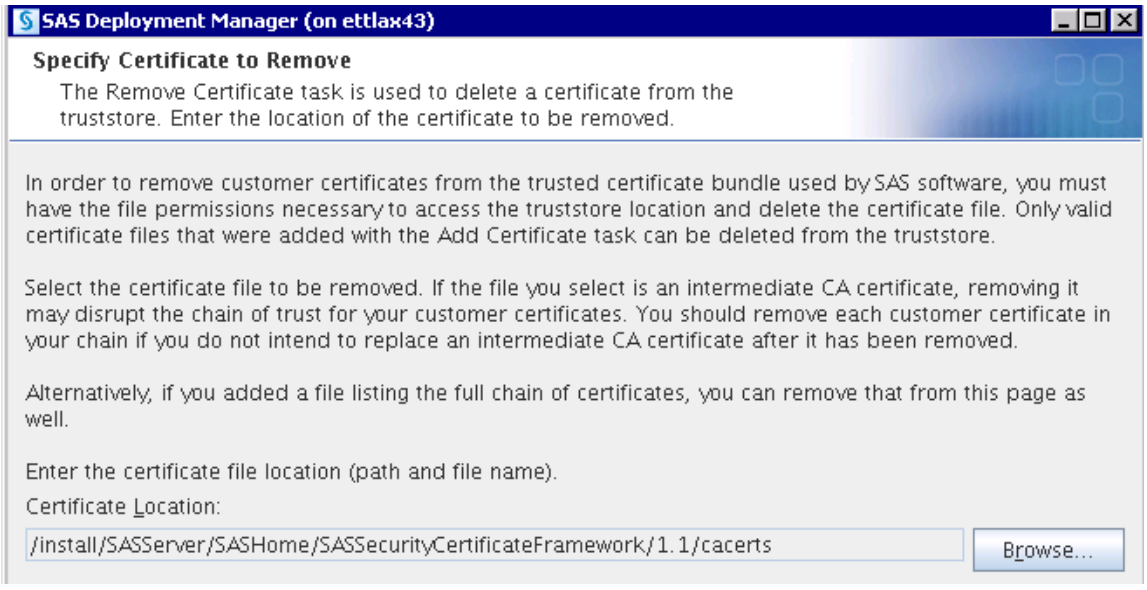
Starting with SAS 9.4M3, you can use SAS Deployment Manager to remove certificates from the trusted CA bundle.

To remove a certificate from the trusted CA bundle, you must have file permissions to access the truststore location. Only certificate files that were added using "Add a Certificate to the Trusted CA Bundle"can be deleted.

TIP `Your certificates were added to` *`SAS-installation-directory/`* `SASSecurityCertificateFramework/1.1/cacerts`. Select this directory when removing your certificates.

Note: If the file that you are removing is an intermediate certificate, removing it might disrupt the chain of trust for your customer certificate. If you do not plan to replace this intermediate certificate, you should remove each customer certificate in the chain.

Figure 7.3 *Remove a Certificate from the Trusted CA Bundle Using SAS Deployment Manager*



TIP The log files created by the remove certificate task are located at `<SASHOME>/InstallMisc/InstallLogs/certframe*`.

Refer to the *SAS Deployment Wizard and SAS Deployment Manager 9.4: User's Guide* for detailed information about using SAS Deployment Manager to manage your certificates.

SAS Deployment Manager Criteria for Validating Certificates

The following criteria must be met for the validation to complete. Otherwise, errors are generated. See Chapter 14, "Troubleshooting," on page 199 for possible errors that might be generated.

- Each certificate's issuer must be added to the trusted certificate bundle before the certificate can be validated.
- Certificates must be X.509 certificates formatted in Base-64 encoding that have .pem, .crt, or .cer extensions.
- The issuing CA is a trusted CA.
- The issuing CA's public key validates the issuer's digital signature.
- The current date is within the certificate's validity period.

Add Your Certificates to the SAS Private JRE

A Java process uses a slightly different technology to establish trust. Within the Java API is a TrustManager that is responsible for managing the trust material that is used when making trust decisions. This uses a specific Java keystore, referred to as the truststore to maintain a repository of CA certificates that are used to establish trust. The default truststore that is used by a Java Runtime Environment (JRE) is located in the **`lib/security/cacerts`** file.

Prior to SAS 9.4M3, there is no SAS Deployment Manager task to help you manage any certificates that you provide. To establish trust for a Java process, the CA certificates must be added to the cacerts file or to JVM arguments that are used to point to a different Java truststore. If you are providing your own certificates, then you must add the CA root certificate and all of its intermediate certificates to the SAS Private JRE using the `keytool -importcert` command.

Note: For Windows machines, you must also add the CA root and intermediate certificates to the Windows certificates stores. For more information, see "Add Your Certificates to the Windows CA Store" on page 112.

To add CA root and intermediate certificates, perform these steps:

1. Log on to the primary machine as the SAS Installer user.

2 Change the directory to where your keytool commands reside.

3 Enter the following command. Refer to the table for information that you must provide.

```
./keytool -import -trustcacerts -alias MyCA -file ca_cert.pem -keystore
SASHome/SASPrivateJavaRuntimeEnvironment/9.4/jre/lib/security/cacerts -
storepass password
```

Note: The entire keytool command must be entered on one line.

TIP For more information about the keytool command, determine what version of Java you are using. See SAS 9.4 Support for Java. Refer to the Oracle documentation for Keytool for Java 8, Keytool for Java 7, and Security Tools and Commands for Java 11.

Table 7.4 User-Supplied Values for the Keytool Command

Value	Description	Examples
SAS-installation-directory	Location on the machine where SAS is installed	**C:\Program Files\SASHome\SASPrivateJavaRuntimeEnvironment\9.4\jre\lib\security**
myhost	Fully qualified machine name	my_server.example.com
path-to-keystore.jks	Absolute path to the keystore Refer to"Certificate Locations" on page 81.	**/opt/certs/my_keystore.jks**

4 Repeat step 3 to add your CA intermediate certificates.

5 Verify that the CA certs have been added to the SAS private JRE. ./keytool -list -keystore SASHome/SASPrivateJavaRuntimeEnvironment/9.4/jre/lib/security/cacerts:

```
path-to-keytool-command/keytool -list -keystore SASHOME/
SASPrivateJavaRuntimeEnvironment/9.4/jre/lib/security/cacerts
```

You should see output similar to the following:

```
intca, Oct 15, 2022, trustedCertEntry,
Certificate fingerprint (SHA1): 12:4D:C9:88:CD:D5:F3:E9:9E:29:D8:AB:F1:00:AD:
93:60:61:11:45 cn=twca root certification authority,ou=root ca,o=taiwan-ca,
c=tw, Jun 2, 2020, trustedCertEntry, Certificate fingerprint (SHA1): CF:9E:
87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
rootca, Oct 15, 2022, trustedCertEntry,
Certificate fingerprint (SHA1): 57:7B:1E:D7:16:5D:5F:44:EB:79:AB:40:FA:98:49
:DD:DF:4F:B5:F7 cn=microsec e-szigno root ca,ou=e-szigno ca,o=microsec ltd.,
l=budapest,c=hu, Jun 2, 2020, trustedCertEntry,
```

6 Repeat steps 1 through 5 on each machine.

7 If you have any Windows machines in your SAS deployment, proceed to “Add Your Certificates to the Windows CA Store” on page 112.

How Clients and Servers Validate Certificates

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.

The following list provides some details about the validation process that occurs between clients and servers.

1 Digital certificates for the CA, the server, and the client (optional) are generated, and the CA trust list is created. Refer to “ Generate Digital Certificates Using OpenSSL ” on page 83.

2 The client connects to a TLS-enabled server.

3 The TLS-enabled server sends its certificate to the client along with all the intermediate CA certificates. The server certificate files are provided in an accessible directory. SAS uses the SSLCERTLOC=, SSLPVTKEYLOC=, and SSLPVTKEYPASS= options to locate the server certificate. A PKCS12 formatted file that contains both the public and private certificates in one file can also be used with the SSLPKCS12LOC= and SSLPKCS12PASS= options.

   The system options are specified in the server's invocation command. For more information, see “SAS System Options for Encryption” on page 133.

4 The client verifies the server’s certificate against the Certificate Authority (CA) list. The client has to know about all of the CAs in the server’s certificate chain in order to validate the server certificate.

   The CA certificate files are provided in either the file that is pointed to by SSLCALISTLOC= or on UNIX in an accessible directory that is pointed to by the SSLCACERTDIR= system option.

5 The server can also validate the client’s certificates. Refer to the previous steps.

How to Use FIPS 140-2 Capable OpenSSL Libraries with SAS 9.4M8 on UNIX and Linux

IMPORTANT Starting with SAS 9.4M8, SAS no longer delivers OpenSSL libraries for SAS Foundation servers. SAS uses supported cryptographic libraries that are installed on the operating system to provide encryption for data in motion.

Before enabling FIPS with SAS 9.4M8, consider the following:

1. What are the policies for enabling FIPS on the operating system and placing the software in FIPS mode?

   Note: For example, Red Hat Enterprise Linux (RHEL) requires that the entire system be put into FIPS mode in order for the OpenSSL libraries to be in FIPS mode.

2. Determine what OpenSSL libraries are being used by the operating system.

   Are there FIPS 140-2 certified or FIPS 140-2 capable modules available for use with the OpenSSL libraries that are installed?

3. Is there a FIPS module installed and enabled?

4. Do you need to build a FIPS capable module and download it?

Use the following instructions to enable and use FIPS 140-2 enabled modules.

1. Use the following OpenSSL command to determine the OpenSSL version being used:

   ```
   openssl version
   ```

2. Verify that the OpenSSL library version that is installed on the operating system is FIPS 140-2 capable. Refer to your operating system's FIPS 140-2 information and certifications.

3. Determine whether you need to build and install a FIPS 140-2 capable module.

   Following are some FIPS certified OpenSSL libraries and references to how the OpenSSL libraries are installed and FIPS is enabled.

   - FIPS support is fully integrated into OpenSSL 3.0. A separate download is no longer needed and FIPS support is built by default. However, the operating system administrator needs to take steps to ensure that your software application is using the FIPS module in OpenSSL 3.0. For more information, see OpenSSL 3.0.

- FIPS Object Module 2.0 is compatible with OpenSSL releases 1.0.1 and 1.0.2. However, the OpenSSL 2.0 FIPS Object Module requires a separate download that has to be built separately and then integrated into your main OpenSSL 1.0.2 build. For more information, see "Process to Build FIPS 140-2 Capable OpenSSL on UNIX and Linux".

4 Enable FIPS or ensure that FIPS is enabled on the operating system. Each operating system provider has different policies in place to enable FIPS compliant OpenSSL libraries. You first want to make sure that the operating system is FIPS enabled (in FIPS mode) before using the ENCRYPTFIPS= system option with the SAS 9.4M8 software. Refer to your operating system documentation for instructions on enabling FIPS.

5 Set the OpenSSL library path to the FIPS enabled libraries.

6 SAS software can run in FIPS mode and use FIPS 140-2 validated modules. Starting with SAS 9.4M8, the operating system should be using supported cryptographic libraries that are FIPS 140-2 validated.. FIPS must be enabled on the operating system first starting with SAS 9.4M8.

The SAS system options that can affect if the SAS software is FIPS enabled are as follows:

- SAS system option NETENCRALG= must be set to SSL to use OpenSSL cryptographic libraries.
- SAS system option ENCRYPTFIPS= can be set either on the SAS server or client. The following table shows the results of whether FIPS is enabled on the operating system and the result of using the ENCRYPTFIPS= SAS system option.

Table 7.5 *Interaction of OS FIPS Mode and SAS ENCRYPTFIPS System Option*

OS FIPS Mode ENABLED	ENCRYPTFIPS Specified	Result
No	No	All encryption algorithms are allowed
No	Yes	Error is generated.
Yes	No	Runs in FIPS Mode. However, an error can occur when algorithms that are unsupported by FIPS are encountered.
Yes	Yes	Runs in FIPS mode with No errors.

Process to Build FIPS 140-2 Capable OpenSSL on UNIX and Linux

For SAS deployments prior to SAS 9.4M8, SAS ships OpenSSL libraries on UNIX and Linux operating systems. The last version of OpenSSL that was shipped with SAS Foundation and is available by hot fix is OpenSSL version 1.0.2. This OpenSSL version also has a compatible FIPS 140-2 certified module.

Note: OpenSSL version 1.0.2 can also be used with SAS 9.4M8.

The OpenSSL 2.0 FIPS Object Module requires a separate download that has to be built separately and then integrated into your main OpenSSL 1.0.2 build. You can use the following information to download the FIPS 140-2 certified module (for example, openssl-fips-2.0.16.tar.gz) and compile it.

- OpenSSL utility at OpenSSL Source to download the FIPS module.
- The latest version of the Security Policy is also needed. See FIPS Security Policy Version 2.0.16.

If you have downloaded FIPS 140-2 compliant OpenSSL libraries, your system administrator needs to set the environment path variables to pick up this software.

1. Go to the `<SASHome>/SASFoundation/9.4/bin` directory. This directory contains the sasenv script that sets the environment variables that are required by SAS.
2. Modify the sasenv_local file to customize environment variable values. Set the location of the FIPS 140-2 compliant libraries in the sasenv_local file. Depending on your operating system, set the LD_LIBRARY_PATH and the SHLIB_PATH to be the same, and set LIBPATH on AIX.

   For example, you might add the following code to the sasenv_local file:

   ```
   export LD_LIBRARY_PATH=<FIPS-library-path>:$LD_LIBRARY_PATH
   ```

   For more information, see "Contents of the !SASROOT Directory" in *SAS Companion for UNIX Environments*.

   Note: Different operating systems require the use of different library file extensions. For example, HP-UX, Linux, and Solaris use libcrypto.so.1.0.0 and libssl.so.1.0.0. AIX uses libcrypto.so and libssl.so. Refer to your operating system vendor documentation when using the vendor's OpenSSL libraries. There might be additional procedures that need to be followed to make the libraries work properly in your environment.
3. Prepend the customized library path in the script that is run before invoking SAS.
4. Use the SAS Deployment Wizard to enable FIPS after building your libraries.
5. Note that SAS system option NETENCRALG= must be set to SSL to configure a FIPS 140-2 capable system.

6 Assign the ENCRYPTFIPS system option.

CAUTION

Use caution when using ENCRYPTFIPS Turning on the ENCRYPTFIPS option is not generally recommended, unless absolutely required by your site's policy. If the ENCRYPTFIPS option is turned on, the SAS server-based TLS clients attempt to load a special subset of OpenSSL libraries, which are contained as part of the OpenSSL FIPS Object Module. Because these libraries are not present by default, you must follow the preceding process to download and compile them in accordance with the specific instructions specified by the FIPS standard. See "ENCRYPTFIPS" on page 135.

8

Configure TLS Certificates and FIPS on Windows

Steps to Add Certificates for TLS to Windows

Follow the steps in this topic to create or obtain digital certificates and import them to the Windows Certificates store and to the SAS truststore.

1 Configure TLS on Windows.

2 Request a Digital Certificate from the Microsoft Certificate Authority or Request a Digital Certificate from a Certificate Authority That Is Not Microsoft.

3 Add Your Certificates to the Windows CA Store.

4 Import the Client Certificate into the Windows Personal Machine Store.

5 (Optional) Convert between PEM and DER Certificate File Formats .

6 Use the SAS Deployment Manager to Manage Certificates in the Trusted CA Bundle.

7 (Optional) Configure FIPS 140-2 Capable TLS on Windows.

TLS on Windows: System and Software Requirements

The system and software requirements for using TLS on the Windows operating environment are as follows:

- a computer that runs Windows 2000 (or later).
- depending on your configuration, access to the internet and a web browser.
- the TCP/IP communications access method.
- Microsoft Certificate Services add-on software.
- if you run your own CA, the Microsoft Certificate Authority application (which is accessible from your web browser).
- for SAS/CONNECT, a client session that runs on a computer that has a Trusted CA Certificate. This is necessary in order for a SAS/CONNECT client session to connect to a SAS/CONNECT server session via a Windows spawner using TLS encryption.

  The Windows spawner must run on a server that has a Trusted CA Certificate and a Personal Certificate.
- knowledge of your site's security policy, practices, and technology. The properties of the digital certificates requested depends on the security policies that have been adopted at your site.

Configure TLS and Request Digital Certificates on Windows

Configure TLS on Windows

Complete information about configuring your Windows operating environment for TLS is contained in the Windows installation documentation and at How to enable TLS 1.2.

The following keywords might be helpful when searching the Microsoft website:

- digital certificate services
- digital certificate authority
- digital certificate request
- site security planning

After generating or obtaining the CA, server, and client certificates that you need to configure TLS, configure the SAS Foundation and SAS IOM servers and spawners to use TLS and specify the certificate files and locations being used.

For more information about IOM servers, see TLS support for IOM Servers.

Request a Digital Certificate from the Microsoft Certificate Authority

The method for requesting a digital certificate depends on the CA that you use. If you are using the Microsoft Certificate Authority, use the Certificate Request wizard to request a digital certificate from an active enterprise CA. The Certificate Request wizard lists all digital certificate types that the user can install.

First, perform the following steps to bring up the Microsoft Management Console (MMC):

1. Click the Windows **Start** button, select **Run**, enter `mmc`, and click **OK**.
2. In the Console window, select **File** ⇨ **Add/Remove Snap-in**.
3. Select **Certificates** from the list of available snap-ins, and click **Add**.
4. In the window that appears, click **My User Account** to request a user certificate or **Computer Account** to request a computer certificate. Click **Next**.
5. If you selected **Computer Account**, click **Local Computer** (the computer this console is running on) and click **Finish**.

