Encryption in SAS® 9.4, Sixth Edition

SAS® Documentation

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

November 2016

9.4-P1:secref

Contents

About This Book

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

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 <option(s)> <argument-2 <option(s)> ...>
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-option(s)*)>
<*value-range-set-1* <(*picture-1-option(s)*)>
<*value-range-set-2* <(*picture-2-option(s)*)> *…*>>;

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

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

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

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 in SAS 9.4

Overview

Encryption in SAS is affected by the following changes and enhancements in SAS:

- SAS/SECURE is included with Base SAS, instead of being licensed and ordered separately.
- The new encoding type SAS004 (uses AES encryption with 64-bit salt) provides increased security for stored passwords.
- Increased security is provided for SAS data on disk.
- Enhanced logging features are introduced for encryption. These enhancements include new loggers and better debugging and traceback features that are now part of the SAS Logging Facility.
- Digital certificates can be imported to a central location on a Windows client or server.
- In the first maintenance release for SAS 9.4, 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.
- New 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 support logging. The default location is specified by the SSLCALISTLOC= system option.

  Note: These environment variables are available through hot fixes in some maintenance releases.
- Starting in the first maintenance release for SAS 9.4, 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.
- In the third maintenance release of SAS 9.4, CA certificates are now located in the trustedcerts.pem file for UNIX and z/OS. The SSLCALISTLOC= option on UNIX and z/OS now points to the trustedcerts.pem file by default.
- The SAS_SSL_MIN_PROTOCOL environment variable supported on UNIX, Windows, and z/OS, and the SAS_SSL_CIPHER_LIST environment variable supported on UNIX and z/OS have been added.

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

- In the third maintenance release of SAS 9.4, 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 trusted Mozilla CA bundle (provided by SAS) for all hosts. After SAS installation, you can use the SAS Deployment Manager to add your own trusted certificates to this list.
- In the third maintenance release of SAS 9.4, information about setting up a FIPS-2 environment has been updated in the SAS Deployment Wizard.
- In the fourth maintenance release of SAS 9.4, the OpenSSL libraries provided by SAS have been updated. For SAS 9.4 and all maintenance releases of SAS 9.4, updated versions of OpenSSL are provided and updated through hot fixes for UNIX and z/OS.

General Enhancements

- For software delivery purposes, SAS/SECURE is a product within the SAS System. In SAS 9.4, SAS/SECURE is included with the Base SAS software. In prior releases, SAS/SECURE was an add-on product that was licensed separately. This change makes strong encryption available in all deployments (except where prohibited by import restrictions).
- If you use SAS/SECURE, you can use a new encoding type 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 Chapter 1, "Technologies for Encryption," on page 3 and Chapter 4, "PWENCODE Procedure," on page 53.
- 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 Language Reference: Concepts*.
- 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 now 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 20.
- In the first maintenance release and the second maintenance release for SAS 9.4, for TLS encryption, 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 foundation servers. This default location is specified by the SSLCALISTLOC= option in configuration files. For more information, see "SSLCALISTLOC= System Option" on page 30.
- In the third maintenance release of SAS 9.4, 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.

- 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 Environment Variable" on page 46 and "SSL_CERT_DIR Environment Variable" on page 48.

- Starting in the first maintenance release for SAS 9.4, UNIX and z/OS clients and servers now 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 Environment Variable" on page 50.

- In the third maintenance release of SAS 9.4, two new environment variables are available: SAS_SSL_MIN_PROTOCOL, supported on UNIX, Windows, and z/OS, and SAS_SSL_CIPHER_LIST, supported on UNIX and z/OS. For more information, see "SAS_SSL_MIN_PROTOCOL Environment Variable" on page 43 and "SAS_SSL_CIPHER_LIST Environment Variable" on page 45.

- On a Windows server or client, the user can import digital certificates to a Machine Store as well as to a Personal Store. See "TLS on Windows: Setting Up Digital Certificates " on page 106.

- In the third maintenance release of SAS 9.4, 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 trusted CA bundle (provided by SAS) 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 trusted CA bundle, on UNIX, the SAS Deployment Manager task updates the trustedcerts.pem and the trustedcerts.jks files, and on z/OS, the SAS Deployment Manager tasks update the trustedcerts.pem file.

  See "Add Your Certificates to the Windows CA Stores" on page 109 , "Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager" on page 96 and, . For the specific details about these SAS Deployment Manager tasks, see the *SAS® Deployment Wizard and SAS® Deployment Manager 9.4: User's Guide*.

- In the third maintenance release of SAS 9.4, information has been added about setting the FIPS security settings on a Windows server. See "TLS on Windows: FIPS 140-2 Capable OpenSSL " on page 117.

- In the third maintenance release of SAS 9.4, 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 information about FIPS in this document, see "FIPS 140-2 Standards Compliance" on page 5, "TLS: FIPS 140-2 Compliant Installation and Configuration" on page 14, and "TLS on UNIX: Building FIPS 140-2 Capable OpenSSL " on page 103.

- In the fourth maintenance release of SAS 9.4, the OpenSSL libraries provided by SAS have been updated. For SAS 9.4 and all maintenance releases of SAS 9.4, 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 in SAS products and about the advisories under consideration.

For a quick reference of the OpenSSL version supported for each version of SAS Foundation, see Mapping Between SAS Version and OpenSSL Version.

Note: Windows versions of SAS support the TLS versions that Windows supports.

Documentation Enhancements

In the fourth maintenance release of SAS 9.4, we have moved information about certificate management 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 101.
- See "Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager" on page 96.
- See "Overview of Certificate Management Using the SAS Deployment Manager" on page 82.

Accessibility

For information about the accessibility of this product, see Accessibility Features of the Windowing Environment for SAS 9.4 at support.sas.com.

Part 1

Encryption in SAS 9.4

Chapter 1

Technologies for Encryption

Overview of Encryption

Security Concepts

SAS provides strategies for protecting information that is associated with a SAS deployment. Some components supporting this protection are based on third-party components that are incorporated into the SAS product delivery, and some are SAS-specific components. SAS provides products and third-party strategies for protecting data and credentials (user IDs and passwords) that are exchanged in a networked environment. 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 transit or data at rest.

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.

  Encoding obfuscates the data. Your data should be protected by other security controls as well. Use file system permissions or other access control mechanisms. Encoding does not provide data confidentiality.

  Examples are SAS002, SAS003, and SAS004 encoding and SAS Proprietary 32-bit fixed key encoding.

encryption
: Encryption transforms data into another format in such a way that only specific individual(s) 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.

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

  Examples are SHA-256 and 512 hashing algorithms.

salting
: Salt is data used as an additional input to the encryption algorithm. When the salt is being used, the first eight bytes of the encrypted data are reserved for the salt. The salt value is generated at random when encrypting a file and read from the encrypted file when it is decrypted.

  Examples are AES with 16-bit salt (SAS003) and AES with 64-bit salt (SAS004).

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 recommended to use industry-standard encryption and hashing algorithms. These methods provide stronger protection and are available in all deployments, except where prohibited by import restrictions.

  Note: Industry-standard algorithms are provided by SAS/SECURE. For details about supported algorithms and availability, see, "Providers of Encryption" on page 6.

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-at-rest encryption protects data at rest. The emphasis is on protection of passwords in configuration files and in the metadata repository, and on encryption of SAS data sets.
- Data-in-motion encryption protects data in transit. The emphasis is on protection of passwords and data in transit. You can also choose to protect all traffic in transit between SAS servers and SAS desktop clients.

  Note: To ensure that only FIPS-validated encryption algorithms are used, set the ENCRYPTFIPS system option. See "ENCRYPTFIPS System Option" on page 23.

FIPS 140-2 Standards Compliance

Overview

FIPS 140-2 standards are supported for SAS/SECURE and Transport Layer Security (TLS) encryption technologies. 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 140-1 or FIPS 140-2 certificate. The certificate specifies the exact module name, hardware, software, firmware, and applet version numbers. For more information, see FIPS PUB 140-2 SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES.

There are four levels of security: from Level 1 (lowest) to Level 4 (highest). The security requirements cover areas related to the secure design and implementation of a cryptographic module. These areas include basic design and documentation, module interfaces, authorized roles and services, physical security, software security, operating system security, key management, cryptographic algorithms, electromagnetic interference or electromagnetic compatibility (EMI/EMC), and self-testing.

For installation and configuration details about FIPS 140-2, see "SAS/SECURE FIPS 140-2 Compliant Installation and Configuration" on page 11, "TLS: FIPS 140-2 Compliant Installation and Configuration" on page 14, and "ENCRYPTFIPS System Option" on page 23.

How SAS Implements FIPS

The ENCRYPTFIPS option is provided by SAS primarily as a mechanism to help ensure that your SAS system is configured to leverage the encryption algorithms and cipher suites specified by the FIPS 140-2 standard and that libraries will be validated for

compliance when loaded. With this 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. ENCRYPTFIPS makes sure IOM uses AES and that SAS/CONNECT uses AES or SSL.

However, turning off the SAS system option ENCRYPTFIPS does not impact the ability of SAS to use FIPS approved encryption algorithms available with SAS/SECURE, 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 will attempt to load a special subset of OpenSSL libraries, contained as part of the OpenSSL FIPS Object Module. These libraries are not present by default and would 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. See "TLS on UNIX: Building FIPS 140-2 Capable OpenSSL " on page 103 for additional information.

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 option is enabled and the libraries are not present, then the SAS system produces an error when a SAS server needs to act as a TLS client and communicate over HTTPS protocol.

Note: When the ENCRYPTFIPS option is turned on, SAS Internal Passwords are stored using the SHA-256 hashing algorithm.

Providers of Encryption

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. SAS/SECURE and TLS provide a higher level of security.

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: SAS/SECURE and data set support of AES, which is also shipped with Base SAS software, provides a higher level of security.

  For detailed information, see "SAS Data File Encryption" in *SAS Language Reference: Concepts*.

- 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 SAS Proprietary algorithm is strong enough to protect your data from casual viewing. It provides a medium level of security. SAS/SECURE and TLS provide a higher level of security.

  Note: SAS recommends that you use the highest levels of security possible.

  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. For example, TLS or AES can be used.

  Data-at-rest passwords are secured in the following ways.

  - Login passwords on-disk in the metadata can use SASProprietary (SAS002) or AES (SAS003 or SAS004). 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 MD5 or SHA-256 hashing. By default, SHA256 is used.

    Note: If SAS/SECURE is not present, MD5 is used. If MD5 hashing is specified in the configuration file, it overrides the default SHA-256 hashing.
  - Passwords on-disk in configuration files can use SASProprietary or AES. By default, SASProprietary (SAS002) is used.

  Note: Configuration file passwords can be upgraded to AES (SAS003 or SAS004). 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.

Refer to the *SAS 9.4 Intelligence Platform: Security Administration Guide*, “Encryption for Data at Rest” and “Encryption for Data in Motion” chapters for detailed information.

SAS Proprietary Encryption System Requirements

SAS supports SAS Proprietary Encryption under these operating environments:

- UNIX
- 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 Installation and Configuration

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

Configure SAS Proprietary Encryption as follows:

- SAS Proprietary Encryption for SAS data sets is implemented with the ENCRYPT= data set option set to YES. You can use the ENCRYPT= data set option only when you are creating a SAS data file. You must also assign a password when encrypting a data file with SAS Proprietary Encryption. At a minimum, you must specify the READ= data set option or the PW= data set option at the same time you specify ENCRYPT=YES. Because passwords are used in this encryption technique, you

cannot change any password on an encrypted data set without re-creating the data set.

Note: Beginning with the first maintenance release of 9.4, 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* and *SAS Guide to Metadata-Bound Libraries*.

For detailed information, see "SAS Data File Encryption" in *SAS Language Reference: Concepts*.

- 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 SASProprietary algorithm be used for encryption of all data that is exchanged with connecting clients.

  For detailed information, see "NETENCRYPT System Option" on page 26 and "NETENCRYPTALGORITHM System Option" on page 26.

For an example of configuring and using SAS Proprietary Encryption in your environment, see "SAS Proprietary Encryption for SAS/SHARE: Example " on page 64. For an example of configuring SAS Data File encryption using the SASProprietary algorithm, see "SAS Data File Encryption" in *SAS Language Reference: Concepts*.

SAS/SECURE

SAS/SECURE Overview

SAS/SECURE software provides industry standard encryption capabilities in addition to the SASProprietary algorithm.

On UNIX, Windows, and z/OS, SAS/SECURE supports the following encryption algorithms:

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

Note: The algorithms listed above are supported by SAS/SECURE on Windows by using the Microsoft Cryptographic API libraries that are included with the operating system.

Refer to "Encryption Algorithms" on page 17 for more information about encryption algorithms supported for use with SAS/SECURE.

SAS/SECURE provides encryption for the following:

- data in transit

SAS/SECURE enables you to provide stronger protection for data in transit than is provided by SAS Proprietary Encryption. This affects communications among SAS servers and between SAS servers, SAS desktop clients, and SAS web applications.

Refer to "NETENCRYPT System Option" on page 26 and "NETENCRYPTALGORITHM System Option" on page 26 for details.

- stored login passwords

  SAS/SECURE also enables you to provide stronger protection for stored login passwords than is provided by SAS Proprietary encoding. By default, the stored login passwords are stored using SAS002 encoding. With SAS/SECURE, you can use SAS003 or SAS004 encoding methods, which use industry-standard algorithms for stored passwords. The SAS003 encoding method uses AES with 16-bit salt and the SAS004 encoding method uses AES with 64-bit salt. You can use the PWENCODE procedure (specify the METHOD= option) to upgrade to stronger encryption, AES (SAS003 or SAS004).

  Refer to Chapter 4, "PWENCODE Procedure," on page 53 for details.

- internal account passwords stored in the metadata repository

  SAS/SECURE also enables you to provide stronger protection for internal account passwords stored in the metadata repository. You should use a minimum of SHA-256 hashing.

  CAUTION:
  Passwords that are stored in SAS003 format, SAS004 format, or SHA-256 hashing become unusable and inaccessible if SAS/SECURE is unavailable. If you choose to discontinue use of SAS/SECURE, you must revert stored passwords to the less secure format before you discontinue using the software.

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

  You can instruct SAS/SECURE to use only services that are part of the Federal Information Processing Standard (FIPS) 140-2 standard. When SAS system option ENCRYPTFIPS is configured, SAS/SECURE uses only FIPS 140-2 validated encryption and hashing algorithms from libraries that are validated when loaded. AES is the encryption algorithm and SAS003 is the encoding format (for stored passwords) used with FIPS 140-2 enabled SAS/SECURE software. The SHA-256 hashing algorithm is used with FIPS 140-2 enabled software for stored internal account passwords in the metadata server.

  Refer to "How SAS Implements FIPS" on page 5."ENCRYPTFIPS System Option" on page 23Also see and "SAS/SECURE FIPS 140-2 Compliant Installation and Configuration" on page 11 for details.

