Encryption in SAS® 9.4, Sixth Edition
Contents

About This Book ................................................................. vii
What's New in Encryption in SAS 9.4 ................................. xiii
Accessibility ................................................................. xix

PART 1  Encryption in SAS 9.4  1

Chapter 1 • Technologies for Encryption .............................. 3
  Overview of Encryption ...................................................... 3
  FIPS 140-2 Standards Compliance ................................. 5
  Providers of Encryption .................................................. 6
  Encryption Algorithms ................................................... 18
  Comparison of Encryption Technologies .................. 20
  Encryption: Implementation ........................................ 20
  Encryption: SAS Logging Facility .................................... 21
  Encrypting ODS Generated PDF Files ........................ 22

Chapter 2 • SAS System Options for Encryption .................. 25
  Dictionary ................................................................. 25

Chapter 3 • SAS Environment Variables ............................... 53
  Overview of Environment Variables ........................................ 53
  Dictionary ................................................................. 53

Chapter 4 • PWENCODE Procedure .................................... 59
  Overview: PWENCODE Procedure .................................. 59
  Concepts: PWENCODE Procedure .................................. 59
  Syntax: PWENCODE Procedure ...................................... 61
  Examples: PWENCODE Procedure .................................... 63

Chapter 5 • Encryption Technologies: Examples ................... 69
  SAS Proprietary Encryption for SAS/SHARE: Example ............... 70
  SAS/SECURE for SAS/CONNECT: Example .......................... 70
  TLS for a SAS/CONNECT UNIX Spawner: Example ............... 71
  TLS for a SAS/CONNECT Windows Spawner: Example .......... 73
  TLS on a z/OS Spawner on a SAS/CONNECT Server: Example .... 75
  TLS for SAS/SHARE on UNIX: Example ............................ 77
  TLS for SAS/SHARE on Windows: Examples ..................... 79
  TLS for SAS/SHARE on z/OS: Example ...................................... 80
  SSH Tunnel for SAS/CONNECT: Example ........................ 81
  SSH Tunnel for SAS/SHARE: Example ...................................... 82

PART 2  Installing and Configuring TLS and Certificates 83
Chapter 6 • Certificates Explained .................................................. 85
   About Certificates ........................................................................ 85
   Certificate File Formats ............................................................. 86
   Overview of CA Certificate Management Using the SAS Deployment Manager ........................................................................ 88
   Best Practices When Generating Certificates ................................ 89
   Certificate Implementation: How TLS Client and Servers Negotiate ......................................................................................... 90
   How SAS Validates Certificates between Clients and Servers ........ 91

Chapter 7 • Installing and Configuring TLS and Certificates on UNIX .............................................................. 93
   TLS on UNIX: System and Software Requirements ....................... 93
   Certificate Locations ....................................................................... 94
   Preparation for Setting Up Digital Certificates ............................. 95
   Generate Digital Certificates Using OpenSSL .............................. 95
   Convert between PEM and DER File Formats Using OpenSSL ........ 104
   Convert PEM Files to a PFX File Using OpenSSL ........................... 104
   Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager ................................................................. 104
   Add Your Certificates to the SAS Private JRE ............................... 110
   How Clients and Servers Validate Certificates .............................. 112
   Building FIPS 140-2 Capable OpenSSL on UNIX ........................... 113

Chapter 8 • Installing and Configuring TLS and Certificates on Windows .......................................................... 115
   Steps to Add Certificates for TLS to Windows .............................. 115
   TLS on Windows: System and Software Requirements ................ 116
   Configure TLS and Request Digital Certificates on Windows ........ 116
   Add Your Certificates to the Windows CA Store ........................... 119
   Import the Client Certificate into the Windows Personal Machine Store ......................................................................................... 120
   Convert between PEM and DER File Formats for TLS .................. 121
   Use the SAS Deployment Manager to Manage Certificates in the
      Trusted CA Bundle ...................................................................... 122
   Validating Certificates between Clients and Servers .................... 122
   Configure FIPS 140-2 Capable TLS on Windows ............................ 122

Chapter 9 • Installing and Configuring TLS and Certificates on z/OS .............................................................. 125
   TLS on z/OS: System and Software Requirements ....................... 125
   TLS on z/OS: Setting Up Digital Certificates ............................... 125
   Use the SAS Deployment Manager to Manage Certificates in the
      Trusted CA Bundle ...................................................................... 130

PART 3  Troubleshooting ................................................... 131

Chapter 10 • Troubleshooting .......................................................... 133
   ERROR: Unable to load extension: (tkessl) ................................. 133
   ERROR: SSL provider not in FIPS mode ....................................... 133
   ERROR: HTTP proxy handshake failed ....................................... 133
   ERROR: Cannot load SSL Support ............................................. 134
   ERROR: 14090086:SSL routines:
      SSL3_GET_SERVER_CERTIFICATE: certificate verify failed .......... 134
   Failed to Find the Following Issuer of this Certificate in Truststore ......................................................................................... 134
   Verify that the File Contains Certificates in the Proper Encoding ......................................................................................... 135
   ERROR: Cannot negotiate encryption algorithm ............................ 135
   SEVERE: java.security.cert.CertPathValidatorException: signature check failed ........................ 136
PART 4  Appendices  137

Appendix 1 • Environment Variables Used Before SAS 9.4M5  ........................................... 139
  Overview ................................................................................................................................. 139
  Dictionary .............................................................................................................................. 139

Appendix 2 • Supported Ciphers for the SSLMODE= Option  .............................................. 147

  Recommended Reading .......................................................................................................... 153
  Glossary ................................................................................................................................. 155
  Index ..................................................................................................................................... 159
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** <unique-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:

```sas
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
elementary, 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:

```sas
inFILE file-specification obs = 100;
libname libref 'SAS-library';
```
What's New in Encryption in SAS 9.4

Overview

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

- TLS version 1.2 is the protocol supported in SAS 9.4. Earlier versions of TLS and SSL are insecure.

- 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.

- SAS/SECURE is included with Base SAS, instead of being licensed and ordered separately.

- Encoding type SAS004 is added to (uses AES encryption with 64-bit salt) provide increased security for stored passwords.

- Increased security is provided for SAS data 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.

The following changes and enhancements were introduced in SAS maintenance releases.

- In SAS 9.4M1, 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.

- In SAS 9.4M1, environment variables SSL_CERT_DIR and SSLCACERTDIR were introduced. They are used to point to the location of certificates. These environment variables are supported on UNIX and support logging. The default location is specified by the SSLCALISTLOC= system option.

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

- Starting in SAS 9.4M1, 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 SAS 9.4M3, CA certificates are located in the trustedcerts.pem file for UNIX and z/OS. The SSLCALISTLOC= option on UNIX and z/OS points to the trustedcerts.pem file by default.

• In SAS 9.4M3, the SAS_SSL_MIN_PROTOCOL= environment variable was added and is supported on UNIX, Windows, and z/OS. This environment variable should be set to support TLS 1.2. Earlier versions of TLS are insecure.

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

• In SAS 9.4M3, the SAS_SSL_CIPHER_LIST environment variable supported on UNIX and z/OS was added.

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

• In SAS 9.4M3, 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 SAS 9.4M3, information about setting up a FIPS-2 environment has been updated in the SAS Deployment Wizard.

• In SAS 9.4M5, system options SSLCACERTDATA=, SSLCACERTDIR=, SSLCIPHERLIST=, and SSLSNIHOSTNAME= have been added for LINUX, UNIX, and Z/OS. The SSLMODE= system option has been added for LINUX, UNIX, Z/OS, and Windows.

• In SAS 9.4M5, by default the SNI is sent to the web servers in the TLS handshake. The environment variable, SSL_USE_SNI is now used only to disable SNI.

• In SAS 9.4M5, when encrypting data at rest, you can now specify data set option ENCRYPT=AES2. AES2 is another key generation algorithm for AES encryption.

• In SAS 9.4M5, encoding type SAS005 uses AES encryption with a 256-bit fixed key and a 64-bit random salt value. SAS005 increases security for stored passwords by using the SHA-256 hashing algorithm and is hashed for additional iterations.

• In SAS 9.4M5, for more security, you can use SHA256-10000 for internal account passwords used in metadata. SHA256-10000 is the same as SHA256, but is hashed for additional iterations.

• In SAS 9.4M5, environment variable SAS_VIYA_TOKEN is added and provides the OAuth token that is needed for a user to access SAS Viya services.

• In SAS 9.4M5, the CAS_CLIENT_SSL_CA_LIST environment variable is added to allow SAS 9.4 acting as a client to use the same TLS certificates as those used by SAS Viya.

• In SAS 9.4M6, TLS is supported on all IOM servers.

• In SAS 9.4M6, the SAS Private JRE is based on Java 8 technology. Java 8 supports TLS 1.2. When using the SAS Deployment Manager to add CA certificates, the certificates supplied should be generated using algorithms supported by TLS 1.2.

  Note: SAS 9.4 versions that precede SAS 9.4M6 use a SAS Private JRE that is based on Java 7.
General Enhancements

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

- TLS version 1.2 is the protocol supported in SAS 9.4. Earlier versions of TLS and SSL are insecure. Starting in SAS 9.4M5, the default version of TLS is 1.2. For SAS software releases prior to SAS 9.4M5, you must configure SAS to use TLS 1.2 by setting the SAS_SSL_MIN_PROTOCOL= environment variable.
  
  Note: The SAS_SSL_MIN_PROTOCOL= environment variable is available in SAS releases prior to SAS 9.4M3 by applying security hot fixes.

- 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 code base version that is used to build the TLS libraries provided by SAS for each release, see Table 1.1 on page 14.

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

- For software delivery purposes, SAS/SECURE is a product within SAS. 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 the SAS004 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 59.

- 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 following changes and enhancements were made to encryption in SAS maintenance releases:

- In SAS 9.4M1 and SAS 9.4M2 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 32.
• In 9.4M3, 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 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 141 and “SSL_CERT_DIR Environment Variable” on page 143.

• Starting in SAS 9.4M1, 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 57.

• In SAS 9.4M3, SAS_SSL_MIN_PROTOCOL= environment variable was added and is supported on UNIX, Windows, and z/OS. This environment variable should be set to support TLS 1.2. Earlier versions of TLS are insecure. For more information, see “SAS_SSL_MIN_PROTOCOL= Environment Variable” on page 54.

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

• In SAS 9.4M3, environment variable SAS_SSL_CIPHER_LIST is available and is supported on UNIX and z/OS. For more information, see “SAS_SSL_CIPHER_LIST Environment Variable” on page 139.

• On a Windows server or client, the user can import digital certificates to a Machine Store as well as to a Personal Store. See “Configure TLS and Request Digital Certificates on Windows” on page 116.

• In SAS 9.4M3, 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 Store” on page 119, “Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager” on page 104 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 SAS 9.4M3, information has been added about setting the FIPS security settings on a Windows server. See “Configure FIPS 140-2 Capable TLS on Windows” on page 116.

• In SAS 9.4M3, 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 15, and “Building FIPS 140-2 Capable OpenSSL on UNIX” on page 113.

- In SAS 9.4M5, the following system options have been added for LINUX, UNIX, and Z/OS. These system options can also be used as environment variables. See Chapter 2, “SAS System Options for Encryption,” on page 25:
  - SSLCACERTDATA= specifies the trusted certification authority (CA) certificate in base64 encoded string.
  - SSLCIPHERLIST= specifies a list of cipher suites to use.
    Note: This system option replaces the SAS_SSL_CIPHER_LIST environment variable.
  - SSLSNIHOSTNAME= enables the client to use Server Name Indication (SNI) in the TLS handshake to identify the server name that it is trying to connect to. SSLSNIHOSTNAME= is used when you want to change the name of the host.
  - SSLCACERTDIR= specifies the location of the trusted certificate authorities (CA) found in OpenSSL format.
    Note: This system option replaces the SAS_SSL_CERT_DIR environment variable.
  - SSLMODE= specifies the SAS data-in-motion standard being used for all data-in-transit across network boundaries.
    - In SAS 9.4M5, the SSLMODE= system option has been added for Windows.
    - In SAS 9.4M5, by default the SNI is sent to the web servers in the TLS handshake. The environment variable, SSL_USE_SNI is now used only to disable SNI.
    - In SAS 9.4M5, when encrypting data at rest, you can now specify data set option ENCRYPT=AES2. AES2 is another key generation algorithm for AES encryption. See “ENCRYPT= Data Set Option” in SAS Data Set Options: Reference and “SAS Data File Encryption” in SAS Language Reference: Concepts.
    - In SAS 9.4M5, encoding type SAS005 is added and uses AES encryption with a 256-bit fixed key and a 64-bit random salt value. SAS005 increases security for stored passwords by using the SHA-256 hashing algorithm and is hashed for additional iterations. You can specify SAS005 as a method to encode passwords using PROC PWENCODE. You can also use this encoding for passwords stored in metadata and in configuration files. See Chapter 4, “PWENCODE Procedure,” on page 59.
    - In SAS 9.4M5, for more security, you can use SHA256-10000 for internal account passwords stored in metadata. SHA256-10000 is the same as SHA256, but is hashed for additional iterations.
    - In SAS 9.4M5, environment variable SAS_VIYA_TOKEN is added and provides the OAuth token that is needed for a user to access SAS Viya services. This environment variable enables you to pre-load your token before starting your SAS session. See “SAS_VIYA_TOKEN Environment Variable” on page 56.
    - In SAS 9.4M5, the CAS_CLIENT_SSL_CA_LIST environment variable can be specified to allow a SAS 9.4 client to use the same TLS certificates as those used by SAS Viya. The environment variable points to the path and filename of the file that contains the list of trusted certificate authority (CA) certificates.
      Note: Starting in the December 2017 release of SAS 9.4M5, the CAS_CLIENT_SSL_CA_LIST environment variable does not need to be set.

For more information, see CAS TLS Environment Variables.
• The OpenSSL libraries provided by SAS are updated on an ongoing basis. For 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. For a quick reference, see Table 1.1 on page 14.

• In SAS 9.4M6, TLS is supported on Integrated Object Model (IOM) servers and server processes that provide IOM Bridge access. The following servers and processes support TLS:
  • SAS Metadata Server
  • SAS OLAP Server
  • workspace server
  • SAS Stored Process Server
  • pooled workspace server
  • object spawner

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

• In SAS 9.4M6, the SAS Private JRE is based on Java 8 technology. Java 8 supports TLS 1.2. When using the SAS Deployment Manager to add CA certificates, the certificates supplied should be generated using algorithms supported by TLS 1.2. See the error that can occur when you try to add a certificate to the trust store using SAS Deployment Manager and an unexpected algorithm has been used to sign the certificate.

For more information, see SAS 9.4 Support for Java.

---

**Documentation Enhancements**

In SAS 9.4M4, 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 110.

• See “Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager” on page 104.
For information about the accessibility of this product, see Accessibility Features of the Windowing Environment for SAS 9.4 at support.sas.com.
What's New in Encryption in SAS 9.4
Part 1

Encryption in SAS 9.4

Chapter 1
  Technologies for Encryption ........................................... 3

Chapter 2
  SAS System Options for Encryption ................................. 25

Chapter 3
  SAS Environment Variables ......................................... 53

Chapter 4
  PWENCODE Procedure .............................................. 59

Chapter 5
  Encryption Technologies: Examples ............................... 69
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 motion 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. PROC PWENCODE, for example, encodes passwords.

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

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

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

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 and TLS. 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 is data stored in databases, file servers, endpoint devices, and various storage networks. This data can be on-premises, virtual, or in the cloud. This data is usually protected in conventional ways by firewalls. Numerous layers of defense are needed, and encrypting sensitive data is another layer. The emphasis is on protection of passwords in configuration files, in the metadata repository, and on encryption of SAS data sets.
Data in motion is data that is being transmitted to another location. Data is most vulnerable while in transit. Sensitive data in motion should be encrypted. You can protect all traffic in transit between servers and clients.

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 15, and “ENCRYPTFIPS System Option” on page 25.

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 the ENCRYPTFIPS option enabled, SAS verifies that all of your SAS servers have been configured to use the FIPS approved Advanced Encryption Standard (AES) libraries or the TLS protocol.

Note: Prior to SAS 9.4M6, only AES encryption was used to secure IOM servers when ENCRYPTFIPS was set.
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 “Building FIPS 140-2 Capable OpenSSL on UNIX” on page 113 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) encoding and AES (SAS003-SAS005). By default, metadata stores passwords on login objects with SAS003, but returns passwords using SAS002 by default.
- Internal account passwords on disk in the metadata can use SHA-256, SHA246-10000, and MD5 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 (SAS002) encoding or AES (SAS003-SAS005). By default, SASProprietary (SAS002) is used.
  Note: Configuration file passwords can be upgraded to AES (SAS003-SAS005).
  SAS003 uses a 256-bit key plus 16-bit salt value to encode passwords. SAS004 uses a 256-bit key plus 64-bit salt value to encode passwords. SAS005 uses 256-bit key plus 64-bit salt value and more iterations than SAS005.

Refer to the "Encryption for Data at Rest" and "Encryption for Data in Motion" chapters in SAS Intelligence Platform: Security Administration Guide for detailed information. For information about encrypting data sets using SASProprietary, see “SAS Data File Encryption” in SAS Language Reference: Concepts.

**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 SAS 9.4M1, a metadata-bound library administrator can require that all data files in the bound library be encrypted with either AES or SAS Proprietary encryption. For more information, see “Requiring Encryption for Metadata-Bound Data Sets” in Base SAS Procedures Guide 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 NETENCRIPTALGORITHM=SASPROPRIETARY. The NETENCRIPTALGORITHM= option must be set before the LIBNAME statement establishes the connection to the server. On the server, you set the NETENCYP option to specify that encryption is required by any client that accesses this server. The NETENCRIPTALGORITHM= 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 28 and “NETENCRYPTALGORITHM= System Option” on page 28.

For an example of configuring and using SAS Proprietary Encryption in your environment, see “SAS Proprietary Encryption for SAS/SHARE: Example” on page 70. 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 18 for more information about encryption algorithms supported for use with SAS/SECURE.

SAS/SECURE provides encryption for the following:

• data in motion
SAS/SECURE enables you to provide stronger protection for data in motion 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 28 and “NETENCRYPTALGORITHM= System Option” on page 28 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-SAS005 encoding methods, which use industry-standard algorithms for stored passwords. The SAS003 encoding method uses AES with 16-bit salt, SAS004 and SAS005 encoding method uses AES with 64-bit salt. However, SAS005 uses more iterations for more security. You can use the PWENCODE procedure (specify the METHOD= option) to upgrade to stronger encryption, AES (SAS003-SAS005).

Refer to Chapter 4, “PWENCODE Procedure,” on page 59 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-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 25. Also see and “SAS/SECURE FIPS 140-2 Compliant Installation and Configuration” on page 11 for details.

- AES Encryption of SAS Data Sets

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

**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 SAS. 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 Configuration**
SAS/SECURE is delivered on every SAS installation. In SAS 9.4, SAS/SECURE is installed with the Base SAS software. However, the default is SAS Proprietary Encryption. Whether SAS/SECURE is used depends on the options that are set.

To use the higher form of encryption provided by SAS/SECURE for communications and networking (data in motion), 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 28 and “NETENCRYPTIONALGORITHM= System Option” on page 28.

**SAS/SECURE FIPS 140-2 Compliant Installation and Configuration**

To configure a FIPS 140-2 compliant system, you must use SAS/SECURE or TLS. On UNIX, z/OS, or Windows, set the following options:

- When using SAS/SECURE, specify SAS system options `ENCRYPTFIPS` and `NETENCRALG=AES`.
- When using TLS, specify SAS system options `ENCRYPTFIPS` and `NETENCRALG=AES` or `NETENCRALG=SSL`.

You can then 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.

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

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

**CAUTION:**

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 25
- “`NETENCRYPTALGORITHM= System Option`” on page 28
- “`FIPS 140-2 Standards Compliance`” on page 5
- “`TLS: FIPS 140-2 Compliant Installation and Configuration`” on page 15

**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 is a protocol that provides network data privacy, data integrity, and authentication.

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

TLS uses X.509 certificates and hence asymmetric cryptography to assure the party with whom they are communicating, and to exchange a symmetric key. As a consequence of choosing X.509 certificates, certificate authorities and a public key infrastructure are necessary to verify the relation between a certificate and its owner, as well as to
generate, sign, and administer the validity of certificates. For more information about client and server negotiations using certificates, refer to “Certificate Implementation: How TLS Client and Servers Negotiate” on page 90.

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:

- Linux
- 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 and OpenSSL Version Support**
TLS 1.2 is the supported TLS version for SAS 9.4. Earlier versions of TLS and SSL are insecure. Contact Technical Support if you need an earlier version of TLS.

Starting in SAS 9.4M5, TLS version 1.2 is the default TLS protocol supported.

Prior to SAS 9.4M5, to configure SAS to use TLS 1.2, set environment variable SAS_SSL_MIN_PROTOCOL= to TLS 1.2. See “SAS_SSL_MIN_PROTOCOL= Environment Variable” on page 54.

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

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

- On Windows, SAS uses the Windows Secure Channel (Schannel) library that comes with the Windows operating system for TLS encryption.
• On UNIX and Linux, SAS provides the OpenSSL libraries needed to run TLS. To find the OpenSSL code base version that is used to build the TLS libraries provided by SAS for each release, see Table 1.1 on page 14.

• SAS provides the libraries needed to run TLS on z/OS.

For SAS 9.4 and all maintenance releases of SAS 9.4, updated versions of OpenSSL are provided in the latest releases of the software and are updated as needed 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 software fixes.

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

<table>
<thead>
<tr>
<th>SAS Version</th>
<th>OpenSSL Version Library</th>
<th>Library Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.4M6 hot fix</td>
<td>1.02r</td>
<td>libcrypto.so.1.0.0, libssl.so.1.0.0</td>
</tr>
<tr>
<td>9.4 - 9.4M5b hot fixes</td>
<td>1.0.2n</td>
<td>libcrypto.so.1.0.0, libssl.so.1.0.0</td>
</tr>
<tr>
<td>9.3 - 9.3M2 hot fixes</td>
<td>1.0.2n</td>
<td>libcrypto.so.1.0.0, libssl.so.1.0.0</td>
</tr>
<tr>
<td>9.2M3 hot fix</td>
<td>0.9.8zh</td>
<td>not applicable</td>
</tr>
</tbody>
</table>

The OpenSSL libraries supplied by SAS for 9.4 on UNIX and Linux are external libraries. You can use OpenSSL to compile your own libraries.

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

The OpenSSL libraries shipped with SAS for UNIX and Linux are not FIPS 140-2 compliant. However, you can compile a FIPS 140-2 compliant version of OpenSSL and install it. See “Building FIPS 140-2 Capable OpenSSL on UNIX” on page 113.

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

The trustedcerts.pem file is in X.509 Base-64 encoded format. System option SSLCALISTLOC= points to the location of this file. The path to the certificate file is
In SAS 9.4M6, the SAS Private JRE is based on Java 8 technology. Java 8 supports TLS 1.2. When using the SAS Environment Manager to add CA certificates, the certificates supplied should be generated using algorithms supported by TLS 1.2. For more information, see SAS 9.4 Support for Java.

Note: SAS 9.4 versions that precede SAS 9.4M6 use a SAS Private JRE that is based on Java 7.

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

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

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

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

- Chapter 7, “Installing and Configuring TLS and Certificates on UNIX,” on page 93
- Chapter 8, “Installing and Configuring TLS and Certificates on Windows,” on page 115
- Chapter 9, “Installing and Configuring TLS and Certificates on z/OS,” on page 125

For examples of configuring and using TLS in your environment, see Chapter 5, “Encryption Technologies: Examples,” on page 69.

**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 and Linux are not FIPS 140-2 compliant. However, you can compile a FIPS 140-2 compliant version of OpenSSL and install it. For more information, see “Building FIPS 140-2 Capable OpenSSL on UNIX” on page 113 and “Configure FIPS 140-2 Capable TLS on Windows” on page 122.

After compiling the FIPS compliant libraries, you will need to specify the NETENCRYPTALGORITHM (set to AES or SSL) system option. You will then be able to run in FIPS compliant mode and connect to other servers and clients that are also configured with FIPS.

**CAUTION:**

Use ENCRYPTFIPS with caution. Turning on the ENCRYPTFIPS option is not generally recommended for TLS, unless absolutely required by your site’s policy. If the ENCRYPTFIPS option is turned on, the SAS server-based TLS clients 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 “Building FIPS 140-2 Capable OpenSSL on UNIX” on page 113.
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 28
- “ENCRYPTFIPS System Option” on page 25

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:
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 81 and “SSH Tunnel for SAS/SHARE: Example ” on page 82.

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 asymmetric-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 asymmetric-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. An 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.

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
Comparison of Encryption Technologies

The following table compares the features of encryption technologies:

<table>
<thead>
<tr>
<th>Features</th>
<th>SAS Proprietary</th>
<th>SAS/SECURE</th>
<th>TLS</th>
<th>SSH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption and authentication</td>
<td>Encryption only</td>
<td>Encryption only</td>
<td>Encryption and authentication</td>
<td>Encryption only</td>
</tr>
<tr>
<td>Encryption level</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Algorithms supported</td>
<td>SAS Proprietary fixed encoding</td>
<td>RC2, RC4, DES, TripleDES, AES. Default is SAS Proprietary</td>
<td>RC2, RC4, DES, TripleDES, AES, and others</td>
<td>Product dependent</td>
</tr>
<tr>
<td>Installation required</td>
<td>No (part of Base SAS)</td>
<td>No (delivered with Base SAS)</td>
<td>Delivered with Base SAS. You can replace this version.</td>
<td>Yes</td>
</tr>
<tr>
<td>Operating environments supported</td>
<td>UNIX Windows z/OS</td>
<td>UNIX Windows z/OS</td>
<td>UNIX Windows z/OS</td>
<td>UNIX Windows z/OS</td>
</tr>
<tr>
<td>SAS version support</td>
<td>8 and later</td>
<td>8 and later</td>
<td>9 and later</td>
<td>8.2 and later</td>
</tr>
</tbody>
</table>

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 25. For examples, see Chapter 5, “Encryption Technologies: Examples,” on page 69.

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 1.3 Selected Security-Related Loggers

<table>
<thead>
<tr>
<th>Logger</th>
<th>SAS/SECURE Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>App.tk.eam</td>
<td>Logs security information.</td>
</tr>
<tr>
<td>App.tk.eam.ssl</td>
<td>Logs TLS encryption information including the OpenSSL protocol and cipher suites being used.</td>
</tr>
<tr>
<td>App.tk.eam.sas</td>
<td>Logs SAS Proprietary encryption information.</td>
</tr>
<tr>
<td>App.tk.eam.rsa</td>
<td>Logs RC2, RC4, DES, DES3, and AES encryption information.</td>
</tr>
<tr>
<td>App.tk.eam.rsa.pbe</td>
<td>Enables or disables the password-based encryption processing that creates a key.</td>
</tr>
<tr>
<td>App.tk.eam.rsa.capi</td>
<td>Logs RC2, RC4, DES, and DES3 encryption information for Windows C API.</td>
</tr>
<tr>
<td>App.tk.eam.rsa.cc</td>
<td>Logs RC2, RC4, DES, DES3, and AES encryption information for RSA BSAFE® Crypto-C.</td>
</tr>
<tr>
<td>App.tk.eam.rsa.ccme</td>
<td>Logs AES encryption information for RSA BSAFE® Crypto-C ME. This log is for FIPS.</td>
</tr>
<tr>
<td>App.tk.eam.rsa.icsf</td>
<td>Logs AES encryption information for IBM Integrated Cryptographic Service Facility (ICSF). This log is for FIPS.</td>
</tr>
</tbody>
</table>

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.
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.


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>
<thead>
<tr>
<th>Task</th>
<th>System Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies whether text and graphics from PDF documents can be read by screen readers for the visually impaired</td>
<td>PDFACCESS</td>
</tr>
<tr>
<td>Controls whether PDF documents can be assembled</td>
<td>PDFASSEMBLY</td>
</tr>
<tr>
<td>Controls whether PDF document comments can be modified</td>
<td>PDFCOMMENT</td>
</tr>
<tr>
<td>Controls whether the contents of a PDF document can be changed</td>
<td>PDFCONTENT</td>
</tr>
<tr>
<td>Controls whether text and graphics from a PDF document can be copied</td>
<td>PDFCOPY</td>
</tr>
<tr>
<td>Controls whether PDF forms can be filled in</td>
<td>PDFFILLIN</td>
</tr>
<tr>
<td>Specifies the password to use to open a PDF document and the password used by a PDF document owner</td>
<td>PDFPASSWORD=</td>
</tr>
<tr>
<td>Controls the resolution used to print the PDF document</td>
<td>PDFPRINT=</td>
</tr>
<tr>
<td>Task</td>
<td>System Option</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Controls the printing permissions for PDF documents</td>
<td>PDFSECURITY=</td>
</tr>
</tbody>
</table>

*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.
Dictionary

ENCRYPTFIPS System Option ...................................................... 25
NETENCRYPT System Option .......................................................... 28
NETENCRYPTIONALGORITHM= System Option ...................................... 28
NETENCRYPTKEYLEN= System Option ............................................... 31
SSLALISTLOC= System Option ....................................................... 32
SSLCERTDIR= System Option .......................................................... 34
SSLCERTDATA= System Option ......................................................... 36
SSLCERTISS= System Option ............................................................ 37
SSLCERTLOC= System Option ............................................................ 38
SSLCERTSERIAL= System Option ....................................................... 39
SSLCERTSUBJ= System Option .......................................................... 40
SSLCLIENTAUTH System Option ....................................................... 41
SSLCRLCHECK System Option .......................................................... 41
SSLCIPHERLIST= System Option ....................................................... 42
SSLCRLLOC= System Option ............................................................ 43
SSLMODE= System Option ............................................................... 44
SSLMINPROTOCOL= System Option ..................................................... 45
SSLPKCS12LOC= System Option ....................................................... 47
SSLPKCS12PASS= System Option ....................................................... 48
SSLPVTKKEYLOC= System Option ...................................................... 48
SSLPVTKKEYPASS= System Option ..................................................... 49
SSLREQCERT= System Option ........................................................... 50
SSLSNIHOSTNAME= System Option ..................................................... 51

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 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.

**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 “Building FIPS 140-2 Capable OpenSSL on UNIX” on page 113.

**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.
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 site 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 TLS algorithm (value specified is SSL) is supported for FIPS 140-2 encryption. An error is logged when an unsupported algorithm is specified.

ERROR: When SAS option ENCRYPTFIPS is ON the option value for SAS option 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 netencryptalgorithm="aes"

Example 3
Here is an example of configuring the ENCRYPTFIPS option on Windows:

    -encryptfips -netencralg "AES"

See Also

• “NETENCRYPTALGORITHM= System Option” on page 28
• “FIPS 140-2 Standards Compliance” on page 5
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**: NETENCRYPTHALGORITHM=

**Example**: “SAS/SECURE for SAS/CONNECT: Example ” on page 70

**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 NETENCRYPTHALGORITHM= 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.

**NETENCRYPTHALGORITHM= System Option**

Specifies the algorithm or algorithms to be used for encrypted client/server data transfers.
**NETENCRYPTALGORITHM= System Option**

<table>
<thead>
<tr>
<th>Client:</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server:</td>
<td>Required</td>
</tr>
<tr>
<td>Valid in:</td>
<td>Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line</td>
</tr>
<tr>
<td>Category:</td>
<td>Communications: Networking and Encryption</td>
</tr>
<tr>
<td><strong>PROC OPTIONS GROUP=</strong></td>
<td>Communications</td>
</tr>
<tr>
<td><strong>Alias:</strong></td>
<td>NETENCRALG</td>
</tr>
<tr>
<td><strong>Operating environment:</strong></td>
<td>UNIX, Windows, z/OS</td>
</tr>
<tr>
<td>See:</td>
<td>“NETENCRYPT System Option” on page 28, “ENCRYPTFIPS System Option” on page 25</td>
</tr>
<tr>
<td><strong>Examples:</strong></td>
<td>“TLS for a SAS/CONNECT Windows Spawner: Example ” on page 73</td>
</tr>
<tr>
<td></td>
<td>“TLS on a z/OS Spawner on a SAS/CONNECT Server: Example” on page 75</td>
</tr>
<tr>
<td></td>
<td>“TLS for a SAS/CONNECT UNIX Spawner: Example ” on page 71</td>
</tr>
</tbody>
</table>

### Syntax

NETENCRYPTALGORITHM=algorithm | (“algorithm-1”... “algorithm-n”)

### Syntax Description

algorithm | (“algorithm-1”... “algorithm-n”)

specifies the algorithm or algorithms that can be used for encrypting data that is transferred between a client and a server across a network. When you specify two or more encryption algorithms, use a space or a comma to separate them, and enclose the algorithms in parentheses.

**Note:** If you are running SAS/CONNECT in a SAS Intelligence Platform environment using the SAS Metadata Server and configured encryption using the SAS Management Console, you can specify only one encryption algorithm. There is a workaround if you receive “ERROR: Cannot negotiate encryption algorithm” on page 135.

The following values can be specified:

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

**Note:** All values except SSL are algorithms. SSL is the predecessor protocol to TLS. TLS is a protocol that allows two communicating peers to select a mutually agreed upon algorithm that is used to encrypt all traffic.
Restrictions
If you do not have SAS/SECURE, an error is generated if algorithm AES is specified.

Starting in SAS 9.4M6, Integrated Object Model (IOM) servers support the NETENCRYPTALG=SSL). In software releases before 9.4M6, TLS is not supported on IOM servers.

When ENCRYPTFIPS is specified, only the TLS protocol (value specified in NETENCRYPTALG= is SSL) 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

<table>
<thead>
<tr>
<th>Server Settings</th>
<th>Client Settings</th>
<th>Connection Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONETENCRYPT</td>
<td>No settings</td>
<td>If the client is capable of encryption, the client/server connection is encrypted.</td>
</tr>
<tr>
<td>NETENCRLAG=alg</td>
<td></td>
<td>Otherwise, the connection is not encrypted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NETENCRYPT</td>
<td>No settings</td>
<td>If the client is capable of encryption, the client/server connection is encrypted.</td>
</tr>
<tr>
<td>NETENCRLAG=alg</td>
<td></td>
<td>Otherwise, the client/server connection fails.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No settings</td>
<td>NONETENCRYPT</td>
<td>A client/server connection is not encrypted.</td>
</tr>
<tr>
<td>NETENCRLAG=alg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No settings</td>
<td>NETENCRYPT</td>
<td>A client/server connection fails.</td>
</tr>
<tr>
<td>NETENCRLAG=alg</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Example

In the following example, the client and the server specify different values for the NETCRIPTALGORITHM= option.

The client specifies two algorithms in the following OPTIONS statement:

```plaintext
options netencryptalgorithm=(rc2 tripledes);
```

The server specifies three algorithms and requires encryption in the following OPTIONS statement:

```plaintext
options netencrypt netencryptalgorithm=(ssl des tripledes);
```

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

### NETCRIPTKEYLEN= 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**: NETENCRYKEY=
- **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