6 If you selected **My User Account,** click **Finish**.

7 Click **OK**.

Perform the following steps to use the Certificate Request Wizard for requesting a certificate from an active enterprise CA that is configured to issue the digital certificate:

1 In the Console window, expand **Certificates (Local Computer)** or **Current User** on the left.

2 Expand **Personal**. Click **Certificates**.

3 Right-click the **Personal** folder. Click **All Tasks**. Click **Request New Certificate**.

4 From the Certificate Request Wizard, click **Next**.

5 On the **Certificate Types** page, select the certificate template that you want to request (server, client, CA). The list is limited to the certificate templates for which either the current user or local machine have Read and Enroll permissions. After you select the certificate template, click **Next.**

6 On the **Certificate Friendly Name and Description** page, in the **Friendly Name** box, enter a descriptive name for the requested certificate. Click **Next**.

7 Click **Finish**.

8 Click **OK**. After the CA issues the requested digital certificate, the digital certificate is automatically installed in the Certificate store.

Request a Digital Certificate from a Certificate Authority That Is Not Microsoft

The method of requesting a digital certificate depends on the CA that you use. If you are using a CA that is not Microsoft, you can create an offline request using the Certificate manager console or any third-party application that generates digital certificates. Users can perform the following tasks to request digital certificates that are not issued by the Microsoft CA.

1 First, perform the following steps to bring up the Microsoft Management Console (MMC):

   a Click the Windows **Start** button, select **Run**, enter `mmc`, and click **OK**.

   b In the Console window, select **File** ⇨ **Add/Remove Snap-in**.

   c Select **Certificates** from the list of available snap-ins, and click **Add**.

   d In the window that appears, click **My User Account** to request a user certificate or **Computer Account** to request a computer certificate. Click **Next**.

   e If you selected **Computer Account**, click **Local Computer** (the computer this console is running on) and click **Finish**.

   f If you selected **My User Account,** click **Finish**.

g Click **OK**.

2 Create an Offline Certificate Request using the Windows Certificate manager console by performing the following steps:

a From the Certificate manager console, expand **Certificates (Local Computer)** , and expand **Personal**. Right-click **Certificates**, navigate to **All tasks**, **Advanced Operations**, and select **Create custom request**.

b The Certificate Enrollment Wizard opens. Review the “Before You Begin” section and click **Next**.

c Under **Custom request**, leave the default "No template" option. Click **Next**.

d On the **Certificate Information** page, expand **Details**, and click **Properties**.

e From the **Certificate Properties** page, select the **General** tab and fill out the **Friendly name** and **Description** values. These values are not required but are useful to distinguish your certificate from the other certificates that you have installed.

f From the **Certificate Properties** page, select the **Subject** tab. Add values to the **Subject name** and **Alternative name** attributes.

For **Common Name**, specify a Fully Qualified Domain Name (FQDN). For **DNS**, specify a Fully Qualified Domain Name. Use best practices when creating certificates on page 70.

g From the **Certificate Properties** page, select the **Extensions** tab. Add the key usage and extended key usage that you need. This usage includes server and client authentication.

h From the **Certificate Properties** page, select the **Private Key** tab. Expand **Cryptographic Service Provider**. Select the cryptographic service provider (for example, RSA). Expand **Key options** and select the key size. Expand **Select Hash Algorithm**. For **Hash Algorithm**, select an SHA algorithm.

i Click **OK**.

j On the **Where do you want to save the offline request?** page, give your certificate request (CSR) file a name and save it to a location on your computer. Make sure the file format is set to Base 64.

k Click **Finish**.

3 Use OpenSSL to generate a certificate signing request. See “ Generate Digital Certificates Using OpenSSL ” on page 83.

Note: The Windows operating environment can import digital certificates that were generated in the UNIX operating environment. To convert from UNIX (PEM format) to Windows (DER format) before importing, see “Convert between PEM and DER File Formats for TLS” on page 115.

4 Submit your certificate request to a Certificate Authority to process your request and issue a certificate. The certificate request is a text file. Usually, you are required to copy the text from the file and enter it into an online submission form on the Certificate Authority website. Contact your Certificate Authority directly for instructions on the process for submitting your certificate request.

5 When your CA has processed your request and issued the certificate, download it to your server so that it can be imported. See Add the certificates to the Windows CA store on page 112.

Add Your Certificates to the Windows CA Store

If you are providing your own self-signed or site-signed certificates, then you must add the CA root certificate and all of its intermediate certificates to the Windows certificates stores using the Windows Certificates snap-in.

Note: If you have not already done so, you must add your CA root and intermediate certificates to the trusted CA bundle or to the SAS Private JRE. For more information, see "Add a Certificate to the Trusted CA Bundle" on page 94 or "Add Your Certificates to the SAS Private JRE" on page 99.

To add CA root and intermediate certificates to the Windows certificates stores on your local computer (Certificates (Local Computer)), perform these steps:

1 Click the Windows **Start** button, select **Run**, enter `mmc`, and click **OK**.

2 In the Console window, select **File** ⇨ **Add/Remove Snap-in**.

3 Select **Certificates** from the list of available snap-ins, and click **Add**.

4 In the window that appears, select **Computer account**, and click **Next**.

5 In the window box that appears, click **Finish**.

6 In the window that appears, click **OK**.

7 In the Console window, expand **Certificates (Local Computer)** on the left.

8 Right-click **Trusted Root Certification Authorities**, and select **All Tasks** ⇨ **Import**.

9 On the Certificate Import Wizard page, click **Next**.

10 On the second wizard page, click **Browse**, navigate to the location that contains your CA root certificate and select the appropriate certificate. Click **Next**.

11 Make sure that **Place all certificates in the following store** is selected, and click **Next**.

12 Click **Finish**.

13 Click **OK**.

14 In the Console window, expand **Trusted Root Certification Authorities** to make sure that the certificate that you imported is listed.

15 Add the intermediate certificates. Right-click **Intermediate Certification Authorities**, and select **All Tasks** ⇨ **Import**.

16 On the Certificate Import Wizard page, click **Next**.

17 On the second wizard page, click **Browse**, navigate to the location that contains your intermediate certificates, and select the appropriate certificate. Click **Next**.

18 Make sure that **Place all certificates in the following store** is selected, and click **Next**. The Certificate store shows Intermediate Certification Authorities.

19 Click **Finish**.

20 Click **OK**.

21 In the Console window, expand **Intermediate Certification Authorities** to make sure that the certificate that you imported is listed.

22 Repeat previous instructions on all Windows machines in your SAS deployment.

Import the Client Certificate to the Windows Personal Machine Store

Import the client certificates into the Windows Personal store on the local machine using the Microsoft Management Console (MMC). After the certificates have been imported, you will then need to grant permission to the authenticated users who use the client certificates private key. Perform the following steps to import the client certificates and grant permission to authenticated users:

1 Start the Microsoft Management Console (MMC). Right-click the Windows **Start** menu and select **Run**.

2 In the Run window, enter `mmc`, and press **OK**.

3 Add the Certificate manager console.

   a From the Microsoft Management Console window, click **File** and **Add/Remove Snap-in** from the drop-down menu.

   b In the Add or Remove snap-ins window, select **Certificates** from the **Available snap-ins** list.

   c Add **Certificates** to the **Selected snap-ins** list.

   d Select **Computer account** and click **Next**.

   e Select **Local** computer (the computer that this console is running on). Click **Finish**.

   f Click **OK**.

4 Expand the **Certificates (Local Computer)** list and click **Personal**.

5. From the left pane, right-click and select the **Certificates** node within the **Certificates (Local Computer)**, **Personal**, **Certificates** hierarchy to view a list of certificates. Select **All Tasks**, and then **Import** from the drop-down menu.

6. From the Certificate Import Wizard, confirm that **Local Machine** is selected for the **Store Location**, and click **Next**.

   a. Select **Browse** to locate the file to import. Expand the list of file types and select **All Files** from the drop-down menu.

   b. From the list of files, select the certificates that you want to import. In this example, the client certificate is selected. The certificate is in PFX format that contains the certificate and private key file, customerCert.pfx. Click **Next**.

   c. From the **Private Key Protection** page of the Certificate Import Wizard, enter the password for customerCert.pfx. Select **Include all extended properties**. Do not select **Enable strong private key protection** and do not select **Mark this key as exportable**. Click **Next**.

   d. From the **Certificate Store** page of the Certificate Import Wizard, select **Place all certificates in the following store**. Place the certificates in the Personal Certificate store. Click **Next**.

   e. From the **Completing the Certificate Import Wizard** page, click **Finish**.

   f. Verify that you receive the message `The import was successful`, and click **OK**.

7. Observe that the imported certificates are now listed in the Microsoft Management Console.

In order to use the client's private key, the client certificate's private key must be readable. Perform the following tasks to grant Read permission to the authenticated users who use the client certificate's private key.

1. Start the Microsoft Management Console (MMC). Right-click the Window's **Start** menu and select **Run**.

2. From the Run window, enter `mmc`, and press **OK**.

3. Right-click the client certificate that you recently imported.

4. On the pop-up menu, select **All Tasks**, **Manage Private Keys**. A Permissions window appears.

5. From the Permissions window, within the **Security** tab, add **Authenticated Users**.

6. Ensure that authenticated users have Read permission. Select **Allow Read item**, deselect **Allow Full control**, and deselect **Allow Special permissions**. Click **OK**.

Convert between PEM and DER File Formats for TLS

By default, TLS files are created in Privacy Enhanced Mail (PEM) format. TLS files that are created in Windows operating environments are created in Distinguished Encoding Rules (DER) format.

Under Windows, you can import a file that is created in either PEM or DER format. However, a digital certificate that is created in DER format must be converted to PEM format before it can be included in a trust list on UNIX.

You can use OpenSSL. Here is an example of converting a server digital certificate from DER input format to PEM output format:

```
openssl x509 -inform DER -outform PEM -in server.der -out server.pem
```

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

```
openssl x509 -inform PEM -outform DER -in server.pem -outserver.der
```

Note: Files with the .cert, .cer, or .crt extensions are recognized by Windows as a certificate. For more information, see "Certificate Encoding Formats and Extensions" on page 66.

Use the SAS Deployment Manager to Manage Certificates in the Trusted CA Bundle

For information, see "Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager" on page 92.

Validating Certificates between Clients and Servers

Clients and servers exchange and validate each other's digital certificates. The following list provides some details:

1. Digital certificates for the CA, the server, and the client are generated and imported into the appropriate certificate store. Refer to "Configure TLS and Request Digital Certificates on Windows" on page 109.
2. The Windows client verifies the TLS-enabled server's certificate against the Certificate Authority (CA) list. The client has to know about all of the CAs in the server's certificate chain in order to validate the server certificate. The Windows CA certificate is installed using Microsoft Certificate Services. The certificate must be a trusted root certificate in the user or machine certificate store.
3. The client connects to a TLS-enabled server.
4. The TLS-enabled server sends its certificate to the client. The Window's server certificate is installed using Microsoft Certificate Services and is located in the user or machine certificate store. SAS uses the SSLCERTISS/SSLCERTSERIAL or the SSLCERTSUBJ/ SSLCERTISS system options to locate the server certificate.

   The system options are specified in the server's invocation command. For more information, see "SAS System Options for Encryption" on page 133.
5. The server can also validate the client's certificates. Refer to the previous steps.

Enable Elliptical Curve Ciphers for Use with TLS 1.2

You might need to enable Elliptic Curve Cryptography (ECC) cipher suites that are available when using TLS 1.2. You can enable ECC using Windows PowerShell or through Group Policy. See Enabling Elliptic Curves.

For information about the TLS versions and cipher suites that are supported by SAS, see "Cryptographic Library Support Starting with SAS 9.4M8" and "TLS Versions and Cipher Suites Supported Starting with SAS 9.4M8".

For information about the cipher suites that are supported for various versions of Windows Schannel SSP, see Cipher Suites in TLS/SSL (Schannel SSP).

Configure FIPS 140-2 Capable TLS on Windows

SAS uses Security Support Provider (SSP) to provide TLS for Windows. Schannel SSP implements the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) internet standard authentication protocols.

Starting with SAS 9.4M8, SAS no longer ships cryptographic libraries that are used for Windows to provide TLS. Instead, SAS uses the cryptographic libraries that are provided and installed on the Windows operating system. The cryptographic modules for FIPS are also not provided by SAS. Administrators need to verify that all cryptographic modules that are installed are FIPS 140-2 validated. See FIPS 140-2 Validation for information about enabling FIPS 140-2 and about FIPS validation on Windows.

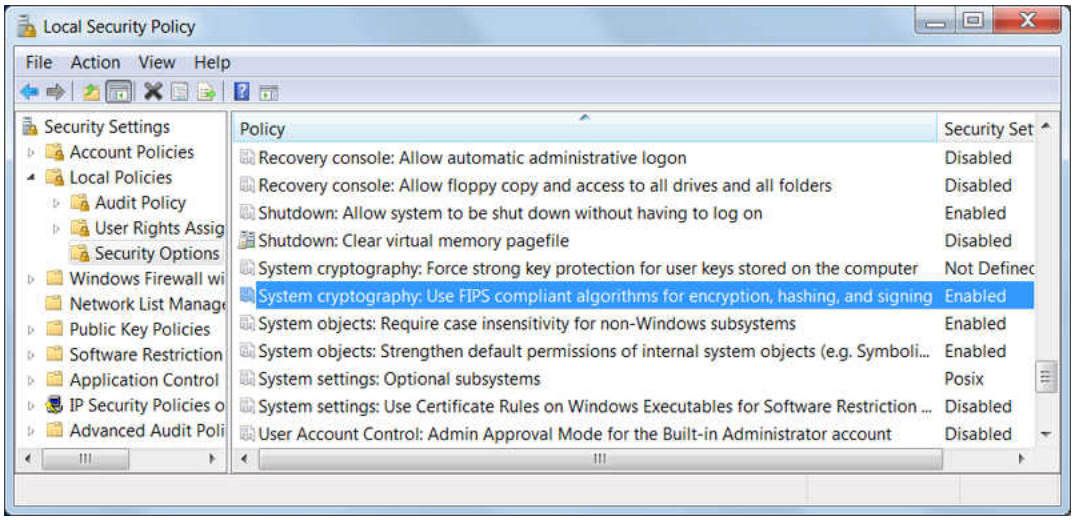
Prior to SAS 9.4M8, SAS provided the cryptographic libraries that are used by Windows to provide TLS and FIPS capable libraries.

To put the library into FIPS 140-2 compliant mode, enable the **System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing** setting under your Local Security Policy or as part of Group Policy. This setting informs applications that they should use only cryptographic algorithms that are FIPS 140-2 compliant and in compliance with FIPS approved modes of operation.

To check that your Windows server is configured for FIPS, go to the Windows **Start Menu** ⇨ **Search** and enter “Local Security Policy”. The Local Security Policy window appears.

1. In the left pane of the **Security Policies**, expand **Local Policies**.
2. Click **Security Options**.
3. In the right pane, scroll down to **System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing**. Make sure that the item is enabled.

Figure 8.1 *FIPS Encryption Enabled on Windows*



9

Configure TLS Certificates and FIPS on z/OS

System and Software Requirements to Configure TLS on z/OS

IMPORTANT Starting with SAS 9.4M8, SAS supports the use of IBM System SSL to provide cryptographic services for z/OS. SAS no longer provides OpenSSL libraries to support TLS on z/OS, but instead uses System SSL. For more information, see "IBM System SSL Provides OpenSSL Capabilities for z/OS in SAS 9.4M8".

The system and software requirements for using TLS on z/OS operating environments are as follows:

- a computer that runs z/OS. For use with SAS 9.4M8, you need a z/OS system that is configured to run IBM System SSL.
- the TCP/IP communications access method.
- The RADCERT command can be used to generate certificates for use with z/OS. The RACF system administrator provides access to this utility.

  Starting with SAS 9.4M8, you can also use the gskkyman utility to generate certificates for use with System SSL. The gskkyman utility can be used to create self-signed certificates and to generate certificate requests. It can also be used to create a key database (.kdb) and a corresponding stash file (.sth), as well as to import existing certificates into a key database. For more information, see Certificate and key management
- knowledge of your site's security policy, practices, and technology. The properties of the digital certificates that you request are based on the security policies that have been adopted at your site.

Setting Up Digital Certificates on z/OS Using RACDCERT

IMPORTANT Starting with SAS 9.4M8, SAS supports the use of IBM System SSL to provide cryptographic services for z/OS. SAS no longer provides OpenSSL libraries to support TLS on z/OS, but instead uses System SSL. For more information, see "IBM System SSL Provides OpenSSL Capabilities for z/OS in SAS 9.4M8".

Because of this change, certificates for use with System SSL must be of type .p12, .kdb. or SAF keyring. See "How to Use SAS 9.4M8 with IBM System SSL ".

Step 1 Authorize Access to the RACDCERT Command

To use z/OS as your trusted Certificate Authority (CA), the RACF administrator must provide authorized access to the RACDCERT command in order to set up the CA and to create and sign certificates. The trusted administrator must have CONTROL access to profiles in the FACILITY class.

For this example, DIGICERT is the trusted CA that is used for this example.

- IRR.DIGTCERT.ADD
- IRR.DIGTCERT.DELETE
- IRR.DIGTCERT.EXPORT
- IRR.DIGTCERT.GENCERT
- IRR.DIGTCERT.LIST

Step 2 Create the Digital Certificate for the CA

The tasks that you perform to generate a CA certificate, the server certificate, and the client certificate 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 by using RACDCERT. After Proton, Inc. becomes a CA, it can serve as a CA for issuing digital certificates to clients (users) and servers on its network.

Perform these tasks:

Note: The instructions here apply to UNIX and Linux.