- AES Encryption on Data Sets

  AES encryption on SAS Data Files is available in SAS 9.4. AES produces a stronger encryption by using a key value that can be up to 64 characters long. You specify ENCRYPT=AES when creating a data set. Instead of passwords that are stored in the data set (SAS Proprietary Encryption), AES uses a key value that is not stored in the data set. You must use the ENCRYPTKEY= data set option when creating or accessing an AES encrypted data set unless the metadata-bound library administrator has securely recorded the encryption key in metadata to which the data set is bound. You cannot change the ENCRYPTKEY= key value on an AES encrypted data set without re-creating the data set.

Note: Beginning with the first maintenance release of 9.4, 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* and *SAS Guide to Metadata-Bound Libraries*.

For detailed information about AES Encryption on SAS Data Files, see "AES Encryption" in *SAS Language Reference: Concepts*.

SAS/SECURE System Requirements

SAS supports SAS/SECURE under these operating environments:

- UNIX
- Windows
- z/OS

SAS/SECURE Software Availability

For software delivery purposes, SAS/SECURE is a product within the SAS System. In SAS 9.4, SAS/SECURE is included with the Base SAS software. In prior releases, SAS/SECURE was an add-on product that was licensed separately. This change makes strong encryption available in all deployments (except where prohibited by import restrictions).

SAS/SECURE Export Restrictions

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

SAS/SECURE for UNIX, Windows, and z/OS includes the following encryption algorithms:

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

SAS/SECURE for Windows uses the encryption algorithms that are available in Microsoft CryptoAPI. The level of the SAS/SECURE encryption algorithms under Windows depends on the level of the encryption support in Microsoft CryptoAPI under Windows.

Note: For AES, SAS does not use Windows libraries by default. It tries to use the RSA libraries that are FIPS certified.

SAS/SECURE Installation and Configuration

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

In SAS 9.4, SAS/SECURE is installed with the Base SAS software. However, the default encryption is now SAS Proprietary Encryption.

To use the higher form of encryption provided by SAS/SECURE for communications and networking, specify the NETENCRYPT system option and set the NETENCRALG= system option to a value of RC2, RC4, DES, TRIPLEDES, AES, or SSL. Refer to "NETENCRYPT System Option" on page 26 and "NETENCRYPTALGORITHM System Option" on page 26.

For examples of configuring and using SAS/SECURE with SAS/CONNECT and SAS/SHARE, see Chapter 5, "Encryption Technologies: Examples," on page 63.

For detailed information about AES Encryption on SAS Data Files, see "AES Encryption" in *SAS Language Reference: Concepts*.

SAS/SECURE FIPS 140-2 Compliant Installation and Configuration

To configure a FIPS 140-2 compliant system, you must use SAS/SECURE or TLS. When using SAS/SECURE, specify SAS system options ENCRYPTFIPS and NETENCRALG= (set to AES) for UNIX, z/OS, or Windows. When using TLS, specify SAS system options ENCRYPTFIPS and NETENCRALG= (set to AES or SSL) for UNIX, z/OS, or Windows. Therefore, you can connect only servers and clients that are also configured with AES or SSL. Errors are generated when other encryption algorithms are specified.

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

In the FIPS 140-2 compliant mode, the SHA-256 hashing algorithm is used for stored password protection. Therefore, you can connect only servers and clients that are enabled for FIPS 140-2.

CAUTION:

In 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 SHA-256 hashing algorithm.

See the following information for details about FIPS.

- "ENCRYPTFIPS System Option" on page 23
- "NETENCRYPTALGORITHM System Option" on page 26
- "FIPS 140-2 Standards Compliance" on page 5
- "TLS: FIPS 140-2 Compliant Installation and Configuration" on page 14

Transport Layer Security (TLS)

Transport Layer Security (TLS) Overview

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

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

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

generate, sign, and administer the validity of certificates. For more information about client and server negotiations using certificates, refer to "Certificate Implementation: How TLS Client and Servers Negotiate" on page 83.

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 Concepts

The following concepts are fundamental to understanding TLS:

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

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

  Note: z/OS provides the PACDCERT command and PKI Services for implementing a CA.

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.

TLS System Requirements

SAS supports TLS under these operating environments:

- UNIX
- Windows
- z/OS

Note: The TLS software is included in the SAS installation software only for countries that allow the importation of encryption software.

TLS Software Availability

In the fourth maintenance release of SAS 9.4, the OpenSSL libraries provided by SAS have been updated. For SAS 9.4 and all maintenance releases of SAS 9.4, updated versions of OpenSSL 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 in SAS products and about the advisories under consideration.

For a quick reference of the OpenSSL version supported for each version of SAS Foundation, see Mapping Between SAS Version and OpenSSL Version.

The TLS version supported by default is now TLS 1.2.

Note: If you need to override the default protocol, you can set the SAS_SSL_MIN_PROTOCOL= environment variable. For information, see "SAS_SSL_MIN_PROTOCOL Environment Variable" on page 43.

OpenSSL is shipped with Base SAS for UNIX and z/OS. Windows versions of SAS support the TLS versions that Windows supports.

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

- For Windows, SAS uses the SChannel library that comes with the Windows operating system.
- SAS provides the libraries needed to run TLS on UNIX. To find the OpenSSL code base version that is used to build the TLS libraries provided by SAS for each release, see Mapping Between SAS Version and OpenSSL Version.

  Note: Different operating systems require the use of different library file extensions. For example, HPUX, 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.
- SAS provides the libraries needed to run TLS on z/OS.

TLS Installation and Configuration

TLS for UNIX, z/OS, and Windows is shipped with Base SAS. No additional software installation is required. For SAS 9.4 and all maintenance releases of SAS 9.4, 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.

In the third maintenance release of SAS 9.4, the SAS Deployment Manager used to automate the process of adding and removing 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 is then used to add existing certificates to the Trusted CA Bundle and to remove digital certificates from the Trusted CA Bundle.

In the third maintenance release of SAS 9.4, this new 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, which is now **`<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem`**.

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

- Chapter 7, "Installing and Configuring TLS and Certificates on UNIX," on page 85
- Chapter 8, "Installing and Configuring TLS and Certificates on Windows," on page 105
- Chapter 9, "Installing and Configuring TLS and Certificates on z/OS," on page 119

For examples of configuring and using TLS in your environment, see Chapter 5, "Encryption Technologies: Examples," on page 63.

TLS: FIPS 140-2 Compliant Installation and Configuration

You can configure TLS to run in FIPS 140-2 compliant mode. The libraries supplied by Windows are FIPS 140-2 compliant. The OpenSSL libraries shipped with SAS for UNIX 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 "TLS on UNIX: Building FIPS 140-2 Capable OpenSSL " on page 103 and "TLS on Windows: FIPS 140-2 Capable OpenSSL " on page 117.

After compiling the FIPS compliant libraries, you will need to specify the NETENCRALG= (set to AES or SSL) system option. You will then be able to run in FIPS compliant mode and connect only 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 will 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 "TLS on UNIX: Building FIPS 140-2 Capable OpenSSL " on page 103.

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

Note: The TLS version shipped with SAS for z/OS is not FIPS 140-2 compliant. However, you can use SAS/SECURE with AES to provide FIPS on z/OS.

See the following for more information about FIPS.

- "How SAS Implements FIPS" on page 5
- "NETENCRYPTALGORITHM System Option" on page 26
- "ENCRYPTFIPS System Option" on page 23

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.

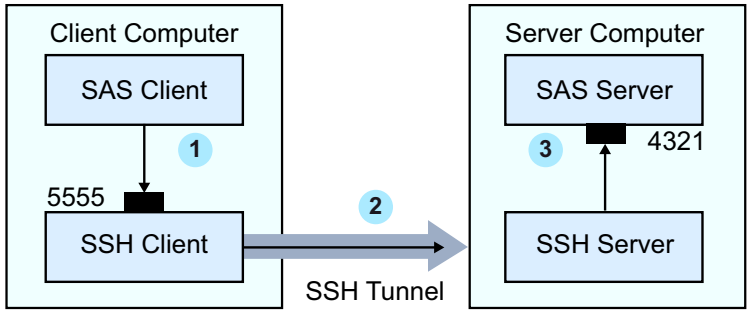
To build the OpenSSL software, refer to the following resources:

- 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 1.1 *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 "SSH Tunnel for SAS/CONNECT: Example " on page 75 and "SSH Tunnel for SAS/SHARE: Example " on page 76.

Encryption Algorithms

The following encryption algorithms are provided with Base SAS:

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 that encrypts data in blocks of 64 bits. A *block cipher* is an encryption algorithm that divides a message into blocks and encrypts each block. The RC2 key size ranges from 8 to 256 bits. SAS/SECURE uses a configurable key size of 40 or 128 bits. (The NETENCRYPTKEYLEN system option is used to configure the key length.) The RC2 algorithm expands a single message by a maximum of 8 bytes. RC2 is an algorithm developed by RSA Data Security, Inc.

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

RC4
: is a stream cipher. A *stream cipher* is an encryption algorithm that encrypts data one byte at a time. The RC4 key size ranges from 8 to 2048 bits. SAS/SECURE uses a configurable key size of 40 or 128 bits. (The NETENCRYPTKEYLEN system

option is used to configure the key length.) RC4 is an algorithm developed by RSA Data Security, Inc.

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 a single 56-bit key. This has the effect of encrypting the data by using a 168-bit key. TripleDES expands a single message by a maximum of 8 bytes. TripleDES is defined in the American National Standards Institute (ANSI) X9.52 specification.

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.

SHA-1 (Secure Hash Algorithm)
: produces a 160-bit (20-byte) hash value. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long. This algorithm was developed by the U.S. National Security Agency (NSA) and published in 2001 by the NIST as a U.S. Federal Information Processing Standard (FIPS) PUB 180-1.

  Note: SHA-1 is a FIPS 140-2 compliant hashing algorithm.

SHA-256 (Secure Hash Algorithm)
: is essentially a 256-bit block cipher algorithm that encrypts the intermediate hash value using the message block as key. 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: SHA-256 is a FIPS 140-2 compliant hashing algorithm.

Comparison of Encryption Technologies

The following table compares the features of encryption technologies:

Table 1.1 *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 authentication	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 others	Product dependent
Installation required	No (part of Base SAS)	No (delivered with Base SAS)	Delivered with Base SAS. You can replace this version.	Yes
Operating environments supported	UNIX Windows z/OS	UNIX Windows z/OS	UNIX Windows z/OS	UNIX 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 Chapter 2, "SAS System Options for Encryption," on page 23. For examples, see Chapter 5, "Encryption Technologies: Examples," on page 63.

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: The logging of the SAS Deployment Manager add and remove certificate tasks for managing of the Trusted CA Bundle is located at **`<SASHOME>/InstallMisc/InstallLogs/certframe*`**.

Table 1.2 *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.
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 RSA BSAFE® Crypto-C.

Logger	SAS/SECURE Information
App.tk.eam.rsa.ccme	Logs AES encryption information for RSA 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.

Note: 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 filename 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.

See Also

For 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= option are NONE or HIGH. SAS encrypts PDF documents using a 128-bit encryption algorithm. With PDFSECURITY=HIGH, at least one password must be set using the PDFPASSWORD= system option. A password is required to open a PDF file that has been generated with ODS.

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

The following table lists the PDF system options that are available to restrict or allow users' ability to access, assemble, copy, or modify ODS PDF files. Other SAS system options control whether the user can fill in forms and set the print resolution. These system options are documented in *SAS System Options: Reference*.

Table 1.3 *PDF System Options*

Task	System Option
Specifies whether text and graphics from PDF documents can be read by screen readers for the visually impaired	PDFACCESS | NOPDFACCESS
Controls whether PDF documents can be assembled	PDFASSEMBLY | NOPDFASSEMBLY

Task	System Option
Controls whether PDF document comments can be modified	PDFCOMMENT | NOPDFCOMMENT
Controls whether the contents of a PDF document can be changed	PDFCONTENT | NOPDFCONTENT
Controls whether text and graphics from a PDF document can be copied	PDFCOPY | NOPDFCOPY
Controls whether PDF forms can be filled in	PDFFILLIN | NOPDFFILLIN
Specifies the password to use to open a PDF document and the password used by a PDF document owner	PDFPASSWORD =
Controls the resolution used to print the PDF document	PDFPRINT=
Controls the printing permissions for PDF documents	PDFSECURITY=

Note: The SAS/SECURE software and TLS libraries are included in the SAS installation software only for countries that allow the importation of encryption software.

Chapter 2

SAS System Options for Encryption

Dictionary

ENCRYPTFIPS System Option

Specifies that the SAS/SECURE and TLS security services use FIPS 140-2 validated 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

Restriction: The ENCRYPTFIPS option is not supported on z/OS for TLS.

Operating environment: UNIX, Windows, z/OS

See: NETENCRYPTALGORITHM

Syntax

ENCRYPTFIPS

Syntax Description

ENCRYPTFIPS

specifies that SAS/SECURE and TLS services are using FIPS 140-2 compliant encryption algorithms.

Note: Turning on the ENCRYPTFIPS option is not generally recommended for TLS, unless absolutely required by your sites policy.

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

Note: SAS Internal Passwords are stored using the SHA-256 hashing algorithm when this option is specified.

The ENCRYPTFIPS option is provided by SAS primarily as a mechanism to help ensure that your SAS system 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 your SAS servers have been configured to use the FIPS approved Advanced Encryption Standard (AES) libraries or the SSL protocol. ENCRYPTFIPS makes sure IOM uses AES and that SAS/CONNECT uses AES or SSL.

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

CAUTION:

Use ENCRYPTFIPS with caution. Turning on the ENCRYPTFIPS option is not generally recommended for TLS, unless absolutely required by your sites policy. If the ENCRYPTFIPS option is turned on, the SAS server based TLS clients will 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 "TLS on UNIX: Building FIPS 140-2 Capable OpenSSL " on page 103.

Restriction 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.

Notes When configuring the ENCRYPTFIPS option on a Microsoft Windows 2003 server, refer to "SAS/SECURE FIPS 140-2 Compliant Installation and Configuration" on page 11 for instructions on resolving the environment variable issue.

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.

NOENCRYPTFIPS
specifies that the SAS/SECURE and TLS security services are not limited to FIPS 140-2 verified algorithms.

Details

The ENCRYPTFIPS option limits the services provided by SAS/SECURE and TLS to those services that are part of the FIPS 140-2 specification.

Note: Turning on the ENCRYPTFIPS option is not generally recommended for TLS, unless absolutely required by your sites policy.

Read more about Security Requirements for Cryptographic Modules at FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION. Refer to "FIPS 140-2 Standards Compliance" on page 5 for an overview of FIPS 140-2 standards.

There is an interaction between the ENCRYPTFIPS option and the NETENCRYPTALGORITHM option. Only the AES algorithm or the SSL algorithm 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*.

Examples

Example 1

Here is an example of configuring the ENCRYPTFIPS option on UNIX:

```
-encryptfips -netencryptalgorithm aes;
```

Example 2

Here is an example of configuring the ENCRYPTFIPS option on z/OS:

```
encryptfips netecryptalgorithm="aes"
```

Example 3

Here is an example of configuring the ENCRYPTFIPS option on Windows:

```
-encryptfips -netencralg "AES"
```

See Also

- "NETENCRYPTALGORITHM System Option" on page 26
- "FIPS 140-2 Standards Compliance" on page 5

- "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:	"SAS/SECURE for SAS/CONNECT: Example " on page 64

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 System Option" on page 26, "ENCRYPTFIPS System Option" on page 23

Examples: "TLS for a SAS/CONNECT Windows Spawner: Example " on page 67
"TLS on a z/OS Spawner on a SAS/CONNECT Server: Example" on page 69
"TLS for a SAS/CONNECT UNIX Spawner: Example " on page 65

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.