```plaintext
NETCRIPTKEYLEN= 0
```
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.

SSLALISTLOC= 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 SAS 9.4M3, the default path set for the SSLALISTLOC= system option on UNIX and z/OS foundation servers is <SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem. The trustedcerts.pem file contains the list of trusted CA Certificates, the Mozilla Bundle provided by SAS at installation.

In SAS 9.4M1 and SAS 9.4M2, the default path set for the SSLALISTLOC= system option on UNIX and z/OS foundation servers is SAS-configuration-directory/Levnl/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 94.

Examples: “TLS for a SAS/CONNECT UNIX Spawner: Example ” on page 71
SSLALISTLOC= System Option

"TLS on a z/OS Spawner on a SAS/CONNECT Server: Example" on page 75

Syntax

SSLALISTLOC=“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 SSLALISTLOC= 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 86.

From SAS 9.4 to SAS 9.4M2, the default setting for the SSLALISTLOC= system option on UNIX and z/OS foundation servers is SAS-configuration-directory/Levlen/certs/cacert.pem. The cacert.pem file contains the list of trusted CA Certificates.

In SAS 9.4M3, the default path set for the SSLALISTLOC= 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 SSLALISTLOC= system option. Starting SAS 9.4M3, the SSLALISTLOC= 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 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 104.

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 128. For information about creating a trust list on UNIX, refer to “Step 5. Create a Certificate Chain in PEM Format Using OpenSSL” on page 102 and “Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager” on page 104.

Note: System option SSLCACERTDIR= points to a directory that contains all of the public certificate file(s) of all CA(s) in the trust chain. One file exists for each CA in the trust chain. These can be used instead of using the SSLALISTLOC= system option. Refer to “SSLCACERTDIR= System Option” on page 34.

For Foundation servers such as workspace servers and stored process servers (that is, servers in a deployment), if certificates are used, SAS searches for certificates in the following specific order:
1. SAS looks for SAS system option SSLCALISTLOC= to find the file trustedcerts.pem.

2. SAS looks for the SSLCALISTLOC= system option to find the file trustedcerts.pem.

3. If trustedcerts.pem exists and SSLCERTDIR= system option is set, SAS checks trustedcerts.pem first before it searches this directory.

4. If trustedcerts.pem does not exist, but the certificates are in the directory defined by SSLCERTDIR=, then SAS ignores SSLCALISTLOC=.

5. If trustedcerts.pem does not exist, and the SSLCERTDIR= system option is not set, SAS reports an error.

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

---

**SSLCACERTDIR= System Option**

Specifies the location of the trusted certificate authorities (CA) found in OpenSSL format.

- **Client:** Optional
- **Server:** Optional
- **Valid in:** Configuration file, SAS invocation
- **Operating environment:** UNIX, LINUX, z/OS

**Categories:**
- Communications: Networking and Encryption
- System Administration: Security

**Default:** The default location for certificates is set using the SSLCALISTLOC= system option.

**Note:** This system option is added in SAS 9.4M5. This option can also be specified as an environment variable. In prior releases, an environment variable of a similar name was used.

**Tips:**
- OpenSSL looks up the CA certificate based on the x509 hash value of the certificate. SSLCACERTDIR= requires that the certificates are located in the specified directory where the certificate names are the value of a hash that OpenSSL generates.
- If you are upgrading from a version of OpenSSL that is older than 1.0.0, you need to update your certificate directory links. Starting with code base 1.0.0, SHA hashing is used instead of MD5. You can use the OpenSSL C_REHASH utility to re-create symbolic links to files named by the hash values.
- You can discover the hash value for a CA and then create a link to the file named after the certificate’s hash value. Note that you must add ".0" to the hash value.

```bash
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:

```bash
openssl verify -CApath /u/myuser/sslcerts cacert1.pem
```

**See:** “Defining Environment Variables in UNIX Environments” in *SAS Companion for UNIX Environments* and “TKMVSENV File” in *SAS Companion for z/OS*

**Example:** The SSCACERTDIR system option points to the directory where the CA certificate is located.

```bash
-SSLCACERTDIR=/u/myuser/sslcerts/
```

### Syntax

```
SSLCACERTDIR="file-path"
```

### Syntax Description

**“file-path”**

specifies the directory location where the public certificates for all of the trusted certificate authorities (CA) in the trust chain are filed. There is one file for each CA. The names of the files are the value of a hash that OpenSSL generates.

**Note:** OpenSSL generates different hash values for each OpenSSL version. For example, OpenSSL 0.9.8 generates different hash values than does OpenSSL 1.0.2.

### Details

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

SSLCACERTDIR= requires the certificates to be in the directory where their names are the value of a hash that OpenSSL generates.

Each CA certificate file must be PEM-encoded (base64). For more information, see “Certificate File Formats” on page 86.

For Foundation servers such as workspace servers and stored process servers (that is, servers in a deployment), if certificates are used, SAS searches for certificates in the following specific order:

1. SAS looks for SAS system option SSLCALISTLOC= to find a file. This file holds your trusted certificates.
2. SAS looks for the SSLCALISTLOC environment variable to find a file. This file holds your trusted certificates.
3. If trustedcerts.pem exists and the SSLCACERTDIR= system option is 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 the SSLCACERTDIR= system option or environment variable, then SAS ignores SSLCALISTLOC=.
5. If trustedcerts.pem does not exist, and the SSLCACERTDIR= system option is not set, SAS reports an error.
In SAS 9.4, SAS 9.4M1, and SAS 9.4M2, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is `SAS-configuration-directory/Levn/certs/cacert.pem`. The cacert.pem file contains the list of trusted certificates.

In SAS 9.4M3, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is `<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem`. The trustedcerts.pem file contains the list of trusted certificates.

Note: A trusted CA certificate is required at the client in order to validate a server's digital certificate. The trusted CA certificate must be from the CA that signed the server certificate.

### SSLCACERTDATA= System Option

Specifies the base64 encoded x509 text that represents a single CA certificate

**Client:** Optional  
**Server:** Optional  
**Valid in:** Configuration file, SAS invocation, SAS/CONNECT spawner start-up if this option is used as an environment variable  
**Categories:** Communications: Networking and Encryption, System Administration: Security  
**Operating environment:** UNIX, LINUX, z/OS  
**Note:** This system option is added in SAS 9.4M5. This option can also be specified as an environment variable. In prior releases, an environment variable of a similar name was used.  
**See:** “Defining Environment Variables in UNIX Environments” in SAS Companion for UNIX Environments and “TKMVSENV File” in SAS Companion for z/OS

### Syntax

```
SSLCACERTDATA="encoded-string"
```

### Syntax Description

*encoded-string*

specifies the base64 encoded x509 text that represents a single certificate authority (CA) certificate. This string is in PEM format. The text string starts with "-----BEGIN CERTIFICATE-----" and ends with "-----END CERTIFICATE-----" lines of text.

### Details

This option provides a way to programmatically specify a CA certificate rather than having to point to a file that contains the certificate information.

The certificate must be PEM-encoded (base64) format. For more information, see “Certificate File Formats” on page 86.
Example

Here is an example of how you might use the SSLCACERTDATA= System Option to specify a certificate.

```sas
data _null_
  length certInfo $3200.;
  input txt $67.;
  retain certInfo;
  if _N_ = 1 then
    certInfo=txt;
  else
    certInfo=catx('0a'x,certInfo,txt);
  call symput('certInfo',trim(left(certInfo)));
  datalines;
  -----BEGIN CERTIFICATE-----
  MIICbzCCAfagAwIBAgIJAP7q5/tk7+laMAoGCCqGSM49BAMCMHYxCzAJBgNVBAYT
  A1VTMQwQDVQQDDBxkZWQwNjBuMjIwMjIwMjIwMjIwMjIwMjIwMjIwMjIwMjIw
  M0AgQGlDh0dCP0gQDMwDgYDVR0PAQH/AgEAMeQGCCsGAQUFBzAChzowVzELMAkGA1UE
  BhoIhW5kJRYwFAYDVQQK
  -----END CERTIFICATE-----;
run;
options SSLCACERTDATA="&certInfo";
```

**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 79

Syntax

SSLCERTISS="issuer-of-digital-certificate"

Syntax Description

“issuer-of-digital-certificate”

specifies the name of the issuer of the digital certificate that should be used by TLS.

Details

The SSLCERTISS= option is used with the SSLCERTSERIAL= option to uniquely identify a digital certificate from the Microsoft Certificate Store.

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

You can use OpenSSL to print the issuer of the certificate. Note the value of serial= in the output.

C:\>openssl.exe x509 -in customer.pem -issuer -noout

Note the issuer of the certificate. For example, if issuer= /DC=com/DC=Company/ CN=Company SHA2 Issuing CA02, you will need the value of CN=, which is Company SHA2 Issuing CA02.

SSLCERTLOC= System Option

Specifies the location of the digital certificate for the machine’s public key. This is used for authentication.

Client: Optional
Server: Required
Valid in: Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category: Communications: Networking and Encryption

PROC OPTIONS GROUP=

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 71
“TLS on a z/OS Spawner on a SAS/CONNECT Server: Example” on page 75
“TLS for SAS/SHARE on UNIX: Example ” on page 77
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 86.

SSLHOST= System Option

Specifies the host name to use in the SSL client 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=
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 79

Syntax

SSLHOST="host-name"

Syntax Description

"host-name"

specifies the host name to use in the SSL client certificate.

SSLPORT= System Option

Specifies the port number to use for SSL communication.

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=
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 79

Syntax

SSLPORT=port-number

Syntax Description

"port-number"

specifies the port number to use for SSL communication.

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=
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 79

Syntax

SSLCERTSERIAL="serial-number"

Syntax Description

"serial-number"

specifies the serial number of the digital certificate that should be used by TLS.

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=
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 79

Syntax

SSLCERTSERIAL="serial-number"

Syntax Description

"serial-number"

specifies the serial number of the digital certificate that should be used by TLS.
Details

The SSLCERTSERIAL= option is used with the SSLCERTISS= option to uniquely identify a digital certificate from the Microsoft Certificate Store.

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

You can use OpenSSL to print the serial number of the certificate. Note the value of serial= in the output.

```
C:\>openssl.exe x509 -in customer.pem -serial -noout
```

SSLCERTSUBJ= System Option

Specifies the subject name of the digital certificate that TLS should use.

<table>
<thead>
<tr>
<th>Client:</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server:</td>
<td>Optional</td>
</tr>
<tr>
<td>Valid in:</td>
<td>Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line</td>
</tr>
<tr>
<td>Category:</td>
<td>Communications: Networking and Encryption</td>
</tr>
</tbody>
</table>

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 73

Syntax

SSLCERTSUBJ=“subject-name”

Syntax Description

“subject-name” specifies the subject name of the digital certificate that TLS should use.

Details

The SSLCERTSUBJ= option is used to search for a digital certificate from the Microsoft Certificate Store.

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

Specifies whether a server should perform client authentication.

**Server:** Optional  
**Valid in:** Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line  
**Category:** Communications: Networking and Encryption  
**PROC OPTIONS GROUP=** Communications  
**Operating environment:** UNIX, Windows, z/OS  
**Tip:** When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in SAS/CONNECT User’s Guide.

**Syntax**

SSLCLIENTAUTH | NOSSLCLIENTAUTH

**Syntax Description**

**SSLCLIENTAUTH**  
specifies that the server should perform client authentication.

| **TIP** | If you enable client authentication, a certificate for each client is needed. |

**NOSSLCLIENTAUTH**  
specifies that the server should not perform client authentication.

**Default** NOSSLCLIENTAUTH is the default.

**Details**

Server authentication is always performed, but the SSLCLIENTAUTH option enables a user to control client authentication. This option is valid only when used on a server.

SSLCRLCHECK System Option

Specifies whether a Certificate Revocation List (CRL) is checked when a digital certificate is validated.

**Client:** Optional  
**Server:** Optional  
**Valid in:** Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT spawner command line  
**Category:** Communications: Networking and Encryption
PROC OPTIONS
   GROUP= Communications
   Operating environment: UNIX, Windows, z/OS

Tip: When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in SAS/CONNECT User’s Guide.

Syntax

SSLCRLCHECK | NOSSLCRLCHECK

Syntax Description

SSLCRLCHECK
   specifies that CRLs are checked when digital certificates are validated.

NOSSLCRLCHECK
   specifies that CRLs are not checked when digital certificates are validated.

Details

A Certificate Revocation List (CRL) is published by a Certificate Authority (CA) and contains a list of revoked digital certificates. The list contains only the revoked digital certificates that were issued by a specific CA.

The SSLCRLCHECK option is required at the server only if the SSLCLIENTAUTH option is also specified at the server. Because clients check server digital certificates, this option is relevant for the client.

SSLCIPHERLIST= System Option

Specifies the ciphers that can be used on UNIX and z/OS for OpenSSL.

Client: Optional
Server: Optional
Valid in: Configuration file, command line
          SAS/CONNECT spawner start-up if this option is used as an environment variable
Categories: Communications: Networking and Encryption
           System Administration: Security
Restriction: If SSLMODE option is set, this option is ignored.
Operating environment: UNIX, LINUX, and z/OS
Notes: This system option is added in SAS 9.4M5. This option can also be specified as an environment variable. In prior releases, an environment variable of a similar name was used.

This system option must be set before TLS is loaded. It cannot be changed after TLS is loaded. You must set the environment variable before the SAS/CONNECT spawner is started and the system option before SAS is started on the client.
Tip: You can also define SET commands for Windows by using the System Properties dialog box that you access from the Control Panel.

See: “Defining Environment Variables in UNIX Environments” in SAS Companion for UNIX Environments and “TKMVSENV File” in SAS Companion for z/OS

Example: Specify the system option:

-SSLCIPHERLIST=HIGH

Syntax

SSLCIPHERLIST=openssl_cipher_list

Syntax Description

openssl-cipher-list

The SSLCIPHERLIST= system option specifies the ciphers that can be used on UNIX and z/OS for OpenSSL. Refer to the OpenSSL Ciphers document to see how to format the openssl-cipher-list and for a complete list of the ciphers that work with your TLS version. The OpenSSL Cipher information can be found at OpenSSL 1.0.2 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 21.

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

Details

This system option is available on UNIX and z/OS platforms. This system option can be specified anytime before TLS is used. After TLS is loaded, it cannot be changed.

Refer to the OpenSSL documentation on ciphers for information about the ciphers that can be specified for this system option. This information can be found at OpenSSL 1.0.2 Ciphers.

Note: For Windows, you can use group policy settings to configure TLS Cipher Suite Order. See Cipher Suites in TLS/SSL (Schannel SSP) for information about the TLS Cipher Suite order.

SSLRLOC= 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.

SSLMODE= System Option

Sets the allowed TLS version and cipher suites to be used for TLS.

Client: Optional
Server: Optional
Valid in: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window, SAS configuration
SAS/CONNECT spawner start-up if this option is used as an environment variable
Category: Communications: Networking and Encryption
PROC OPTIONS GROUP= Communications
Default: SSLMODESP800131A
Interaction: When system option SSLMODE= is set, system option SSLMINPROTOCOL is ignored.
Operating environment: UNIX, LINUX, z/OS
Tip: When additional encryption options are specified on the spawner command line, the options must be included in the -SASCMD value. The spawner does not automatically pass the encryption values. For detailed information, see SASCMD for your operating environment in SAS/CONNECT User’s Guide.
See: “Defining Environment Variables in UNIX Environments” in SAS Companion for UNIX Environments and “TKMVSENV File” in SAS Companion for z/OS
For a list of ciphers that are supported for each of the modes that can be specified for the SSLMODE= system option, see “SSLMODE= System Option Supported Ciphers” on page 147.
Example: -ssimode SSLMODESP800131A
Syntax

**SSLMODE=ssl-mode**

**Syntax Description**

*ssl-mode*

sets the allowed TLS version and cipher suites to be used for TLS. Valid *ssl-modes* are:

**SSLMODESUITEB128**

is the mode of operation that uses the cipher suites specified in the NIST Suite B Cryptography using 128 AES encryption.

**SSLMODESUITEB192**

is the mode of operation that uses the cipher suites specified in the NIST Suite B Cryptography using 192 AES encryption.

**SSLMODESP800131A**

is the DEFAULT configuration mode for TLS communication. The SSLMODESP800131A mode is the mode of operation that uses the cipher suites specified in [NIST Special Publication 800-131A](https://csrc.nist.gov/publications/detail/sp/800-131a/rev-1/final).

When system option SSLMODE= is set, system option SSLMINPROTOCOL= is ignored. If SSLMODE= is not set, SAS checks SSLMINPROTOCOL= system option and uses the protocol set. If neither system option is set, SAS uses the default cipher mode SSLMODESP800131A.

**SSLMODEDEPRECATED**

allows for older cipher suites. These are cipher suites that do not meet NIST standards.

**CAUTION:**

Cipher suites that do not meet NIST standards should not be used.

**Details**

SAS uses the National Institute of Standards and Technology (NIST) Special Publication 800-131A (SP800-131A) as the minimum compliance standard for TLS and to extend the FIPS standards. TLS version 1.2 is the default version of TLS that SAS supports. For details of SP800-131A, see [NIST Special Publicatio...](https://csrc.nist.gov/publications/detail/sp/800-131a/rev-1/final).

Suite B cryptography allows TLS client and server applications to specify a profile compliant with Suite B Cryptography as defined in [RFC 5430: Suite B Profile for Transport Layer Security (TLS)](https://tools.ietf.org/html/rfc5430). Suite B cryptography specifies the cryptographic algorithms that can be used in a "Suite B Compliant" TLS V1.2 session. Suite B requires the key establishment and authentication algorithms that are used in TLS V1.2 sessions to be based on Elliptic Curve Cryptography, and the encryption algorithm to be AES.