1 Request a digital CA certificate. Here is an example of a request:

```
RACDCERT GENCERT CERTAUTH +
SUBJECTSDN( +
  CN('proton.com') +
  C('US') +
  SP('North Carolina') +
  L('Cary') +
  O('Proton Inc.') +
  OU('IDB') +
) +
ALTNAME( +
  EMAIL('Joe.Bass@proton.com') +
) +
```

```
WITHLABEL('Proton CA')
```

2 Export the CA certificate in PEM format:

```
RACDCERT CERTAUTH EXPORT(LABEL('Proton CA')) +
DSN(CA.CERT)
```

3 Copy the certificate to the UNIX file system. Use the TSO OPUT and OCOPY commands to copy the files to your UNIX file system:

Note: TLS certificate and key files must reside in the z/OS UNIX file system. The OpenSSL library cannot read z/OS data sets.

```
cp //ca.cert ca.cert
```

4 Convert the certificate file to ASCII format:

Note: PEM format certificate files must be converted to ASCII format. The OpenSSL library code used with SAS cannot read EBCDIC text.

```
iconv -f ibm-1047 -t iso8859-1 ca.cert >ca.cert.ascii
```

The creation of the CA digital certificate is complete.

A root CA digital certificate is self-signed, which means that the digital certificate is signed using the private key that corresponds to the public key that is in the digital certificate. Except for root CAs, digital certificates are usually signed using a private key that corresponds to a public key that belongs to someone else, usually the CA.

Prior to SAS 9.4M8, the location of the CA digital certificate is specified using the SSLCERTLOC= system option, which is automatically set to **<SASHOME>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem**.

Step 3 Create the Server and Client Digital Certificates

Perform these tasks to create a digital certificate for a server and a client. The steps are identical for the server and the client. This example shows the tasks for the server.

1 Request a signed server certificate.

Here is an example of a request for a signed server certificate for user SERVER that runs on **proton.zos.com**.

```
RACDCERT GENCERT ID(SERVER) +
SUBJECTSDN( +
  CN('proton.zos.com') +
  C('US') +
  SP('North Carolina') +
  L('Cary') +
  O('Proton Inc.') +
  OU('IDB') +
) +
```

```
ALTNAME( +
  EMAIL('Joe.Bass@proton.com') +
) +
WITHLABEL('Proton Server') +
SIGNWITH(CERTAUTH LABEL('Proton CA'))
```

2 Export the server certificate and key that are specified in PKCS#12 DER encoding package format:

Note: The PKCS#12 DER encoding package is the format used by the RACDCERT utility to encode the exported certificate and private key for an entity, such as a server. It is a binary format.

```
RACDCERT ID(SERVER) EXPORT(LABEL('Proton Server')) +
DSN(SERVER.P12) +
PASSWORD('abcd')
```

3 Copy the certificate to the UNIX file system:

Note: The PKCS#12 DER encoding package file must reside in the z/OS UNIX file system. The OpenSSL library cannot read z/OS data sets. Because the file is already in binary format, its conversion to ASCII is unnecessary.

```
cp //server.p12 server.p12
```

The creation of the server digital certificate and key is complete.

A PKCS#12 DER encoding package is the format that RACDCERT uses to export a certificate and a key for an entity. The exported package file contains both the certificate and the key. The content of the package file is secure by using the password that is specified in the RACDCERT EXPORT command.

Prior to SAS 9.4M8, specify a server or client PKCS#12 package using the SSLPKCS12LOC= system option. Specify the password for the package using the SSLPKCS12PASS= option. Starting with SAS 9.4M8, you can specify a server or client PKCS#12 package using the SSLKEYRINGFILE= system option. Specify the password for the package using the SSLKEYRINGPW= option.

Note: For the server, the Common Name must be the name of the computer that the server runs on (for example, **proton.zos.com**).

Step 4. View Digital Certificates

To view a digital certificate, issue these commands:

```
RACDCERT CERTAUTH LIST(LABEL('Proton CA'))
RACDCERT ID(SERVER) LIST(LABEL('Proton Server'))
```

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.

To read the certificate files, issue these commands:

```
RACDCERT CHECKCERT(CA.CERT)
RACDCERT CHECKCERT(SERVER.P12) PASS('abcd')
```

Step 5 Create a CA Trust List Using OpenSSL

IMPORTANT The information in this topic applies to SAS 9.4 versions prior to SAS 9.4M8. Starting with SAS 9.4M8, to use the SAS provided truststore and add your CA certificates to the bundle of trusted CA certificates, see SAS Usage Note 69954. These instructions provide a way to download the truststore in a format that System SSL can use (trustedcerts.kdb and trustedcerts.sth) and provide instructions for adding additional CA certificates to the truststore.

Note: Starting with SAS 9.4M3, you can use SAS Deployment Manager after Installation to add certificates to the bundle of trusted CA certificates (trustedcerts).

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.

If there is only one CA to trust, in the client application specify the name of the file that contains the OpenSSL CA 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. They can be added to the file in any order. To manually create a new trust list, use the following template:

```
(Your Server Certificate)

-----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-----
```

Because the digital certificate is encoded, it is unreadable. Therefore, the content of the digital certificate in this example is represented as `<PEM encoded certificate>` . The content of each digital certificate is delimited using 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, and CA's return .crt files (Microsoft returns .cer files). Instead of manually cutting and pasting these files together (regardless of your file extension), you can use the UNIX **cat** command to concatenate the certificate authority files together. For example, you can take an intermediate authority certificate file, a root authority certificate file, and primary certificate file and concatenate them into a single PEM file. All the certificates must be encoded in PEM format and in ASCII format.

An example of concatenating certificates is as follows:

```
cat server.pem > certchain.pem
cat intermediateCA.pem >> certchain.pem
cat rootCA.pem >> certchain.pem
```

Note: You can place these files in any order.

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 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 cert.der -inform der -text -noout
```

Note: If you are including a digital certificate that is stored in DER format, you must first convert it to PEM format. For more information, see "Convert between PEM and DER File Formats Using OpenSSL" on page 91.

Certificate Management Using the Bundle of Trusted CA Certificates (trustedcerts)

Use SAS Deployment Manager to Manage Certificates in the Trusted CA Bundle

For z/OS deployments prior to SAS 9.4M8, the SAS Deployment Manager can be used to add certificates to the SAS provided trusted CA bundle of certificates. See "Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager" on page 92. For instructions for using the SAS Deployment Manager, see the *SAS Deployment Wizard and SAS Deployment Manager 9.4: User's Guide*.

How to Use the Trusted CA Bundle of Certificates (trustedcerts) with 9.4M8 for System SSL

Starting with SAS 9.4M8, SAS supports the use of IBM System SSL to provide cryptographic services for z/OS. SAS no longer provides OpenSSL libraries to support TLS on z/OS, but instead uses System SSL. Certificate files for use with System SSL must be of a key database file (.kdb) type or PKCS#12 file (.p12) type. SAS Deployment Manager uses a file of PEM format (trustedcerts.pem).

Starting with SAS 9.4M8, to use the SAS provided truststore and add your CA certificates to the bundle of trusted CA certificates, see SAS Usage Note 69954. These instructions provide a way to download the truststore in a format that System SSL can use (trustedcerts.kdb and trustedcerts.sth) and provide instructions for adding additional CA certificates to the truststore.

Convert between PEM and PKCS#12 File Formats Using OpenSSL

Starting with SAS 9.4M8, because System SSL and its utilities for generating certificates do not accept certificate files in PEM format, you might need to convert a

PEM file to a PKCS#12 file. Here is an example command to convert a .pem file to a .p12 file using OpenSSL. The resulting .p12 file is an ASCII file.

```
openssl pkcs12 -export -in certificate.pem -out certificate.p12 -nokeys -
password password
```

How to Use SAS 9.4M8 with IBM System SSL

Overview

Starting with SAS 9.4M8, SAS no longer delivers the OpenSSL libraries that are used to provide TLS on z/OS. Instead, SAS 9.4M8 uses the cryptographic libraries that are provided by System SSL to provide encryption for data in motion. IBM System SSL uses OpenSSL libraries. The GSK API's are used to interface with SAS software. For more information, see "IBM System SSL Provides OpenSSL Capabilities for z/OS in SAS 9.4M8".

Here are items to be aware of when using System SSL:

- FIPS 140-2 is supported using System SSL.
- TLS 1.3 is not yet supported using System SSL.
- To use the SAS provided truststore and add your CA certificates to the trusted bundle of CA certificates, see SAS Usage Note 69954. These instructions provide a way to download the truststore in a format that System SSL can use. You cannot use the SAS Deployment Manager to manage certificates for z/OS in SAS 9.4M8.
- Certificate and key management with SAS 9.4M8 use the naming convention of "keyring". However, the concept of using keyrings is not required. You can still provide certificates and keys using a PKCS#12 file. See SSLKEYRINGFILE= for information about all the types of files that can be used.

  You can follow the instructions in SAS Usage Note 69954 to convert files to PKCS#12 format if needed.

Configure Certificates Using SAS System Options That Work with System SSL

Configure certificates and perform the following:

1. Generate certificates or provide the certificates for use with System SSL on z/OS. You can use RACDCERT and the gskkyman utility to manage certificates. See Certificate and key management.

You need to generate certificates that meet the criteria specified using SAS system option SSLKEYRINGFILE=.

2 Starting with SAS 9.4M8, new SAS system options are used to manage certificates. These are SSLKEYRINGFILE=, SSLKEYRINGLABEL=, SSLKEYRINGPW=, and SSLKEYRINGSTASHFILE=. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8".

   For an example where the SAS/CONNECT spawner is configured to use the SAS system options, see "Example 1: SAS/CONNECT on z/OS Using System SSL with SAS 9.4M8".

3 To provide tracing and debugging for System SSL, you can specify SAS system options SSLGSKTRACE=, SSLHWDETECTMESSAGE=, SSLICSFERRORMESSAGE=, and SSLGSKTRACEFILE=. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8".

   Logging options that are provided by SAS can be used to provide more information. See "Encryption: SAS Logging Facility".

FIPS 140-2 Support Is Provided with System SSL on z/OS with SAS 9.4M8

IBM System SSL provides support for FIPS 140-2. To configure FIPS, perform the following:

1 Ensure that FIPS is enabled on the operating system. See SSL started task.

2 Make sure that the certificates that you are providing meet System SSL standards for FIPS. See Certificates and Certificate Stores.

3 Use the SAS system options to specify the FIPS 140-2 certificates that are being used. These are SSLKEYRINGFILE=, SSLKEYRINGLABEL=, SSLKEYRINGPW=, and SSLKEYRINGSTASHFILE=. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8".

4 Use SAS system option ENCRYPTFIPS= to place SAS software in FIPS mode.

5 If you are using the gskkyman utility to create a key database, specify FIPS mode when prompted.

Following are additional links to IBM documentation when using FIPS with System SSL:

- SYSTEM SSL and FIPS 140–2
- TLS Protocol
- System SSL module verification setup

Example 1: SAS/CONNECT on z/OS Using System SSL with SAS 9.4M8

Start a SAS/CONNECT spawner on z/OS.

This example shows how to configure a SAS/CONNECT spawner on z/OS with SAS 9.4M8 using SAS system options that work with System SSL:

```
export STEPLIB=INSTALL_HLQ.LIBRARY:INSTALL_HLQ.LIBE
cntspawn -service 1234 -mgmtport 4321 -sascmd "/path/spawnmvs.sh -
sslkeyringfile /path-to-cert/cert.p12 -sslkeyringlabel cert-label-name
-sslkeyringpw password -netencrypt -netencralg ssl"
-netencrypt -netencralg ssl -sslkeyringfile /path/cert.p12 -
sslkeyringlabel cert-label-name -sslkeyringpw password
```

Perform a SAS/CONNECT sign-on to the z/OS spawner from a SAS 9.4M8 client on z/OS:

```
options comamid=tcp;
OPTIONS SSLKEYRINGFILE="/path/cert.p12";
OPTIONS SSLKEYRINGLABEL="cert-label-name";
OPTIONS SSLKEYRINGPW="password";
%let mvshost=targethost.mvs.com;
signon mvshost.1234 user=user1 pwd="password"
```

From a SAS 9.4M8 client on Linux, perform a SAS/CONNECT sign-on to the spawner on z/OS. Specify the following:

```
options SSLCALISTLOC="/path/sas.ca.cert2";
%let mvshost=targethost.mvs.com;
```

Example 2: SAS/SHARE on z/OS Using System SSL with SAS 9.4M8

The following example shows a server that uses a key database with default key (avoids the need to specify SSLKEYRINGLABEL=) for encrypting data. The key password is stored in a stash file.

```
//SYSIN    DD  DATA,DLM=$$
%let tcpsec=_secure_;
OPTIONS SSLKEYRINGFILE='/path-to-cert/cert.kdb';
OPTIONS SSLKEYRINGSTASHFILE='/path-to-cert/cert.sth';
options comamid=tcp;
options netencrypt;
options netencralg=ssl;
proc server id=shrid authenticate=required;run;
```

```
$$
```

The client needs access to the CA certificate. This certificate is stored in a key database and uses a stash file for the password. The remote server is the IBM Database 2 (Db2).

```
//SYSIN   DD  DATA,DLM=$$
%let tcpsec=_secure_;
OPTIONS SSLKEYRINGFILE='/path-to-cert/serverhost.ca.kdb';
OPTIONS SSLKEYRINGSTASHFILE='/path-to-cert/serverhost.ca.sth';
options comamid=tcp;
options netencrypt;
options netencralg=ssl;
proc sql;
connect to remote(server=serverhostid user=userid pw=password
dbms=db2 dbmsarg=(ssid=db2));
select * from connection to remote(select * from CLASS);
quit;
$$
```

PART 5

Reference

10

SAS System Options for Encryption

SAS System Options Deprecated for Use with SAS 9.4M8 on z/OS

The following SAS System Options are no longer supported for use with z/OS starting with SAS 9.4M8. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8" on page 167 for SAS system options that are used to provide TLS encryption for z/OS starting with SAS 9.4M8.

- SSLCACERTDIR on page 143
- SSLCALISTLOC on page 145
- SSLCERTLOC
- SSLCIPHERLIST on page 154
- SSLCRLLOC
- SSLPKCS12LOC
- SSLPKCS12PASS
- SSLPVTKEYLOC
- SSLPVTKEYPASS
- SSLSNIHOSTNAME

PROC HTTP Can Use TLS System Options Locally

Beginning with SAS 9.4M6, TLS environment variables and system options can be applied locally using the PROC HTTP SSLPARMS statement. For more information, see "SSLPARMS" in *Base SAS Procedures Guide*.

Dictionary

ENCRYPTFIPS System Option

Specifies that the SAS encryption services use FIPS 140-2 validated encryption algorithms.

Client:	Optional
Server:	Optional
Valid in:	SAS invocation, configuration file, SAS/CONNECT spawner command line
Categories:	Communications: Networking and Encryption System Administration: Security
PROC OPTIONS GROUP=	Communications SECURITY
Default:	NOENCRYPTFIPS
Restrictions:	When the ENCRYPTFIPS option is specified, the NETENCRYPTALGORITHM= system option must be set to AES or SSL. If a different algorithm is specified, an error message is output. The ENCRYPTFIPS option is not supported on z/OS for TLS prior to SAS 9.4M8..
Operating environment:	UNIX, Linux, Windows, z/OS
Notes:	The ENCRYPTFIPS option is supported on z/OS for TLS starting with SAS 9.4M8. However, note that SAS uses System SSL to provide the cryptographic libraries used and installed on the operating system. The ENCRYPTFIPS option is configured only at start-up. However, you can see that the option is configured when you view the OPTIONS statement or the SAS System Options window.

Examples:	Here is an example of configuring the ENCRYPTFIPS option on UNIX:
	`-encryptfips -netencryptalgorithm SSL`
	Here is an example of configuring the ENCRYPTFIPS option on z/OS:
	`encryptfips netecryptalgorithm="aes"`
	Here is an example of configuring the ENCRYPTFIPS option on Windows:
	`-encryptfips -netencralg "SSL"`

Syntax

ENCRYPTFIPS

Syntax Description

ENCRYPTFIPS

specifies that SAS encryption is using FIPS 140-2 compliant encryption algorithms.

With this option enabled, SAS verifies that all of the SAS servers have been configured to use the FIPS approved cryptographic libraries. ENCRYPTFIPS makes sure that the AES algorithm or TLS protocols are used in FIPS mode.

CAUTION

Use ENCRYPTFIPS with caution. Turning on the ENCRYPTFIPS option is not generally recommended for TLS, unless it is required by your site's security policy. If the ENCRYPTFIPS option is turned on, the SAS server-based TLS clients attempt to load a special subset of OpenSSL libraries, which is contained as part of the OpenSSL FIPS Object Module. Because these libraries might not be present by default, you first need to verify that you have built and installed FIPS validated cryptographic libraries. Starting with SAS 9.4M8, you also need to enable FIPS on the operating system before you use the ENCRYPTFIPS system option.

NOENCRYPTFIPS

specifies that the cryptographic libraries are not limited to FIPS 140-2 verified algorithms.

Turning off the SAS system option ENCRYPTFIPS does not impact the ability of SAS to use FIPS approved encryption algorithms such as the Advanced Encryption Standard (AES), nor does it prevent SAS from leveraging strong FIPS approved cipher suites when acting as a TLS client.

Details

The ENCRYPTFIPS option is provided by SAS primarily as a mechanism to help ensure that SAS is configured to leverage the encryption algorithms and cipher suites specified by the FIPS 140-2 standard and that libraries are validated for compliance when loaded. With this option enabled, SAS verifies that all of the SAS servers have been configured to use the FIPS approved cryptographic libraries. ENCRYPTFIPS makes sure that the AES algorithm or TLS protocols are used in FIPS mode.

The ENCRYPTFIPS option limits the encryption services to those services that are part of the FIPS 140-2 specification. Read more about Security Requirements for Cryptographic Modules at Federal Information Processing Standards Publication. Refer to “FIPS 140-2 Standards Compliance” on page 15 for an overview of FIPS 140-2 standards.

Information to note when using the ENCRYPTFIPS system option:

- With this option enabled, SAS verifies that all of the SAS servers have been configured to use the FIPS approved cryptographic libraries. ENCRYPTFIPS makes sure that the AES algorithm or TLS protocols are used in FIPS mode.

  CAUTION

  Use ENCRYPTFIPS with caution. Turning on the ENCRYPTFIPS option is not generally recommended for TLS, unless it is required by your site's security policy. If the ENCRYPTFIPS option is turned on, the SAS server-based TLS clients attempt to

load a special subset of OpenSSL libraries, which is contained as part of the OpenSSL FIPS Object Module. Because these libraries might not be present by default, you first need to verify that you have built and installed FIPS validated cryptographic libraries. Starting with SAS 9.4M8 you also need to enable FIPS on the operating system before you use the ENCRYPTFIPS system option.

- SAS Internal Passwords are stored using the SHA256 hashing algorithm when this option is specified.
- Starting with SAS 9.4M8, z/OS supports FIPS 140-2 using System SSL . Prior to SAS 9.4M8, FIPS supported using only SAS/SECURE and AES encryption .
- There is an interaction between the ENCRYPTFIPS option and the NETENCRYPTALGORITHM= option. Only the AES algorithm or the TLS algorithm (value specified is SSL) is supported for FIPS 140-2 encryption. An error is logged when an unsupported algorithm is specified.