The following algorithms can be used:

- RC2
- RC4
- DES
- TripleDES
- SAS Proprietary
- SSL
- AES

Restrictions If you do not have SAS/SECURE, an error is generated if algorithm AES is specified.

The SSL option is not applicable to the Integrated Object Model (IOM) servers.

When ENCRYPTFIPS is specified, only the SSL algorithm or the AES algorithm can be specified. Otherwise, an error message is output.

Details

The NETENCRYPTALGORITHM option must be specified in the server session.

Use this option to specify one or more encryption algorithms that you want to use to protect the data that is transferred across the network. If more than one algorithm is specified, the client session negotiates the first specified algorithm with the server session. If the client session does not support that algorithm, the second algorithm is negotiated, and so on.

If either the client session or the server session specifies the NETENCRYPT option (which makes encryption mandatory) but a common encryption algorithm cannot be negotiated, the client cannot connect to the server.

If the NETENCRYPTALGORITHM option is specified in the server session only, then the server's values are used to negotiate the algorithm selection. If the client session supports only one of multiple algorithms that are specified in the server session, the client can connect to the server.

There is an interaction between either NETENCRYPT or NONETENCRYPT and the NETENCRYPTALGORITHM option.

Table 2.1 *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 des tripledes);
```

The client and the server negotiate an algorithm that they share in common, TripleDES, for encrypting data transfers.

NETENCRYPTKEYLEN= System Option

Specifies the key length 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

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 that the maximum key length that is supported at both the client and the server is used.

40
specifies a key length of 40 bits for the RC2 and RC4 algorithms.

128
specifies a key length of 128 bits for the RC2 and RC4 algorithms. If either the client or the server does not support 128-bit encryption, the client cannot connect to the server.

Details

The NETENCRYPTKEYLEN= option supports only the RC2 and RC4 algorithms. The SAS Proprietary, DES, TripleDES, SSL, and AES algorithms are not supported.

By default, if you try to connect a computer that is capable of only a 40-bit key length to a computer that is capable of both a 40-bit and a 128-bit key length, the connection is made using the lesser key length. If both computers are capable of 128-bit key lengths, a 128-bit key length is used.

Using longer keys consumes more CPU cycles. If you do not need a high level of encryption, set NETENCRYPTKEYLEN=40 to decrease CPU usage.

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

Operating environment: UNIX, z/OS

Notes: In the third maintenance release of SAS 9.4, 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 9.4, the first maintenance release of SAS 9.4, and the second maintenance release of SAS 9.4, 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 86.

Examples: "TLS for a SAS/CONNECT UNIX Spawner: Example " on page 65

"TLS on a z/OS Spawner on a SAS/CONNECT Server: Example" on page 69

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

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 File Formats" on page 80.

From SAS 9.4 to the second maintenance release of SAS 9.4, 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.

In the third maintenance release of SAS 9.4, 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 in the third maintenance release of SAS 9.4, the SSLCALISTLOC= system option should not be overridden or changed unless directed by 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 96.

For the specifics and an example of how to create a trust list on z/OS, refer to "Step 5. Create a CA Trust List Using OpenSSL" on page 122. For information about creating a trust list on UNIX, refer to "Step 5. Create a Certificate Chain in PEM Format Using OpenSSL" on page 93 and "Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager" on page 96.

Note: Environment variables SSLCACERTDIR and SSL_CERT_DIR point 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 Environment Variable" on page 46 and "SSL_CERT_DIR Environment Variable" on page 48.

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 a specific order. SAS searches for certificates in the following order:

1. SAS looks for SAS system option SSLCALISTLOC= to find the file trustedcerts.pem.
2. SAS looks for the SSLCALISTLOC environment variable to find the file trustedcerts.pem.
3. If trustedcerts.pem exists and SSL_CERT_DIR and SSLCACERTDIR environment variables are set, SAS checks trustedcerts.pem first before it searches these directories.
4. If trustedcerts.pem does not exist, but the certificates are in the directory defined by SSL_CERT_DIR or SSLCACERTDIR, then SAS ignores SSLCALISTLOC=.

5. If trustedcerts.pem does not exist, and the SSL_CERT_DIR and SSLCACERTDIR environment variables are not set, SAS reports an error.

Note: A trusted CA certificate is required at the client in order to validate a server's digital certificate. The trusted CA certificate must be from the CA that signed the server certificate. The SSLCALISTLOC= option is required at the server only if the SSLCLIENTAUTH option is also specified at the server.

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: "TLS for SAS/SHARE on Windows: Examples " on page 73

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.

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, SAS/CONNECT spawner command line

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Operating environment: UNIX, 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*.

Examples: "TLS for a SAS/CONNECT UNIX Spawner: Example " on page 65

"TLS on a z/OS Spawner on a SAS/CONNECT Server: Example" on page 69

"TLS for SAS/SHARE on UNIX: Example " on page 71

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

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 File Formats" on page 80.

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: "TLS for SAS/SHARE on Windows: Examples " on page 73

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.

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: "TLS for a SAS/CONNECT Windows Spawner: Example " on page 67

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.

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

Operating environment: UNIX, 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

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.

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

Operating environment: UNIX, 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*.

Examples: "TLS on a z/OS Spawner on a SAS/CONNECT Server: Example" on page 69

"TLS for SAS/SHARE on z/OS: Example " on page 74

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, 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*.

Examples: "TLS on a z/OS Spawner on a SAS/CONNECT Server: Example" on page 69

"TLS for SAS/SHARE on z/OS: Example " on page 74

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.

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

Operating environment: UNIX, 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*.

Examples: "TLS for a SAS/CONNECT UNIX Spawner: Example " on page 65

"TLS for SAS/SHARE on UNIX: Example " on page 71

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 File Formats" on page 80.

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, 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*.

Examples: "TLS for a SAS/CONNECT UNIX Spawner: Example " on page 65

"TLS for SAS/SHARE on UNIX: Example " on page 71

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.

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.

SSLREQCERT= System Option

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

Operating environment: UNIX

Example: 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.

DEMAND
specifies that a server certificate is requested, and if no valid certificate is provided, the session terminates. DEMAND is the default setting.

NEVER
specifies that the Authentication Server does not ask for a certificate.

TRY
specifies that the client requests a server certificate, and if no certificate is provided, the session proceeds normally. If an invalid certificate is provided, the session terminates.

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.

Chapter 3

SAS Environment Variables for Encryption

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 File" 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 Environment Variables" in *SAS Companion for Windows*.

Dictionary

SAS_SSL_MIN_PROTOCOL Environment Variable

Specifies the minimum TLS or SSL protocol that can be negotiated when using OpenSSL.

Client: Optional

Server: Optional

Valid in: Configuration file, command line

Categories: Communications: Networking and Encryption

System Administration: Security

Default: TLS 1.2. Starting in the fourth maintenance release of SAS 9.4, the minimum OpenSSL protocol default is TLS 1.2.

Operating environment: UNIX, z/OS, and Windows

Notes: This environment variable must be set before TLS or SSL are loaded. It cannot be changed after TLS or SSL are loaded. You must set the environment variable before the SAS/CONNECT spawner is started and before SAS is started on the client.

This environment variable is available in all SAS 9.3 and SAS 9.4 versions of software if hot fixes are applied.

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*, "TKMVSENV File" in *SAS Companion for z/OS*, "Using Environment Variables" in *SAS Companion for Windows*

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 minimum TLS protocol version that is negotiated between UNIX, z/OS, and Windows servers when using OpenSSL. Valid values for TLS 1.2 are TLS1.2 and TLSv1.2.

See the SAS Security Bulletin on OpenSSL for the most current information about the versions of OpenSSL used in SAS products and about the advisories under consideration. For a list of the versions of OpenSSL libraries provided by SAS, see OpenSSL Version to SAS Software Release.

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.

TLS1.2 is the default minimum OpenSSL protocol version used to negotiate between client and servers in the fourth maintenance release of SAS 9.4. You can specify an earlier fallback value, but it is not recommended.

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: 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 or SSL are loaded. It cannot be changed after TLS or SSL 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*, "TKMVSENV File" in *SAS Companion for z/OS*

Examples: Export the environment variable on UNIX hosts for the Bourne Shell:

```
export SAS_SSL_CIPHER_LISTSS=TLS1.2
```

Set the environment variable on UNIX hosts for the C Shell environment:

```
SETENV SAS_SSL_CIPHER_LISTS HIGH
```

Set the environment variable at SAS invocation for UNIX hosts:

```
-set SAS_SSL_CIPHER_LISTS 3DES:RC2"
```

Set the environment variable on Windows hosts

```
SET SAS_SSL_CIPHER_LISTS 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 or SSL version. The OpenSSL Cipher information can be found at OpenSSL 1.0.1 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 20.

Note: If you set a minimum protocol that does not allow some ciphers, you might get an error.

For Windows, you can configure the SSL Cipher Suite Order in the group policy settings. Search the https://msdn.microsoft.com/en-US/ website for information about how to set the SSL or TLS Cipher Suite Order.

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 1.0.1 Ciphers.

Note: For Windows, you can configure the SSL Cipher Suite Order in the group policy settings. Search the https://msdn.microsoft.com/en-US/ website for information about how to set the SSL or 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= System Option" on page 30.

Operating environment: UNIX

Note: 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
```

See: "Defining Environment Variables in UNIX Environments" in *SAS Companion for UNIX Environments*

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 File Formats" on page 80.

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 a specific order. SAS searches for certificates in the following order:

1. SAS looks for SAS system option SSLCALISTLOC= to find the file trustedcerts.pem.
2. SAS looks for the SSLCALISTLOC environment variable to find the file trustedcerts.pem.
3. If trustedcerts.pem exists and SSL_CERT_DIR and SSLCACERTDIR environment variables are set, SAS checks trustedcerts.pem first before it searches the directory.
4. If trustedcerts.pem does not exist, but the certificates are in the directory defined by SSL_CERT_DIR or SSLCACERTDIR, then SAS ignores SSLCALISTLOC=.
5. If trustedcerts.pem does not exist, and the SSL_CERT_DIR and SSLCACERTDIR environment variables are not set, SAS reports an error.

In SAS 9.4, the first maintenance release of SAS 9.4, and the second maintenance release of SAS 9.4, 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.

In the third maintenance release of SAS 9.4, 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= System Option" on page 30.

Operating environment: UNIX

Note: 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
```

See: "Defining Environment Variables in UNIX Environments" in *SAS Companion for UNIX Environments*

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 File Formats" on page 80.

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 a specific order. SAS searches for certificates in the following order:

1. SAS looks for SAS system option SSLCALISTLOC= to find the file trustedcerts.pem.
2. SAS looks for the SSLCALISTLOC environment variable to find the file trustedcerts.pem.
3. If trustedcerts.pem exists and SSL_CERT_DIR and SSLCACERTDIR environment variables are set, SAS checks trustedcerts.pem first before it searches the directory.
4. If trustedcerts.pem does not exist, but the certificates are in the directory defined by SSL_CERT_DIR or SSLCACERTDIR, then SAS ignores SSLCALISTLOC=.
5. If trustedcerts.pem does not exist, and the SSL_CERT_DIR and SSLCACERTDIR environment variables are not set, SAS reports an error.

In SAS 9.4, the first maintenance release of SAS 9.4, and the second maintenance release of SAS 9.4, 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.

In the third maintenance release of SAS 9.4, 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_USE_SNI Environment Variable

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.

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 not sent as part of the TLS handshake.

Restrictions: The SSL_USE_SNI environment variable is supported only on UNIX.

Windows always sends SNI to the web servers. Some web servers do not support SNI and fail to connect when the TLS SNI extension is present.

Operating environment: UNIX

Note: When this environment variable is specified, the TLS SNI extension is sent to the web server.

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

UNIX clients and servers now support TLS Server Name Indication (SNI). The client uses SNI in the first message of the TLS handshake (connection setup) to identify the server name that it is trying to connect to.

Default SNI is disabled by default on UNIX. To enable 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.

Note: When SSL_USE_SNI is set, some server connections might fail. For example, PROC IOMOPERATE using an Apache proxy (HTTP CONNECT) might not connect when the SSL_USE_SNI environment variable is set. In that case, disable the environment variable.

See Also

For more information, see "Troubleshooting TLS" on page 125.

Chapter 4

PWENCODE Procedure

Overview: PWENCODE Procedure

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

PROC PWENCODE uses encoding to disguise 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.

PROC PWENCODE is intended to prevent casual, non-malicious viewing of passwords. You should not depend on PROC PWENCODE for all your data security needs; a determined and knowledgeable attacker can decode the encoded passwords. Data should be protected by other security controls such as file system permissions or other access control mechanisms.

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 56
"Example 2: Using an Encoded Password in a SAS Program" on page 57
"Example 3: Saving an Encoded Password to the Paste Buffer" on page 59
"Example 4: Specifying Method= SAS003 to Encode a Password" on page 60

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 56

"Example 2: Using an Encoded Password in a SAS Program" on page 57

"Example 3: Saving an Encoded Password to the Paste Buffer" on page 59

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 57

METHOD=*encoding-method*
specifies the encoding method. Here are the supported values for *encoding-method*:

Table 4.1 *Supported Encoding Methods*

Encoding Method	Description	Supported Data Encryption Algorithm
`sas001`	Uses base64 to encode passwords.	None
`sas002`, which can also be specified as **`sasenc`**	Uses a 32-bit key to encode passwords.	SASProprietary, which is included in SAS software.
`sas003`	Uses a 256-bit key plus 16-bit salt to encode passwords.	AES (Advanced Encryption Standard), which is supported in SAS/SECURE.

Encoding Method	Description	Supported Data Encryption Algorithm
sas004	Uses a 256-bit key plus 64-bit salt value to encode passwords.	AES (Advanced Encryption Standard), which is supported in SAS/SECURE.

Note: SAS/SECURE is a product that enables you to protect data through the use of industry-standard encryption and hashing algorithms. For more information, see "SAS/SECURE Software Availability" in *Encryption in SAS*.

If the METHOD= option is omitted, the default encoding method is used. When the FIPS 140-2 compliance option, -encryptfips, is specified, the encoding method defaults to **sas003**. For all other cases, encoding method **sas002** is the default method used. SAS002 is also the default method used if you specify an invalid method.

Note: The METHOD= option supports the SAS003 and SAS004 values, but only if you have SAS/SECURE.

The SAS003 and the SAS004 encoded passwords consist of a 256-bit key plus a salt value. These salt values are random. Therefore, each time you use PROC PWENCODE to encode the same password, you get a different salt value and therefore a different encoded password.

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

```
proc pwencode in='my password';
run;
```

Program Description

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

Other features: 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.