For a list of ciphers that are supported for each of the modes that can be specified for the SSLMODE= system option, see “**SSLMODE= System Option Supported Ciphers**” on page 147.

**SSLMINPROTOCOL= System Option**

Specifies the minimum TLS protocol that can be negotiated when using OpenSSL.
Client: Optional
Server: Optional
Valid in: Configuration file, SAS invocation, OPTIONS statement, System Options Window
SAS/CONNECT spawner start-up if this option is used as an environment variable
Categories: Communications: Networking and Encryption
System Administration: Security
Default: TLS 1.2.
Restriction: If the SSLMODE= option is set, this option is ignored.
Operating environment: UNIX, LINUX, z/OS, and Windows

Note: This system option is added in SAS 9.4M5. In prior releases of SAS 9.4, environment variable SAS_SSL_MIN_PROTOCOL was used.
Tip: You can also define SET commands for Windows by using the System Properties dialog box that you access from the Control Panel.
See: “Defining Environment Variables in UNIX Environments” in SAS Companion for UNIX Environments and “TKMVSENV File” in SAS Companion for z/OS
Example: Specify this system option as follows:

-sslminprotocol="TLS1.2"

Syntax

"SSLMINPROTOCOL= protocol"

Syntax Description

protocol

specifies the minimum TLS protocol version that can be negotiated between UNIX, z/OS, and Windows servers when using OpenSSL. TLS 1.2 is the default value in SAS 9.4M5 and later SAS releases. Protocols earlier than TLS 1.2 are insecure.

Valid values that can be supplied for this option are TLS1.2 and TLSv1.2. These are the values that can be specified, but are insecure: SSL3, SSLV3, TLS, TLS1, TLSV1, TLS1.0, TLSV1.0, TLS1.1, and TLSV1.1.

See the SAS Security Bulletin on OpenSSL for the most current information about the versions of OpenSSL used in SAS products and about the advisories under consideration. For a quick reference, see Table 1.1 on page 14.

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

TLS version 1.2 is the recommended protocol supported in SAS 9.4. In SAS 9.4M5, the default TLS version and the supported ciphers are set with the SSLMODE= option. The SSLMODE= option defaults to SSLMODESP800131A, which uses TLS 1.2 to negotiate between client and servers. If you are using TLS 1.2, you do not need to specify the
SSLMINPROTOCOL system option. Specifying a version of TLS that is older than TLS 1.2 is not recommended.

The SSLMINPROTOCOL= system option 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.

**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 75
  - “TLS for SAS/SHARE on z/OS: Example” on page 80

**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 SSLPVKEYLOC= 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 75
“TLS for SAS/SHARE on z/OS: Example” on page 80

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=

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 71
“TLS for SAS/SHARE on UNIX: Example ” on page 77

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 86.

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=

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 71
“TLS for SAS/SHARE on UNIX: Example ” on page 77

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.

Note: SAS recommends that you use a password or passphrase to encrypt the private key.

Details

The SSLPVTKEYPASS= option is required only when the private key is encrypted. OpenSSL performs key encryption.

Note: No SAS system option is available to encrypt private keys.

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.

---

**SSLSNIHOSTNAME= System Option**

Enables the client to specify the Server Name Indication (SNI) in the TLS handshake that identifies the server name that it is trying to connect to.

- **Client:** Optional
- **Server:** Optional
- **Valid in:** SAS invocation, configuration file
  SAS/CONNECT spawner start-up if this option is used as an environment variable
- **Categories:** Communications: Networking and Encryption
  System Administration: Security
- **Default:** The default is the name of the host being contacted.
- **Operating environment:** UNIX, LINUX, z/OS
- **Notes:** This system option is added in SAS 9.4M5. This option can also be specified as an environment variable. In prior releases, an environment variable of a similar name was used. The TLS SNI extension is always sent to the web server.
- **See:** “Defining Environment Variables in UNIX Environments” in SAS Companion for UNIX Environments and “TKMVSENV File” in SAS Companion for z/OS
- **Example:** Specify the system option as follows:

  -SSLSNIHOSTNAME="www.example.org"

**Syntax**

"**SSLSNIHOSTNAME**= hostname"
Syntax Description

SSLXNHOSTNAME=

specifies the host name that is used for the Server Name Indication (SNI) TLS extension. If it is not specified, the target host name is used. The client uses SNI in the first message of the TLS handshake (connection setup) to identify the server name that it is trying to connect to.

Details

The client uses SNI in the TLS handshake to identify the server name that it is trying to connect to. When making a TLS connection, the client requests a digital certificate from the web server. After the server sends the certificate, the client examines it and compares the name that it was trying to connect to with the name or names included in the certificate. If a match is found, the connection proceeds as normal.

See Also

For more information, see Chapter 10, “Troubleshooting,” on page 133.
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.

Note: In SAS 9.4M5, environment variables SAS_SSL_CIPHER_LIST and SAS_SSL_CERT_DIR have been replaced by system options of similar names. See Chapter 2, “SAS System Options for Encryption,” on page 25.

Note: In SAS 9.4M5, the SSL_USE_SNI environment variable has changed. This option now turns off SNI.

Dictionary

CAS_CLIENT_SSL_CA_LIST Environment Variable

Specifies the path and filename of the file that contains the list of trusted certificate authorities (CAs).
**Client:** Optional  
**Server:** Optional  
**Valid in:** Server configuration file, cas.settings file, and operating system command line  
**Category:** System Administration: Security  
**Operating environment:** LINUX  
**Notes:** This environment variable is available in SAS 9.4M5. This environment variable is used by CAS client, Lua client, Python client, CAS server, SAS 9.4 client.

**Example:**
```
export CAS_CLIENT_SSL_CA_LIST='/opt/sas/viya/config/etc/SASSecurityCertificateFramework/cacerts/vault-ca.crt'
```

---

### Syntax

\[
\text{CAS\_CLIENT\_SSL\_CA\_LIST}=\]

### Syntax Description

\[
\text{CAS\_CLIENT\_SSL\_CA\_LIST='path/certificate-file'}
\]

Specifies the path and filename of the file that contains the list of trusted certificate authorities (CAs). This environment variable can be used by the CAS server or by the client (SAS 9.4 can act as a SAS Viya client) connecting to the CAS server. For the server, this environment variable points to the trust list used to accept connections to the server. For the client, this environment variable points to the trust list that the client uses to connect to the server.

For details about how to manage client certificates between SAS 9.4 and SAS Viya, see Configure SAS 9.4 Clients to Work with SAS Viya.

---

### SAS\_SSL\_MIN\_PROTOCOL= Environment Variable

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

**Client:** Optional  
**Server:** Optional  
**Valid in:** Configuration file, SAS invocation, OPTIONS statement, System Options Window  
**Categories:** Communications: Networking and Encryption  
**System Administration: Security**  
**Operating environment:** UNIX, LINUX, z/OS, and Windows  
**Notes:** This environment variable is used to set the TLS protocol for SAS software releases prior to SAS 9.4M5 to the supported TLS version 1.2. This environment variable was introduced in SAS 9.4M3 and is available in prior SAS software releases by applying security hot fixes.

This environment variable must be set before TLS is loaded. It cannot be changed after TLS is loaded. You must set the environment variable before the SAS/CONNECT spawner is started and before SAS is started on the client.
Tip: You can also define SET commands for Windows by using the System Properties dialog box that you access from the Control Panel.

See: “Defining Environment Variables in UNIX Environments” in SAS Companion for UNIX Environments and “TKMVSENV File” in SAS Companion for z/OS

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

```
export SAS_SSL_MIN_PROTOCOL=TLS1.2
```

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

```
SETENV SAS_SSL_MIN_PROTOCOL TLS1.2
```

Set the environment variable at SAS invocation for UNIX hosts:

```
-set "SAS_SSL_MIN_PROTOCOL=TLS1.2"
```

Set the environment variable on Windows hosts

```
SET SAS_SSL_MIN_PROTOCOL=TLS1.2
```

Syntax

```
SAS_SSL_MIN_PROTOCOL= protocol
"SAS_SSL_MIN_PROTOCOL= protocol"
SAS_SSL_MIN_PROTOCOL protocol
```

Syntax Description

**protocol**

specifies the TLS protocol version that can be negotiated between UNIX, z/OS, and Windows servers when using OpenSSL for SAS releases prior to SAS 9.4M5. Because protocols earlier than TLS 1.2 are insecure, set this environment variable to TLS1.2 or TLSv1.2 for SAS 9.4M4 and earlier releases of SAS.

*Note:* To use the SAS_SSL_MIN_PROTOCOL environment variable prior to SAS 9.4M3, you must apply security hot fixes. Starting in SAS 9.4M5, this environment variable is no longer needed to set the TLS.

For TLS 1.2, you can specify values TLS1.2 and TLSv1.2. These other values can be specified, but are insecure: SSL3, SSLV3, TLS, TLS1, TLSV1, TLS1.0, TLSV1.0, TLS1.1, and TLSV1.1.

See SAS Security Bulletin on OpenSSL for the most current information about the versions of OpenSSL used in SAS products and about the advisories under consideration. For a quick reference, see Table 1.1 on page 14.

**CAUTION:**

It is highly recommended that you use TLS 1.2 or above. Versions prior to TLS 1.2 have known security vulnerabilities.

*Note:* A message is written to the SAS log when an invalid value is specified.

Details

The SAS_SSL_MIN_PROTOCOL= environment variable enables you to set a minimum TLS protocol that will be negotiated. During the first TLS handshake attempt, the highest supported protocol version is offered. If this handshake fails, earlier protocol versions are offered instead. Only TLS versions 1.2 and above are secure.
SAS_VIYA_TOKEN Environment Variable

Enables you to pre-load your token before starting your SAS session.

- **Client:** Optional
- **Server:** Optional
- **Valid in:** SAS invocation, OPTIONS statement
- **Categories:**
  - Communications: Networking and Encryption
  - System Administration: Security
- **Restriction:** Access tokens expire every 12 hours.
- **Interaction:** This environment variable is used with the SERVICESBASEURL= system option.
- **Operating environment:** LINUX

**Note:** This environment variable is available in SAS 9.4M5.

**Example:** Export the environment variable into your shell:

```bash
options set=SAS_VIYA_TOKEN="access_token"
```

**Syntax**

```
SAS_VIYA_TOKEN=
```

**Syntax Description**

```
SAS_VIYA_TOKEN="access_token"
```

specifies the OAuth token that is needed for a user to access SAS Viya services. This environment variable is used to load your access_token. Before starting SAS, assign the value of the access_token to the SAS_VIYA_TOKEN environment variable in the shell where SAS is executed. You can also assign the access token by specifying `options set=SAS_VIYA_TOKEN="access_token"` in your SAS program.

The SAS_VIYA_TOKEN environment variable must be set before any calls to SAS Viya are attempted. You cannot change the value of the SAS_VIYA_TOKEN environment variable after using the token. When the token expires or when you need a different token, start a new SAS session.

**Note:** This environment variable is not needed for SAS 9.4 code that is executed on a SAS Viya compute server using the compute service. The user’s OAuth token is automatically provided.

**Details**

The SAS_VIYA_TOKEN environment variable is used to provide access to SAS Viya services. In order to access the following services, you need to specify a token using this environment variable:

- access SAS Viya credentials services from SAS 9.4. When the AUTHDOMAIN= option is set in the CAS statement or in a LIBNAME statement, an attempt is first made to retrieve credentials from the Credentials service before searching the metadata. For more information, see the AUTHDOMAIN option in the CAS
statement. Refer to the server documentation that you are connecting to for the
details of using the AUTHDOMAIN option in the LIBNAME statement.

- store and retrieve files within the file service in the SAS Viya system using the
  FILESRVC Access Method FILENAME Statement. See “FILENAME Statement,

Providing the token and setting the SERVICESBASEURL= system option enables some
SAS system capabilities to use SAS Viya services. For more information, see

SSL_USE_SNI Environment Variable

Disables the use of Server Name Indication (SNI) in the TLS handshake for the client.

| Client: | Optional |
| Server: | Optional |
| Valid in: | SAS invocation, configuration file |
| Categories: | Communications: Networking and Encryption System Administration: Security |
| Default: | By default, the TLS SNI extension is sent as part of the TLS handshake. |
| Restrictions: | In SAS 9.4M5, the SAS system option SSLSNIHOSTNAME is used to specify the Server Name Indication (SNI) that identifies the server name that it is trying to connect to. This environment variable is now used to turn off SNI which is sent by default. See Chapter 2, “SAS System Options for Encryption,” on page 25 for details. The SSL_USE_SNI environment variable is supported only on UNIX. |
| Operating environment: | UNIX, LINUX |

See: “Defining Environment Variables in UNIX Environments” in SAS Companion for UNIX Environments

Examples:

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

```bash
export SSL_USE_SNI=1
```

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

```bash
SETENV SSL_USE_SNI
```

Set the environment variable at SAS invocation for UNIX hosts:

```bash
sas -dms -set SSL_USE_SNI
```

Syntax

SSL_USE_SNI

Syntax Description

SSL_USE_SNI

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

SNI is enabled by default on UNIX. To disable SNI, specify the SSL_USE_SNI environment variable.

Details

The client uses SNI in the TLS handshake to identify the server name that it is trying to connect to. When making a TLS connection, the client requests a digital certificate from the web server. After the server sends the certificate, the client examines it and compares the name that it was trying to connect to with the name or names included in the certificate. If a match is found, the connection proceeds as normal.

See Also

For more information, see Chapter 10, “Troubleshooting,” on page 133.
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

Example 1: Encoding a Password

Example 2: Using an Encoded Password in a SAS Program

Example 3: Saving an Encoded Password to the Paste Buffer

Example 4: Specifying Method= SAS003 to Encode a Password

Example 5: Specifying Method= SAS005 to Encode a Password
the tag and decode the string before using it. Encoding a password enables you to write SAS programs without having to specify a password in plaintext.

Note: PROC PWENCODE passwords can contain up to a maximum of 512 characters, which include alphanumeric characters, spaces, and special characters. Data set passwords, however, must follow SAS naming rules. For information about SAS naming rules, see “Rules for Most SAS Names” in SAS Language Reference: Concepts.

The encoded password is never written to the SAS log in plain text. Instead, each character of the password is replaced by an X in the SAS log.

**Encoding versus Encryption**

Encoding techniques disguise passwords and the approach is intended to prevent casual, non-malicious viewing of passwords. With encoding, one character set is translated to another character set through some form of table lookup.

Encryption, by contrast, involves the transformation of data from one form to another through the use of mathematical operations and, usually, a "key" value. Encryption is generally more difficult to break than encoding. Several options for PROC PWENCODE designate encryption techniques that align with industry standards. These options support longer encryption keys (for example, 256-bit). Salting and multiple iterations are provided to the AES encryption algorithm to create passwords that are harder to break.

Encoding methods for PROC PWENCODE are now SAS001 – SAS005. Starting in SAS 9.4M5, PROC PWENCODE provides stronger password protection using the SAS005 method of encoding.

Password protection is an important part of your security strategy, but you should not rely only on password protection for all your data security needs; a determined and knowledgeable attacker can break passwords. Data should also be protected by other security controls such as file system permissions, other access control mechanisms, and encryption of data at rest and in transit.

**Encoding Methods**

Starting in SAS 9.4M5, the SAS005 method for encoding passwords is added. When SAS005 is specified for PROC PWENCODE, a more secure 256-bit fixed key is generated. SAS005, like SAS004, uses a 256-bit fixed key plus a 64-bit random salt. However, it is hashed for additional iterations.

<table>
<thead>
<tr>
<th>Encoding Method</th>
<th>Uses Data Encryption Algorithm</th>
<th>Encoded Password/key Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sas001</td>
<td>None</td>
<td>Uses base64 to encode passwords.</td>
</tr>
<tr>
<td>sas002, which can also be specified as sasenc</td>
<td>SASProprietary, which is included in SAS software.</td>
<td>Uses a 32-bit fixed key.</td>
</tr>
<tr>
<td>sas003</td>
<td>AES (Advanced Encryption Standard), which is supported in SAS/SECURE.</td>
<td>Uses a 256-bit fixed key plus a 16-bit random salt value.</td>
</tr>
</tbody>
</table>
Encoding Method | Uses Data Encryption Algorithm | Encoded Password/key Description
--- | --- | ---
**sas004** | AES (Advanced Encryption Standard), which is supported in SAS/SECURE. | Uses a 256-bit fixed key and a 64-bit random salt value.

**sas005** | AES (Advanced Encryption Standard), which is supported in SAS/SECURE. | Uses a 256-bit fixed key, a 64-bit random salt value, and is hashed for additional iterations.

**Note:** The METHOD= option supports the SAS003, SAS004, and SAS005 values, but only if you have SAS/SECURE. SAS/SECURE enables you to protect data through the use of industry-standard encryption and hashing algorithms. Shipment of SAS/SECURE is restricted by some countries. SAS Proprietary encoding is available with all SAS software. For more information, see SAS/SECURE on page 8.