```
ERROR: When SAS option ENCRYPTFIPS is ON the option value for SAS
option
ERROR: NETENCRYPTALGORITHM must be a single value of AES or SSL.
ERROR: Invalid option value.
NOTE: Unable to initialize the options subsystem.
```

- When the ENCRYPTFIPS option is specified, a message is logged informing the user that FIPS 140-2 encryption is enabled. This log can be viewed in the log for SAS window at the DEBUG and or TRACE levels. Refer to "The SAS Log" in *SAS Language Reference: Concepts* and "Administering Logging for SAS/CONNECT" in *SAS/CONNECT User's Guide*.

See Also

- "NETENCRYPTALGORITHM="
- "FIPS 140-2 Standards Compliance"
- "Security Options" in *SAS/CONNECT User's Guide*

NETENCRYPT System Option

Specifies whether client/server data transfers are encrypted.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Default:	NONETENCRYPT

Operating environment: UNIX, Windows, z/OS

See: NETENCRYPTALGORITHM=

Example: “Using the RC4 Algorithim for Encryption of SAS/CONNECT: Example ” on page 41

Syntax

NETENCRYPT | **NONETENCRYPT**

Syntax Description

NETENCRYPT
specifies that encryption is required.

NONETENCRYPT
specifies that encryption is not required, but is optional.

Details

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.

Client: Optional

Server: Required

Valid in: Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP=	Communications
Alias:	NETENCRALG
Operating environment:	UNIX, Windows, z/OS
See:	"NETENCRYPT" on page 137, "ENCRYPTFIPS" on page 135
Examples:	"Using TLS for Encryption of a SAS/CONNECT Windows Spawner: Example " on page 44 "Using TLS for Encryption of a z/OS Spawner on a SAS/CONNECT Server: Example" on page 47 "Using TLS for Encryption of a SAS/CONNECT UNIX Spawner: Example " on page 42

Syntax

NETENCRYPTALGORITHM=*algorithm* | *("algorithm-1"... "algorithm-n")*

Syntax Description

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. When you specify two or more encryption algorithms, use a space or a comma to separate them, and enclose the algorithms in parentheses.

Note: If you are running SAS/CONNECT in a SAS Intelligence Platform environment using the SAS Metadata Server and configured encryption using the SAS Management Console, you can specify only one encryption algorithm. There is a workaround if you receive "ERROR: Cannot Negotiate Encryption Algorithm" on page 202.

The following encryption values can be used. The SSL value is used to set either a TLS or SSL protocol. These encryption types can be specified on the client, server, or spawner.

Note: All values except SSL are algorithms. SSL is the predecessor protocol to TLS.

- AES
- DES
- RC2
- RC4
- TripleDES
- SASProprietary
- SSL

Alias NETENCRALG=

Restrictions If you do not have the cryptographic libraries that support AES and specify algorithm AES, an error is generated.

Starting with SAS 9.4M6, Integrated Object Model (IOM) servers support the NETENCRYPTALG=SSL. In software releases before SAS 9.4M6, TLS is not supported on IOM servers.

When ENCRYPTFIPS is specified, only the TLS protocol (value specified in NETENCRYPTALG= SSL) or the AES algorithm can be specified. Otherwise, an error message is output.

Example options NOSCRIPT netencrypt netencryptalgorithm=(ssl des tripledes)

Details

Use the NETENCRYPTALGORITHM= 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.

For SAS/CONNECT, the following guidelines apply when setting the NETENCRYPTALGORITHM= system option:

- If the SAS/CONNECT client specifies an algorithm(s), then the first algorithm in its list is negotiated for use if the SAS/CONNECT spawner or server supports it. If that protocol or algorithm is not supported, the next algorithm in the SAS/CONNECT client list is negotiated. If there are no compatible encryption types, the connection fails.
- If there are no SAS/CONNECT client encryption values specified, the encryption protocol or encryption algorithm used is determined by what is set on the SAS/CONNECT spawner. If there is a list of values specified, the SAS/CONNECT spawner uses the first value specified in the list. If that protocol or algorithm is not supported, the next one in the SAS/CONNECT spawner list is negotiated. If there are no compatible encryption types, the connection fails.
- If neither the SAS/CONNECT client nor the SAS/CONNECT spawner/server specify an algorithm, the default encryption type is used.

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.

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.

Table 10.27 shows the interactions of NETENCRYPT or NONETENCRYPT and the NETENCRYPTALGORITHM= option.

Note: The values for the NETENCRALG= option can be either encryption algorithms or the SSL protocol. In this table *alg* is used for both.

Table 10.1 *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-1*	NETENCRALG=*alg-2*	Regardless of whether NETENCRYPT or NONETENCRYPT is specified, a client/server connection fails.

Example

In the following example, the client and the server specify different values for the NETENCRYPTALGORITHM= option.

The client specifies two algorithms in the following OPTIONS statement:

```
options netencryptalgorithm=(rc2 tripledes);
```

The server specifies three algorithms and requires encryption in the following OPTIONS statement:

```
options netencrypt netencryptalgorithm=(ssl aes tripledes);
```

The client and the server negotiate an algorithm that they share in common, TripleDES in this example, for encrypting data transfers.

NETENCRYPTKEYLEN= System Option

Specifies the symmetric key size that is used by the encryption algorithm for encrypted client/server data transfers.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Alias:	NETENCRKEY=
Default:	0
Restriction:	The NETENCRYPTKEYLEN= option supports only the RC2 and RC4 algorithms. The SASProprietary, DES, TripleDES, and AES algorithms and SSL protocol do not use this system option.
Operating environment:	UNIX, Windows, z/OS
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.

Syntax

NETENCRYPTKEYLEN= 0 | 40 | 128

Syntax Description

0
specifies the maximum key length that is supported at both the client and the server.

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.

Details

The NETENCRYPTKEYLEN= option supports only the RC2 and RC4 algorithms.

> **IMPORTANT** The SASProprietary, DES, TripleDES, and AES algorithms and the SSL protocol do not use this SAS system option.

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.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, SAS invocation, SAS/CONNECT spawner start-up if this option is used as an environment variable
Categories:	Communications: Networking and Encryption System Administration: Security
Default:	The default location for certificates is set using the SSLCALISTLOC= system option. Certificates are located in one .pem file. By contrast, The SSLCACERTDIR= system option allows the customer to specify a location where multiple certificate files reside. See “SSLCALISTLOC=” on page 145.
Restriction:	Beginning with SAS 9.4M7, when lockdown is in effect, the location specified by this SAS system option (or equivalent environment variable) is added by default to the LOCKDOWN allowlist. This system option can be set in a SAS configuration file or specified at start up on the command line.
Operating environment:	UNIX and Linux. Prior to SAS 9.4M8, also supported for z/OS. See “SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8”.
Note:	This system option is added with SAS 9.4M5. This option can also be specified as an environment variable. In prior releases, an environment variable of a similar name was used.
Tips:	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 in the following example), use the following OpenSSL command:

```
openssl verify -CApath /u/myuser/sslcerts cacert1.pem
```

See: "Defining Environment Variables in UNIX Environments" in *SAS Companion for UNIX Environments* and "TKMVSENV Options under z/OS" in *SAS Companion for z/OS*

Example: The SSLCACERTDIR system option points to the directory where the CA certificate is located.

```
-SSLCACERTDIR=/u/myuser/sslcerts/
```

Syntax

SSLCACERTDIR="*file-path*"

Syntax Description

"file-path"

specifies the directory 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. 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.

Details

System option SSLCACERTDIR= points to a directory that contains all of the public certificate files of all CAs in the trust chain. One file exists for each CA in the trust chain.

SSLCACERTDIR= requires the certificates to be in the directory where their names are the value of a hash that OpenSSL generates.

Each CA certificate file must be PEM-encoded (base64). For more information, see "Certificate Encoding Formats and Extensions" on page 66.

For Foundation servers such as workspace servers and stored process servers (that is, servers in a deployment), if certificates are used, SAS searches for certificates in the following specific order:

1. SAS looks for SAS system option SSLCALISTLOC= to find a file. This file holds your trusted certificates.

2 If trustedcerts.pem exists and the SSLCACERTDIR= system option is set, SAS checks trustedcerts.pem first before it searches the directory.

3 If trustedcerts.pem does not exist, but the certificates are in the directory defined by the SSLCACERTDIR= system option, then SAS ignores SSLCALISTLOC=.

4 If trustedcerts.pem does not exist, and the SSLCACERTDIR= system option is not set, SAS reports an error.

With SAS 9.4, SAS 9.4M1, and SAS 9.4M2, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is ***SAS-configuration-directory*/Levn/certs/cacert.pem**. The cacert.pem file contains the list of trusted certificates.

Starting with SAS 9.4M3, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is **<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem**. The trustedcerts.pem file contains the list of trusted certificates.

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).

Client:	Required
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Restriction:	Beginning with SAS 9.4M7, when lockdown is in effect, the location specified by this SAS system option (or equivalent environment variable) is added by default to the LOCKDOWN allowlist. This system option can be set in a SAS configuration file or specified at start up on the command line.
Operating environment:	UNIX and Linux. Prior to SAS 9.4M8, also supported for z/OS. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8".
Notes:	Starting with SAS 9.4M3, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is <SASHome>/SASSecurityCertificateFramework/1.1/cacerts/ trustedcerts.pem. The trustedcerts.pem file contains the list of trusted CA Certificates, the Mozilla Bundle provided by SAS at installation. In SAS 9.4M1 and SAS 9.4M2, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is *SAS-configuration-directory*/Lev*n*/certs/cacert.pem. The cacert.pem file contains the list of trusted CA Certificates.

Tip: When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.

See: "Certificate Locations" on page 81.

Examples: "Using TLS for Encryption of a SAS/CONNECT UNIX Spawner: Example " on page 42
"Using TLS for Encryption of a z/OS Spawner on a SAS/CONNECT Server: Example" on page 47

Syntax

SSLCALISTLOC="*file–path*"

Syntax Description

"*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.

Details

IMPORTANT This SAS system options is not supported for use with z/OS starting with SAS 9.4M8. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8" on page 167.

The SSLCALISTLOC= 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 chain. The CA file must be PEM-encoded (base64). For z/OS, the file must be formatted as ASCII and must reside in a UNIX file system. For more information, see "Certificate Encoding Formats and Extensions" on page 66.

From SAS 9.4 to SAS 9.4M2, the default setting for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is ***SAS-configuration-directory*/Levn/certs/cacert.pem**. The cacert.pem file contains the list of trusted CA Certificates.

Starting with SAS 9.4M3, the default path set for the SSLCALISTLOC= system option on UNIX foundation servers is **<SASRoot>/SASHome/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem**. By default, the trustedcerts.pem file contains a managed set of trusted root certificates provided by Mozilla. If additional CA certificates are required, they can be added using the SAS Deployment Manager (SAS Deployment Manager).

CAUTION
Do not change the SSLCALISTLOC= system option. Starting SAS 9.4M3, the SSLCALISTLOC= system option should not be overridden or changed unless directed by SAS Technical Support or PSD. In addition, the trustedcerts.pem file should not be altered by any means other than by using the new SAS Deployment Manager tasks for

adding and removing certificates to Trusted CA Bundle. If the file is changed outside of using these tasks, the provided Trusted CA Bundle might not be supported and maintenance of those changes is not guaranteed. See "Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager" on page 92.

For the specifics and an example of how to create a trust list on z/OS, refer to Create a CA Trust List Using OpenSSL on page 124. For information about creating a trust list on UNIX, refer to Create a Certificate Chain in PEM Format Using OpenSSL on page 89 and "Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager" on page 92.

Note: System option SSLCACERTDIR= points to a directory that contains all of the public certificate file(s) of all CA(s) in the trust chain. One file exists for each CA in the trust chain. These can be used instead of using the SSLCALISTLOC= system option. Refer to "SSLCACERTDIR=" on page 143.

For Foundation servers such as workspace servers and stored process servers (that is, servers in a deployment), if certificates are used, SAS searches for certificates in the following specific order:

1. SAS looks for SAS system option SSLCALISTLOC= to find the file trustedcerts.pem.
2. If trustedcerts.pem exists and SSLCACERTDIR= system option is set, SAS checks trustedcerts.pem first before it searches this directory.
3. If trustedcerts.pem does not exist, but the certificates are in the directory defined by SSLCACERTDIR=, then SAS ignores SSLCALISTLOC=.
4. If trustedcerts.pem does not exist, and the SSLCACERTDIR= system option is 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.

SSLCERTISS= System Option

Specifies the name of the issuer of the digital certificate that TLS should use.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications

Operating environment: Windows

Tip: When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.

Example: "Using TLS for Encryption of SAS/SHARE on Windows: Examples " on page 55

Syntax

SSLCERTISS="*issuer-of-digital-certificate*"

Syntax Description

"*issuer-of-digital-certificate*"
specifies the name of the issuer of the digital certificate that should be used by TLS.

Details

The SSLCERTISS= option is used with the SSLCERTSERIAL= option to uniquely identify a digital certificate from the Microsoft Certificate Store.

Note: You can also use the SSLCERTSUBJ= option to identify a digital certificate instead of using the SSLCERTISS= and SSLCERTSERIAL= options.

You can use OpenSSL to print the issuer of the certificate. Note the value of `serial=` in the output.

```
C:\>openssl.exe x509 -in customer.pem -issuer -noout
```

Note the issuer of the certificate. For example, if `issuer= /DC=com/DC=Company/CN=Company SHA2 Issuing CA02`, then you know that the value of `CN=` is needed, which is `Company SHA2 Issuing CA02`.

SSLCERTLOC= System Option

Specifies the location of the digital certificate for the machine's public key. This is used for authentication.

Client: Optional

Server: Required

Valid in: Configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP=	Communications
Restriction:	Beginning with SAS 9.4M7, when lockdown is in effect, the location specified by this SAS system option (or equivalent environment variable) is added by default to the LOCKDOWN allowlist. This system option can be set in a SAS configuration file or specified at start up on the command line.
Operating environment:	UNIX and Linux. Prior to SAS 9.4M8, also supported for z/OS. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8".
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.
Examples:	"Using TLS for Encryption of a SAS/CONNECT UNIX Spawner: Example " on page 42 "Using TLS for Encryption of a z/OS Spawner on a SAS/CONNECT Server: Example" on page 47 "Using TLS for Encryption of SAS/SHARE on UNIX: Example " on page 53

Syntax

SSLCERTLOC="*file-path*"

Syntax Description

"file-path"
specifies the location of a file that contains a digital certificate for the machine's public key. This is used by servers to send to clients for authentication.

Details

> **IMPORTANT** This SAS system option is not supported for use with z/OS starting with SAS 9.4M8. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8" on page 167.

The SSLCERTLOC= option is required for a server. It is required at the client only if the SSLCLIENTAUTH option is specified at the server.

If you want the spawner to locate the appropriate digital certificate, you must specify both the -SSLCERTLOC= and -SSLPVTKEYLOC= options in the -SASCMD script.

The certificate must be PEM-encoded (base64). Under z/OS, the file must be formatted as ASCII and must reside in a UNIX file system. For more information, see "Certificate Encoding Formats and Extensions" on page 66.

SSLCERTSERIAL= System Option

Specifies the serial number of the digital certificate that TLS should use.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	Windows
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.
Example:	"Using TLS for Encryption of SAS/SHARE on Windows: Examples " on page 55

Syntax

SSLCERTSERIAL="*serial-number*"

Syntax Description

"*serial-number*"
specifies the serial number of the digital certificate that should be used by TLS.

Details

The SSLCERTSERIAL= option is used with the SSLCERTISS= option to uniquely identify a digital certificate from the Microsoft Certificate Store.

Note: You can also use the SSLCERTSUBJ= option to identify a digital certificate instead of using the SSLCERTISS= and SSLCERTSERIAL= options.

You can use OpenSSL to print the serial number of the certificate. Note the value of `serial=` in the output.