Program

```
proc pwencode in='my password' method=sas003;
run;
```

Program Description

Encode the password using SAS003.

```
proc pwencode in='my password' 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}08D7B93810D390916F615117D71B2639B4BE

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Chapter 5

Encryption Technologies: Examples

SAS Proprietary Encryption for SAS/SHARE: Example

SAS/SHARE Client

In this example, the NETENCRYPTALGORITHM= option is set to saspropriety 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=saspropriety;
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 be used for encryption of all data that is exchanged with connecting clients.

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

SAS/SECURE for 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 $*
```

TLS for 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 5.1 *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 SSL encryption algorithm
-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

SAS Commands and Arguments	Function
-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 *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

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 5.2 *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

SAS Options, Statements, and Arguments	Client Access Tasks
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 *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

TLS for 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 5.3 *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 SSL encryption algorithm.
-SSLCERTSUBJ "*apex.pc.com*"	Specifies the subject name that is used to search for a certificate from the Microsoft Certificate Store.

SAS Command and Arguments	Function
-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 *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

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 5.4 *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.

TLS on a z/OS Spawner on a SAS/CONNECT Server: Example

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 -SSLCALISTLIC 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 5.5 *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 SSL 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 *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

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 command-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 5.6 *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 *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

TLS for SAS/SHARE on UNIX: Example

Start-up of a Multi-UserSAS/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 5.7 *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 SSL 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 5.8 *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.

TLS for SAS/SHARE on Windows: Examples

Start-up of a Multi-UserSAS/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 5.9 *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 SSL 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 5.10 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.

TLS for SAS/SHARE on z/OS: Example

Start-up of a Multi-UserSAS/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/certificates/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 5.11 SAS Options, Statements, and Arguments for Server Start-Up Tasks

SAS Options, Statements, and Arguments	Server Start-Up Tasks
TCPSEC= _SECURE_	Secures the server
NETENCRALG=*SSL*	Specifies SSL 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 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 5.12 *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.

SSH Tunnel 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 *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

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.

SSH Tunnel for SAS/SHARE: Example

Start-up of a Multi-UserSAS/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 2

Installing and Configuring TLS and Certificates

Chapter 6

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 provided by the requestor of a digital certificate. If the registration authority verifies the requestor's information, the CA can then 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

  You go to a commercial third-party certificate authority, such as VeriSign, Symantec, or Comodoto and purchase a certificate.

- site-signed

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

- self-signed

  You serve as your own certificate authority.

Figure 6.1 *Types of Certificates*

Certificate File Formats

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

- Encodings (also used as extensions)

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

Examples are CA certificate files or an entire certificate chain 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 95, "TLS on Windows: Converting between PEM and DER File Formats for TLS" on page 116.

DER

Distinguished Encoding Rules (.der) is used for binary DER encoded certificates. A PEM file is just a Base64 encoded DER file. OpenSSL can convert these to PEM. 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 95, "TLS on Windows: Converting between PEM and DER File Formats for TLS" on page 116.

PKCS12 .P12

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.

- Common Extensions

CRT

The CRT extension is used for certificates. 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 CER file is recognized by Windows Explorer as a certificate. It is an alternate form of CRT (Microsoft Convention). You can use MS to convert CRT to CER. You can encode both to DER-encoded CER or to base64[PEM]-encoded CER.

Note: If you export a certificate using the Windows export wizard, the CER-formatted file is Base64 Encoded x.509 and is the equivalent to PEM.

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

CSR

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

KEY

The KEY extension is used both for public and private PKCS#8 keys. The keys can be encoded as binary DER or as ASCII PEM.

Overview of Certificate Management Using the SAS Deployment Manager

Starting in the third maintenance release of SAS 9.4, 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 when you are setting up protocols such as TLS. When providing your own signed certificates, you must add the CA root and intermediate certificates to the trusted CA bundle using the SAS Deployment Manager. See "Add Your Certificates to the Trusted CA Bundle" on page 97.

You will also need to add your self-signed certificates to the trusted CA bundle.

Note: In the second maintenance release for SAS 9.4 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 101.

Note: Regardless of your release of SAS 9.4, on Windows, when providing your own signed certificates, you must add the CA root and intermediate certificates to the Windows certificates stores using the Windows Certificates Snap-in. See "Add Your Certificates to the Windows CA Stores" on page 109.

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 additional information about the Mozilla Bundle of Certificates, see Mozilla CA Certificate Store. The current list of included root certificates can be found at Mozilla Included CA Certificate List.

When you use the SAS Deployment Manager task to add custom CA certificates, your 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 and UNIX hosts. After you add files using the SAS Deployment Manager, the three files contain the CA certificates redistributed by SAS from Mozilla as well as the certificates that you just added. The same process occurs when the SAS Deployment Manager task to used to remove the same custom CA certificates. The three files are regenerated. 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 a detailed discussion of these files and the tasks to add and remove certificates.

When the initial installation of SAS Software is complete on UNIX and z/OS platforms, the SSLCALISTLOC option is set by default to point to the trustedcerts.pem file.

Note: THE SSLCALISTLOC option should not be overridden or changed unless directed by technical support. In addition, the trustedcerts.pem file should not be altered by any means other than by using the new SAS Deployment Manager add and remove certificate tasks. If the file is changed by another means, the provided trusted CA bundle might not be supported and maintenance of those changes is not guaranteed.

CAUTION:

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

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 PEM, DER, and PKCS12 file formats. For more details, see "Certificate File Formats" on page 80.
- 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:
  - making sure the certificate has not expired.
  - making 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*.
  - making 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.
  - making sure the certificate has not been revoked.

How SAS Validates Certificates between Clients and Servers

Certificates must be validated between the clients and servers. The following SAS system options, environment variables, or Windows selections are set to provide information about the signer's public key.

- For SAS servers on UNIX or z/OS:

  Certificates can be in one of two locations:

- All certificates must be in one file in PEM format that is referenced by the SSLCALISTLOC= option. The option points to the signer's public key (a file in PEM format). When a server or client receive a certificate, they have to validate the certificate using the signer's public key.

  Normally, a website is required to send all intermediate certificates when they send the server certificate. If they do, the SSLCALISTLOC= just needs to contain the root CA certificate. If it does not, then all intermediate CA certificates need to be put into the file.

  See "SSLCALISTLOC= System Option" on page 30.

- For UNIX, all certificates must be in an OpenSSL CA certificates directory pointed to by the SSL_CERT_DIR or SSLCACERTDIR environment variables.

  SSL_CERT_DIR is the OpenSSL environment variable and SSLCACERTDIR is the SAS environment variable. The layout of this directory is specified by OpenSSL, where the certificates are in PEM format and referenced by their hash values.

  See "SSLCACERTDIR Environment Variable" on page 46 and "SSL_CERT_DIR Environment Variable" on page 48.

- For the SAS servers on Windows:

  The certificate must be in the Windows System truststore.

  Note: Many certificates are already pre-populated on Windows machines.

Chapter 7

Installing and Configuring TLS and Certificates on UNIX

TLS on UNIX: System and Software Requirements

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

- a computer that runs UNIX.
- Internet access and a web browser.
- the TCP/IP communications access method when connecting to SAS/CONNECT or SAS/SHARE servers. See "Access Methods" 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 above. See *SAS® Deployment Wizard and SAS® Deployment Manager 9.4: User's Guide*.
- access to the SAS Deployment Wizard if you plan to create a FIPS-2 capable environment when you install SAS 9.4. See *SAS® Deployment Wizard and SAS® Deployment Manager 9.4: User's Guide*.
- access to the OpenSSL utility at OpenSSL Source. You will need access if you plan to use OpenSSL for the following actions:
  - You plan to apply to become a CA.
  - You plan to build FIPS 140-2 capable OpenSSL.
  - Your site administrator plans to generate a site certificate and private key.

Note: The SAS 9.4 versions of OpenSSL provided by SAS are not FIPS compliant. Refer to "TLS on UNIX: Building FIPS 140-2 Capable OpenSSL " on page 103 for details.

Certificate Locations

SSLCALISTLOC= points to one file that contains a list of root certificates. 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.

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.

  In the third maintenance release of SAS 9.4, the sasv9.cfg file in UNIX 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 now points to **<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem**. The file where the trusted certificates reside is named trustedcacerts.pem. For syntax, see "SSLCALISTLOC= System Option" on page 30.

- On UNIX, all certificates must be in an OpenSSL CA certificates directory pointed to by the SSL_CERT_DIR or SSLCACERTDIR environment variables.

  SSL_CERT_DIR is the OpenSSL environment variable and SSLCACERTDIR is the SAS environment variable. The layout of this directory is specified by OpenSSL where the certificates are in PEM format.

  If you use SSL_CERT_DIR or SSLCACERTDIR, 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 information, see "SSLCACERTDIR Environment Variable" on page 46 and "SSL_CERT_DIR Environment Variable" on page 48.

Preparation for Setting Up Digital Certificates

The process of setting up TLS on UNIX involves setting up Digital Certificates. The 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 trust list of certificates.

In the third maintenance release of SAS 9.4, the process for adding certificates to the trusted CA list has been made easier. You can now use the SAS Deployment Manager at SAS installation to add your existing digital certificates to the Trusted Certificate Bundle of Mozilla certificates (trustedcerts.pem). For more information, see "Add Your Certificates to the Trusted CA Bundle" on page 97.

Here is much of the process that needs to happen to set up digital certificates:

- If your server comes with an instance of OpenSSL, locate that directory. You will need that information to set UNIX 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 89.
- To prepare the certificate(s) to add to the trusted CA bundle, you can, as needed, create a root certificate. See " Setting Up Digital Certificates Using OpenSSL " on page 87.
- 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 93 and "Step 6. Verify Certificates in the Trust Chain Using OpenSSL" on page 95.
- Use the SAS Deployment Manager after SAS installation to add your root and intermediate certificates to the trusted CA bundle and validate the certificates. See "Add Your Certificates to the Trusted CA Bundle" on page 97.

Setting Up Digital Certificates Using OpenSSL

Step 1. Generate a New RSA Private Key and Certificate Signing Request (CSR)

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.

We are using an example to show you how to generate a private key and certificate in PEM format. SAS on UNIX and z/OS platforms requires that you use PEM format.

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 following example, Proton, Inc. is the organization that is applying to become a CA. A certificate request is sent to a certificate authority to get it signed, thereby becoming a CA. After Proton, Inc. becomes a CA, it can serve as a CA for issuing other digital certificates to clients and servers on its network. The certificates generated by the Proton, Inc. CA are considered site-signed certificates.

Note: You can also sign the certificate yourself if you have your own certificate authority or create a self-signed certificate.

Perform the following tasks:

1. In this example, we are using an OpenSSL conf file. You do not have to use this file. You can submit your options with the OpenSSL command or allow OpenSSL to prompt you.

   Edit your existing openssl.cnf file or create an openssl.cnf file. OpenSSL by default looks for a configuration file in **`/usr/lib/ssl/openssl.cnf`**. It is good practice to add **`-config ./openssl.cnf`** to the commands OpenSSL CA or OpenSSL REQ to ensure that OpenSSL is reading the correct file.

   Note: You can find where the openssl.cnf file is located by submitting the OpenSSL command.

   ```
   openssl version -d
   ```

Here is an example of some of the information that can be specified in the openssl.cnf file. Here is a partial file example. 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
organizatio      itName  = optional
```

2. Select the directory where OpenSSL was built.
3. Initialize OpenSSL.

   ```
   $ openssl
   ```

4. Issue the appropriate command to request a digital certificate. In the example below, we are creating an RSA private key and generating a Certificate Signing Request all at once.

   Table 7.1 *OpenSSL Commands for Generating an RSA Private Key and Certificate Signing Request*

Request Certificate for	OpenSSL Command
CA	req -config ./openssl.cnf -new -out ca.csr -newkey rsa:2048 -keyout cakey.pem -nodes -sha256

Request Certificate for	OpenSSL Command
Server	req -config ./openssl.cnf -new -out server.csr -newkey rsa:2048 -keyout serverkey.pem -sha256
Client	req -config ./openssl.cnf -new -out client.csr -newkey rsa:2048 -keyout clientkey.pem -sha256

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

Table 7.2 *Arguments and Values Used in OpenSSL Commands*

OpenSSL Arguments and Values	Functions
req	Requests a certificate
-config ./openssl.cnf	Specifies the storage location for the configuration details for the OpenSSL program
-new	Identifies the request as new
-out ca.csr	Specifies the storage location for the certificate request
-newkey rsa:2048	Generates a new private key along with the certificate request that is 2048 bits in length using the RSA algorithm.
-keyout cakey.pem	Specifies the storage location for the private key
-nodes	Prevents the private key from being encrypted
-sha256	Specifies that the SHA256 hash algorithm be used. Use SHA256 for FIPS 140-2. Without this option, the default is SHA-1.

5. Informational messages are displayed and prompts for additional information appear according to the specific request.

   To accept a default value, press the Enter key. To change a default value, type the appropriate information and press the Enter key.

   Note: Unless the -NODES option is used in the OpenSSL command when creating a digital certificate request, OpenSSL prompts you for a password before allowing access to the private key. 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:

```
OpenSSL> req -config ./openssl.cnf -new -out ca.req -newkey rsa:2048
-keyout privkey.pem -nodes
Using configuration from ./openssl.cnf
```

```
Generating a 2048 bit RSA private key
.........................++++++
.........................................++++++
writing new private key to 'cakey.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [North Carolina]:
Locality Name (city) [Cary]:
Organization Name (company) [Proton Inc.]:
Organizational Unit Name (department) [IDB]:
Common Name (YOUR name) []: proton.com
Email Address []:Joe.Bass@proton.com
Please enter the following 'extra' attributes to be sent with
your certificate request
A challenge password []:
An optional company name []:
OpenSSL>
```

The request for a digital certificate is complete.

Note: For the server, the Common Name must be the name of the computer that the server runs on. In our examples, we are using proton.com.

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 7.3 *OpenSSL Commands for Generating Digital Certificates on UNIX*

Generate Certificate for	OpenSSL Command
CA	x509 -req -in ca.csr -signkey cakey.pem -out ca.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.

Generate Certificate for	OpenSSL Command
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 7.4 *Arguments and Values Used in OpenSSL Commands to Generate a Certificate*

OpenSSL Arguments and Values	Functions
x509	Identifies the certificate display and signing utility
-req	Specifies that a certificate be generated from the request
ca	Identifies the Certificate Authority utility
-config ./openssl.cnf	Specifies the storage location for the configuration details for the OpenSSL utility
-in filename.csr	Specifies the storage location for the input for the certificate request
-out filename.pem	Specifies the storage location for the certificate
-signkey cakey.pem	Specifies the private key that is used to sign the certificate that is generated by the certificate request
-md sha256	Specifies that the SHA256 hash algorithm be used. Use SHA256 for FIPS 140-2. Without this option, the default is SHA-1.

2. Informational messages are displayed and prompts for additional information appear according to the specific request.

   To accept a default value, press the Enter key. To change a default value, type the appropriate information, and press the Enter key.

   Here is a sample of the messaging 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.'
organizationalUnitName:PRINTABLE:'IDB'
commonName            :PRINTABLE:'proton.com'
Certificate is to be certified until Oct 16 17:48:27 2014 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/cakey.pem -out /tmp/enccakey.pem
```

OpenSSL will prompt us 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: In the third maintenance release of SAS 9.4, you can use the SAS Deployment Manager after installation to add to the trusted CA bundle of certificates. For more information, see "Add Your Certificates to the Trusted CA Bundle" on page 97.