---

### Syntax: PWENCODE Procedure

```sas
PROC PWENCODE IN='password' <OUT=fileref> <METHOD=encoding-method>;
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Task</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC PWENCODE</td>
<td>Encode a password</td>
<td>Ex. 1, Ex. 2, Ex. 3, Ex. 4</td>
</tr>
</tbody>
</table>

---

### PROC PWENCODE Statement

Encodes a password.

**Examples:**

"Example 1: Encoding a Password" on page 63
"Example 2: Using an Encoded Password in a SAS Program" on page 63
"Example 3: Saving an Encoded Password to the Paste Buffer" on page 65
"Example 4: Specifying Method= SAS003 to Encode a Password" on page 66
"Example 5: Specifying Method= SAS005 to Encode a Password" on page 67

**Syntax**

```sas
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

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 63

“Example 2: Using an Encoded Password in a SAS Program” on page 63

“Example 3: Saving an Encoded Password to the Paste Buffer” on page 65

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 63

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

• SAS001
• SAS002
• SAS003
• SAS004
• SAS005

The SAS003, SAS004, and SAS005 encoded passwords uses a 256-bit fixed key plus a random salt value is applied to the encoding method. Therefore, each time you use PROC PWENCODE to encode the same password, you get a different encoded password, because the salt values are random.

For more information about each of these encoding methods, see “Encoding Methods” on page 60.

Note: The METHOD= option supports the SAS003, SAS004, and SAS005 values, but only if you have SAS/SECURE. SAS/SECURE enables you to protect data through the use of industry-standard encryption and hashing algorithms. Shipment of SAS/SECURE is restricted by some countries. SAS Proprietary encoding is available with all SAS software. For more information, see SAS/SECURE on page 8.

If the METHOD= option is omitted, the default encoding method is used. The default method is sas002 in most cases. SAS002 is also the default method used if you specify an invalid method.
When the FIPS 140-2 compliance option, -encryptfips, is specified, the encoding method defaults to **sas003**. For more information about FIPS, see “FIPS 140-2 Standards Compliance” on page 5.

**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**

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

**Program Description**

**Encode the password.**

```sas
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

**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.

Example 4: Specifying Method= SAS003 to Encode a Password

Features: METHOD= argument

Details
This example shows a simple case of encoding a password using the SAS003 encoding method and writing the encoded password to the SAS log. SAS003 uses a 16-bit salt to encode a password.

Program
proc pwencode in='mypassword' method=sas003;
run;
Program Description

**Encode the password using SAS003.** The encoded password is a 256-bit key with a 16 bit random salt.

```sas
proc pwencode in='mypassword' method=sas003;
run;
```

Log

Note that each character of the password is replaced by an X in the SAS log. SAS003 encoding uses AES encryption plus a 16-bit salt. Because SAS003 uses random salting, each time you run the following code, a different password is generated.

```
8   proc pwencode in=XXXXXXXXXXXXX method=sas003;
29   run;

[SAS003]4837B146585CED2C9FED14A3C946D68E4389
NOTE: PROCEDURE PWENCODE used (Total process time):  
real time           0.00 seconds  
cpu time            0.00 seconds
```

---

**Example 5: Specifying Method= SAS005 to Encode a Password**

**Features:** METHOD= argument

**Details**

This example shows a simple case of encoding a password using the **sas005** encoding method and writing the encoded password to the SAS log. SAS005 uses a 256-bit fixed key that uses a 64-bit random salt to encode the password.

**Program**

```sas
proc pwencode in='mypassword' method=sas005;
run;
```

**Program Description**

**Encode the password using SAS005.**

```sas
proc pwencode in='mypassword' method=sas005;
run;
```

**Log**

Note that each character of the password is replaced by an X in the SAS log. SAS005 encoding uses AES encryption with a 256-bit fixed key and a 64-bit random salt value. SAS005 increases security for stored passwords by using the SHA-256 hashing algorithm and is hashed for additional iterations. Because SAS005 uses
random salting, each time you run the following code, a different password is generated.

```sas
proc pwencode in=XXXXXXXXXXXX method=sas005;
run;
```

{SAS005}ADD8AB7108595A7D1A69190D78CDFE6145C1EB849CC7A43D

NOTE: PROCEDURE PWENCODE used (Total process time):
real time 0.01 seconds
cpu time 0.01 seconds
Chapter 5
Encryption Technologies: Examples

SAS Proprietary Encryption for SAS/SHARE: Example .......................... 70
  SAS/SHARE Client .................................................. 70
  SAS/SHARE Server ................................................. 70

SAS/SECURE for SAS/CONNECT: Example ............................................. 70
  SAS/CONNECT Client on UNIX ...................................... 70
  SAS/CONNECT Server on UNIX .................................... 70

TLS for a SAS/CONNECT UNIX Spawner: Example ................................. 71
  Start-up of a UNIX Spawner on a SAS/CONNECT Server ................. 71
  Connection of a SAS/CONNECT Client to a UNIX Spawner .............. 72

TLS for a SAS/CONNECT Windows Spawner: Example ............................. 73
  Start-up of a Windows Spawner on a Single-User SAS/CONNECT Server 73
  Connection of a SAS/CONNECT Client to a Windows Spawner on a SAS/CONNECT Server ................................................. 74

TLS on a z/OS Spawner on a SAS/CONNECT Server: Example .................. 75
  Start-up of a z/OS Spawner on a SAS/CONNECT Server ................. 75
  Connection of a SAS/CONNECT Client to a z/OS Spawner .............. 77

TLS for SAS/SHARE on UNIX: Example .............................................. 77
  Start-up of a Multi-User SAS/SHARE Server ........................ 77
  SAS/SHARE Client Access of a SAS/SHARE Server .................... 78

TLS for SAS/SHARE on Windows: Examples ......................................... 79
  Start-up of a Multi-User SAS/SHARE Server ........................ 79
  SAS/SHARE Client Access of a SAS/SHARE Server .................... 79

TLS for SAS/SHARE on z/OS: Example ............................................... 80
  Start-up of a Multi-User SAS/SHARE Server ........................ 80
  SAS/SHARE Client Access of a SAS/SHARE Server .................... 81

SSH Tunnel for SAS/CONNECT: Example ............................................ 81
  Start-up of a UNIX Spawner on a Single-User SAS/CONNECT Server .... 81
  Connection of a SAS/CONNECT Client to a UNIX Spawner on a SAS/CONNECT Server .................................................. 81

SSH Tunnel for SAS/SHARE: Example ............................................... 82
  Start-up of a Multi-User SAS/SHARE Server ........................ 82
  SAS/SHARE Client Access of a SAS/SHARE Server .................... 82
SAS Proprietary Encryption for SAS/SHARE: Example

SAS/SHARE Client

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

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

SAS/SHARE Server

This example shows how to set the options for encryption services on a SAS/SHARE server. The NETENCRIPT 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.

```sas
options netencrypt netencryptalgorithm=sasproprietary;
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.

```sas
options netencryptalgorithm=rc4;
options remote=unixnode comamid=tcp;
signon;
```

SAS/CONNECT Server on UNIX

The following command starts a spawner on the computer that runs the server. The -NETENCRIPT 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 startup command.

```bash
cntsspawn -service spawner -netencrypt -netencryptalgorithm rc4 -sascmd mystartup
```

The spawner executes a UNIX shell script that executes the commands to start SAS.
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
-sslprivkeyloc /users/server/certificates/serverkey.pem
-sslprivkeypass starbuck1
-sslnalistloc /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

<table>
<thead>
<tr>
<th>SAS Commands and Arguments</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNTSPAWN</td>
<td>Starts the spawner</td>
</tr>
<tr>
<td>-SERVICE unxspawn</td>
<td>Specifies the spawner service (configured in the services file)</td>
</tr>
<tr>
<td>-NETENCRYPTALGORITHM SSL</td>
<td>Specifies the TLS encryption algorithm</td>
</tr>
<tr>
<td>-SSLCERTLOC/users/server/certificates/server.pem</td>
<td>Specifies the file path for the location of the server's public certificate</td>
</tr>
<tr>
<td>-SSLPRVKEYLOC/users/server/certificates/serverkey.pem</td>
<td>Specifies the file path for the location of the server's private key</td>
</tr>
<tr>
<td>-SSLPRVKEYPASS password</td>
<td>Specifies the password to access the server's private key if the private key is encrypted with a password</td>
</tr>
</tbody>
</table>
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 *SAS/CONNECT User’s Guide*.

**Connection of a SAS/CONNECT Client to a UNIX Spawner**

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

The following example shows how to connect a client to a spawner that is running on a SAS/CONNECT server:
```
options netencryptalgorithm=ssl;
options sslcalistloc="/users/johndoe/certificates/sas.pem";
%let machine=unixspawn;
signon machine.spawner user=_prompt_;
```

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

<table>
<thead>
<tr>
<th>SAS Options, Statements, and Arguments</th>
<th>Client Access Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>NETENCRYPTALGORITHM=SSL</td>
<td>Specifies the encryption algorithm</td>
</tr>
</tbody>
</table>
SAS Options, Statements, and Arguments

SSLCALISTLOC=sas.pem
Specifies the CA trust list

SIGNON=unxs_spawn
Specifies the server and service to connect to

USER=_PROMPT_
Prompts for the user ID and password to be used for authenticating the client to the server

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

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

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>
<thead>
<tr>
<th>SAS Command and Arguments</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNTSPAWN</td>
<td>Starts the spawner.</td>
</tr>
<tr>
<td>-INSTALL</td>
<td>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.</td>
</tr>
<tr>
<td>-NETENCRYPTALGORITHM SSL</td>
<td>Specifies the TLS encryption algorithm.</td>
</tr>
<tr>
<td>-SSLCERTSUBJ &quot;apex.pc.com&quot;</td>
<td>Specifies the subject name that is used to search for a certificate from the Microsoft Certificate Store.</td>
</tr>
</tbody>
</table>
**SAS Command and Arguments**

<table>
<thead>
<tr>
<th>Function</th>
<th>SAS Command and Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifies the name of an executable file that starts a SAS session when you sign on without a script file.</td>
<td>-SASCMD <em>mysas.bat</em></td>
</tr>
<tr>
<td>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.</td>
<td>-SERVUSER <em>user-ID</em></td>
</tr>
<tr>
<td>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.</td>
<td>-SERVPASS <em>password</em></td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

```sas
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**

<table>
<thead>
<tr>
<th>SAS Options, Statements, and Arguments</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>NETENCRYPTALGORITHM=SSL</td>
<td>Specifies the encryption algorithm</td>
</tr>
<tr>
<td>SIGNON=server-ID</td>
<td>Specifies which server to connect to</td>
</tr>
<tr>
<td>USER=<em>PROMPT</em></td>
<td>Prompts for the user ID and password to be used for authenticating the client to the server</td>
</tr>
</tbody>
</table>

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'
//STPLIB   DD  DISP=SHR,DSN=<customer.high.level.pfx>.LIBRARY
//STPLIB   DD  DISP=SHR,DSN=<customer.high.level.pfx>.LIBE
//SYSPRINT DD  SYSOUT=*                          
//SYSTYPEM DD  SYSOUT=*                          
//TKMVJSNL DD  SYSOUT=*                          
//SYSPRINT DD  SYSOUT=*                          
//SYSPRINT DD  SYSOUT=*                          
//SIN      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

<table>
<thead>
<tr>
<th>SAS Commands and Arguments</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNTSPAWN</td>
<td>Starts the spawner</td>
</tr>
<tr>
<td>-SERVICE 4321</td>
<td>Specifies the spawner service that is listening on port 4321</td>
</tr>
<tr>
<td>- NETENCRIPTALGORITHM SSL</td>
<td>Specifies the TLS encryption algorithm</td>
</tr>
<tr>
<td>-SSLPKCS12LOC /users/server/certificates/serverkey.p12</td>
<td>Specifies the file path for the location of the server's PKCS #12 DER encoding package</td>
</tr>
<tr>
<td>-SSLPKCS12PASS password</td>
<td>Specifies the password to access the server's private key in the PKCS #12 package</td>
</tr>
<tr>
<td>-SSLCALISTLOC /users/server/certificates/sas.pem</td>
<td>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.</td>
</tr>
<tr>
<td>-SASCMD /users/server/command.sh</td>
<td>Specifies the name of an executable file that starts a SAS session when you sign on without a script file</td>
</tr>
</tbody>
</table>

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`:

```bash
#!/bin/sh
args=$*
if [ -n "$NETENCRALG" ] ; then
   args="$args -netencralg $NETENCRALG"
fi
if [ -n "$SASDAEMONPORT" ] ; then
   args="$args -sasdaemonport $SASDAEMONPORT"
fi
if [ -n "$SASCLIENTPORT" ] ; then
   args="$args -sasclientport $SASCLIENTPORT"
fi
export TSOOUT=
export SYSPROC=SAS.CLIST
/bin/tso -t %sas -dmr -noterminal
-sslpkcs12loc /users/server/certificates/serverkey.p12
-sslpkcs12pass password $args
```

For complete information about starting a z/OS spawner, see *SAS/CONNECT User’s Guide*. 
Connection of a SAS/CONNECT Client to a z/OS Spawner

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

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

```
options 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>
<thead>
<tr>
<th>SAS Options and Arguments</th>
<th>Client Access Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMAMID=TCP</td>
<td>Specifies the TCP/IP access method</td>
</tr>
<tr>
<td>NETENCRYPTALGORITHM=SSL</td>
<td>Specifies the encryption algorithm</td>
</tr>
<tr>
<td>SSLCALISTLOC=sas.pem</td>
<td>Specifies the CA trust list</td>
</tr>
<tr>
<td>SIGNON=server-ID.service</td>
<td>Specifies the server and service to connect to</td>
</tr>
<tr>
<td>USER=<em>PROMPT</em></td>
<td>Prompts for the user ID and password to be used for authenticating the client to the server</td>
</tr>
</tbody>
</table>

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

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

TLS for SAS/SHARE on UNIX: Example

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

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

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

```
%let tcpsec=_secure_
options netencryptalgorithm=ssl;
options sslcertloc="/users/johndoe/certificates/server.pem";
options sslpvtkeyloc="/users/johndoe/certificates/serverkey.pem";
options sslpvtkeypass="password";
proc server id=shrserv;
run;
```
The following table lists the SAS option or statement that is used for each task to start a server.

**Table 5.7  SAS Options and Statements for Server Start-Up Tasks**

<table>
<thead>
<tr>
<th>SAS Options and Statements</th>
<th>Server Start-Up Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPSEC= <em>SECURE</em></td>
<td>Secures the server</td>
</tr>
<tr>
<td>NETENCRLG=SSL</td>
<td>Specifies TLS as the encryption algorithm</td>
</tr>
<tr>
<td>SSLCERTLOC=server.pem</td>
<td>Specifies the filepath for the location of the server's certificate</td>
</tr>
<tr>
<td>SSLPVTKKEYLOC=serverkey.pem</td>
<td>Specifies the filepath for the location of the server's private key</td>
</tr>
<tr>
<td>SSLPVTKKEYPASS=&quot;password&quot;</td>
<td>Specifies the password to access server's private key</td>
</tr>
<tr>
<td>PROC SERVERID=shrserv</td>
<td>Starts the server</td>
</tr>
</tbody>
</table>

*Note:* As an alternative to using the SSLPVTKKEYPASS= 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:

```plaintext
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**

<table>
<thead>
<tr>
<th>SAS Options and Arguments</th>
<th>Client Access Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLCALISTLOC=cacerts.pem</td>
<td>Specifies the CA trust list</td>
</tr>
<tr>
<td>SERVER=machine.shrserv</td>
<td>Specifies the machine and server to connect to</td>
</tr>
<tr>
<td>USER=<em>PROMPT</em></td>
<td>Prompts for the user ID and password to be used for authenticating the client to the server</td>
</tr>
</tbody>
</table>
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-User SAS/SHARE Server**

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

```sas
%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>
<thead>
<tr>
<th>SAS Options, Statements, and Arguments</th>
<th>Server Start-Up Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPSEC= <em>SECURE</em></td>
<td>Secures the server</td>
</tr>
<tr>
<td>COMAMID=TCP</td>
<td>Specifies the TCP/IP access method</td>
</tr>
<tr>
<td>NETENCRALG=SSL</td>
<td>Specifies TLS as the encryption algorithm</td>
</tr>
<tr>
<td>SSLCERTISS=&quot;Glenn's CA&quot;</td>
<td>Specifies the name of the issuer of the digital certificate that TLS should use</td>
</tr>
<tr>
<td>SSLCERTSERIAL=&quot;0a1dcfa30000000000015&quot;</td>
<td>Specifies the serial number of the digital certificate that TLS should use</td>
</tr>
<tr>
<td>PROC SERVERID=shrserv;</td>
<td>Starts the server</td>
</tr>
</tbody>
</table>

**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:

```sas
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

<table>
<thead>
<tr>
<th>SAS Options and Arguments</th>
<th>Client Access Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMAMID=TCP</td>
<td>Specifies the TCP/IP access method</td>
</tr>
<tr>
<td>SERVER=machine.shrserv</td>
<td>Specifies the machine and server to connect to</td>
</tr>
<tr>
<td>USER=<em>PROMPT</em></td>
<td>Prompts for the user ID and password to be used for authenticating the client to the server</td>
</tr>
</tbody>
</table>

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-User SAS/SHARE Server

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

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

```sas
%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

<table>
<thead>
<tr>
<th>SAS Options, Statements, and Arguments</th>
<th>Server Start-Up Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPSEC= <em>SECURE</em></td>
<td>Secures the server</td>
</tr>
<tr>
<td>NETENCRALG=SSL</td>
<td>Specifies TLS as the encryption algorithm</td>
</tr>
<tr>
<td>SSLPKCS12LOC=server.p12</td>
<td>Specifies the filepath for the location of the server's private key</td>
</tr>
<tr>
<td>SSLPKCS12PASS=&quot;password&quot;</td>
<td>Specifies the password to access server's private key</td>
</tr>
<tr>
<td>PROC SERVERID=shrserv</td>
<td>Starts the server</td>
</tr>
</tbody>
</table>
**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:

```plaintext
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>
<thead>
<tr>
<th>SAS Options and Arguments</th>
<th>Client Access Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLCALISTLOC=cacerts.pem</td>
<td>Specifies the CA trust list</td>
</tr>
<tr>
<td>SERVER=machine.shrserv</td>
<td>Specifies the machine and server to connect to</td>
</tr>
<tr>
<td>USER=<em>PROMPT</em></td>
<td>Prompts for the user ID and password to be used for authenticating the client to the server</td>
</tr>
</tbody>
</table>

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.