```
C:\>openssl.exe x509 -in customer.pem -serial -noout
```

SSLCERTSUBJ System Option

Specifies the subject name of the digital certificate that TLS should use.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	Windows
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.
Example:	"Using TLS for Encryption of a SAS/CONNECT Windows Spawner: Example " on page 44

Syntax

SSLCERTSUBJ="*subject-name*"

Syntax Description

"subject-name"
specifies the subject name of the digital certificate that TLS should use.

Details

The SSLCERTSUBJ= option is used to search for a digital certificate from the Microsoft Certificate Store.

Note: You can also use the SSLCERTISS= and SSLCERTSERIAL= options instead of the SSLCERTSUBJ= option to identify a digital certificate.

SSLCLIENTAUTH System Option

Specifies whether a server should perform client authentication.

Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	UNIX, Windows, z/OS
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.

Syntax

SSLCLIENTAUTH | **NOSSLCLIENTAUTH**

Syntax Description

SSLCLIENTAUTH
specifies that the server should perform client authentication.

> **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.

Details

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.

SSLCRLCHECK System Option

Specifies whether a Certificate Revocation List (CRL) is checked when a digital certificate is validated.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	UNIX, Windows, z/OS
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.

Syntax

SSLCRLCHECK | **NOSSLCRLCHECK**

Syntax Description

SSLCRLCHECK
specifies that CRLs are checked when digital certificates are validated.

NOSSLCRLCHECK
specifies that CRLs are not checked when digital certificates are validated.

Details

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.

SSLCIPHERLIST= System Option

Specifies the ciphers that can be used on UNIX and z/OS for OpenSSL.

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
Restriction:	If SSLMODE option is set, this option is ignored.
Operating environment:	UNIX and Linux. Prior to SAS 9.4M8, also supported for z/OS. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8".
Notes:	This system option is added with SAS 9.4M5. This option can also be specified as an environment variable. In prior releases, an environment variable of a similar name was used. 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 the system option before SAS is started on the client.
Tip:	You can also define SET commands for Windows by using the System Properties windows that you access from the Control Panel.
See:	"Defining Environment Variables in UNIX Environments" in *SAS Companion for UNIX Environments* and "TKMVSENV Options under z/OS" in *SAS Companion for z/OS*
Example:	Specify the system option: `-SSLCIPHERLIST=HIGH`

Syntax

SSLCIPHERLIST=*openssl_cipher_list*

Syntax Description

openssl-cipher-list

The SSLCIPHERLIST= system option specifies the ciphers that can be used on UNIX and z/OS 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 Cipher information can be found at 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. For more information, see "Encryption: SAS Logging Facility" on page 36.

Note: If you set a minimum protocol that does not allow some ciphers, you might get an error.

Details

This system option is available on UNIX and z/OS platforms. This system option can be specified anytime before TLS is used. After TLS is loaded, it cannot be changed.

Refer to the OpenSSL documentation on ciphers for information about the ciphers that can be specified for this system option. This information can be found at OpenSSL Ciphers.

Note: For Windows, you can use group policy settings to configure TLS Cipher Suite Order. See Cipher Suites in TLS/SSL (Schannel SSP) for information about the TLS Cipher Suite order.

SSLCRLLOC= System Option

Specifies the location of a Certificate Revocation List (CRL).

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS configuration, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Restriction:	Beginning with SAS 9.4M7, when lockdown is in effect, the location specified by this SAS system option (or equivalent environment variable) is added by default to the LOCKDOWN allowlist. This system option can be set in a SAS configuration file or specified at start up on the command line.
Operating environment:	UNIX and Linux. Prior to SAS 9.4M8, also supported for z/OS. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8".
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.

Syntax

SSLCRLLOC="*file-path*"

Syntax Description

"file-path"
specifies the location of a file that contains a Certificate Revocation List (CRL).

Details

The SSLCRLLOC= option is required only when the SSLCRLCHECK option is specified.

SSLMODE= System Option

Sets the allowed cipher suites to be used for TLS.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, SAS invocation, OPTIONS statement, SAS System Options window, SAS configuration, SAS/CONNECT spawner start-up if this option is used as an environment variable
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Default:	SSLMODESP800131A
Interaction:	When system option SSLMODE= is set, system option SSLMINPROTOCOL is ignored.
Operating environment:	UNIX, LINUX, z/OS
Note:	This system option is added starting with SAS 9.4M5.
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.
See:	"Defining Environment Variables in UNIX Environments" in *SAS Companion for UNIX Environments* and "TKMVSENV Options under z/OS" in *SAS Companion for z/OS* For a list of ciphers that are supported for each of the modes that can be specified for the SSLMODE= system option, see "Before SAS 9.4M8: SSLMODE= System Option Supported Ciphers" on page 215.
Example:	-sslmode SSLMODESP800131A

Syntax

SSLMODE=*ssl-mode*

Syntax Description

ssl-mode

sets the allowed cipher suites to be used for TLS version that you are using. Valid *ssl-mode*s and the ciphers supported for each mode with SAS 9.4M8 are listed for each *ssl-mode*.

Note: Valid *ssl-mode*s and the ciphers supported for each mode prior to SAS 9.4M8 are shown in "Before SAS 9.4M8: SSLMODE= System Option Supported Ciphers".

SSLMODESP800131A

is the DEFAULT configuration mode for TLS communication.

When system option SSLMODE= is set, system option SSLMINPROTOCOL= is ignored. If SSLMODE= is not set, SAS checks SSLMINPROTOCOL= system option and uses the protocol set. If neither system option is set, SAS uses the default cipher mode SSLMODESP800131A.

The following TLS 1.3 ciphers are supported when the SSLMODESP800131A mode is set:

- TLS_AES_256_GCM_SHA384
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_CHACHA20_POLY1305_SHA256

The following TLS 1.2 ciphers are supported when the SSLMODESP800131A mode is set:

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

SSLMODESUITEB128

The following TLS 1.3 ciphers are supported when the SSLMODESUITEB128 mode is set:

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_AES_128_CCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_CHACHA20_POLY1305_SHA256

The following TLS 1.2 ciphers are supported when the SSLMODESUITEB128 mode is set:

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

SSLMODESUITEB192

The following TLS 1.3 ciphers are supported when the SSLMODESUITEB192 mode is set:

- TLS_AES_256_GCM_SHA384
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_CHACHA20_POLY1305_SHA256

The TLS 1.2 cipher that is supported when the SSLMODESUITEB192 mode is set is TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384.

SSLMODEDEPRECATED

allows for older cipher suites. These are cipher suites that do not meet NIST standards.

CAUTION

Cipher Suites must meet NIST standards Cipher suites that do not meet NIST standards should not be used.

The following TLS 1.3 ciphers are supported when the SSLMODEDEPRECATED mode is set:

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384
- TLS_AES_128_CCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_CHACHA20_POLY1305_SHA256

The following TLS 1.2 ciphers are supported when the SSLMODEDEPRECATED mode is set:

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_128_CBC_SHA256

- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA

Details

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. For details of SP800-131A, see NIST Special Publication 800-131A.

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 mode prior to SAS 9.4M8, see “Before SAS 9.4M8: SSLMODE= System Option Supported Ciphers” on page 215.

SSLMINPROTOCOL= System Option

Specifies the minimum TLS protocol that can be negotiated when using OpenSSL.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, SAS invocation, OPTIONS statement, System Options Window, SAS/CONNECT spawner start-up if this option is used as an environment variable
Categories:	Communications: Networking and Encryption System Administration: Security
Default:	SAS 9.4M8 uses the Operating System OpenSSL library. The minimum TLS protocol is based on that OpenSSL library. For example, if the OpenSSL version being used is OpenSSL 1.1.1, TLS 1.2 is the minimum protocol used. With SAS 9.4M5 - SAS 9.4M7, the default value is TLS 1.2. Starting with SAS 9.4M8, TLS 1.3 is supported as a minimum value.
Restriction:	If the SSLMODE= option is set, this option is ignored.
Operating environment:	UNIX, LINUX, z/OS, and Windows
Note:	This system option is added starting with SAS 9.4M5. In prior releases of SAS 9.4, environment variable SAS_SSL_MIN_PROTOCOL was used.
Tip:	You can also define SET commands for Windows by using the System Properties windows that you access from the Control Panel.

See: "Defining Environment Variables in UNIX Environments" in *SAS Companion for UNIX Environments* and "TKMVSENV Options under z/OS" in *SAS Companion for z/OS*

Example: Specify this system option as follows:

```
-sslminprotocol="TLS1.3"
```

Syntax

"**SSLMINPROTOCOL=** *protocol*"

Syntax Description

protocol

specifies the minimum TLS protocol version that can be negotiated between UNIX, z/OS, and Windows servers when using OpenSSL.

Valid values that can be supplied for this option are TLS1.2, TLSv1.2, TLS1.3, TLSv1.3. These are the values that can be specified, but are unsecure: SSL3, SSLV3, TLS, TLS1, TLSV1, TLS1.0, TLSV1.0, TLS1.1, and TLSV1.1.

See the SAS Security Bulletin on OpenSSL for the most current information about the versions of OpenSSL used with SAS products and about the advisories under consideration. For a quick reference, see Table 3.2 on page 26.

CAUTION
It is highly recommended that you use TLS 1.2 or above. Versions prior to TLS 1.2 have known security vulnerabilities.

Note: A message is written to the SAS log when an invalid value is specified.

Details

The SSLMINPROTOCOL= system option enables you to set a minimum TLS protocol that can be negotiated. During the first TLS handshake attempt, the highest supported protocol version is offered. If this handshake fails, earlier protocol versions are offered instead.

TLS version 1.2 or higher is the recommended protocol supported with SAS 9.4. Starting with SAS 9.4M5, the default TLS version and the supported ciphers are set with the SSLMODE= option. The SSLMODE= option defaults to SSLMODESP800131A for the ciphers that can be used to negotiate between client and servers. Starting with SAS 9.4M8, TLS 1.2 and TLS 1.3 ciphers are supported. Specifying a version of TLS that is older than TLS 1.2 is not recommended.

SAS 9.4M8 uses the Operating System OpenSSL library. The minimum TLS protocol is based on that OpenSSL library. For example, if the OpenSSL version being used is OpenSSL 1.1.1, TLS 1.2 is the minimum protocol used. However, TLS 1.3 is also supported.

With SAS 9.4M5 - SAS 9.4M7, SSLMINPROTOCOL system option defaults to TLS 1.2.

> **IMPORTANT** Protocols earlier than TLS 1.2 are unsecure.

SSLPKCS12LOC= System Option

Specifies the location of the PKCS#12 encoding package file.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Restriction:	Beginning with SAS 9.4M7, when lockdown is in effect, the location specified by this SAS system option (or equivalent environment variable) is added by default to the LOCKDOWN allowlist. This system option can be set in a SAS configuration file or specified at start up on the command line.
Operating environment:	UNIX and Linux. Prior to SAS 9.4M8, also supported for z/OS. See “SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8”.
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User’s Guide*.
Examples:	“Using TLS for Encryption of a z/OS Spawner on a SAS/CONNECT Server: Example” on page 47 “Using TLS for Encryption of SAS/SHARE on z/OS Prior to SAS 9.4M8: Example ” on page 56

Syntax

SSLPKCS12LOC=“*file-path*”

Syntax Description

“file-path”

specifies the location of the PKCS#12 DER encoding package file that contains the certificate and the private key.

z/OS specifics If you run in a z/OS operating environment, this file must be in the UNIX file system. The OpenSSL library cannot read MVS data sets.

Details

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 the -SASCMD script if you want the spawner to locate the appropriate digital certificate.

SSLPKCS12PASS= System Option

Specifies the password that TLS requires for decrypting the private key.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	UNIX and Linux. Prior to SAS 9.4M8, also supported for z/OS. See “SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8”.
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User’s Guide*.
Example:	Beginning with SAS 9.4M7, SAS001–SAS005 are supported passwords.

```
-netencrypt -netencralg ssl -sslpkcs12loc
/certificate-path/certificate.p12 -sslpkcs12pass
{SAS004}5871DA1957E3F051029C20C5D2F3D1F8DED3F1F9DED875EF
```

Examples:	“Using TLS for Encryption of a z/OS Spawner on a SAS/CONNECT Server: Example” on page 47 “Using TLS for Encryption of SAS/SHARE on z/OS Prior to SAS 9.4M8: Example ” on page 56

Syntax

SSLPKCS12PASS=*password*

Syntax Description

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.

Beginning with SAS 9.4M7, SAS encoded passwords (SAS001–SAS005) are supported passwords.

Details

The SSLPKCS12PASS= option is required only when the PKCS#12 DER encoding package is encrypted. The z/OS RACDCERT EXPORT command always encrypts package files when exporting the certificate and the private key.

You must specify both the SSLPKCS12LOC= option and the SSLPKCS12PASS= option in the -SASCMD script if you want the spawner to locate the appropriate digital certificate.

SSLPVTKEYLOC= System Option

Specifies the location of the private key that corresponds to the digital certificate.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Restriction:	Beginning with SAS 9.4M7, when lockdown is in effect, the location specified by this SAS system option (or equivalent environment variable) is added by default to the LOCKDOWN allowlist. This system option can be set in a SAS configuration file or specified at start up on the command line.
Operating environment:	UNIX and Linux. Prior to SAS 9.4M8, also supported for z/OS. See “SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8”.
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User’s Guide*.

Examples: "Using TLS for Encryption of a SAS/CONNECT UNIX Spawner: Example " on page 42
"Using TLS for Encryption of SAS/SHARE on UNIX: Example " on page 53

Syntax

SSLPVTKEYLOC="*file-path*"

Syntax Description

"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.

Details

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). Under z/OS, the file must be formatted as ASCII and must reside in a UNIX file system. For more information, see "Certificate Encoding Formats and Extensions" on page 66.

You must specify both the SSLCERTLOC= option and the SSLPVTKEYLOC= option in the -SASCMD script if you want the spawner to locate the appropriate digital certificate.

SSLPVTKEYPASS= System Option

Specifies the password that TLS requires for decrypting the private key.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	UNIX and Linux. Prior to SAS 9.4M8, also supported for z/OS. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8".
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.
Example:	Beginning with SAS 9.4M7, SAS001–SAS005 are supported passwords.

```
-netencrypt -netencralg ssl -sslpvtkeyloc
/certificate-path/certificate.pem -sslpvtkeypass
{SAS004}5871DA1957E3F051029C20C5D2F3D1F8DED3F1F9DED875EF
```

Examples: "Using TLS for Encryption of a SAS/CONNECT UNIX Spawner: Example " on page 42
"Using TLS for Encryption of SAS/SHARE on UNIX: Example " on page 53

Syntax

SSLPVTKEYPASS="*password*"

Syntax Description

"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.

Beginning with SAS 9.4M7, SAS encoded passwords (SAS001–SAS005) are supported passwords.

Note: SAS recommends that you use a password or passphrase to encrypt the private key.

Details

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.

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.

Client:	Optional
Server:	Optional
Valid in:	SAS invocation, configuration file, SAS/CONNECT spawner start-up if this option is used as an environment variable
Categories:	Communications: Networking and Encryption System Administration: Security
Default:	The default is the name of the host being contacted.

Operating environment: UNIX and Linux. Prior to SAS 9.4M8, also supported for z/OS. See "SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8".

Notes: This system option is added starting with SAS 9.4M5. This option can also be specified as an environment variable. In prior releases, an environment variable of a similar name was used.

The TLS SNI extension is always sent to the web server.

See: "Defining Environment Variables in UNIX Environments" in *SAS Companion for UNIX Environments* and "TKMVSENV Options under z/OS" in *SAS Companion for z/OS*

Example: Specify the system option as follows:

```
-SSLSNIHOSTNAME="www.example.org"
```

Syntax

"**SSLSNIHOSTNAME=** *hostname*"

Syntax Description

SSLSNIHOSTNAME=

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.

Details

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.

See Also

For more information, see Chapter 14, "Troubleshooting," on page 199.

11

SAS System Options Used for IBM z/OS System SSL with SAS 9.4M8

Overview

Starting with SAS 9.4M8, z/OS uses IBM System SSL to provide encryption for data-in-motion. GSK (Global Security Kit) is the API that SAS uses.

The system options contained in this section include options to provide certificate management and other options to trace and debug issues.

Note: The SAS System options for certificate management do not require the use of keyrings. The SAS System options that are used for certificate and key management with SAS 9.4M8 use the naming convention of "keyring" as part of their names. However, use of "keyrings" is not required to work with SAS 9.4M8. See “SSLKEYRINGFILE=” for information about all the types of files that can be used.

Dictionary

SSLGSKTRACE= System Option

Enables GSK tracing.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	z/OS
Example:	SSLGSKTRACE=0xFFFF SSLGSKTRACEFILE=`/u/sasusr1/gskssl.%.trc`

Syntax

SSLGSKTRACE==*gsk-tracing*

Syntax Description

gsk-tracing

enables GSK tracing. Values that can be specified for *gsk-tracing* are as follows:

- 0xffff to trace all GSK functions.
- 0x01 to trace function entry
- 0x02 to trace function exit
- 0x04 to trace errors
- 0x08 to include informational messages
- 0x10 to include EBCDIC data dumps
- 0x20 to include ASCII data dumps
- 0x00 is the default value.

Details

IBM provides a method to write TLS logging information to a file. SAS system options SSLGSKTRACE= and SSLGSKTRACEFILE= enables logging and exposes them to the user. The SSLGSKTRACE= system option sets flags to the GSK_TRACE z/OS environment variable. The SSLGSKTRACEFILE= system option points to where the trace output should be written and sets the GSK_TRACE_FILE z/OS environment variable.

The SSLGSKTRACEFILE= system option must be set to the name of a UNIX System Services file in a directory for which the executing application has Write permission. The default trace file is **`/tmp/gskssl.%.trc`**.

The current process identifier is included as part of the trace file name when the name contains a percent sign (%). For example, if GSK_TRACE_FILE is set to **`/tmp/gskssl.%.trc`** and the current process identifier is 247, then the trace file name is **`/tmp/gskssl.247.trc`**.

Example

Specify the following to enable GSK messaging:

```
SSLGSKTRACE=0xFFFF
SSLGSKTRACEFILE="/u/sasusr1/gskssl.%.trc"
```

When the job ends, the trace can be converted to a readable format using the gsktrace utility.