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 - ssl.crt)

-----BEGIN CERTIFICATE-----

<PEM encoded certificate>

-----END CERTIFICATE-----

(Your Intermediate CA Certificate(s))

-----BEGIN CERTIFICATE-----

<PEM encoded certificate>

-----END CERTIFICATE-----

(Your Root CA Certificate)

-----BEGIN CERTIFICATE-----

<PEM encoded certificate>

-----END CERTIFICATE-----
```

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

Generally, OpenSSL returns .pem files, 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 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. 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 certificate.der -inform der -text -noout
```

Note: If you are including a digital certificate that is stored in DER format into your certificate chain, you must first convert it to PEM format. For more information, see"Convert between PEM and DER File Formats Using OpenSSL" on page 95.

Step 6. Verify Certificates in the Trust Chain Using OpenSSL

Clients and servers exchange and validate each other's digital certificates. All of the CA certificates that are needed to validate a server certificate compose a trust chain. All CA certificates in a trust chain have to be available for server certificate validation. 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 environment variable or the SSL_CERT_DIR environment variable.

For more information, see "SSLCACERTDIR Environment Variable" on page 46, "SSL_CERT_DIR Environment Variable" on page 48, and "SSLCALISTLOC= System Option" on page 30.

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: In the third maintenance release of SAS 9.4, 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 Your Certificates to the Trusted CA Bundle" on page 97.

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 server.pem -out
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 server.der -out server.pem
```

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

Overview

In the third maintenance release of SAS 9.4, you can add your certificates by using the SAS Deployment Manager after installation.

Note: In the second maintenance release for SAS 9.4 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 101.

The SAS Deployment Manager installation process provides the following:

- a Mozilla bundle of Trusted CA certificates. It is provided in the cacerts.pem file located in new directory **<SASHome>/SASSecurityCertificateFramework/1.1/cacerts**.
- the ability to manage the trusted CA bundle 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 97.

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

cacerts.jks
: contains the Mozilla bundle of CA certificates 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 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 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 Your Certificates to the Trusted CA Bundle

Starting with the third maintenance release for SAS 9.4, 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. If you are using self-signed certificates, the self-signed certificate needs to be added to trusted CA bundle as well. You do this using SAS Deployment Manager.

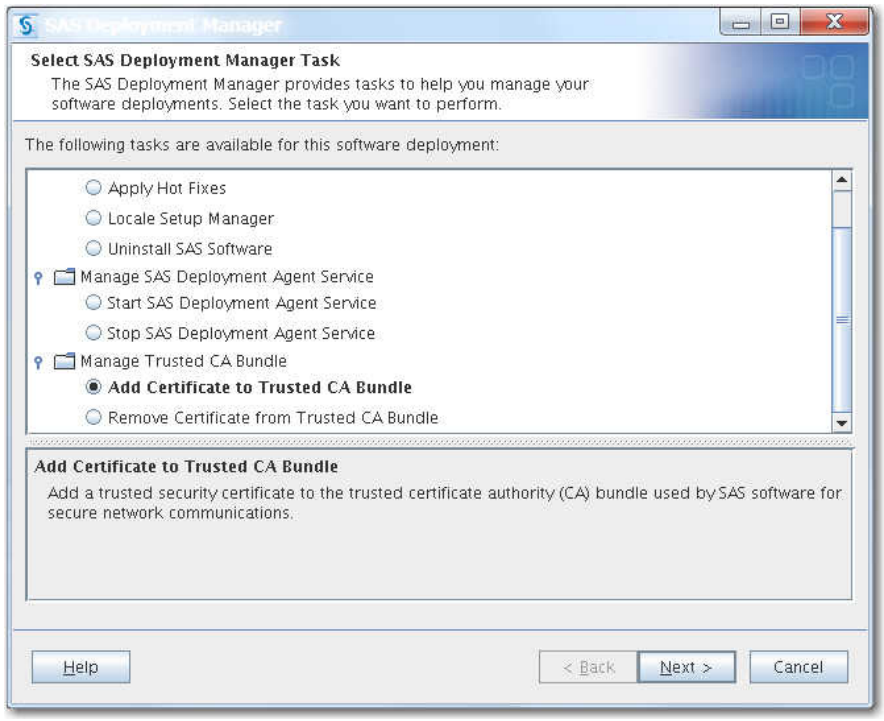
Note: You can add only one certificate at a time with the deployment manager. You must rerun the deployment manager each time you add a certificate to the trusted CA bundle.

Note: 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 Stores" on page 109.

Note: These steps are to be performed for sites that are running the third maintenance for SAS 9.4 and later. If you are running the second maintenance release for SAS 9.4 or earlier, see "Add Your Certificates to the SAS Private JRE" on page 101.

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

1. Log on to the primary middle-tier 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**.

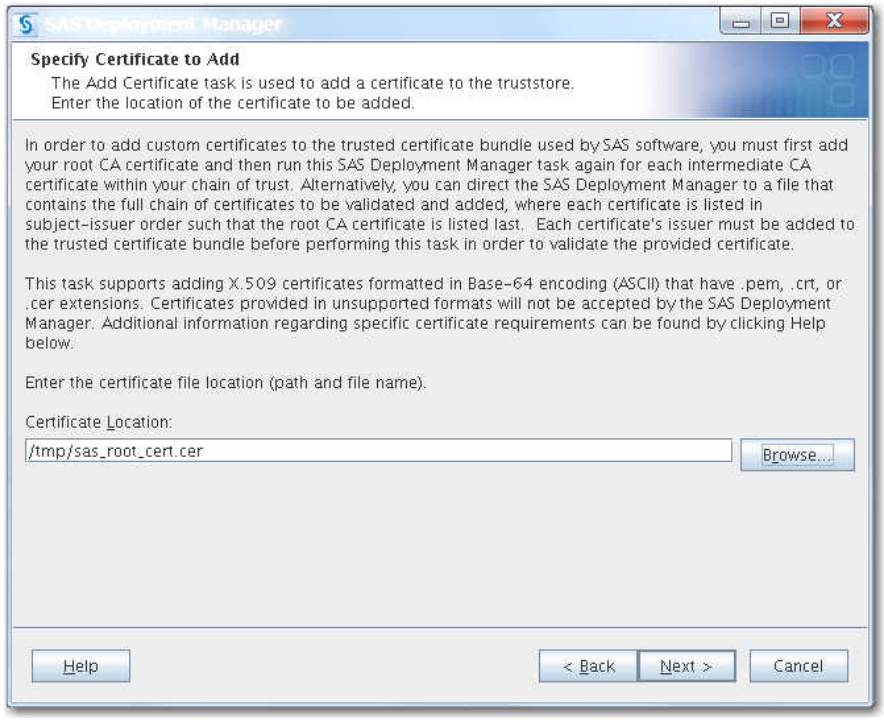


4. Specify the path to your CA root certificate, and click **Next**.

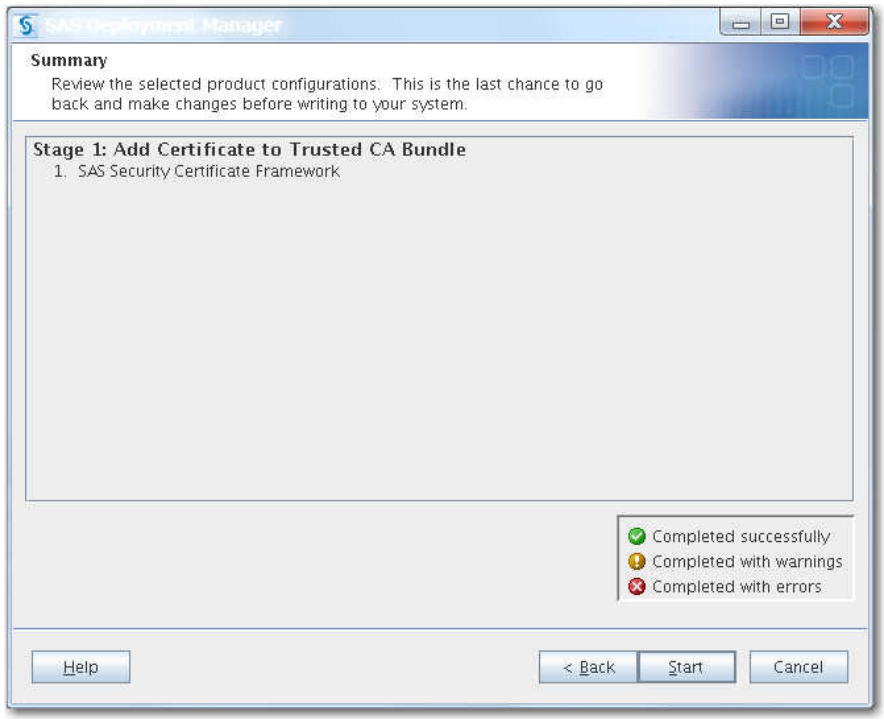
   Note: **Certificate Location** is the location that you established in "Certificate Locations" on page 86.

   The CA root certificate must be in base64 encoding (ASCII) and have a PEM, CRT, or CER file extension. For more information, see "Certificate File Formats" on page 80.

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

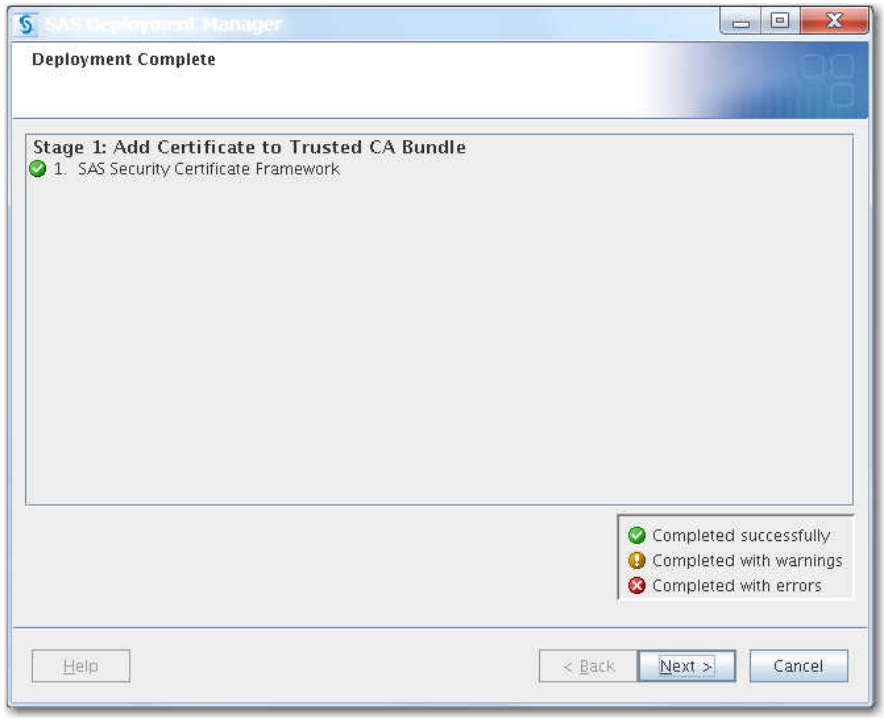


5. On the Summary page, click **Start**.



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 created by the add certificate task are located at **<SASHOME>/InstallMisc/InstallLogs/certframe***.

7. On the Additional Resources page, click **Finish** to close the 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:

   path-to-keytool-command/keytool -list -keystore /_SAS-installation-directory_/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.jks.

   For example:

   /usr/java/jdk1.8.0_45/bin/keytool -list -keystore /opt/SASHome/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.jks

   You should see output similar to the following:

   ```
   intca, Oct 15, 2015, 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, 2015, 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, 2015, 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, 2015, trustedCertEntry,
   ```

10. Repeat steps 1 through 9 on each machine.

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

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

Remove Your Certificates from the Trusted CA Bundle

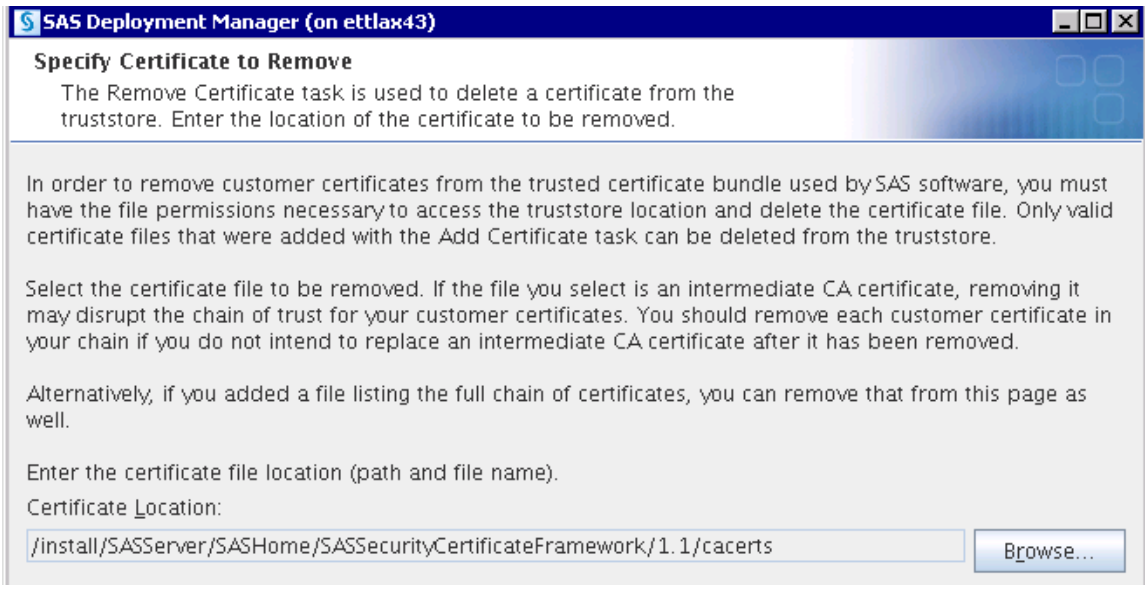
Starting with the third maintenance release for SAS 9.4, you can use SAS Deployment Manager to remove certificates from the trusted CA bundle.

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 the Add Certificate Task 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 the 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 the 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 "Troubleshooting TLS" on page 125 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

Prior to the third maintenance release for SAS 9.4, there is no SAS Deployment Manager task to help you manage any certificates that you provide. 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: If you have any 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 Stores" on page 109.

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

1. Log on to the primary middle-tier machine as the SAS Installer user.
2. Change the directory to where your **keytool** commands reside.

   For example:

   ```
   cd /usr/java/jdk_version/bin
   ```

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

   ```
   ./keytool -importcert -keystore "SAS-installation-directory
   /SASPrivateJavaRuntimeEnvironment/9.4/jre/lib/security/cacerts"
   -storepass changeit -alias myhost -file path-to-keystore.jks
   ```

   Note: The **keytool** command must be on one line. It is shown on more than one line in the preceding code sample for display purposes only.

   TIP For more information about the **keytool** command, see http://docs.oracle.com/javase/7/docs/technotes/tools/windows/keytool.html.

   Table 7.5 *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 86.	**/opt/certs/my_keystore.jks**

4. Repeat step 3 to add your CA intermediate certificates.
5. To verify that your CA root and intermediate certificates were successfully added, enter the following command:

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

   For example:

   ```
   /usr/java/jdk1.8.0_45/bin/keytool -list -keystore
   /opt/SASHome/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.jks
   ```

   You should see output similar to the following:

   ```
   intca, Oct 15, 2015, 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, 2015, 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, 2015, 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, 2015, 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 Stores" on page 109.