```plaintext
cntspawn -service 4321
```

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

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

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

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

```plaintext
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.

```sas
%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-User SAS/SHARE Server**

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

```sas
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:

```bash
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.

```sas
%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 .................................................. 85

Chapter 7
Installing and Configuring TLS and Certificates on UNIX ......... 93

Chapter 8
Installing and Configuring TLS and Certificates on Windows ...... 115

Chapter 9
Installing and Configuring TLS and Certificates on z/OS ........ 125
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.
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.509 v3 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 104, “Convert between PEM and DER File Formats for TLS” on page 121.

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 104, “Convert between PEM and DER File Formats for TLS” on page 121.

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 CA Certificate Management Using the SAS Deployment Manager

Starting in SAS 9.4M3, a bundle of root digital certificates is provided to get TLS up and working at SAS installation. SAS provides a bundle of certificates from Mozilla that can be used as the default trust provider 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 106.

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

Note: In SAS 9.4M2 and earlier, when providing your own signed certificates, you must add the CA root and intermediate certificates to the SAS Private JRE using the Java keytool -importcert command. See “Add Your Certificates to the SAS Private JRE” on page 110.

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 Store” on page 119.

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 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 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.

---

**Best Practices When Generating Certificates**

SAS recommends the following best practices when you are creating certificates and when managing certificates and securing your private keys.

- When generating new certificates, provide the following information for the certificate signing request.
  - Provide fixed host names.
  - Provide fully qualified domain names (FQDN).
  - Provide subject alternative names (SAN), including IP addresses.

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

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

- Use algorithms supported by TLS 1.2 when generating certificates. In SAS 9.4M6, Java 8 is supported. Java 8 supports TLS 1.2. SAS Deployment Manager checks to ensure that the CA certificates being added comply with the supported TLS 1.2 algorithms. For more information, see *The Transport Layer Security (TLS) Protocol Version 1.2* and *Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*.

- 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.

- If you are using SAS 9.4 to start a SAS/CONNECT session to a SAS Viya CAS server, the server certificate needs subject alternative name (SAN) extension entries in the server certificate for each name the host can be known by. The certificate needs the physical host name and DNS alias listed in the SAN.

- For Windows, if you are providing a certificate to enable TLS on a JAVA client, provide a certificate in a PFX formatted file. The file contains the following information.
  - The PFX file has a private key embedded within it.
  - The private key within the PFX file is protected with a password.
  - The PFX file contains all certificates in the certification path (the PFX file contains the certificates that comprise the CA chain).

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

- If your custom root certificate is site-signed or is not already included in the Mozilla bundle of trusted CA certificates, then you need to add the root certificate to the trustedcerts files. See “Add Your Certificates to the Trusted CA Bundle” on page 106.
On UNIX, you should also place a copy of the root certificate that you are adding to the trustedcerts files in the same directory. The root certificate should be Base 64 encoded and have a .crt file extension.

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

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

- Encrypt your private key when possible.
- Password-protect your private key file.

---

**Certificate Implementation: How TLS Client and Servers Negotiate**

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

- TLS needs public and private key pairs. The server sends its public key to the client. The client can then send its public key to the server. However, the private key is never sent anywhere.

- Public keys are stored in files commonly called certificates and private keys are stored in files commonly called keys. TLS uses certificates to describe the public and private key pairs to use. TLS uses certificates defined by the X.509 standard. These certificates contain information that includes the subject (usually the host name) and the Public Key Signature signed by a Certificate Authority (CA).

Certificates come in PEM, DER, and PKCS12 file formats. For more details, see “Certificate File Formats” on page 86.

- 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 32.

  • For UNIX, all certificates must be in an OpenSSL CA certificates directory pointed to by the SSLCACERTDIR= system option. The layout of this directory is specified by OpenSSL, where the certificates are in PEM format and referenced by their hash values.

  See “SSLCACERTDIR= System Option” on page 34.

• 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 or Linux.
• Internet access and a web browser.

Certificate Locations

Preparation for Setting Up Digital Certificates

Generate Digital Certificates Using OpenSSL
Step 1. Generate a New RSA Private Key and Certificate Signing Request (CSR) in PEM Format
Step 2 (Optional). Generate a Public Certificate from an Existing Certificate
Step 3. Secure Your Private Key File
Step 4. Check Your Digital Certificate Using OpenSSL
Step 5. Create a Certificate Chain in PEM Format Using OpenSSL
Step 6. Verify Certificates in the Trust Chain Using OpenSSL
Step 7. End OpenSSL

Convert between PEM and DER File Formats Using OpenSSL

Convert PEM Files to a PFX File Using OpenSSL

Manage Certificates in the Trusted CA Bundle Using the SAS Deployment Manager
Overview
Add Your Certificates to the Trusted CA Bundle
Remove Your Certificates from the Trusted CA Bundle
SAS Deployment Manager Criteria for Validating Certificates

Add Your Certificates to the SAS Private JRE

How Clients and Servers Validate Certificates

Building FIPS 140-2 Capable OpenSSL on UNIX

- 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.

- In SAS 9.4M6, Java 8 is supported. Java 8 supports TLS 1.2. When using the SAS Deployment Manager to add CA certificates, the certificates supplied must use algorithms supported by TLS 1.2. For more information, see The Transport Layer Security (TLS) Protocol Version 1.2 and Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.

- access to the SAS Deployment Manager if you plan to add digital certificates when you install SAS 9.4M3 and later. 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.
  - Your site administrator plans to generate a site-signed certificate and private key.
  - You plan to build your own OpenSSL libraries.
  - You plan to build FIPS 140-2 capable OpenSSL libraries.

Note: The SAS 9.4 versions of OpenSSL provided by SAS are not FIPS compliant. Refer to “Building FIPS 140-2 Capable OpenSSL on UNIX” on page 113 for details.

Certificate Locations

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

Certificates must be in one of the following locations:

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

  In SAS 9.4M3, 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 points to <SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem. The file where the trusted certificates reside is named trustedcerts.pem. For syntax, see “SSLCAListLoc= System Option” on page 32.

- On UNIX, all certificates must be in an OpenSSL CA certificates directory pointed to by the SSLCACERTDIR= system option. The layout of this directory is specified by OpenSSL where the certificates are in PEM format.
If you use the SSLCACERTDIR= system option, create a link to the file named after the certificate’s hash value.

```bash
$ 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= System Option” on page 34.

---

### 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 SAS 9.4M3, the process for adding certificates to the trusted CA list has been made easier. You can 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 106.

Here is information that you might need to add digital certificates for use with SAS 9.4.

- 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.
- (Optional) Create an openssl.cnf file. This file stores the locations of your CA keys. For a partial example of this file, see Figure 7.1 on page 97.
- To prepare the certificate(s) to add to the trusted CA bundle, you can, as needed, create a root certificate. See “Generate Digital Certificates Using OpenSSL” on page 95.
- To prepare the certificate(s) to add to the trusted CA bundle, you can, as needed, create a certificate trust list and verify the trust list. See “Step 5. Create a Certificate Chain in PEM Format Using OpenSSL” on page 102 and “Step 6. Verify Certificates in the Trust Chain Using OpenSSL” on page 103.
- Use best practices when generating certificates on page 89.
- 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 106.

---

### Generate Digital Certificates Using OpenSSL

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

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

In the OpenSSL.cnf file shown below in one of the OpenSSL examples, Proton, Inc. is the organization that is applying to become a CA. A certificate request is sent to a certificate authority to get it signed, thereby becoming a CA. After Proton, Inc. becomes a CA, it can serve as a CA for issuing other digital certificates to clients and servers on its network. The CA’s role is to accept certificate applications, authenticate applications, issue certificates, and maintain status information about certificates issued. The certificates generated by the Proton, Inc. CA are considered site-signed certificates.

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

1. In this example, we are using an OpenSSL configuration file.

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

   Edit your existing openssl.cnf file or create an openssl.cnf file. OpenSSL by default looks for a configuration file in /usr/lib/ssl/openssl.cnf. It is 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 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**

```bash
#  # OpenSSL example configuration file.
#  # This is being used for generation of certificate requests.
#
#
[ default_ca ] = CA_default  # The default ca section

# Where everything is kept
dir = ./demoCA
certs = $dir/certs

crl_dir = $dir/crl

database = $dir/index.txt

# database index file.
new_certs_dir = $dir/newcerts

# default place for new certs.
certificate = $dir/cacert.pem
serial = $dir/serial

crl = $dir/crl.pem

# The private key
private_key = $dir/private/cakey.pem

# private random number file
RANDFILE = $dir/private/.rand

# The extensions to add to the cert
x509_extensions = usr_cert

default_days = 365

default_crl_days = 30

# how long to certify for
# how long before next CRL
# 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

organizationUnitName = optional
```

2. Select the directory where OpenSSL was built.

3. Initialize OpenSSL.

   ```bash
   $ openssl
   ```

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

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

   in the OpenSSL command.

Here are a few examples of generating a CSR:

- In this example, we are generating a self-signed CA certificate with subject alternative names. The request creates a private key, from which it generates a Certificate Signing Request and signs it with the private key.

  ```bash
  openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
  ```
In this example, we are generating a certificate signing request (CSR) and 2048-bit RSA key file that we will then send to a certificate signing authority to be signed. This example is a server certificate.

```bash
```

In this example, we are generating a certificate signing request (CSR) and 2048-bit RSA key file. This example uses an openssl.conf file. any extensions are specified in that file.

```bash
```

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

**Table 7.1 Arguments and Values Used in OpenSSL Commands**

<table>
<thead>
<tr>
<th>OpenSSL Arguments and Values</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>req</td>
<td>Requests a certificate.</td>
</tr>
<tr>
<td>-config ./openssl.cnf</td>
<td>Specifies the name and location of the OpenSSL configuration file. You can use it to specify extensions, basic constraints, and many other options.</td>
</tr>
<tr>
<td>-new</td>
<td>Identifies the request as new.</td>
</tr>
<tr>
<td>-out customer.csr</td>
<td>Specifies the location of the certificate signing request.</td>
</tr>
<tr>
<td>-newkey rsa:2048</td>
<td>Generates a new private key along with the certificate request that is 2048 bits in length using the RSA algorithm.</td>
</tr>
<tr>
<td>-keyout customer.key</td>
<td>Specifies the storage location for the private key.</td>
</tr>
<tr>
<td>-nodes</td>
<td>Prevents the private key from being encrypted.</td>
</tr>
</tbody>
</table>

*Note:* SAS recommends that you encrypt the private key.
OpenSSL Arguments and Values

<table>
<thead>
<tr>
<th>Functions</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>-sha256</td>
<td>Specifies that the SHA256 hash algorithm be used. Use SHA256 for FIPS 140-2. Without this option, the default is SHA-1.</td>
</tr>
<tr>
<td>-addext</td>
<td>Used to add extensions to the OpenSSL request. Some of the extensions can be Subject Alternative Names (SAN), basic constraints, and certificate policies. For more information, see Supported Extensions.</td>
</tr>
</tbody>
</table>

5. When you submit your CSR, informational messages are displayed and prompts for additional information appear according to the specific request.

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

*Note:* Unless the -NODES option is used in the OpenSSL command when creating a digital certificate request, OpenSSL prompts you for a password before allowing access to the private key. It is highly recommended that you supply a password to help protect the private key.

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

```
OpenSSL> req -config ./openssl.cnf -new -out customer.req -newkey rsa:2048 -keyout customer.key -nodes
Using configuration from ./openssl.cnf
Generating a 2048 bit RSA private key
............................+++++
..........................................+++++
writing new private key to 'customer.key'
-----
You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [North Carolina]:
Locality Name (city) [Cary]:
Organization Name (company) [Proton Inc.]:
Organizational Unit Name (department) [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 request 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 7.2 OpenSSL Commands for Generating Digital Certificates on UNIX

<table>
<thead>
<tr>
<th>Generate Certificate for</th>
<th>OpenSSL Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>x509 -req -in customer.csr -signkey customer.key -out customer.pem -sha256</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> This command generates a self-signed certificate.</td>
</tr>
<tr>
<td>Server</td>
<td>ca -config ./openssl.cnf -in server.csr -out server.pem -md sha256</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> This command creates certificates signed by the CA. These are defined in the openssl.cnf file.</td>
</tr>
<tr>
<td>Client</td>
<td>ca -config ./openssl.cnf -in client.csr -out client.pem -md sha256</td>
</tr>
<tr>
<td></td>
<td><em>Note:</em> This command creates certificates signed by the CA. These are defined in the openssl.cnf file.</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>OpenSSL Arguments and Values</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>x509</td>
<td>Identifies the certificate display and signing utility.</td>
</tr>
<tr>
<td>-req</td>
<td>Specifies that a certificate be generated from the request.</td>
</tr>
<tr>
<td>ca</td>
<td>Identifies the Certificate Authority utility.</td>
</tr>
<tr>
<td>-config ./openssl.cnf</td>
<td>Specifies the location of the openssl.cnf file and where the OpenSSL utility is located.</td>
</tr>
<tr>
<td>-in filename.csr</td>
<td>Specifies the location of the input file for the certificate request.</td>
</tr>
</tbody>
</table>
Generate Digital Certificates Using OpenSSL

### OpenSSL Arguments and Values

<table>
<thead>
<tr>
<th>Argument</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-out filename.pem</code></td>
<td>Specifies the location of the certificate file in pem format.</td>
</tr>
<tr>
<td><code>-signkey cakey.pem</code></td>
<td>Specifies the private key that is used to sign the certificate that is generated by the certificate request.</td>
</tr>
<tr>
<td><code>-md sha256</code></td>
<td>Specifies that the SHA256 hash algorithm be used. Use SHA256 for FIPS 140-2. Without this option, the default is SHA-1.</td>
</tr>
</tbody>
</table>

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:

```bash
eopenssl rsa -aes256 -in /tmp/customer.pem -out /tmp/customer.key
```
OpenSSL prompts for the password to use on the private key file.

**Step 4. Check Your Digital Certificate Using OpenSSL**

To check a digital certificate, issue the following command:

```
openssl > x509  -text -in filename.pem
```

A digital certificate contains data that was collected to generate the digital certificate timestamps, a digital signature, and other information. However, because the generated digital certificate is encoded (usually in PEM format), it is unreadable.

**Step 5. Create a Certificate Chain in PEM Format Using OpenSSL**

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

**Note:** In SAS 9.4M3, you can use the SAS Deployment Manager after installation to add to the trusted CA bundle of certificates. For more information, see “Add Your Certificates to the Trusted CA Bundle” on page 106.

**Note:** In SAS 9.4M6, IOM servers support TLS.

If there is only one CA to trust, in the client application, specify the name of the file that contains the OpenSSL CA digital certificate. If multiple CAs are to be trusted, you can copy and paste into a new file the contents of all the digital certificates of CAs to be trusted by the client application. These CAs can be primary, intermediate, or root certificates. Add the root CAs to the client’s truststore.

For the server, do not include the Root CA in the server's certificate chain.

To manually create a new trust list, use the following template:

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

-----BEGIN CERTIFICATE-----
<PEM encoded certificate>
-----END CERTIFICATE-----

(Your Intermediate CA Certificate(s))

-----BEGIN CERTIFICATE-----
<PEM encoded certificate>
-----END CERTIFICATE-----

(Your Root CA Certificate)

-----BEGIN CERTIFICATE-----
<PEM encoded certificate>
-----END CERTIFICATE-----
```

102  Chapter 7 • Installing and Configuring TLS and Certificates on UNIX
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:

```bash
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:

```bash
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:

```bash
openssl x509 -in cert.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 104.

---

**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= system option.

For more information, see “SSLCACERTDIR= System Option” on page 34 and “SSLCALISTLOC= System Option” on page 32.

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

```bash
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 SAS 9.4M3, you can use the SAS Deployment Manager after installation to add your trust chain. The SAS Deployment Manager also validates those certificates. For more information, see “Add Your Certificates to the Trusted CA Bundle” on page 106.
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
<CASHome>/SASSecurityCertificateFramework/1.1/cacerts/server.pem -out
<CASHome>/SASSecurityCertificateFramework/1.1/cacerts/server.der
```

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

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

Convert PEM Files to a PFX File Using OpenSSL

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

```
```

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

Overview

Note:
In SAS 9.4M3, you can add your certificates by using the SAS Deployment Manager after installation. In SAS 9.4M6, IOM servers support TLS. These servers support using the SAS Deployment Manager to manage customer provided trusted CA certificates.

**Note:** In SAS 9.4M2 and earlier, when providing your own signed certificates, you must add the CA root and intermediate certificates to the SAS Private JRE using the Java `keytool -importcert` command. See “Add Your Certificates to the SAS Private JRE” on page 110.

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 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 `<SASHome>/SASSecurityCertificateFramework/1.1/cacerts` directory and updated at installation are described below. How the `cacerts` directory looks is shown in Figure 7.2 on page 105.

- `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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Ext</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>backup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cacerts.jks</td>
<td></td>
<td>161 KB</td>
</tr>
<tr>
<td>trustedcerts.jks</td>
<td></td>
<td>161 KB</td>
</tr>
<tr>
<td>cacerts.pem</td>
<td></td>
<td>244 KB</td>
</tr>
<tr>
<td>trustedcerts.pem</td>
<td></td>
<td>244 KB</td>
</tr>
</tbody>
</table>
Add Your Certificates to the Trusted CA Bundle

Starting with SAS 9.4M3, if you are providing your own site-signed certificates, then you must add the CA root certificate and all of its intermediate certificates to the trusted CA bundle. 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.