```
gsktrace gskssl.17041476.trc > a.txt
```

Sample output follows:

```
01/30/2023-14:51:50 Thd-37 INFO read_v3_server_hello(): Using TLSV1.2 protocol
01/30/2023-14:51:50 Thd-37 INFO read_v3_server_hello(): Using V3 cipher
specification 0039
01/30/2023-14:51:50 Thd-37 INFO read_v3_server_hello(): Using key exchange type 0
01/30/2023-14:51:50 Thd-37 INFO read_v3_server_hello(): Renegotiation Indication
signaled by initial SERVER-HELLO
01/30/2023-14:51:50 Thd-37 INFO gsk_read_v3_record(): Calling read routine for 5
bytes
01/30/2023-14:51:51 Thd-37 INFO gsk_read_v3_record(): 5 bytes received
01/30/2023-14:51:51 Thd-37 INFO gsk_read_v3_record(): Calling read routine for 1
bytes
01/30/2023-14:51:51 Thd-37 INFO gsk_read_v3_record(): 1 bytes received
01/30/2023-14:51:51 Thd-37 INFO read_v3_change_cipher_specs(): Received CHANGE-
CIPHER-SPECS message
01/30/2023-14:51:51 Thd-37 INFO gsk_read_v3_record(): Calling read routine for 5
bytes
01/30/2023-14:51:51 Thd-37 INFO gsk_read_v3_record(): 5 bytes received
01/30/2023-14:51:51 Thd-37 INFO gsk_read_v3_record(): Calling read routine for 64
bytes
01/30/2023-14:51:51 Thd-37 INFO gsk_read_v3_record(): 64 bytes received
01/30/2023-14:51:51 Thd-37 INFO crypto_aes_decrypt_ctx(): Clear key AES 256-bit
decryption performed for 16 bytes
01/30/2023-14:51:51 Thd-37 INFO crypto_aes_decrypt_ctx(): Clear key AES 256-bit
decryption performed for 48 bytes
01/30/2023-14:51:51 Thd-37 INFO read_v3_finished(): Received FINISHED message
01/30/2023-14:51:51 Thd-37 ASCII read_v3_finished(): FINISHED message
```

SSLHWDETECTMESSAGE= System Option

Produce System SSL initialization messages pertaining to the current status of hardware cryptographic support.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	z/OS
Example:	SSLHWDETECTMESSAGE="1"

Syntax

SSLHWDETECTMESSAGE="*1*"

Syntax Description

1

Set to the value of "1" to have GSK initialization messages sent to stderr.

SSLICSFERRORMESSAGE = System Option

Produce ICSF error messages.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	z/OS

Example: SSLICSFERRORMESSAGE="1"

Syntax

SSLICSFERRORMESSAGE="*1*"

Syntax Description

1
Set to the value of "1" to have ICSF error messages output.

SSLGSKTRACEFILE= System Option

specifies the location of the GSK trace file.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	z/OS
Example:	SSLGSKTRACE=0xFFFF SSLGSKTRACEFILE=`/u/sasusr1/gskssl.%.trc`
Example:	"Example" on page 169

Syntax

SSLGSKTRACEFILE==*gsk-tracing-file*

Syntax Description

gsk-tracing-file
specifies the location of *gsk-tracing-file*. The gsktrace program formats the file. Use % to include the process identifier in the name. For example: `/tmp/gskssl.%.trc`

Details

IBM provides a method to write TLS logging information to a file. SAS system options SSLGSKTRACE= and SSLGSKTRACEFILE= enables logging and exposes them to the user. The SSLGSKTRACE sets flags to the GSK_TRACE z/OS environment variable. SSLGSKTRACEFILE= points to where the trace output should be written and sets the GSK_TRACE_FILE z/OS environment variable.

The SSLGSKTRACEFILE= must be set to the name of a UNIX System Services file in a directory for which the executing application has Write permission. The default trace file is `/tmp/gskssl.%.trc`.

The current process identifier is included as part of the trace file name when the name contains a percent sign (%). For example, if GSK_TRACE_FILE is set to `/tmp/gskssl.%.trc` and the current process identifier is 247, then the trace file name is `/tmp/gskssl.247.trc`.

SSLKEYRINGFILE= System Option

Specifies the key file and location.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	z/OS
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.
Example:	cntspawn -service *1234* -mgmtport *4321* -sascmd "*/path/spawnmvs*.sh -entry sas -sslkeyringfile */path/cert*.p12 -sslkeyringlabel *cert-label* -sslkeyringpw *password* -netencrypt -netencralg ssl" -netencrypt -netencralg ssl *-/path/cert*.p12 -sslkeyringlabel *cert-label* -sslkeyringpw *password*

Syntax

SSLKEYRINGFILE=="*keyfile-location*"

Syntax Description

"keyfile-location"

a *keyfile-location* for the key file can be specified as one of the following:

- a key database file. Specify a key database (.kdb) and password or a key database and a password stash file (.sth) for a UNIX File System (UFS) or Networking File System (NFS). The z/OS file is called a key database and by convention, has a file extension of .kdb. This file includes PKI private keys, certificate requests, and certificates.
- a PKCS#12 file. The PKCS#12 file extension is.p12. Specify the PKCS#12 file name along with the password on UFS or NFS. System SSL supports PKCS#12 certificate and private key objects types.
- a System authorization facility (SAF) key ring. Specify a *userid*/*keyringname*.

  Note: If user ID is not provided, then the user running the job is assumed.

  This file type is generated using the z/OS Security Server (RACF) RACDCERT command. RACDCERT installs and maintains PKI private keys and certificates in RACF.

  For information, see Steps for generating a certificate and private key for the RACF address space.
- a PKCS #11 token. Specify the *token-name* as defined in the ICSF TKDS data set. This token contains PKI private keys, certificate requests, and certificates.

Details

A key database is a file on UFS (or NFS) that contains certificates and keys for use by applications using the GSK API.

Certificates and keys can be imported, exported and displayed. A utility that can be used to create, modify, and delete the key database file is the z/OS shell-based program called gskkyman. In order for applications to use the key database file, the application must specify both the file name and its associated password. The password can either be specified directly or through a stash file containing the encrypted password. The stash file provides a level of security where the password does not have to be explicitly specified.

The gskkyman program creates, completes, and manages either a z/OS file or z/OS PKCS #11 token that contains PKI private keys, certificate requests, and certificates. The z/OS file is called a key database and by convention, has a file extension of .kdb.

The gskkyman utility is located in `/bin/gskkyman`. For information about using the z/OS shell based gskkyman utility, see Certificate/Key management. It can be used both as a menu-driven program or, when run with options, can be a single command-line utility.

The z/OS Security Server (RACF) RACDCERT commands can also be used to generate certificate and key files. For information, see z/OS Security Server RACF Security Administrator's Guide.

SSLKEYRINGLABEL= System Option

Specifies the key label.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Operating environment:	z/OS
Tip:	When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.
Example:	cntspawn -service *1234* -mgmtport *4321* -sascmd "/*path/spawnmvs*.sh -entry sas -sslkeyringfile */path/cert*.p12 -sslkeyringlabel *cert-label* -sslkeyringpw *password* -netencrypt -netencralg ssl" -netencrypt -netencralg ssl -/*path/cert*.p12 -sslkeyringlabel *cert-label* -sslkeyringpw *password*

Syntax

SSLKEYRINGLABEL=="*key-label*"

Syntax Description

"key-label"
specifies the label of the key used to authenticate the application.

SSLKEYRINGPW= System Option

Specifies the password that z/OS System SSL requires for decrypting the private key.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line
Category:	Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Interaction: When the SSLKEYRINGPW= option is specified, it takes precedence over the SSLKEYRINGSTASHFILE= option.

Operating environment: z/OS

Tip: When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.

Example: cntspawn -service *1234* -mgmtport *4321* -sascmd "/*path/spawnmvs*.sh -entry sas -sslkeyringfile */path/cert*.p12 -sslkeyringlabel *cert-label* -sslkeyringpw *password* -netencrypt -netencralg ssl" -netencrypt -netencralg ssl -/*path/cert*.p12 -sslkeyringlabel *cert-label* -sslkeyringpw *password*

Syntax

SSLKEYRINGPW=="*password*"

Syntax Description

"password"
specifies the password for the key database to be used. A password is required to be used with system option SSLKEYRINGFILE.

SSLKEYRINGSTASHFILE= System Option

Specifies the name of the database stash file containing the encrypted password.

Client: Optional

Server: Optional

Valid in: Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Interaction: When the SSLKEYRINGPW= option is specified, it takes precedence over the SSLKEYRINGSTASHFILE= option.

Operating environment: z/OS

Tip: When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically

pass the encryption values. For detailed information, see SASCMD for your operating environment in *SAS/CONNECT User's Guide*.

Example: cntspawn -service *1234* -mgmtport *4321* -sascmd "/*path/spawnmvs*.sh -entry sas -sslkeyringfile */path/cert*.p12 -sslkeyringlabel *cert-label* -sslkeyringstashfile*keyringStashFile*.sth -netencrypt -netencralg ssl" -netencrypt -netencralg ssl -/*path/cert*.p12 -sslkeyringlabel *cert-label* -sslkeyringstashfile *keyringStashFile*

Syntax

SSLKEYRINGSTASHFILE=="*file-name*"

Syntax Description

"file-name"

a *file-name*.sth. The extension used with a stash file is .sth. Specifies the name of the database stash file containing the encrypted password.

Note: When the SSLKEYRINGPW= option is specified, it takes precedence over the SSLKEYRINGSTASHFILE= option.

12

SAS Environment Variables

Overview of Environment Variables

UNIX 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" in *SAS Companion for UNIX Environments*.

z/OS environment variables are specified in a SAS data set that is referred to as the TKMVSENV data set file. For more information about setting environment variables in the TKMVSENV file, see"TKMVSENV Options under z/OS" in *SAS Companion for z/OS*.

For Windows, you can choose to define a SAS environment variable using the SET system option or to define a Windows environment variable using the Windows SET command. For more information, see "Using Data Libraries" in *SAS Companion for Windows*.

Starting with SAS 9.4M6, TLS environment variables and system options can be applied locally using the PROC HTTP SSLPARMS statement. For more information, see "SSLPARMS" in *Base SAS Procedures Guide*.

Note: Starting with SAS 9.4M5, environment variables SAS_SSL_CIPHER_LIST and SAS_SSL_CERT_DIR have been replaced by system options of similar names. See "SAS System Options for Encryption" on page 133.

Note: Starting with SAS 9.4M5, the SSL_USE_SNI environment variable has changed. This option turns off SNI.

Dictionary

CAS_CLIENT_SSL_CA_LIST Environment Variable

Specifies the path and file name of the file that contains the list of trusted certificate authorities (CAs).

Client:	Optional
Server:	Optional
Valid in:	Server configuration file, cas.settings file, and operating system command line
Category:	System Administration: Security
Operating environment:	LINUX
Notes:	This environment variable is available starting with SAS 9.4M5. This environment variable is used by CAS client, Lua client, Python client, CAS server, SAS 9.4 client
Example:	`export CAS_CLIENT_SSL_CA_LIST='/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/vault-ca.crt'`

Syntax

CAS_CLIENT_SSL_CA_LIST=

Syntax Description

CAS_CLIENT_SSL_CA_LIST='*path/certificate-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 (SAS 9.4 can act as a SAS Viya 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.

For details about how to manage client certificates between SAS 9.4 and SAS Viya, see Configure SAS 9.4 Clients to Work with SAS Viya.

SAS_SSL_MIN_PROTOCOL= Environment Variable

Specifies the minimum TLS protocol that can be negotiated when using OpenSSL.

Client: Optional

Server: Optional

Valid in: Configuration file, SAS invocation, OPTIONS statement, System Options Window, SAS/CONNECT spawner start-up if this option is used as an environment variable

Categories: Communications: Networking and Encryption
System Administration: Security

Operating environment: UNIX, LINUX, z/OS, and Windows

Notes: This environment variable is used to set the TLS protocol for SAS software releases prior to SAS 9.4M5 to the supported TLS version 1.2. This environment variable was introduced with SAS 9.4M3 and is available in prior SAS software releases by applying security hot fixes.

This environment variable 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.

Tip: You can also define SET commands for Windows by using the System Properties dialog box that you access from the Control Panel.

See: "Defining Environment Variables in UNIX Environments" in *SAS Companion for UNIX Environments* and "TKMVSENV Options under z/OS" in *SAS Companion for z/OS*

Examples: Export the environment variable on UNIX hosts for the Bourne Shell:

```
export SAS_SSL_MIN_PROTOCOL=TLS1.2
```

Set the environment variable on UNIX hosts for the C Shell environment:

```
SETENV SAS_SSL_MIN_PROTOCOL TLS1.2
```

Set the environment variable at SAS invocation for UNIX hosts:

```
-set "SAS_SSL_MIN_PROTOCOL=TLS1.2"
```

Set the environment variable on Windows hosts

```
SET SAS_SSL_MIN_PROTOCOL=TLS1.2
```

Syntax

SAS_SSL_MIN_PROTOCOL= *protocol*
"**SAS_SSL_MIN_PROTOCOL=** *protocol*"
SAS_SSL_MIN_PROTOCOL *protocol*

Syntax Description

protocol

specifies the TLS protocol version that can be negotiated between UNIX, z/OS, and Windows servers when using OpenSSL for SAS releases prior to SAS 9.4M5. Because protocols earlier than TLS 1.2 are insecure, set this environment variable to TLS1.2 or TLSv1.2 for SAS 9.4M4 and earlier releases of SAS.

Note: To use the SAS_SSL_MIN_PROTOCOL environment variable prior to SAS 9.4M3, you must apply security hot fixes. Starting with SAS 9.4M5, this environment variable is no longer needed to set the TLS.

For TLS 1.2, you can specify values TLS1.2 and TLSv1.2. These other values can be specified, but are insecure: SSL3, SSLV3, TLS, TLS1, TLSV1, TLS1.0, TLSV1.0, TLS1.1, and TLSV1.1.

See SAS Security Bulletin on OpenSSL for the most current information about the versions of OpenSSL used with SAS products and about the advisories under consideration. For a quick reference, see Table 3.2 on page 26.

CAUTION
It is highly recommended that you use TLS 1.2 or above. Versions prior to TLS 1.2 have known security vulnerabilities.

Note: A message is written to the SAS log when an invalid value is specified.

Details

The SAS_SSL_MIN_PROTOCOL= environment variable enables you to set a minimum TLS protocol that will be negotiated. During the first TLS handshake attempt, the highest supported protocol version is offered. If this handshake fails, earlier protocol versions are offered instead. Only TLS versions 1.2 and above are secure.

SAS_VIYA_TOKEN Environment Variable

Enables you to pre-load your token before starting your SAS session.

Client:	Optional
Server:	Optional
Valid in:	SAS invocation, OPTIONS statement
Categories:	Communications: Networking and Encryption System Administration: Security
Restriction:	Access tokens expire every 12 hours.

Interaction: This environment variable is used with the SERVICESBASEURL= system option.

Operating environment: LINUX

Note: This environment variable is available starting with SAS 9.4M5.

Example: Export the environment variable into your shell:

```
options set=SAS_VIYA_TOKEN="access_token"
```

Syntax

SAS_VIYA_TOKEN=

Syntax Description

SAS_VIYA_TOKEN="*access_token*"

specifies the OAuth token that is needed for a user to access SAS Viya services. This environment variable is used to load your *access_token*. Before starting SAS, assign the value of the *access_token* to the SAS_VIYA_TOKEN environment variable in the shell where SAS is executed. You can also assign the access token by specifying `options set=SAS_VIYA_TOKEN="access_token"` in your SAS program.

The SAS_VIYA_TOKEN environment variable must be set before any calls to SAS Viya are attempted. You cannot change the value of the SAS_VIYA_TOKEN environment variable after using the token. When the token expires or when you need a different token, start a new SAS session.

Note: This environment variable is not needed for SAS 9.4 code that is executed on a SAS Viya compute server using the compute service. The user's OAuth token is automatically provided.

Details

The SAS_VIYA_TOKEN environment variable is used to provide access to SAS Viya services. In order to access the following services, you need to specify a token using this environment variable:

- access SAS Viya credentials services from SAS 9.4. When the AUTHDOMAIN= option is set in the CAS statement or in a LIBNAME statement, an attempt is first made to retrieve credentials from the Credentials service before searching the metadata. For more information, see the AUTHDOMAIN option in the CAS statement. Refer to the server documentation that you are connecting to for the details of using the AUTHDOMAIN option in the LIBNAME statement.
- store and retrieve files within the file service in the SAS Viya system using the FILESRVC Access Method FILENAME Statement. See "FILENAME: FILESRVC Access Method" in *SAS Global Statements: Reference*.

Providing the token and setting the SERVICESBASEURL= system option enables some SAS system capabilities to use SAS Viya services. For more information, see "SERVICESBASEURL=" in *SAS System Options: Reference*.

SSLREQCERT= Environment Variable

Specifies what checks to perform on server certificates in a TLS session.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Default:	SSLREQCERT= defaults to DEMAND.
Operating environment:	UNIX and Linux
Note:	SSLREQCERT= environment variable is available starting with SAS 9.4M5.
Tip:	PROC HTTP statement SSLPARMS, can be used to set TLS environment variables and system option. For more information, see "SSLPARMS" in *Base SAS Procedures Guide*.
Examples:	-set SSLREQCERT "ALLOW" export SSLREQCERT "ALLOW"

Syntax

SSLREQCERT="*ALLOW* | *DEMAND* | *NEVER* | *TRY*"

Syntax Description

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.

CAUTION

TLS connections are not validated with SSLREQCERT="ALLOW" TLS connections are validated by the SAS session when this option is set when invoking your SAS session. For security purposes, DEMAND is the setting that should be specified.

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.

CAUTION
TLS connections are not validated with SSLREQCERT="NEVER" TLS connections are not validated by the SAS session when this option is set when invoking your SAS session. For security purposes, DEMAND is the setting that should be specified.

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.

CAUTION
When SSLREQCERT="TRY", the session continues, but might be insecure TLS connections are validated by the SAS session when this option is set when invoking your SAS session. For security purposes, DEMAND is the setting that should be specified.

Details

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: Beginning with SAS 9.4M6, TLS environment variables and system options can be applied locally using the PROC HTTP SSLPARMS statement. For more information, see "SSLPARMS" in *Base SAS Procedures Guide*.

SSL_USE_SNI Environment Variable

Disables the use of Server Name Indication (SNI) in the TLS handshake for the client.

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.
Restrictions:	With SAS 9.4M5, the SAS 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. See "SAS System Options for Encryption" on page 133 for details.

The SSL_USE_SNI environment variable is supported only on UNIX.

Operating environment: UNIX, LINUX

See: "Defining Environment Variables in UNIX Environments" in *SAS Companion for UNIX Environments*

Examples: Export the environment variable on UNIX hosts for the Bourne Shell :