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 provides some details of 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 " Setting Up Digital Certificates Using OpenSSL " on page 87.
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 information, see Chapter 2, "SAS System Options for Encryption," on page 23.
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 pointed to by SSLCALISTLOC= or on UNIX in an accessible directory that is pointed to by the SSL_CERT_DIR or SSLCACERTDIR environment variables.
5. The server can also validate the client's certificates. Refer to the previous steps.

TLS on UNIX: Building FIPS 140-2 Capable OpenSSL

SAS ships OpenSSL libraries on UNIX. However, these are not FIPS 140-2 compliant libraries. You must compile a FIPS 140-2 compliant version of OpenSSL and install it. If you plan to build FIPS 140-2 capable OpenSSL for UNIX, access the OpenSSL utility at

OpenSSL Source. Then follow the instructions in OpenSSL FIPS 140-2 Security Policy Version 2.0 to build an OpenSSL FIPS Object Module v2.0.

Note: Different operating systems require the use of different library file extensions. For example, HPUX, 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.

If you are using your own FIPS 140-2 compliant OpenSSL libraries, your system administrator needs to set the environment path variables to pick up this software. 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. When you customize environment variable values, modify the sasenv_local file. 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: Prepend the customized library path in the script that is run before invoking SAS.

Use the SAS Deployment Wizard to configure FIPS after building your libraries. See *SAS® Deployment Wizard and SAS® Deployment Manager 9.4: User's Guide*. Note that SAS system option NETENCRALG= must be set SSL to configure a FIPS 140-2 compliant system.

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 will 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 preceding process to download and compile in accordance with the specific instructions specified by the FIPS standard. See "ENCRYPTFIPS System Option" on page 23 and "FIPS 140-2 Standards Compliance" on page 5.

Chapter 8

Installing and Configuring TLS and Certificates 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.

There is a Microsoft issue that needs attention before configuring ENCRYPTFIPS on Microsoft Windows 2003 servers.

Services that run on a computer that use Microsoft Windows Server 2003 might not recognize Windows environment variable changes. To resolve this issue, perform these steps:

Go to the Microsoft support website http://support.microsoft.com/kb/887693 to find out how to resolve the Windows 2003 Server environment variable issue.

TLS on Windows: Setting Up Digital Certificates

Perform the following tasks to set up digital certificates for TLS:

Step 1. Configure TLS

Complete information about configuring your Windows operating environment for TLS is contained in the Windows installation documentation and at **`www.microsoft.com`**.

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

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

Step 2. Request a Digital Certificate

Methods of Requesting a Digital Certificate

The method of requesting a digital certificate depends on the CA that you use:

- “Request a Digital Certificate from the Microsoft Certificate Authority” on page 106
- “Request a Digital Certificate from a Certificate Authority That Is Not Microsoft” on page 107

Request a Digital Certificate from the Microsoft Certificate Authority

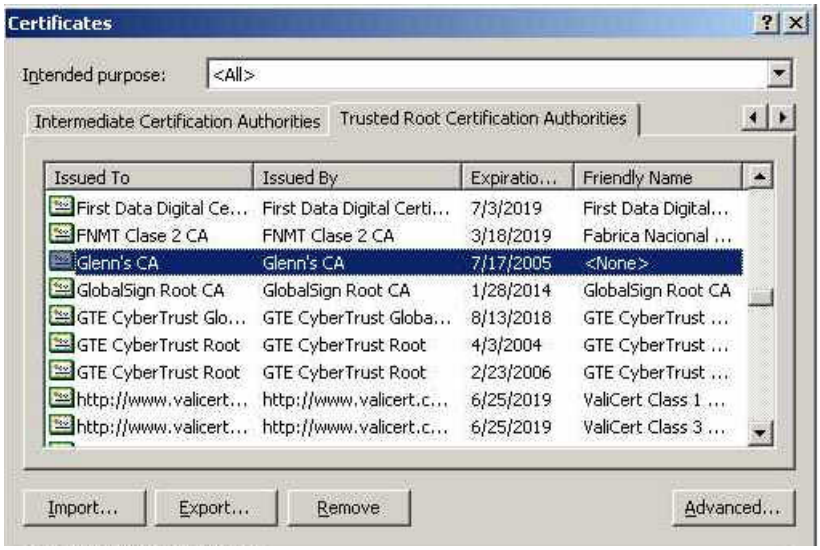
Perform the following tasks to request digital certificates that are issued by the Microsoft Certificate Authority:

1. System administrator: If you are running your own CA, use Microsoft Certificate Services to create an active Certificate Authority (CA).
2. User:
   a. 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.
   b. Select a digital certificate type.
   c. Select security options.

d. Submit the request to an active CA that is configured to issue the digital certificate.

After the CA issues the requested digital certificate, the digital certificate is automatically installed in the Certificate Store. The installed digital certificate is highlighted, as shown in the following display:

Figure 8.1 *Digital Certificate Installation in the Certificate Store*



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

Users should perform the following tasks to request digital certificates that are not issued by the Microsoft CA:

1. Request a digital certificate from a CA.
2. Import the digital certificate to a Certificate Store by using the Certificate Manager Import wizard application from a web browser.

   A digital certificate can be generated by using the Certificate Request wizard or any third-party application that generates digital certificates.

   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 "TLS on Windows: Converting between PEM and DER File Formats for TLS" on page 116.

For details about importing existing digital certificates, see "Import a Digital Certificate to a Certificate Store" on page 107.

Import a Digital Certificate to a Certificate Store

Digital certificates that were issued by a Certificate Authority that is not Microsoft can be imported to an appropriate Certificate Store as follows:

Certificate Type	Certificate Storage Location
Client	Personal Certificate Store or Machine Certificate Store
Server	Personal Certificate Store or Machine Certificate Store

Certificate Type	Certificate Storage Location
CA (self-signed)	Trusted Root Certificate Authorities

Perform the following tasks to import a digital certificate to your Personal Certificate Store:

1. Access the Certificate Manager Import wizard application from your web browser. From the `Tools` drop-down menu, select **Internet Options**.

   Then select the **Content** tab, and click **Certificates**.

   Specify the digital certificate to import to the Personal Certificate Store by selecting the **Personal** tab in the **Certificates** window, as shown in the following display:

   Figure 8.2 *Digital Certificate Selections for a Personal Certificate Store*

2. Click **Import** and follow the instructions to import digital certificates.

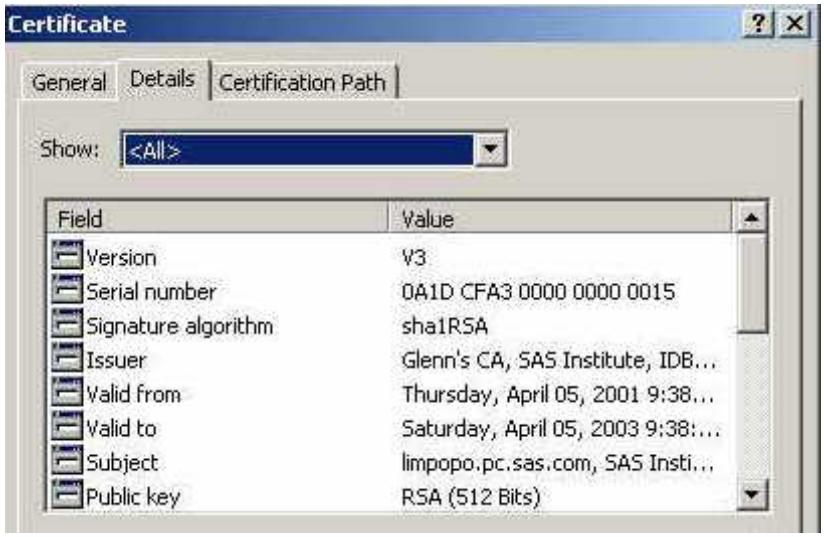
   Repeat this task in order to import the necessary digital certificates for the CA, the server, and the client, as appropriate.

   Note: You can now import a digital certificate to the Machine Certificate store as well as to a Personal Certificate store.

3. After you have completed the selections for your personal Certificate Store, select the appropriate tab to view your selections.

4. To view the details about a digital certificate, select the digital certificate and click **View**. Typical results are shown in the following display:

Figure 8.3 *Digital Certificate Details Tab*



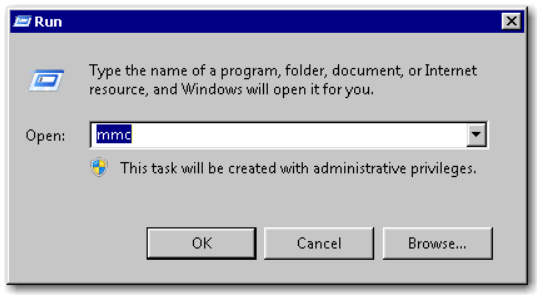
Add Your Certificates to the Windows CA Stores

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.

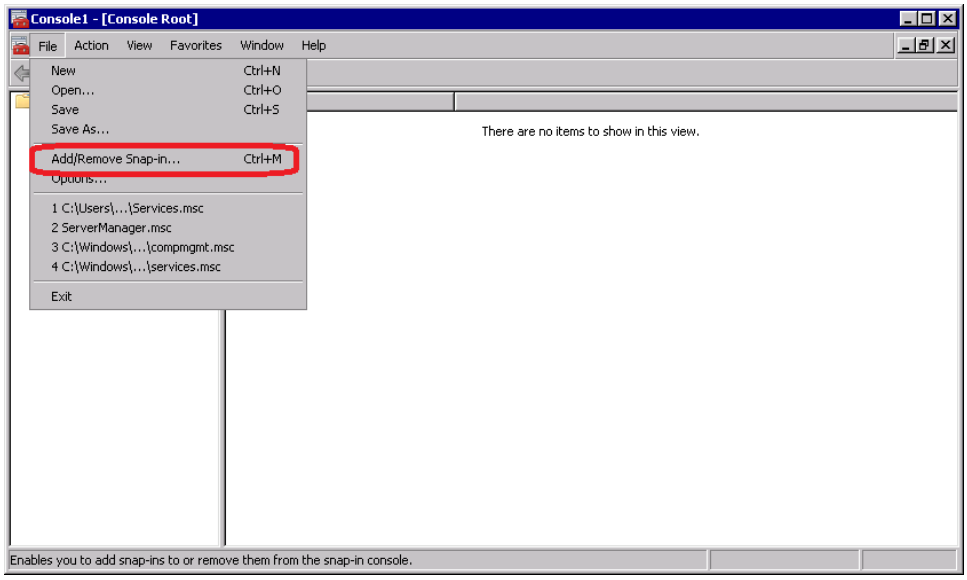
Note: If you have not already done so, you must add your CA root and intermediate certificates to the trusted CA bundle or to the SAS Private JRE. For more information, see "Add Your Certificates to the Trusted CA Bundle" on page 97 or "Add Your Certificates to the SAS Private JRE" on page 101.

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

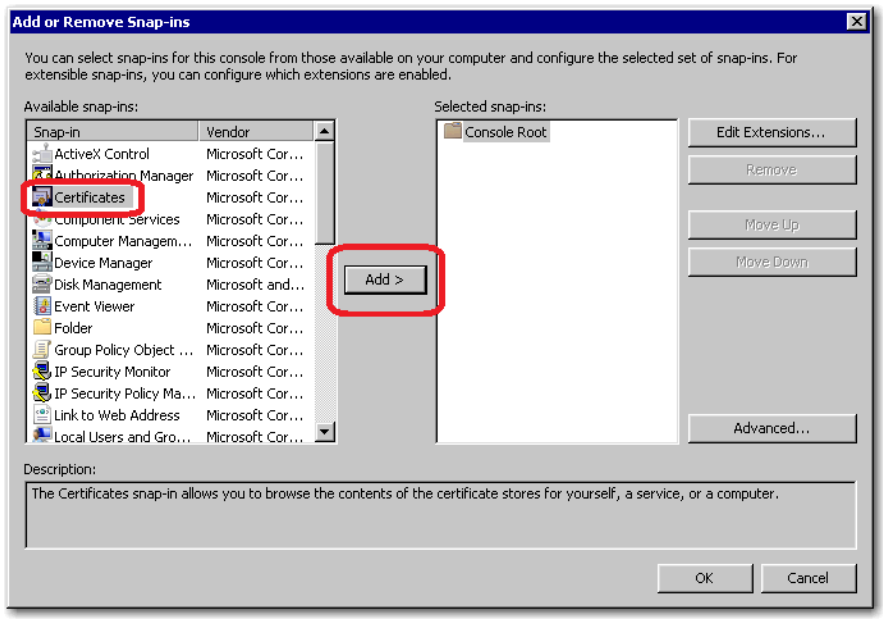
1. Click the Windows **Start** button, select **Run**, enter **`mmc`**, and click **OK**.



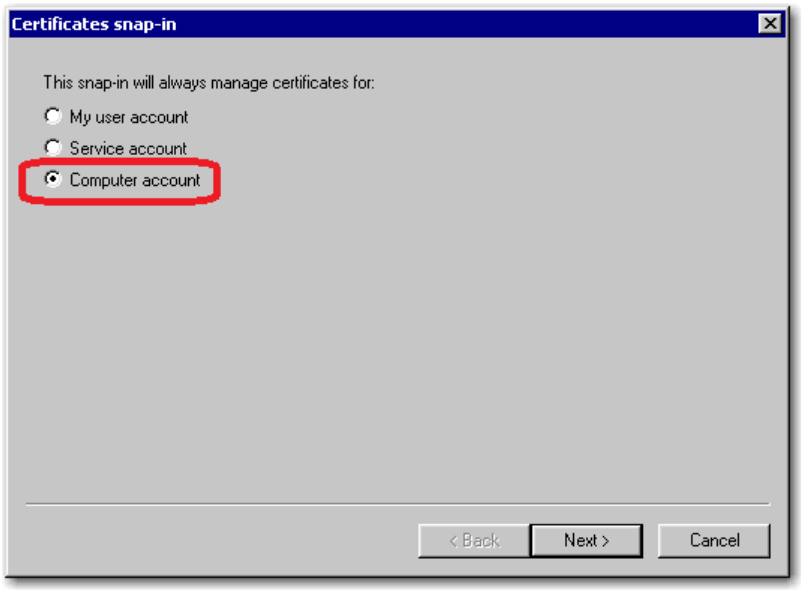
2. In the Console window, select **File** ⇨ **Add/Remove Snap-in**.