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

• In SAS 9.4M6, Java 8 is supported. Java 8 supports TLS 1.2. When using the SAS Environment Manager to add CA certificates, the certificates supplied must use algorithms supported by TLS 1.2. For more details, see The Transport Layer Security (TLS) Protocol Version 1.2 and Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.

• 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.

• If you have any Windows machines, you must also add the CA root and intermediate certificates to the Windows Certificate stores. For more information, see “Add Your Certificates to the Windows CA Store” on page 119.

• If you are importing certificates from SAS Viya, Configure SAS 9.4 Clients to Work with SAS Viya.

• These steps are to be performed for sites that are running SAS 9.4M3 and later. If you are running SAS 9.4M2 or earlier, see “Add Your Certificates to the SAS Private JRE” on page 110.

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 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 94.

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 86.

**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.

   ![Deployment Complete](image)

   **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:

```
:60:61:11:45 cn=twca root certification authority,ou=root ca,o=taiwan-ca,c=tw,
DF:4F:B5:F7 cn=microsec e-signto root ca,ou=e-signto ca,o=microsec ltd.,l=budapest,
c=hu, Jun 2, 2018, 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, proceed to “Add Your Certificates to the Windows CA Store”.

---

### Remove Your Certificates from the Trusted CA Bundle

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

To remove a certificate from the trusted CA bundle, you must have file permissions to access the truststore location. Only certificate files that were added using 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.
SAS Deployment Manager Criteria for Validating Certificates

The following criteria must be met for the validation to complete. Otherwise, errors are generated. See Chapter 10, “Troubleshooting,” on page 133 for possible errors that might be generated.

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

Add Your Certificates to the SAS Private JRE

A Java process uses a slightly different technology to establish trust. Within the Java API is a TrustManager which is responsible for managing the trust material that is used when making trust decisions. This uses a specific Java keystore, referred to as the truststore to maintain a repository of CA certificates that are used to establish trust. The default truststore used by a Java Runtime Environment (JRE) is located in the `lib/security/cacerts` file.
Prior to SAS 9.4M3, there is no SAS Deployment Manager task to help you manage any certificates that you provide. To establish trust for a Java process the CA certificates must be added to the cacerts file or to JVM arguments used to point to a different Java truststore. If you are providing your own certificates, then you must add the CA root certificate and all of its intermediate certificates to the SAS Private JRE using the `keytool -importcert` command.

**Note:** 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 Store” on page 119.

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

1. Log on to the primary machine as the SAS Installer user.
2. Change the directory to where your `keytool` commands reside.
   
   For example:
   
   ```
   cd SASHome/SASPrivateJavaRuntimeEnvironment/9.4/jre/bin/
   ```
3. Enter the following command. Refer to the table for information that you must provide.
   
   ```
   ./keytool -import -trustcacerts -alias MyCA -file ca_cert.pem
   -keystore SASHome/SASPrivateJavaRuntimeEnvironment/9.4/jre/lib/security/cacerts -storepass changeit
   ```
   
   **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>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SAS-installation-directory</code></td>
<td>Location on the machine where SAS is installed</td>
<td>C:\Program Files\SASHome\SASPrivateJavaRuntimeEnvironment\9.4\jre\lib\security</td>
</tr>
<tr>
<td><code>myhost</code></td>
<td>Fully qualified machine name</td>
<td>my_server.example.com</td>
</tr>
<tr>
<td><code>path-to-keystore.jks</code></td>
<td>Absolute path to the keystore</td>
<td>/opt/certs/my_keystore.jks</td>
</tr>
<tr>
<td></td>
<td>Refer to “Certificate Locations” on page 94.</td>
<td></td>
</tr>
</tbody>
</table>

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:
   
   ```
   ./keytool -list -keystore
   ```
You should see output similar to the following:

```
                          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:
                          DD:DF:4P: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 Store” on page 119.

---

**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 “Generate Digital Certificates Using OpenSSL” on page 95.

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 SLPVTKEYPASS= 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 25.

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 SSLCACERTDIR= system option.

5. The server can also validate the client’s certificates. Refer to the previous steps.
Building FIPS 140-2 Capable OpenSSL on UNIX

SAS ships OpenSSL libraries on UNIX. However, these are not FIPS 140-2 compliant libraries. The OpenSSL FIPS 140-2 module must be compiled and installed by the customer to build the OpenSSL libraries in order to ensure FIPS 140-2 compliance.

If you plan to build FIPS 140-2 capable OpenSSL for UNIX, refer to the following information:

- OpenSSL utility at [OpenSSL Source](https://www.openssl.org).
- Documentation for FIPS-140 Object Module SE.

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
```


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

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


**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 25 and “FIPS 140-2 Standards Compliance” on page 5.
Chapter 8
Installing and Configuring TLS and Certificates on Windows

Steps to Add Certificates for TLS to Windows .................................. 115
TLS on Windows: System and Software Requirements .......................... 116
Configure TLS and Request Digital Certificates on Windows ................. 116
  Configure TLS on Windows .......................................................... 116
  Request a Digital Certificate from the Microsoft Certificate Authority .... 117
  Request a Digital Certificate from a Certificate Authority That Is Not Microsoft . 118
Add Your Certificates to the Windows CA Store ................................. 119
Import the Client Certificate into the Windows Personal Machine Store ..... 120
Convert between PEM and DER File Formats for TLS .......................... 121
Use the SAS Deployment Manager to Manage Certificates in the Trusted CA Bundle .......................................................... 122
Validating Certificates between Clients and Servers ............................ 122
Configure FIPS 140-2 Capable TLS on Windows ................................. 122

Steps to Add Certificates for TLS to Windows

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

1. Configure TLS on Windows.
3. Add Your Certificates to the Windows CA Store.
5. (Optional) Convert between PEM and DER File Formats for TLS.
6. Use the SAS Deployment Manager to Manage Certificates in the Trusted CA Bundle.
7. (Optional) Configure FIPS 140-2 Capable TLS on Windows.
TLS on Windows: System and Software Requirements

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

- a computer that runs Windows 2000 (or later).
- depending on your configuration, access to the internet and a web browser.
- the TCP/IP communications access method.
- Microsoft Certificate Services add-on software.
- if you run your own CA, the Microsoft Certificate Authority application (which is accessible from your web browser).
- for SAS/CONNECT, a client session that runs on a computer that has a Trusted CA Certificate. This is necessary in order for a SAS/CONNECT client session to connect to a SAS/CONNECT server session via a Windows spawner using TLS encryption. The Windows spawner must run on a server that has a Trusted CA Certificate and a Personal Certificate.
- knowledge of your site's security policy, practices, and technology. The properties of the digital certificates requested depends on the security policies that have been adopted at your site.

Configure TLS and Request Digital Certificates on Windows

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

Configure TLS on Windows

Complete information about configuring your Windows operating environment for TLS is contained in the Windows installation documentation and at [www.microsoft.com](http://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

After generating or obtaining the CA, server, and client certificates that you will need to configure TLS, configure the SAS IOM servers and spawners to use TLS and specify the certificate files and locations being used. For more information, see [TLS support for IOM Servers](#).
Request a Digital Certificate from the Microsoft Certificate Authority

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

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

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

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

Note: Use algorithms supported by TLS 1.2 when generating certificates. In SAS 9.4M6, Java 8 is supported. SAS 9.4 versions that precede SAS 9.4M6 use a SAS Private JRE that is based on Java 7. Java 8 supports TLS 1.2. SAS Deployment Manager checks to ensure that the CA certificates being added comply with the supported TLS 1.2 algorithms. For more information, see The Transport Layer Security (TLS) Protocol Version 1.2 and Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.

1. In the Console window, expand Certificates (Local Computer) or Current User on the left.
2. Expand Personal. Click Certificates.
3. Right-click the Personal folder. Click All Tasks. Click Request New Certificate.
4. From the Certificate Request Wizard, click Next.
5. On the Certificate Types page, select the certificate template that you want to request (server, client, CA). The list is limited to the certificate templates for which either the current user or local machine have Read and Enroll permissions. After you select the certificate template, click Next.
6. On the Certificate Friendly Name and Description page, in the Friendly Name box, enter a descriptive name for the requested certificate. Click Next.
7. Click Finish.
8. Click OK. After the CA issues the requested digital certificate, the digital certificate is automatically installed in the Certificate Store.
Request a Digital Certificate from a Certificate Authority That Is Not Microsoft

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

1. First, perform the following steps to bring up the Microsoft Management Console (MMC):
   a. Click the Windows Start button, select Run, enter mmc, and click OK.
   b. In the Console window, select File \ Add/Remove Snap-in.
   c. Select Certificates from the list of available snap-ins, and click Add.
   d. In the dialog box that appears, click My User Account to request a user certificate or Computer Account to request a computer certificate. Click Next.
   e. If you selected Computer Account, click Local Computer (the computer this console is running on) and click Finish.
   f. If you selected My User Account, click Finish.
   g. Click OK.

2. Create an Offline Certificate Request using the Windows Certificate manager console by performing the following steps:
   a. From the Certificate manager console, expand Certificates (Local Computer), and expand Personal. Right-click Certificates, navigate to All tasks, Advanced Operations, and select Create custom request.
   c. Under Custom request, leave the default "No template" option. Click Next.
   d. On the Certificate Information page, expand Details, and click Properties.
   e. From the Certificate Properties page, select the General tab and fill out the Friendly name and Description values. These values are not required but are useful to distinguish your certificate from the other certificates that you have installed.
   f. From the Certificate Properties page, select the Subject tab. Add values to the Subject name and Alternative name attributes.
      For Common Name, specify a Fully Qualified Domain Name (FQDN). For DNS, specify a Fully Qualified Domain Name. Use best practices when creating certificates on page 89.
   g. From the Certificate Properties page, select the Extensions tab. Add the key usage and extended key usage that you need. This usage includes server and client authentication.
   h. From the Certificate Properties page, select the Private Key tab. Expand Cryptographic Service Provider. Select the cryptographic service provider (for example, RSA). Expand Key options and select the key size. Expand Select Hash Algorithm. For Hash Algorithm, select an SHA algorithm.
   i. Click OK.
j. On the **Where do you want to save the offline request?** page, give your certificate request (CSR) file a name and save it to a location on your computer. Make sure the file format is set to Base 64.

k. Click **Finish**.

3. Use OpenSSL to generate a certificate signing request. See “**Generate Digital Certificates Using OpenSSL**” on page 95.

   **Note:** The Windows operating environment can import digital certificates that were generated in the UNIX operating environment. To convert from UNIX (PEM format) to Windows (DER format) before importing, see “**Convert between PEM and DER File Formats for TLS**” on page 121.

4. Submit your certificate request to a Certificate Authority to process your request and issue a certificate. The certificate request is a text file. Usually, you are required to copy the text from the file and enter it into an online submission form on the Certificate Authority website. Contact your Certificate Authority directly for instructions on the process for submitting your certificate request.

   **Note:** Use algorithms supported by TLS 1.2 when generating certificates. In SAS 9.4M6, Java 8 is supported. Java 8 supports TLS 1.2. SAS Deployment Manager checks to ensure that the CA certificates being added comply with the supported TLS 1.2 algorithms. For more information, see **The Transport Layer Security (TLS) Protocol Version 1.2 and Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile**.

5. When your CA has processed your request and issued the certificate, download it to your server so that it can be imported. **Add the certificates to the Windows CA store on page 119.**

---

**Add Your Certificates to the Windows CA Store**

If you are providing your own self-signed or site-signed certificates, then you must add the CA root certificate and all of its intermediate certificates to the Windows certificates stores using the Windows Certificates Snap-in.

**Note:** If you have not already done so, you must add your CA root and intermediate certificates to the trusted CA bundle or to the SAS Private JRE. For more information, see “**Add Your Certificates to the Trusted CA Bundle**” on page 106 or “**Add Your Certificates to the SAS Private JRE**” on page 110.

To add CA root and intermediate certificates to the Windows certificates stores on your local computer (Certificates (Local Computer)), perform these steps:

1. Click the Windows **Start** button, select **Run**, enter `mmc`, and click **OK**.
2. In the Console window, select **File ⇒ Add/Remove Snap-in**.
3. Select **Certificates** from the list of available snap-ins, and click **Add**.
4. In the 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.

---

**Import the Client Certificate into the Windows Personal Machine Store**

Import the client certificates into the Windows personal store on the local machine using the Microsoft Management Console (MMC). After the certificates have been imported, you will then need to grant permission to the authenticated users who use the client certificates private key. Perform the following steps to import the client certificates and grant permission to authenticated users:

1. Start the Microsoft Management Console (MMC). Right-click the Windows **Start** menu and select **Run**.

2. In the Run window, enter **mme**, and press **OK**.

3. Add the Certificate manager console.
   a. From the Microsoft Management Console window, click **File** and select **Add/Remove Snap-in** from the drop-down menu.
   b. In the Add or Remove snap-ins window, select **Certificates** from the **Available snap-ins** list.
   c. Add **Certificates** to the **Selected snap-ins** list.
   d. Select **Computer account** and click **Next**.
   e. Select **Local** computer (the computer that this console is running on). Click **Finish**.
   f. Click **OK**.

4. Expand the **Certificates (Local Computer)** list and click **Personal**.

5. From the left pane, right-click and select the **Certificates** node within the **Certificates (Local Computer) > Personal, Certificates** hierarchy to view a list of certificates. Select **All Tasks**, and then **Import** from the drop-down menu.
6. From the Certificate Import Wizard, confirm that Local Machine is selected for the Store Location, and click Next.
   a. Select Browse to locate the file to import. Expand the list of file types and select All Files from the drop-down menu.
   b. From the list of files, select the certificates that you want to import. In our example, we are selecting the client certificate in PFX format that contains the certificate and private key file, customerCert.pfx. Click Next.
   c. From the Private Key Protection page of the Certificate Import Wizard, enter the password for customerCert.pfx. Select Include all extended properties. Do not select Enable strong private key protection and do not select Mark this key as exportable. Click Next.
   d. From the Certificate Store page of the Certificate Import Wizard, select Place all certificates in the following store. Place the certificates in the Personal Certificate store. Click Next.
   e. From the Completing the Certificate Import Wizard page, click Finish.
   f. Verify that you receive the message The import was successful, and click OK.

7. Observe that the imported certificates are now listed in the Microsoft Management Console.

In order to use the client’s private key, the client certificate’s private key must be readable. Perform the following tasks to grant Read permission to the authenticated users who will use the client certificate's private key.

1. Start the Microsoft Management Console (MMC). Right-click the Window’s Start menu and select Run.
2. From the Run window, enter mmc, and press OK.
3. Right-click the client certificate that you recently imported.
4. On the pop-up menu, select All Tasks, Manage Private Keys. A Permissions window appears.
5. From the Permissions window, within the Security tab, add Authenticated Users.
6. Ensure that authenticated users have Read permission. Select Allow Read item, deselect Allow Full control, and deselect Allow Special permissions. Click OK.

---

Convert between PEM and DER File Formats for TLS

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

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

You can use OpenSSL. Here is an example of converting a server digital certificate from DER input format to PEM output format.
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 86.

---

**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 104.

---

**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 “Configure TLS and Request Digital Certificates on Windows” on page 116.

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 25.

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

---

**Configure FIPS 140-2 Capable TLS on Windows**

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 8.1 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](http://www.verisign.com)
- For Thawte, see [www.thawte.com](http://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.

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

   *Note:* TLS certificate and key files must reside in the z/OS UNIX file system. The OpenSSL library cannot read MVS data sets.
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') +
   ) +
   ALTNAMES( +
      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 Intermediated CA Certificate(s))

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

-----BEGIN 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 104.

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 32.

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 104. 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.*
Part 3

Troubleshooting

Chapter 10

Troubleshooting ................................................. 133
Chapter 10
Troubleshooting

ERROR: Unable to load extension: (tkessl)

There are a lot of reasons the library might not load. The best way to debug this error is to turn on logging and get a SAS logging facility (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 “Configure FIPS 140-2 Capable TLS on Windows” on page 122.

ERROR: HTTP proxy handshake failed.

This message is displayed when clients sending TLS Subject Name Identification (SNI) cannot connect to a secured proxy server.
There are servers that do not handle SNI host name checking in a way that allows connecting to secured proxy servers.

- On UNIX servers, starting in SAS 9.4M5, SNI is always sent. You can disable SNI by setting the USE_SSL_SNI environment variable.
- On Windows servers, SNI is always sent. Other than disabling name checking (Subject Alternative Name) on server certificates, there is currently no workaround.

**ERROR: Cannot load SSL Support**

This message is displayed when SAS cannot find required software.

- This message can be generated when TLS certificates cannot be found. If the directory where the certificates are located is specified using the SSLCACERTDIR= environment variable, and the certificate names in the directory are not named using the value of a hash that OpenSSL generates, this message is generated. For more information, see “SSLCACERTDIR= System Option” on page 34.
- This message is generated when requisite software cannot be loaded in an IOM session.

**ERROR:14090086:SSL routines: SSL3_GET_SERVER_CERTIFICATE: certificate verify failed**

This message is displayed when TLS certificates cannot be verified. If the directory where the certificates are located is specified using the SSLCACERTDIR environment variable, and the certificate names in the directory are not named using the value of a hash that OpenSSL generates, this message is generated. For more information, see “SSLCACERTDIR= System Option” on page 34.

**Failed to Find the Following Issuer of this Certificate in Truststore**

This message is displayed when using the SAS Deployment Manager to add TLS certificates to the trust list in the wrong order. First, you need to add the issuer of the certificate or the root certificate. Then you can add the intermediate certificate. You need to run the the SAS Deployment Manager task for each certificate that you need to add.
Verify that the File Contains Certificates in the Proper Encoding

This message is displayed when using the SAS Deployment Manager to add TLS certificates with unacceptable encodings. Certificates must be X.509 certificates formatted in Base-64 encoding that have .pem, .crt, or .cer extensions. For more information, see “Certificate File Formats” on page 86.

ERROR: Cannot negotiate encryption algorithm

This message is displayed when an encryption algorithm cannot be negotiated.

One example of this error being logged occurs if you are running a version of SAS that is earlier than SAS 9.4 and you are not licensed for SAS/SECURE. You might have specified SAS system option NETENCRYPTALG=AES, but without SAS/SECURE, AES is not supported. In this case, your connection will fail. You can still use SASProprietary to encrypt your connection, but not other encryption algorithms in this case.

Note: In SAS 9.4, SAS/SECURE is included as part of Base SAS and includes AES encryption.

Another example is that you have specified an encryption algorithm on the metadata server, and the client is using a different algorithm. For example, you might have assigned AES encryption using the SAS Management Console as the encryption algorithm to use on the metadata server. The metadata server allows only one encryption algorithm to be specified. When a client (for example, SAS/CONNECT) connects to the metadata server and is using an encryption algorithm that is not AES, your connection will fail.

If you need to support multiple network encryption algorithms for a SAS/CONNECT server that is part of a SAS metadata environment, perform the following steps:

1. Edit the ConnectSpawner_usermods.sh file.
   • Set the NETENCRYPTALGORITHM= value to the multiple network encryption algorithms that you need.
   • Add the -SERVICE option to a TCP port that has not been used already.
   • Add the -SASCMD option copied from the current launch command in metadata.

   The edited version of your command might look something like the following code:
   ```
   USERMODS_OPTIONS="-netencryptalgorithm 'AES,SASPROPRIETARY'" -service different-tcp-port -sascmd
   ```

2. Stop and restart the SAS/CONNECT spawner.
3. Sign on using the new port.
This message is displayed when you are using SAS Deployment Manager to add a CA certificate to the truststore where the certificate signature algorithm is not supported. Starting in SAS 9.4M6, SAS supports Java 8, which uses TLS 1.2. When you add a certificate to the truststore using SAS Deployment Manager, the signature of the CA that signs the certificates is checked to see that it uses algorithms that are supported.

An example of when this error might occur is when the customer-provided CA certificate is signed using the RSASSA-PSS signing signature. When that certificate is added to the certificate truststore using the SAS Deployment Manager, an error is generated because the certificate is signed using an algorithm that is unsupported in TLS 1.2. From The Transport Layer Security (TLS) Protocol Version 1.2: “There are certificates that use algorithms and/or algorithm combinations that cannot be currently used with TLS 1.2. For example, a certificate with RSASSA-PSS signature key (id-RSASSA-PSS OID in SubjectPublicKeyInfo) cannot be used because TLS defines no corresponding signature algorithm.” For more information, see Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.

This signature error can be corrected by asking the customer administrator to generate another CA certificate that is signed by a CA using algorithms supported by TLS 1.2 and Java 8.
Appendices

Appendix 1

Environment Variables Used Before SAS 9.4M5

Appendix 2

Supported Ciphers for the SSLMODE= Option
Overview

The environment variables in this section are used in SAS releases prior to SAS 9.4M5. In SAS 9.4M5, system options of a similar name were created. See Chapter 2, “SAS System Options for Encryption,” on page 25.

Dictionary

SAS_SSL_CIPHER_LIST Environment Variable

Specifies the ciphers that can be used on UNIX and z/OS for OpenSSL.

Client: Optional
Server: Optional
Valid in: Configuration file, command line
Categories: Communications: Networking and Encryption
System Administration: Security
Operating environment: UNIX and z/OS

Notes: In SAS 9.4M5, a system option of a similar name was created. See SSLCIPHERLIST= on page 25 for details.
This environment variable is available in all SAS 9.3 and SAS 9.4 versions of software if hot fixes are applied.
This environment variable must be set before TLS is loaded. It cannot be changed after TLS is loaded. You must set the environment variable before the SAS/CONNECT spawner is started and before SAS is started on the client.

**Tip:** You can also define SET commands for Windows by using the System Properties dialog box that you access from the Control Panel.

**Examples:**

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