```
export SSL_USE_SNI=1
```

Set the environment variable on UNIX hosts for the C Shell :

```
SETENV SSL_USE_SNI
```

Set the environment variable at SAS invocation for UNIX hosts:

```
sas -dms -set SSL_USE_SNI
```

Syntax

SSL_USE_SNI

Syntax Description

SSL_USE_SNI
disables support of the TLS Server Name Indication (SNI) for UNIX clients and servers. 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.

Default SNI is enabled by default on UNIX. To disable SNI, specify the SSL_USE_SNI environment variable.

Details

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.

See Also

For more information, see Chapter 14, "Troubleshooting," on page 199.

Chapter 13

PWENCODE Procedure

Overview: PWENCODE Procedure

What Does the PWENCODE Procedure Do?

The PWENCODE procedure enables you to encode passwords. Encoding obfuscates the data. Unlike encryption, encoding is a reversible permutation of the data and uses no keys.

Encoded passwords can be used in place of plaintext passwords in SAS programs that access relational database management systems (RDBMSs) and various servers. Examples are SAS/CONNECT servers, SAS/SHARE servers, SAS Integrated Object Model (IOM) servers, SAS Metadata Servers, and more.

Concepts: PWENCODE Procedure

Using Encoded Passwords in SAS Programs

When a password is encoded with PROC PWENCODE, the output string includes a tag that identifies the string as having been encoded. An example of a tag is `{sas001}`. The tag indicates the encoding method. SAS servers and SAS/ACCESS engines recognize the tag and decode the string before using it. Encoding a password enables you to write SAS programs without having to specify a password in plaintext.

Note: PROC PWENCODE passwords can contain up to a maximum of 512 characters, which include alphanumeric characters, spaces, and special characters. Data set passwords, however, must follow SAS naming rules. For information about SAS naming rules, see "Rules for Most SAS Names" in *SAS Language Reference: Concepts*.

The encoded password is never written to the SAS log in plain text. Instead, each character of the password is replaced by an X in the SAS log.

Encoding versus Encryption

Encoding techniques disguise passwords and the approach is intended to prevent casual, non-malicious viewing of passwords. With encoding, one character set is translated to another character set through some form of table lookup.

Encryption, by contrast, involves the transformation of data from one form to another through the use of mathematical operations and, usually, a "key" value. Encryption is generally more difficult to break than encoding. Several options for PROC PWENCODE designate encryption techniques that align with industry standards. These options support longer encryption keys (for example, 256-bit). Salting and multiple iterations are provided to the AES encryption algorithm to create passwords that are harder to break.

Encoding methods for PROC PWENCODE are now SAS001 – SAS005. Starting in SAS 9.4M5, PROC PWENCODE provides stronger password protection using the SAS005 method of encoding.

Password protection is an important part of your security strategy, but you should not rely only on password protection for all your data security needs; a determined and knowledgeable attacker can break passwords. Data should also be protected by other security controls such as file system permissions, other access control mechanisms, and encryption of data at rest and in transit.

Encoding Methods

Starting in SAS 9.4M5,the SAS005 method for encoding passwords is added. When SAS005 is specified for PROC PWENCODE, a more secure 256-bit fixed key is generated. SAS005, like SAS004, uses a 256-bit fixed key plus a 64-bit random salt. However, it is hashed for additional iterations.

Table 13.1 *Supported Encoding Methods*

Encoding Method	Uses Data Encryption Algorithm	Encoded Password/key Description
`sas001`	None	Uses base64 to encode passwords.
`sas002`, which can also be specified as `sasenc`	SASProprietary, which is included in SAS software.	Uses a 32-bit fixed key.
`sas003`	AES (Advanced Encryption Standard).	Uses a 256-bit fixed key plus a 16-bit random salt value.
`sas004`	AES (Advanced Encryption Standard).	Uses a 256-bit fixed key and a 64-bit random salt value.
`sas005`	AES (Advanced Encryption Standard).	Uses a 256-bit fixed key, a 64-bit random salt value, and is hashed for additional iterations.

IMPORTANT Starting with SAS 9.4M8, SAS Foundation servers use the cryptographic libraries provided and installed on the operating system to provide encryption for data at rest and data in motion. With this change, SAS no longer provides the cryptographic libraries for SAS Foundation servers as part of the SAS Installation. AES encryption is supported using the operating system cryptographic libraries. Prior to SAS 9.4M8, AES encryption was supported as part of SAS/SECURE.

For more information, see “Cryptographic Library Support Starting with SAS 9.4M8”.

Note: The METHOD= option supports the SAS003, SAS004, and SAS005 values. Prior to SAS 9.4M8, you needed SAS/SECURE to support SAS003–SAS005. With SAS 9.4M8, SAS003-SAS005 are supported using the operating system’s cryptopgraphic libraries. SAS Proprietary encoding that supports SAS002 is available with all SAS software. For more information, see SAS/SECURE on page 20.

Syntax: PWENCODE Procedure

PROC PWENCODE IN='*password*' <OUT=*fileref*> <METHOD=*encoding-method*>;

Statement	Task	Example
"PROC PWENCODE"	Encode a password	Ex. 1, Ex. 2, Ex. 3, Ex. 4

PROC PWENCODE Statement

Encodes a password.

Examples:

"Example 1: Encoding a Password" on page 190
"Example 2: Using an Encoded Password in a SAS Program" on page 191
"Example 3: Saving an Encoded Password to the Paste Buffer" on page 193
"Example 4: Specifying Method= SAS003 to Encode a Password" on page 194
"Example 5: Specifying Method= SAS005 to Encode a Password" on page 195

Syntax

PROC PWENCODE IN='*password*' <OUT=*fileref*> <METHOD=*encoding-method*>;

Required Argument

IN='*password*'

specifies the password to encode. The password can contain up to a maximum of 512 characters, which include alphanumeric characters, spaces, and special characters.

Note: Data set passwords must follow SAS naming rules. If the IN=*password* follows SAS naming rules, it can also be used for SAS data sets. For information about SAS naming rules, see "Rules for Most SAS Names" in *SAS Language Reference: Concepts*.

If the password contains embedded single or double quotation marks, use the standard SAS rules for quoting character constants. These rules can be found in the SAS Constants in Expressions chapter of *SAS Language Reference: Concepts*.

Note: Each character of the encoded password is replaced by an X when written to the SAS log.

See "Example 1: Encoding a Password" on page 190

"Example 2: Using an Encoded Password in a SAS Program" on page 191

"Example 3: Saving an Encoded Password to the Paste Buffer" on page 193

Optional Arguments

OUT=*fileref*

specifies a fileref to which the output string is to be written. If the OUT= option is not specified, the output string is written to the SAS log.

Note: The global macro variable

```
_PWENCODE
```

is set to the value that is written to the OUT= fileref or to the value that is displayed in the SAS log.

See "Example 2: Using an Encoded Password in a SAS Program" on page 191

METHOD=*encoding-method*

specifies the encoding method. Here are the supported values for *encoding-method*.

- *SAS001*
- *SAS002*
- *SAS003*
- *SAS004*
- *SAS005*

The SAS003, SAS004, and SAS005 encoded passwords use a 256-bit fixed key plus a random salt value that is applied to the encoding method. Therefore, each time you use PROC PWENCODE to encode the same password, you get a different encoded password, because the salt values are random.

For more information about each of these encoding methods, see "Encoding Methods " on page 187.

Note: The METHOD= option supports the SAS003, SAS004, and SAS005 values. Prior to SAS 9.4M8, you needed SAS/SECURE to provide support of SAS003-SAS005 encoding methogs. With SAS 9.4M8, SAS003-SAS005 are supported using the operating system's cryptopgraphic libraries. SAS Proprietary encoding that supports SAS002 is available with all SAS software. For more information, see SAS/SECURE on page 20.

If the METHOD= option is omitted, the default encoding method is used. The default method is `sas002` in most cases. `SAS002` is also the default method used if you specify an invalid method.

When the FIPS 140-2 compliance option, -encryptfips, is specified, the encoding method defaults to `sas003`. For more information about FIPS, see "FIPS 140-2 Standards Compliance" on page 15.

Examples: PWENCODE Procedure

Example 1: Encoding a Password

Features: IN= argument

Details

This example shows a simple case of encoding a password and writing the encoded password to the SAS log.

Program

Encode the password.

```
proc pwencode in='my password';
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log.

```
19   proc pwencode in=XXXXXXXXXXXXX;
20   run;


{SAS002}DBCC571245AD0B31433834F80BD2B99E16B3C969

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Example 2: Using an Encoded Password in a SAS Program

Features: IN= argument
OUT= option

Details

This example illustrates the following:

- encoding a password and saving it to an external file
- reading the encoded password with a DATA step, storing it in a macro variable, and using it in a SAS/ACCESS LIBNAME statement

Program 1: Encoding the Password

```
filename pwfile
'external-filename';

proc pwencode in='mypass1' out=pwfile;
run;
```

Program Description

Declare a fileref.

```
filename pwfile
'external-filename';
```

Encode the password and write it to the external file. The OUT= option specifies which external fileref the encoded password is written to.

```
proc pwencode in='mypass1' out=pwfile;
run;
```

Program 2: Using the Encoded Password

```
filename pwfile
'external-filename';

options symbolgen;

data _null_;
infile pwfile truncover;
input line :$50.;
call symputx('dbpass',line);
run;

libname x odbc dsn=SQLServer user=testuser password="&dbpass";
```

Program Description

Declare a fileref for the encoded-password file.

```
filename pwfile
'external-filename';
```

Set the SYMBOLGEN SAS system option. This step shows that the actual password cannot be revealed, even when the macro variable that contains the encoded password is resolved in the SAS log. This step is not required in order for the program to work properly.

```
options symbolgen;
```

Read the file and store the encoded password in a macro variable. The DATA step stores the encoded password in the macro variable DBPASS.

```
data _null_;
infile pwfile truncover;
input line :$50.;
call symputx('dbpass',line);
run;
```

Use the encoded password to access a DBMS. You must use double quotation marks (“ ”) so that the macro variable resolves properly.

```
libname x odbc dsn=SQLServer user=testuser password="&dbpass";
```

Log

```
1    filename pwfile 'external-filename';
2    options symbolgen;
3    data _null_;
4    infile pwfile truncover;
5    input line :$50.;
6    call symputx('dbpass',line);
7    run;

NOTE: The infile PWFILE is:
      Filename=external-filename
      RECFM=V,LRECL=256,File Size (bytes)=4,
      Last Modified=12Apr2012:13:23:49,
      Create Time=12Apr2012:13:23:39

NOTE: 1 record was read from the infile PWFILE.
      The minimum record length was 4.
      The maximum record length was 4.
NOTE: DATA statement used (Total process time):
      real time           0.57 seconds
      cpu time            0.04 seconds
8
9   libname x odbc
SYMBOLGEN:  Macro variable DBPASS resolves to {sas002}bXlwYXNzMQ==
9 !                dsn=SQLServer user=testuser password="&dbpass";
NOTE: Libref X was successfully assigned as follows:
      Engine:        ODBC
      Physical Name: SQLServer
```

Example 3: Saving an Encoded Password to the Paste Buffer

Features: IN= argument
OUT= option
FILENAME statement with CLIPBRD access method

DETAILS

This example saves an encoded password to the paste buffer. You can then paste the encoded password into another SAS program or into the password field of an authentication dialog box.

Program

```
filename clip clipbrd;
```

```
proc pwencode in='my password' out=clip;
run;
```

Program Description

Declare a fileref with the CLIPBRD access method.

```
filename clip clipbrd;
```

Encode the password and save it to the paste buffer. The OUT= option saves the encoded password to the fileref that was declared in the previous statement.

```
proc pwencode in='my password' out=clip;
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log.

```
24
25   filename clip clipbrd;
26     proc pwencode in=XXXXXXXXXXXXX out=clip;
27   run;

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Example 4: Specifying Method= SAS003 to Encode a Password

Features: METHOD= argument

Details

This example shows a simple case of encoding a password using the **sas003** encoding method and writing the encoded password to the SAS log. SAS003 uses a 16-bit salt to encode a password.

Program

Encode the password using SAS003. The encoded password is a 256-bit key with a 16 bit random salt.

```
proc pwencode in='mypassword' method=sas003;
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log. SAS003 encoding uses AES encryption plus a 16-bit salt. Because SAS003 uses random salting, each time you run the following code, a different password is generated.

```
8   proc pwencode in=XXXXXXXXXXXXX method=sas003;
29   run;

{SAS003}4837B146585CED2C9FED14A3C946D68E4389

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Example 5: Specifying Method= SAS005 to Encode a Password

Features: METHOD= argument

Details

This example shows a simple case of encoding a password using the `sas005` encoding method and writing the encoded password to the SAS log. SAS005 uses a 256-bit fixed key that uses a 64-bit random salt to encode the password.

Program

Encode the password using SAS005.

```
proc pwencode in='mypassword' method=sas005;
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log. SAS005 encoding uses AES encryption with a 256-bit fixed key and a 64-bit random salt value. SAS005 increases security for stored passwords by using the SHA-256 hashing algorithm and is hashed for additional iterations. Because SAS005 uses random salting, each time you run the following code, a different password is generated.

```
230  proc pwencode in=XXXXXXXXXXXX method=sas005;
231  run;


{SAS005}ADD8AB7108595A7D1A69190D78CDFE6145C1EB849CC7A43D

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

PART 6

Troubleshooting

14

Troubleshooting

ERROR: Unable to load extension: (tkessl)

There are a lot of reasons the library might not load. The best way to debug this error is to turn on logging and get a SAS logging facility log output.

ERROR: SSL Provider Not in FIPS Mode

This message might be displayed on Windows servers when the system cryptography "Use FIPS compliant algorithms for encryption, hashing, and signing" setting is not enabled. Enable this under your Local Security Policy. For more information, see “Configure FIPS 140-2 Capable TLS on Windows” on page 117.

ERROR: HTTP Proxy Handshake Failed

This message is displayed when clients sending TLS Subject Name Identification (SNI) cannot connect to a secured proxy server.

There are servers that do not handle SNI host name checking in a way that allows connecting to secured proxy servers.

- On UNIX servers, starting with SAS 9.4M5, SNI is always sent. You can disable SNI by setting the USE_SSL_SNI environment variable.
- On Windows servers, SNI is always sent. Other than disabling name checking (Subject Alternative Name) on server certificates, there is currently no workaround.

ERROR: Cannot Load SSL Support

This message is displayed when SAS cannot find required software.

- This message can be generated when TLS certificates cannot be found. If the directory where the certificates are located is specified using the SSLCACERTDIR environment variable, and the certificate names in the directory are not named using the value of a hash that OpenSSL generates, this message is generated. For more information, see “SSLCACERTDIR=” on page 143.
- This message is generated when requisite software cannot be loaded in an IOM session.

ERROR:14090086:SSL routines: SSL3_GET_SERVER_CERTIFICATE: Certificate Verify Failed

This message is displayed when TLS certificates cannot be verified. If the directory where the certificates are located is specified using the SSLCACERTDIR environment variable, and the certificate names in the directory are not named using the value of a hash that OpenSSL generates, this message is generated. For more information, see “SSLCACERTDIR=” on page 143.

Failed to Find the Following Issuer of This Certificate in Truststore

This message is displayed when using the SAS Deployment Manager to add TLS certificates to the trust list in the wrong order. First, you need to add the issuer of the certificate or the root certificate. Then you can add the intermediate certificate. You need to run the SAS Deployment Manager task for each certificate that you need to add.

Verify That the File Contains Certificates in the Proper Encoding

This message is displayed when using the SAS Deployment Manager to add TLS certificates with unacceptable encodings. Certificates must be X.509 certificates formatted in Base-64 encoding that have .pem, .crt, or .cer extensions. For more information, see “Certificate Encoding Formats and Extensions” on page 66.

ERROR: Cannot Negotiate Encryption Algorithm

This message is displayed when an encryption algorithm cannot be negotiated.

One example of this error being logged occurs if you are running a version of SAS that is earlier than SAS 9.4 and you are not licensed for SAS/SECURE. You might have specified SAS system option NETENCRYPTALG=AES, but without SAS/SECURE, AES is not supported. In this case, your connection fails. You can still use SASProprietary to encrypt your connection, but not other encryption algorithms in this case.

Note: With SAS 9.4, SAS/SECURE is included as part of Base SAS and includes AES encryption.

Another example is that you have specified an encryption algorithm on the metadata server, and the client is using a different algorithm. For example, you might have assigned AES encryption using the SAS Management Console as the encryption algorithm to use on the metadata server. The metadata server allows only one encryption algorithm to be specified. When a client (for example, SAS/CONNECT) connects to the metadata server and is using an encryption algorithm that is not AES, your connection fails.

If you need to support multiple network encryption algorithms for a SAS/CONNECT server that is part of a SAS metadata environment, perform the following steps:

1. Edit the ConnectSpawner_usermods.sh file.
   - Set the NETENCRYPTALGORITHM= value to the multiple network encryption algorithms that you need.
   - Add the -SERVICE option to a TCP port that has not been used already.
   - Add the -SASCMD option copied from the current launch command in metadata.

   The edited version of your command might look something like the following code:

   ```
   USERMODS_OPTIONS="-netencryptalgorithm '(AES,SASPROPRIETARY)' -service
   different-tcp-port -sascmd
   ```

2. Stop and restart the SAS/CONNECT spawner.
3. Sign on using the new port.

TLS Certificate Verification: ERROR, Certificate Has Expired

Starting with SAS 9.4M3, SAS ships the Mozilla CA trusted bundle of certificates. This bundle of CA certificates contains a CA certificate by Sectigo with an expiration date of May 30, 2020. You need to apply the hot fix described in Problem Note 66213: Certificates that shipped in SAS® 9.4 contains an expired certificate from Sectigo AddTrust (expired on May 30, 2020) to resolve issues related to the expired certificate.

For information about this expired certificate, see Sectigo AddTrust External CA Root Expiring May 30, 2020.

Invalid RSA Key for Encrypting; n (1024) < 2048

When using SAS/SECURE with AES encryption and FIPS 140-2 is required (ENCRYPTFIPS option), a larger key for encryption is needed. Upgrade the server to SAS 9.4M7 and apply hot fixes. When hot fixes are applied, the server can use 4096-bit RSA keys. Hot fixes might also need to be applied to the client. For more information, see SAS Note 68804

Insufficient Access Authority

If you attempt to access a SAS library that is enabled for z/OS Pervasive Encryption and you do not have RACF access to the underlying encryption key label, an error that resembles the following is generated:

```
ICH4081 USER(SASUser) GROUP (group-name) NAME (name)
keyName CL(CSFKEYS)
INSUFFICIENT ACCESS AUTHORITY
ACCESS INTENT (READ) ACCESS ALLOWED (NONE)
```

I/O Error Accessing a z/OS Pervasive Encryption Data Set

Pervasively encrypted data sets have a special bit turned on that verifies that the accessing program is capable of reading the file using the EXCP access method. Therefore, if you try to browse or edit a file from a program that is not capable of reading the file (for example, using ISPF browse or edit), the system reports an I/O error that resembles the following:

```
DSLLIST - Data Sets Matching USERID.ENCRYPT.SASLIB     I/O ERROR
```

WARNING: The OpenSSL 3 "legacy" provider could not be loaded. Certain functions may not work without ciphers from the "legacy" provider

Starting with SAS 9.4M8, SAS Foundation servers use the cryptographic libraries provided and installed on the operating system to provide encryption for data at rest and data in motion. With this change, SAS no longer provides OpenSSL libraries for SAS Foundation Servers.

The operating system's OpenSSL libraries might not provide all of the encryption algorithms previously available in SAS, mainly because they are deemed unsecure. This issue often occurs when running in FIPS 140-2 mode. For example, when using OpenSSL version 3 libraries, RC2, RC4, and other algorithms that are contained in a 'legacy' encryption provider might not be provided. Refer to SAS Note 70230 for more information.

This warning might also be generated when using the PDFSECURITY=HIGH system option with the ODS PDF destination. Refer to SAS Note 70305 for more information.

You might see a warning similar to the following message, which can be safely ignored:

```
WARNING: The OpenSSL 3 "legacy" provider could not be loaded. Certain
functions may not work without ciphers from the "legacy" provider
.
```

PART 7

Appendices

15

Environment Variables Appendix

Overview

The environment variables in this section are used with SAS releases prior to SAS 9.4M5. With SAS 9.4M5, new system options similarly named have been created. See "SAS System Options for Encryption" on page 133.

Dictionary

SAS_SSL_CIPHER_LIST Environment Variable

Specifies the ciphers that can be used on UNIX and z/OS for OpenSSL.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, command line
Categories:	Communications: Networking and Encryption System Administration: Security

Operating environment: UNIX and z/OS

Notes: With SAS 9.4M5, a new system option similarly named has been created. See SSLCIPHERLIST= on page 133 for details.

This environment variable is available in all SAS 9.3 and SAS 9.4 versions of software if hot fixes are applied.

This environment variable 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.

Tip: You can also define SET commands for Windows by using the System Propertieswindow that you access from the Control Panel.

Examples: Export the environment variable on UNIX hosts for the Bourne Shell:

```
export SAS_SSL_CIPHER_LIST=TLSv1.2
```

Set the environment variable on UNIX hosts for the C Shell environment:

```
SETENV SAS_SSL_CIPHER_LIST HIGH
```

Set the environment variable at SAS invocation for UNIX hosts:

```
-set SAS_SSL_CIPHER_LIST '3DES:RC2'
```

Set the environment variable on Windows hosts

```
SET SAS_SSL_CIPHER_LIST SHA256
```

Syntax

SAS_SSL_CIPHER_LIST=*openssl_cipher_list*

Syntax Description

openssl-cipher-list

The SAS_SSL_CIPER_LIST environment variable specifies the ciphers that can be used on UNIX and z/OS 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 Cipher information can be found at 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. For information, see “Encryption: SAS Logging Facility” on page 36.

Note: If you set a minimum protocol that does not allow some ciphers, you might get an error.

Details

This environment variable is available on UNIX and z/OS platforms. This environment variable can be specified anytime before TLS is used. After TLS is loaded, it cannot be changed.

Refer to the OpenSSL documentation on ciphers for information about the ciphers that can be specified for this environment variable. This information can be found at OpenSSL Ciphers.

Note: For Windows, you can use group policy settings to configure TLS Cipher Suite Order. See Cipher Suites in TLS/SSL (Schannel SSP) for information about the TLS Cipher Suite order.

SSLCACERTDIR Environment Variable

Specifies the location of the trusted certificate authorities (CA) found in OpenSSL format.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, SAS invocation, SAS/CONNECT spawner start-up
Categories:	Communications: Networking and Encryption System Administration: Security
Default:	The default location for certificates is set using the SSLCALISTLOC= system option. Certificates are located in one .pem file. By contrast, The SSLCACERTDIR environment variable allows the customer to specify a location where multiple certificate files reside. See “SSLCALISTLOC=” on page 145.
Operating environment:	UNIX
Notes:	Starting with SAS 9.4M5, a new system option similarly named has been created. See SSLCACERTDIR= on page 133 for details. This environment variable is available in all SAS 9.3 and SAS 9.4 versions of software if hot fixes are applied.
Tips:	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
```

Examples: The SSLCACERTDIR environment variable points to the directory where the CA certificate is located. Export the environment variable on UNIX hosts for the Bourne Shell:

```
export SSLCACERTDIR=/u/myuser/sslcerts/
```

Set the environment variable on UNIX 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 UNIX hosts:

```
-set "SSLCACERTDIR=/u/myuser/sslcerts/"
```

Syntax

SSLCACERTDIR="*file-path*"

Syntax Description

"*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. 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.x.

Details

Environment variables SSLCACERTDIR and SSL_CERT_DIR point to a directory that contains all of the public certificate files of all CAs in the trust chain. One file exists for each CA in the trust chain.

SSLCACERTDIR requires the certificates to be in the directory where their names are the value of a hash that OpenSSL generates.

Each CA certificate file must be PEM-encoded (base64). For more information, see "Certificate Encoding Formats and Extensions" on page 66.

For Foundation servers such as workspace servers and stored process servers (that is, servers in a deployment), if certificates are used, SAS searches for certificates in the following specific order:

1. SAS looks for SAS system option SSLCALISTLOC= to find a file. This file holds your trusted certificates.

2. If trustedcerts.pem exists and the SSLCACERTDIR= system option is set, SAS checks trustedcerts.pem first before it searches the directory.
3. If trustedcerts.pem does not exist, but the certificates are in the directory defined by the SSLCACERTDIR= system option, then SAS ignores SSLCALISTLOC=.
4. If trustedcerts.pem does not exist, and the SSLCACERTDIR= system option is not set, SAS reports an error.

With SAS 9.4, SAS 9.4M1, and SAS 9.4M2, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is ***SAS-configuration-directory*/Levn/certs/cacert.pem**. The cacert.pem file contains the list of trusted certificates.

Starting with SAS 9.4M3, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is **<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem**. The trustedcerts.pem file contains the list of trusted certificates.

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.

SSL_CERT_DIR Environment Variable

Specifies the location of the trusted certificate authorities (CA) found in OpenSSL format. This is the OpenSSL environment variable.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, SAS invocation, SAS/CONNECT spawner start-up
Categories:	Communications: Networking and Encryption System Administration: Security
Default:	The default location for certificates is set using the SSLCALISTLOC= system option. Certificates are located in one .pem file. By contrast, The SSLCACERTDIR environment variable allows the customer to specify a location where multiple certificate files reside. See “SSLCALISTLOC=” on page 145.
Operating environment:	UNIX
Notes:	With SAS 9.4M5, a new system option similarly named has been created. See SSLCACERTDIR= on page 133 for details. This environment variable is available in all SAS 9.3 and SAS 9.4 versions of software if hot fixes are applied.
Tips:	OpenSSL looks up the CA certificate based on the x509 hash value of the certificate. SSL_CERT_DIR 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 the 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
```

Examples:

The SSL_CERT_DIR environment variable points to the directory where the CA certificate is located. Export the environment variable on UNIX hosts for the Bourne Shell:

```
export SSL_CERT_DIR=/u/myuser/sslcerts/
```

Set the environment variable on UNIX hosts for the C Shell directory where the CA certificates are located:

```
SETENV SSL_CERT_DIR /u/myuser/sslcerts/
```

Set the environment variable at SAS invocation for UNIX hosts:

```
-set "SSL_CERT_DIR=/u/myuser/sslcerts/"
```

Syntax

SSL_CERT_DIR="*file-path*"

Syntax Description

"*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. 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.x.

Details

Environment variables SSLCACERTDIR and SSL_CERT_DIR point to a directory that contains all of the public certificate files of all CAs in the trust chain. One file exists for each CA in the trust chain.

SSL_CERT_DIR requires the certificates to be in the directory where their names are the value of a hash that OpenSSL generates.

Each CA certificate file must be PEM-encoded (base64). For more information, see "Certificate Encoding Formats and Extensions" on page 66.

For Foundation servers such as workspace servers and stored process servers (that is, servers in a deployment), if certificates are used, SAS searches for certificates in the following specific order:

1. SAS looks for SAS system option SSLCALISTLOC= to find a file. This file holds your trusted certificates.
2. If trustedcerts.pem exists and the SSLCACERTDIR= system option is set, SAS checks trustedcerts.pem first before it searches the directory.
3. If trustedcerts.pem does not exist, but the certificates are in the directory defined by the SSLCACERTDIR= system option, then SAS ignores SSLCALISTLOC=.
4. If trustedcerts.pem does not exist, and the SSLCACERTDIR= system option is not set, SAS reports an error.

With SAS 9.4, SAS 9.4M1, and SAS 9.4M2, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is ***SAS-configuration-directory*/Levn/certs/cacert.pem**. The cacert.pem file contains the list of trusted certificates.

Starting with SAS 9.4M3, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is **<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem**. The trustedcerts.pem file contains the list of trusted certificates.

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.

Appendix 2

Prior 9.4M8 Supported Ciphers for the SSLMODE= System Option

Before SAS 9.4M8: SSLMODE= System Option Supported Ciphers

Note: For Java clients, the list of supported TLS ciphers is determined by the Java Cryptography Extension (JCE) providers installed in the SAS Private Java Runtime Environment.

Prior to SAS 9.4M8, the following ciphers were supported.

Table A6.1 *Ciphers Supported for SSLMODE= Option on UNIX and Linux*

SSLMODE=	TLS Cipher	OpenSSL Cipher Used
SSLMODESUITEB192	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDHE-ECDSA-AES256-GCM-SHA384
SSLMODESUITEB128	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDHE-ECDSA-AES256-GCM-SHA384
	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ECDHE-ECDSA-AES128-GCM-SHA256

SSLMODE=	TLS Cipher	OpenSSL Cipher Used
SSLMODESP800131A	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDHE-ECDSA-AES256-GCM-SHA384
	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ECDHE-ECDSA-AES128-GCM-SHA256
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDHE-RSA-AES128-GCM-SHA256
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDHE-RSA-AES256-GCM-SHA384
	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	ECDHE-ECDSA-AES128-SHA256
	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	ECDHE-ECDSA-AES256-SHA384
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	ECDHE-RSA-AES128-SHA256
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	ECDHE-RSA-AES256-SHA384
	TLS_RSA_WITH_AES_128_GCM_SHA256	AES128-GCM-SHA256
	TLS_RSA_WITH_AES_256_GCM_SHA384	AES256-GCM-SHA384
	TLS_RSA_WITH_AES_128_CBC_SHA256	AES128-SHA256
	TLS_RSA_WITH_AES_256_CBC_SHA384	AES256-SHA256
SSLMODEDEPRECATED	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ECDHE-ECDSA-AES128-GCM-SHA256
	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDHE-ECDSA-AES256-GCM-SHA384
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDHE-RSA-AES128-GCM-SHA256
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDHE-RSA-AES256-GCM-SHA384
	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	ECDHE-ECDSA-AES128-SHA256

SSLMODE=	TLS Cipher	OpenSSL Cipher Used
	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	ECDHE-ECDSA-AES256-SHA384
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	ECDHE-RSA-AES128-SHA256
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	ECDHE-RSA-AES256-SHA384
	TLS_RSA_WITH_AES_128_GCM_SHA256	AES128-GCM-SHA256
	TLS_RSA_WITH_AES_256_GCM_SHA384	AES256-GCM-SHA384
	TLS_RSA_WITH_AES_128_CBC_SHA256	AES128-SHA256
	TLS_RSA_WITH_AES_256_CBC_SHA384	AES256-SHA256
	TLS_DHE_RSA_WITH_AES_128_CBC_SHA	DHE-RSA-AES128-SHA
	TLS_DHE_RSA_WITH_AES_256_CBC_SHA	DHE-RSA-AES256-SHA
	TLS_RSA_WITH_AES_256_CBC_SHA	AES128-SHA
	TLS_RSA_WITH_AES_128_CBC_SHA	AES256-SHA

Table A6.2 *Ciphers Supported for SSLMODE= Option on Windows*

SSLMODE	TLS Cipher	Schannel Algorithms Used	Resulting Ciphers
SSLMODESUITEB192	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	CALG_ECDHE CALG_ECDSA CALG_AES_256 CALG_SHA_384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
SSLMODESUITEB128	TLS_ECDHE_ECDSA_WITH_AES	CALG_ECDHE CALG_ECDSA	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

SSLMODE	TLS Cipher	Schannel Algorithms Used	Resulting Ciphers
	_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	CALG_AES_128 CALG_AES_256 CALG_SHA_384	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
SSLMODESP800131A	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 TLS_RSA_WITH_AES_128_GCM_SHA256 TLS_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_128_CBC_SHA256	CALG_ECDHE CALG_ECDSA CALG_RSA_SIGN CALG_AES_128 CALG_AES_256 CALG_SHA_256 CALG_SHA_384	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SH TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_256_CBC_SHA256

SSLMODE	TLS Cipher	Schannel Algorithms Used	Resulting Ciphers
	TLS_RSA_WITH_AES_256_CBC_SHA384		
SSLMODEDEPRECATED	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 TLS_RSA_WITH_AES_128_GCM_SHA256 TLS_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_256_CBC_SHA384 TLS_DHE_RSA_WITH_AES_128_CBC_SHA	CALG_ECDHE CALG_DH_EPHEM CALG_ECDSA CALG_RSA_SIGN CALG_AES_128 CALG_AES_256 CALG_SHA_256 CALG_SHA_384 CALG_SHA	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 TLS_RSA_WITH_AES_128_GCM_SHA256 TLS_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_256_CBC_SHA256 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_256_CBC_SHA256

SSLMODE	TLS Cipher	Schannel Algorithms Used	Resulting Ciphers
	TLS_DHE_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_128_CBC_SHA		TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA TLS_DHE_RSA_WITH_AES_128_CBC_SHA TLS_DHE_RSA_WITH_AES_256_CBC_SHA
SSLMODEDEPRECATED (continued)	(continued)	(continued)	(continued) TLS_DHE_DSS_WITH_AES_128_CBC_SHA256 TLS_DHE_DSS_WITH_AES_256_CBC_SHA256 TLS_DHE_DSS_WITH_AES_128_CBC_SHA TLS_DHE_DSS_WITH_AES_256_CBC_SHA

Appendix 3

Security Terminology

Security 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 that is 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. .

encryption
: Encryption is a process of protecting data. 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 protocol.

hashing
: Hashes are commonly used to validate passwords without having to store or record the password itself. 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. SHA256 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.

The following terminology is fundamental to understanding basic concepts for using the TLS protocol.

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.