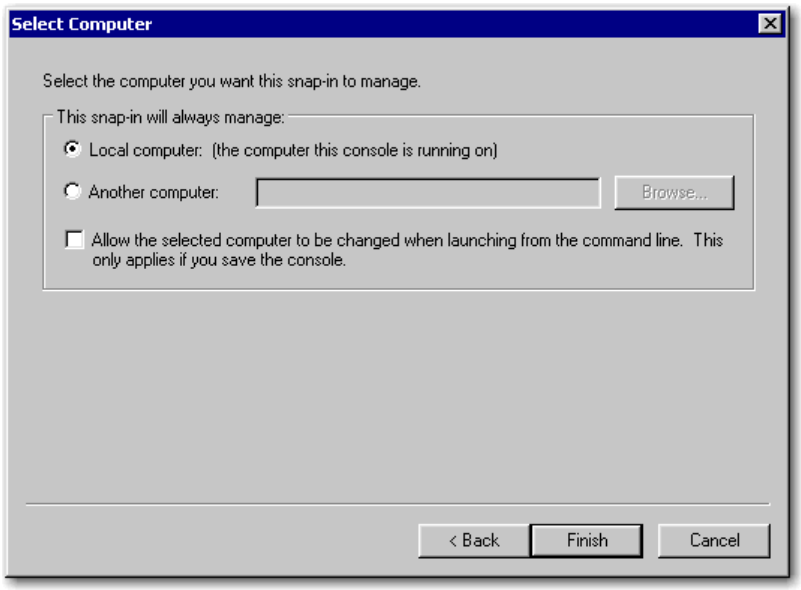
3. Select **Certificates** from the list of available snap-ins, and click **Add**.



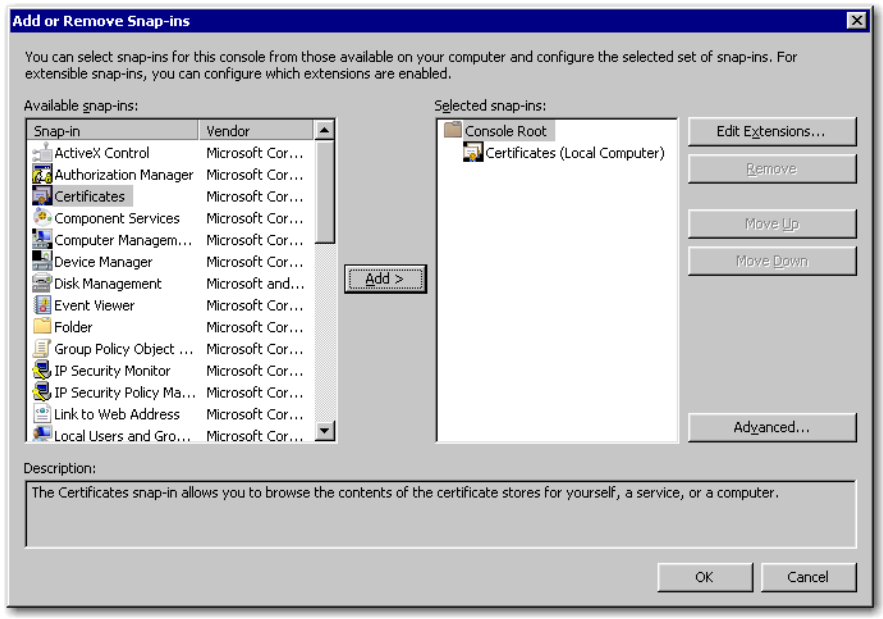
4. In the dialog box that appears, select **Computer account**, and click **Next**.



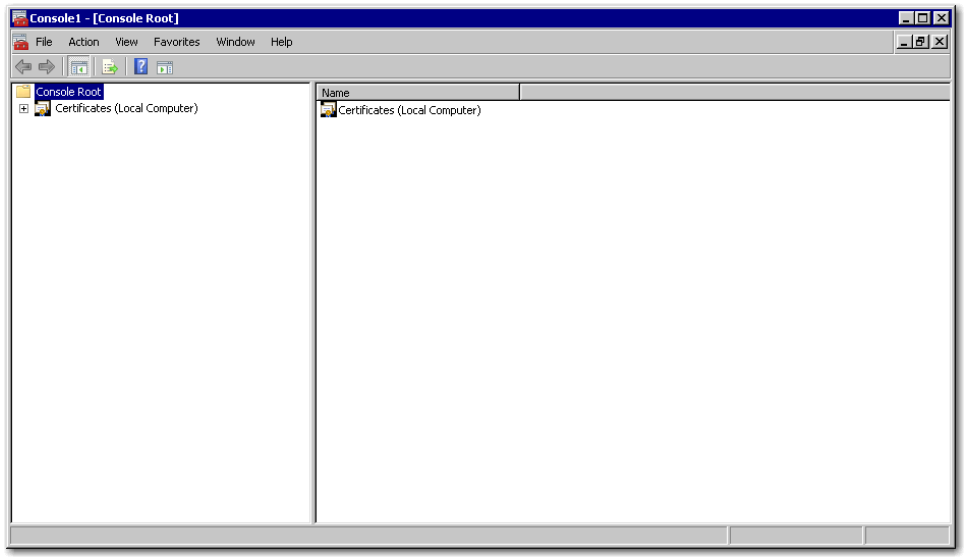
5. In the dialog box that appears, click **Finish**.



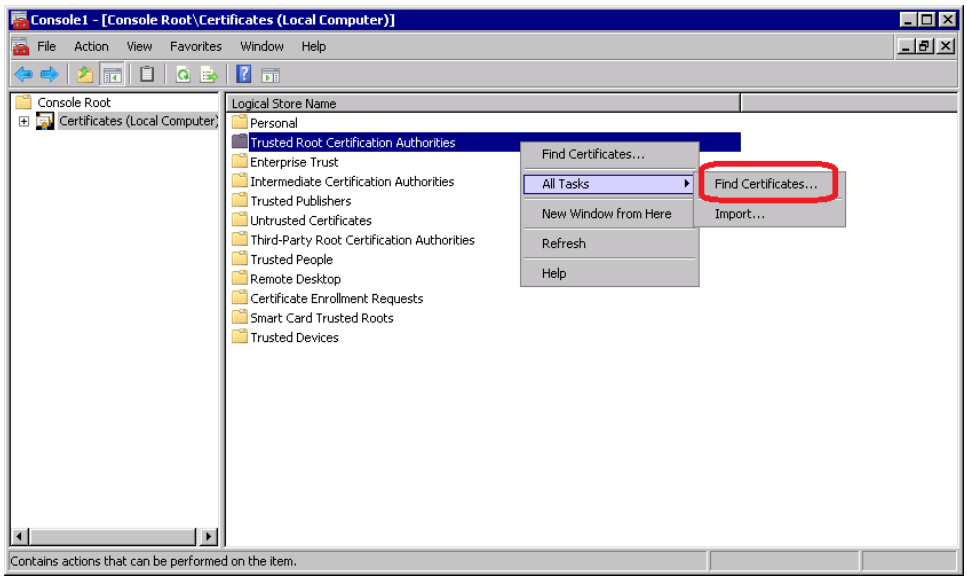
6. In the dialog box that appears, click **OK**.



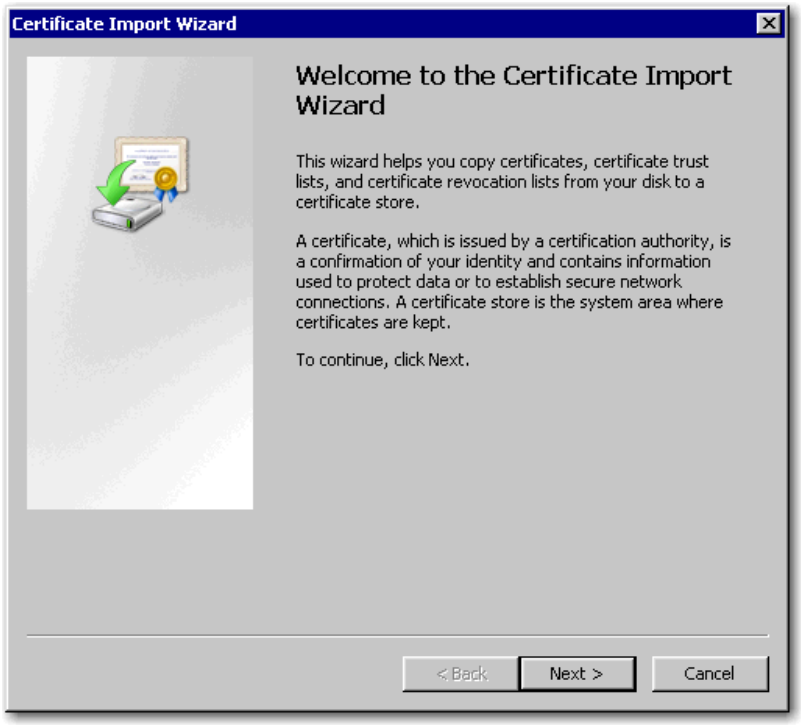
7. In the Console window, expand **Certificates (Local Computer)** on the left.



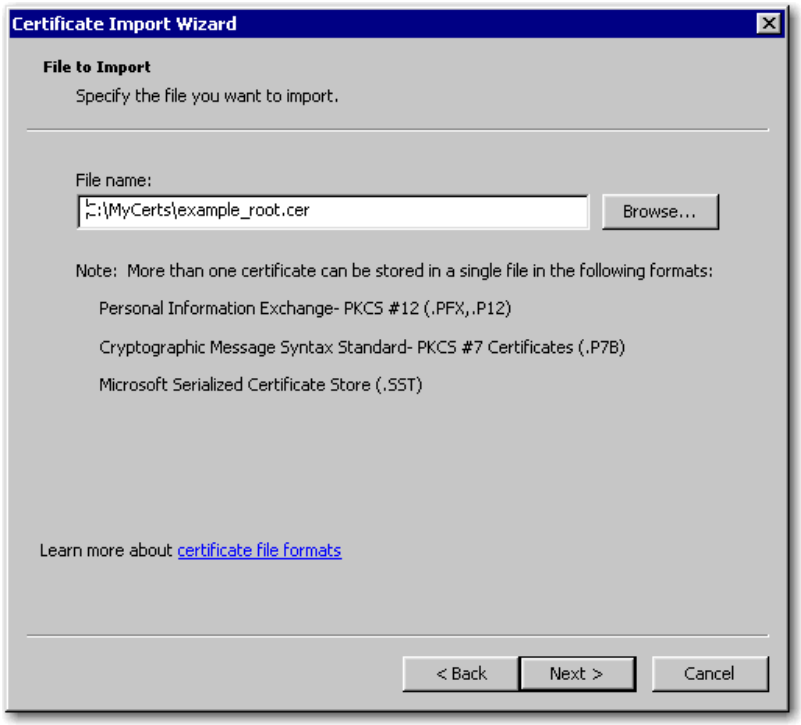
8. Right-click **Trusted Root Certification Authorities**, and select **All Tasks** ⇨ **Import**.



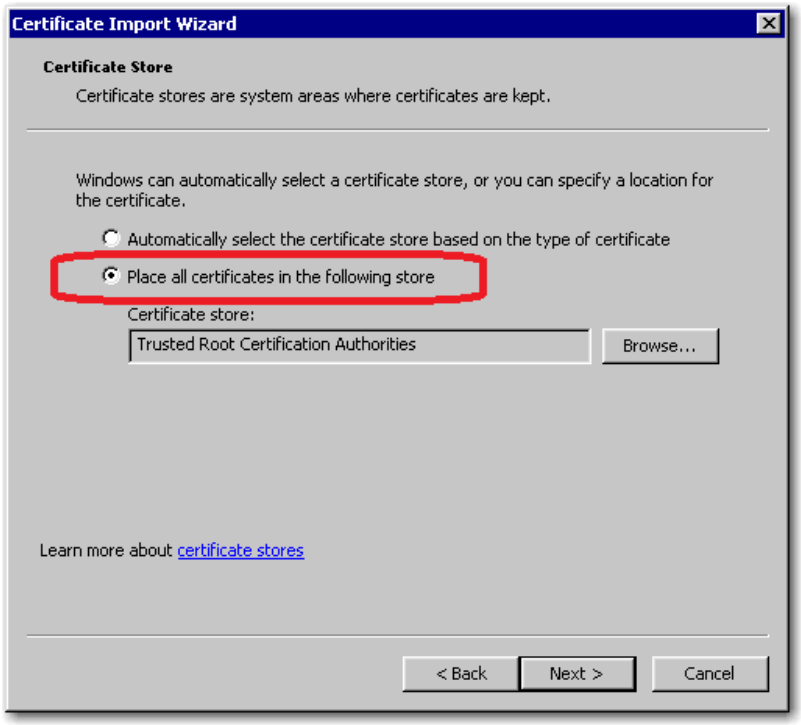
9. On the Certificate Import Wizard page, click **Next**.



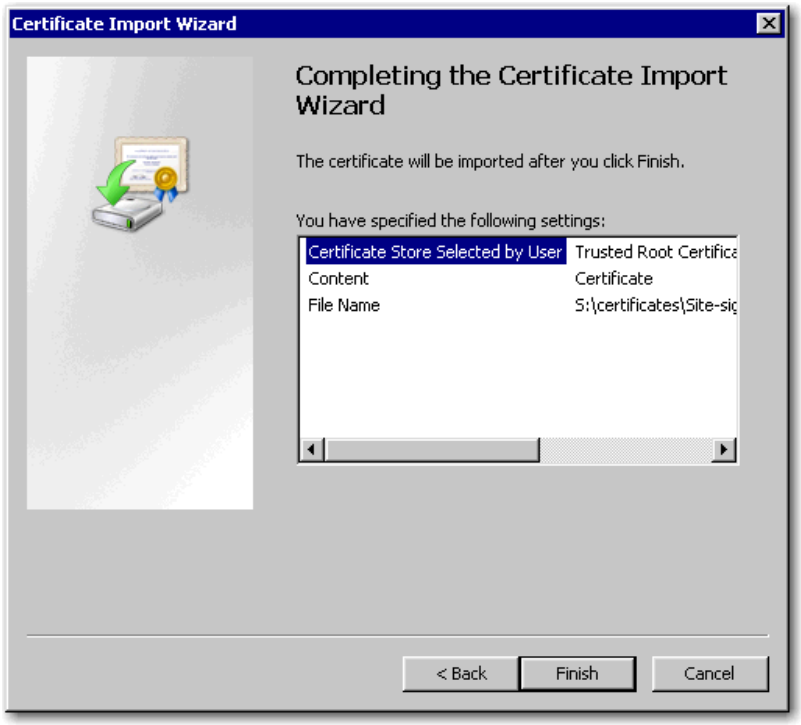
10. On the second wizard page, click **Browse**, navigate to the location that contains your CA root certificate and any intermediate certificates, and select the appropriate certificate. Click **Next**.



11. Make sure that **Place all certificates in the following store** is selected, and click **Next**.

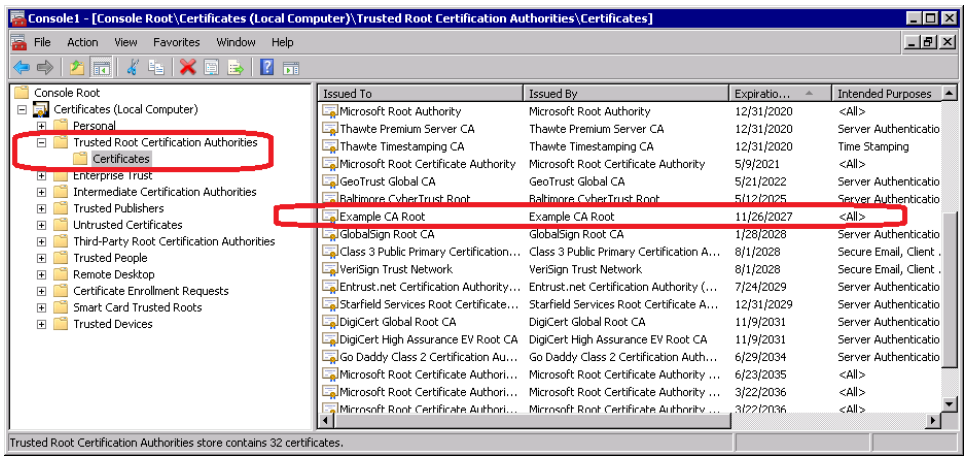


12. Click **Finish**.



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. Repeat steps 8 through 14 for any CA intermediate certificates.

16. Repeat steps 1 through 15 on any additional Windows machines in your SAS deployment.

TLS on Windows: Converting between PEM and DER File Formats for TLS

By default, OpenSSL files are created in Privacy Enhanced Mail (PEM) format. SSL 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.

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 -out
server.der
```

Note: Files with the .cert, .cer, or .crt extensions are recognized by Windows as a certificate. For more information, see "Certificate File Formats" on page 80.

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 96.

TLS on Windows: Validating Certificates between Clients and Servers

Clients and servers exchange and validate each other's digital certificates. The following 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 "TLS on Windows: Setting Up Digital Certificates " on page 106.
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 Chapter 2, "SAS System Options for Encryption," on page 23.

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

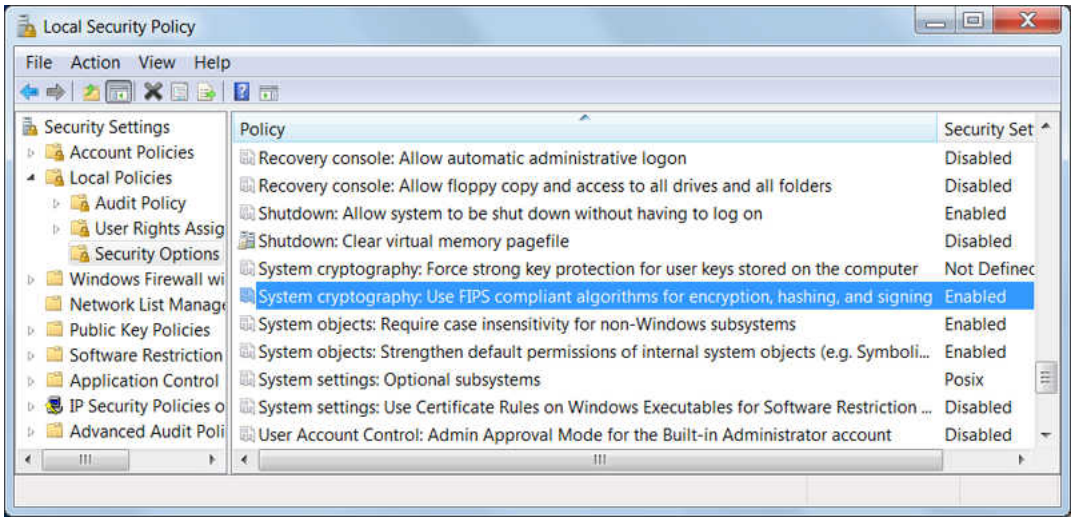
TLS on Windows: FIPS 140-2 Capable OpenSSL

For Windows, the TLS version shipped with SAS is FIPS 140-2 compliant. To put the library into FIPS 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** and make sure that the item is enabled.

Figure 8.4 *FIPS Encryption Enabled on Windows*



Chapter 9

Installing and Configuring TLS and Certificates on z/OS

TLS on z/OS: System and Software Requirements

The system and software requirements for using TLS on z/OS operating environments are as follows:

- a computer that runs z/OS.
- the TCP/IP communications access method.
- if you are planning to use a computer that runs z/OS as the CA, access to the RACDCERT command on z/OS.
- 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.

TLS on z/OS: Setting Up Digital Certificates

Perform these tasks to set up and use TLS:

Step 1. Authorize Access to the RACDCERT Command

To use z/OS as your trusted Certificate Authority (CA), you must authorize access to the RACDCERT command in order to set up the CA and to create and sign certificates. Authorize the trusted administrator using CONTROL access to these profiles in the FACILITY class:

- IRR.DIGTCERT.ADD
- IRR.DIGTCERT.DELETE
- IRR.DIGTCERT.EXPORT
- IRR.DIGTCERT.GENCERT
- IRR.DIGTCERT.LIST

The following sites provide information about alternative CAs:

- For VeriSign, see **www.verisign.com**
- For Thawte, see **www.thawte.com**

Step 2. Create the Digital Certificate for the CA

The tasks that you perform to generate a digital certificate for the CA, the server, and the client are similar. However, the values that you specify are different.

In this example, Proton, Inc. is the organization that is applying to become a CA 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:

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 MVS data sets.

```
cp //ca.cert ca.cert
```

4. Convert the certificate file to ASCII format

   Note: TLS PEM format certificate files must be converted to ASCII format. The OpenSSL library code in 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.

The location of the CA digital certificate is specified using the SSLCALISTLOC= 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 MVS 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.

Specify a server or client PKCS #12 package using the SSLPKCS12LOC= system option. Specify the password for the package using the SSLPKCS12PASS= 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

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 *trust list*.

Note: Starting in the third maintenance release of SAS, you can use the SAS Deployment Manager after Installation to add to the Trusted CA Bundle of Certificates.

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. They can be added to the file in any order. To manually create a new trust list, use the following template:

```
(Your Server Certificate - ssl.crt)

-----BEGIN CERTIFICATE-----

<PEM encoded certificate>

-----END CERTIFICATE-----

(Your Intermediate CA Certificate(s))

-----BEGIN CERTIFICATE-----
```

```
<PEM encoded certificate>

-----END CERTIFICATE-----

(Your Root CA Certificate)

-----BEGIN CERTIFICATE-----

<PEM encoded certificate>

-----END CERTIFICATE-----
```

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, 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 certificate.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 95.

Step 5. Verify Certificates in the Trust Chain Using OpenSSL

Clients and servers exchange and validate each other's digital certificates. All of the CA certificates that are needed to validate a server certificate compose a trust chain. All CA certificates in a trust chain have to be available for server certificate validation. The certificates are combined into one file pointed to by the SSLCALISTLOC= option. "SSLCALISTLOC= System Option" on page 30.

You can use the following OpenSSL command to verify certificates signed by a recognized certificate authority (CA):

```
openssl verify <your-certificate-file>
```

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.

```
openssl verify -verbose -CAfile <your-CA_file>.pem <your-server-cert>.pem
```

Note: In the third release of SAS 9.4, you can use the SAS Deployment Manager after installation to add your trust chain. The SAS Deployment Manager also validates those certificates.

Step 6. End OpenSSL

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

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

For an overview of using the SAS Deployment Manager to add certificates to the Trusted CA Bundle, see "Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager" on page 96. For specific details about using the SAS Deployment Manager to add certificates to the Trusted CA bundle, see the *SAS Deployment Wizard and SAS Deployment Manager 9.4: User's Guide*.

Chapter 10

Troubleshooting

Troubleshooting TLS

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 (log4SAS) 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 “TLS on Windows: FIPS 140-2 Capable OpenSSL ” 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, make sure that the USE_SSL_SNI environment variable is not set.
- 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 SSL 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 Environment Variable" on page 46.
- 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 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 Environment Variable" on page 46.

Failed to Find the Following Issuer of this Certificate in Truststore

This message is displayed when using the SAS Deployment Manager (SDM) to add 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 SDM 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 SDM to add 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 File Formats" on page 80.

Recommended Reading

Here is the recommended reading list for this title:

- *SAS/CONNECT User's Guide*
- *SAS/SHARE User's Guide*
- *SAS Statements: Reference*
- *SAS System Options: Reference*
- *Base SAS Procedures Guide*
- *SAS Language Reference: Concepts*
- *SAS XML LIBNAME Engine: User's Guide*
- *SAS 9.4 Intelligence Platform: Security Administration Guide*
- SAS Companion that is specific to your operating environment
- *SAS Deployment Wizard and SAS Deployment Manager 9.4: User's Guide*
- *Configuration Guide for SAS 9.4 Foundation for Microsoft Windows for x64*
- *Configuration Guide for SAS 9.4 Foundation for Microsoft Windows*
- *Configuration Guide for SAS 9.4 Foundation for z/OS*
- *Configuration Guide for SAS 9.4 Foundation for UNIX Environments*

For a complete list of SAS publications, go to sas.com/store/books. If you have questions about which titles you need, please contact a SAS Representative:

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: sas.com/store/books

Glossary

authentication
See client authentication.

block cipher
a type of encryption algorithm that divides a message into blocks and encrypts each block. *See also* stream cipher.

Certificate Revocation List (CRL)
a list of revoked digital certificates. CRLs are published by Certification Authorities (CAs), and a CRL contains only the revoked digital certificates that were issued by a specific CA.

Certification Authority
a commercial or private organization that provides security services to the e-commerce market. A Certification Authority creates and maintains digital certificates, which help to preserve the confidentiality of an identity. Microsoft, VeriSign, and Thawte are examples of commercial Certification Authorities.

ciphertext
unintelligible data. *See also* encryption.

client authentication (authentication)
the process of verifying the identity of a person or process for security purposes.

credentials
evidence that is submitted to support a claim of identity (for example, a user ID and password) or privilege (for example, a passphrase or encryption key).

CRL
See Certificate Revocation List.

cryptography
the science of encoding and decoding information to protect its confidentiality. *See also* encryption.

data security technology
a set of software features that protect data that is exchanged in client/server data transfers across a network.

DER
See Distinguished Encoding Rules.

digital certificate
an electronic document that binds a public key to an individual or an organization. A digital certificate usually contains a public key, a user's name, an expiration date, and the name of a Certification Authority.

digital signature
a digital code that is appended to a message. The digital signature is used to verify to a recipient that the message was sent by a particular business, organization, or individual, and that the message has not been changed en route. The message can be any kind of file that is transmitted electronically.

Distinguished Encoding Rules (DER)
a format that is used for creating SSL files in Windows operating environments.

encryption
the conversion of data by the use of algorithms or other means into an unintelligible form in order to secure data (for example, passwords) in transmission and in storage.

PEM
See Privacy Enhanced Mail.

PKCS #12
See Public Key Cryptography Standard #12.

plaintext
information that a sender wishes to transmit to a receiver, and that is used as input to an algorithm for the purpose of encryption. *See also* ciphertext.

port forwarding
See tunneling.

Privacy Enhanced Mail (PEM)
a format that is used for creating OpenSSL files.

private key
a number that is known only to its owner. The owner uses the private key to read (decrypt) an encrypted message. *See also* public key, encryption.

public key
a number that is associated with a specific entity such as an individual or an organization. A public key can be known by everyone who needs to have trusted interactions with that entity. A public key is always associated with a single private key, and can be used to verify digital signatures that were generated using that private key.

Public Key Cryptography Standard #12 (PKCS #12)
a personal information exchange syntax standard. It defines a file format that is used to store private keys with accompanying public-key certificates. *See also* Secure Sockets Layer.

public-key cryptography
the science that uses public and private key pairs to protect confidential information. The public key can be known by anyone. The private key is known only to the owner

of the key pair. 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.

SASProprietary algorithm
a fixed encoding algorithm that is included with Base SAS software. The SASProprietary algorithm requires no additional SAS product licenses. It provides a medium level of security.

Secure Shell (SSH)
a network 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.

Secure Sockets Layer (SSL)
an encryption protocol for securely communicating across the Internet. SSL uses encryption algorithms RC2, RC4, DES, TripleDES, and AES.

SSH
See Secure Shell.

SSL
See Secure Sockets Layer.

stream cipher
a type of encryption algorithm that encrypts data one byte at a time. *See also* block cipher.

TLS
See Transport Layer Security.

Transport Layer Security (TLS)
the successor to Secure Sockets Layer (SSL), a cryptographic protocol that is designed to provide communication security over the Internet. TLS uses asymmetric cryptography for authentication and confidentiality of the key exchange, symmetric encryption for data/message confidentiality, and message authentication codes for message integrity. Several versions of the protocols are in widespread use in applications such as web browsing, electronic mail, Internet faxing, instant messaging and voice-over-IP (VoIP). *See also* Secure Sockets Layer.

trust list
a file created by a user that contains the digital certificates for Certification Authorities, if more than one Certification Authority is used.

tunneling (port forwarding)
a secure, encrypted connection between the SSH client, which runs on the same computer as a SAS client, and an SSH server, which runs on the same computer as a SAS server. The SSH client and 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. *See also* Secure Shell.

Index

F

I

K

L

M

N

O

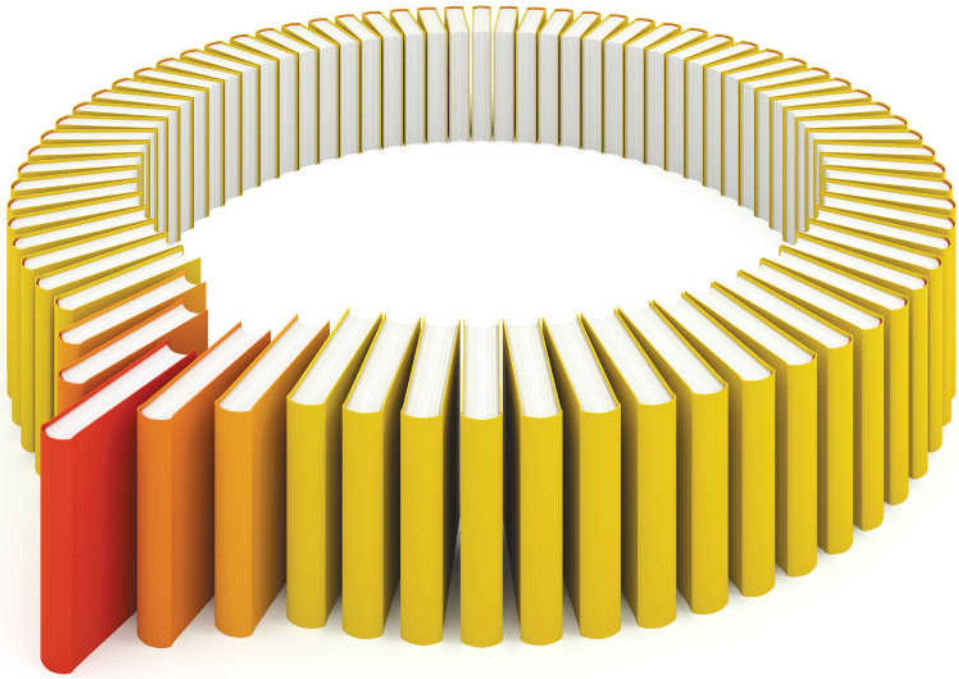
P

R

S

W

Z



Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore
for additional books and resources.