```bash
export SAS_SSL_CIPHER_LIST=TLSv1.2
```

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

```bash
SETENV SAS_SSL_CIPHER_LIST HIGH
```

Set the environment variable at SAS invocation for UNIX hosts:

```
-set SAS_SSL_CIPHER_LIST '3DES:RC2'
```

Set the environment variable on Windows hosts:

```bash
SET SAS_SSL_CIPHER_LIST SHA256
```

**Syntax**

\[ \text{SAS\_SSL\__CIPHER\_LIST} = \text{openssl\_cipher\_list} \]

**Syntax Description**

`openssl-cipher-list`

The `SAS_SSL_CIPHER_LIST` environment variable specifies the ciphers that can be used on UNIX and z/OS for OpenSSL. Refer to the OpenSSL Ciphers document to see how to format the `openssl-cipher-list` and for a complete list of the ciphers that work with your TLS version. The OpenSSL Cipher information can be found at [OpenSSL Ciphers](#).

**Note:** SAS does not support CAMELLIA, IDEA, MD2, and RC5 ciphers.

**Note:** The protocol and cipher information for the actual connection can be seen by setting `dumpCurrentCipherInfo` at the SAS DEBUG level. For information, see “Encryption: SAS Logging Facility” on page 21.

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

**Details**

This environment variable is available on UNIX and z/OS platforms. This environment variable can be specified anytime before TLS is used. After TLS is loaded, it cannot be changed.

Refer to the OpenSSL documentation on ciphers for information about the ciphers that can be specified for this environment variable. This information can be found at [OpenSSL Ciphers](#).

**Note:** For Windows, you can use group policy settings to configure TLS Cipher Suite Order. See Cipher Suites in TLS/SSL (Schannel SSP) for information about the TLS Cipher Suite order.
SSLCACERTDIR Environment Variable

Specifies the location of the trusted certificate authorities (CA) found in OpenSSL format.

Client: Optional
Server: Optional
Valid in: Configuration file, SAS invocation, SAS/CONNECT spawner start-up

Categories:
Communications: Networking and Encryption
System Administration: Security

Default: The default location for certificates is set using the SSLCALISTLOC= system option. Certificates are located in one .pem file. By contrast, The SSLCACERTDIR environment variable allows the customer to specify a location where multiple certificate files reside. See “SSLCALISTLOC= System Option” on page 32.

Operating environment: UNIX

Notes: In SAS 9.4M5, a new system option of a similar name was created. See SSLCACERTDIR= on page 25 for details.
This environment variable is available in all SAS 9.3 and SAS 9.4 versions of software if hot fixes are applied.

Tips: OpenSSL looks up the CA certificate based on the x509 hash value of the certificate. SSLCACERTDIR requires that the certificates are located in the specified directory where the certificate names are the value of a hash that OpenSSL generates.
If you are upgrading from a version of OpenSSL that is older than 1.0.0, you need to update your certificate directory links. Starting with code base 1.0.0, SHA hashing is used instead of MD5. You can use the OpenSSL C_REHASH utility to re-create symbolic links to files named by the hash values.
You can discover the hash value for a CA and then create a link to the file named after the certificate’s hash value. Note that you must add ".0" to the hash value.

ln -s cacert1.pem 'openssl x509 -noout -hash -in
{/u/myuser/sslcerts/cacert1.pem}'.0

If you list the CA file, you see the link between the file named after the certificate’s hash value and the CA file.

lrwxrwxrwx 1 myuser rnd 10 Apr 7 14:42 6730c6a9.0 -> cacert1.pem

To verify the path of the server certificate file (cacert1.pem for our example), use the following OpenSSL command:
openssl verify -CApath /u/myuser/sslcerts cacert1.pem

Examples: The SSLCACERTDIR environment variable points to the directory where the CA certificate is located. Export the environment variable on UNIX hosts for the Bourne Shell:

export SSLCACERTDIR=/u/myuser/sslcerts/

Set the environment variable on UNIX hosts for the C Shell directory where the CA certificates are located:

SETENV SSLCACERTDIR /u/myuser/sslcerts/

Set the environment variable at SAS invocation for UNIX hosts:
-set "SSLACERTDIR=/u/myuser/sslcerts/*"

**Syntax**

```plaintext
SSLACERTDIR="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 SSLACERTDIR 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.

SSLACERTDIR 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 86.

For Foundation servers such as workspace servers and stored process servers (that is, servers in a deployment), if certificates are used, SAS searches for certificates in the following specific order:

1. SAS looks for SAS system option SSLCALISTLOC= to find a file. This file holds your trusted certificates.
2. SAS looks for the SSLCALISTLOC environment variable to find a file. This file holds your trusted certificates.
3. If trustedcerts.pem exists and the SSLACERTDIR= system option is 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 the SSLACERTDIR= system option or environment variable, then SAS ignores SSLCALISTLOC=.
5. If trustedcerts.pem does not exist, and the SSLACERTDIR= system option is not set, SAS reports an error.

In SAS 9.4, SAS 9.4M1, and SAS 9.4M2, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is `SAS-configuration-directory/Lev/n/certs/cacert.pem`. The cacert.pem file contains the list of trusted certificates.

In SAS 9.4M3, the default path set for the SSLCALISTLOC= system option on UNIX and z/OS foundation servers is `<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem`. The trustedcerts.pem file contains the list of trusted certificates.

*Note:* A trusted CA certificate is required at the client in order to validate a server's digital certificate. The trusted CA certificate must be from the CA that signed the server certificate.
**SSL_CERT_DIR Environment Variable**

Specifies the location of the trusted certificate authorities (CA) found in OpenSSL format. This is the OpenSSL environment variable.

- **Client:** Optional
- **Server:** Optional
- **Valid in:** Configuration file, SAS invocation, SAS/CONNECT spawner start-up
- **Categories:** Communications: Networking and Encryption
  System Administration: Security
- **Default:** The default location for certificates is set using the SSLCALISTLOC= system option. Certificates are located in one .pem file. By contrast, The SSLCACERTDIR environment variable allows the customer to specify a location where multiple certificate files reside. See “SSLCALISTLOC= System Option” on page 32.

- **Operating environment:** UNIX

- **Notes:** In SAS 9.4M5, a new system option of a similar name was created. See SSLCERTDIR= on page 25 for details.
  This environment variable is available in all SAS 9.3 and SAS 9.4 versions of software if hot fixes are applied.

- **Tips:** OpenSSL looks up the CA certificate based on the x509 hash value of the certificate. SSL_CERT_DIR requires that the certificates are located in the specified directory where the certificate names are the value of a hash that OpenSSL generates.

If you are upgrading from a version of OpenSSL that is older than 1.0.0, you need to update your certificate directory links. Starting with code base 1.0.0, SHA hashing is used instead of MD5. You can use the OpenSSL C_REHASH utility to re-create symbolic links to files named by the hash values.

You can discover the hash value for the CA and then create a link to the file named after the certificate's hash value. Note that you must add ".0" to the hash value.

```
ln -s cacert1.pem 'openssl x509 -noout -hash -in /u/myuser/sslcerts/cacert1.pem'.0
```

If you list the CA file, you see the link between the file named after the certificate's hash value and the CA file.

```
lrwxrwxrwx 1 myuser rnd 10 Apr 7 14:42 6730c6a9.0 -> cacert1.pem
```

To verify the path of the server certificate file (cacert1.pem for our example), use the following OpenSSL command:

```
openssl verify -CApath /u/myuser/sslcerts cacert1.pem
```

- **Examples:** The SSL_CERT_DIR environment variable points to the directory where the CA certificate is located. Export the environment variable on UNIX hosts for the Bourne Shell:

```
export SSL_CERT_DIR=/u/myuser/sslcerts/
```

Set the environment variable on UNIX hosts for the C Shell directory where the CA certificates are located:

```
SETENV SSL_CERT_DIR /u/myuser/sslcerts/
```
Set the environment variable at SAS invocation for UNIX hosts:

```
-set "SSL_CERT_DIR=/u/myuser/sslcerts/"
```

### Syntax

`SSL_CERT_DIR=“file-path”`

### Syntax Description

“file-path” specifies the location where the public certificates for all of the trusted certificate authorities (CA) in the trust chain are filed. There is one file for each CA. The names of the files are the value of a hash that OpenSSL generates.

**Note:** OpenSSL generates different hash values for each OpenSSL version. For example, OpenSSL 0.9.8 generates different hash values than does OpenSSL 1.x.

### Details

Environment variables `SSLCACERTDIR` and `SSL_CERT_DIR` point to a directory that contains all of the public certificate files of all CAs in the trust chain. One file exists for each CA in the trust chain.

`SSL_CERT_DIR` requires the certificates to be in the directory where their names are the value of a hash that OpenSSL generates.

Each CA certificate file must be PEM-encoded (base64). For more information, see “Certificate File Formats” on page 86.

For Foundation servers such as workspace servers and stored process servers (that is, servers in a deployment), if certificates are used, SAS searches for certificates in the following specific order:

1. SAS looks for SAS system option `SSLCALISTLOC=` to find a file. This file holds your trusted certificates.
2. SAS looks for the `SSLCALISTLOC` environment variable to find a file. This file holds your trusted certificates.
3. If `trustedcerts.pem` exists and the `SSLCACERTDIR=` system option is 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 the `SSLCACERTDIR=` system option or environment variable, then SAS ignores `SSLCALISTLOC=`.
5. If `trustedcerts.pem` does not exist, and the `SSLCACERTDIR=` system option is not set, SAS reports an error.

In SAS 9.4, SAS 9.4M1, and SAS 9.4M2, the default path set for the `SSLCALISTLOC=` system option on UNIX and z/OS foundation servers is `SAS-configuration-directory/Levn/certs/cacert.pem`. The cacert.pem file contains the list of trusted certificates.

In SAS 9.4M3, the default path set for the `SSLCALISTLOC=` system option on UNIX and z/OS foundation servers is `<SASHome>/SASSecurityCertificateFramework/1.1/cacerts/trustedcerts.pem`. The trustedcerts.pem file contains the list of trusted certificates.
Note: A trusted CA certificate is required at the client in order to validate a server’s digital certificate. The trusted CA certificate must be from the CA that signed the server certificate.
Appendix 2
Supported Ciphers for the SSLMODE= Option

Note: For Java clients, the list of supported TLS ciphers is determined by the Java Cryptography Extension (JCE) providers installed in the SAS Private Java Runtime Environment.

Table A2.1  Ciphers Supported for SSLMODE= Option on UNIX and Linux

<table>
<thead>
<tr>
<th>SSLMODE=</th>
<th>TLS Cipher</th>
<th>OpenSSL Cipher Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLMODESUITEB192</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</td>
<td>ECDHE-ECDSA-AES256-GCM-SHA384</td>
</tr>
<tr>
<td>SSLMODESUITEB128</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</td>
<td>ECDHE-ECDSA-AES256-GCM-SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</td>
<td>ECDHE-ECDSA-AES128-GCM-SHA256</td>
</tr>
<tr>
<td>SSLMODESP800131A</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</td>
<td>ECDHE-ECDSA-AES256-GCM-SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</td>
<td>ECDHE-ECDSA-AES128-GCM-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256</td>
<td>ECDHE-RSA-AES128-GCM-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</td>
<td>ECDHE-RSA-AES256-GCM-SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256</td>
<td>ECDHE-ECDSA-AES128-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384</td>
<td>ECDHE-ECDSA-AES256-SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256</td>
<td>ECDHE-RSA-AES128-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384</td>
<td>ECDHE-RSA-AES256-SHA384</td>
</tr>
<tr>
<td>SSLMODE=</td>
<td>TLS Cipher</td>
<td>OpenSSL Cipher Used</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>---------------------</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_GCM_SHA256</td>
<td>AES128-GCM-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_GCM_SHA384</td>
<td>AES256-GCM-SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
<td>AES128-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA384</td>
<td>AES256-SHA256</td>
</tr>
<tr>
<td>SSLMODEDEPRECATE</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</td>
<td>ECDHE-ECDSA-AES128-GCM-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</td>
<td>ECDHE-ECDSA-AES256-GCM-SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256</td>
<td>ECDHE-RSA-AES128-GCM-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</td>
<td>ECDHE-RSA-AES256-GCM-SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256</td>
<td>ECDHE-ECDSA-AES128-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384</td>
<td>ECDHE-ECDSA-AES256-SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256</td>
<td>ECDHE-RSA-AES128-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384</td>
<td>ECDHE-RSA-AES256-SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_GCM_SHA256</td>
<td>AES128-GCM-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_GCM_SHA384</td>
<td>AES256-GCM-SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
<td>AES128-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA384</td>
<td>AES256-SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_DHE_RSA_WITH_AES_128_CBC_SHA</td>
<td>DHE-RSA-AES128-SHA</td>
</tr>
</tbody>
</table>
### SSLMODE= System Option Supported Ciphers

**SSLMODE=** System Option Supported Ciphers

<table>
<thead>
<tr>
<th>SSLMODE=</th>
<th>TLS Cipher</th>
<th>OpenSSL Cipher Used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TLS_DHE_RSA_WITH_AES_256_CBC_SHA</td>
<td>DHE-RSA-AES256-SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA</td>
<td>AES128-SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA</td>
<td>AES256-SHA</td>
</tr>
</tbody>
</table>

**Table A2.2  Ciphers Supported for SSLMODE= Option on Windows**

<table>
<thead>
<tr>
<th>SSLMODE=</th>
<th>TLS Cipher</th>
<th>Schannel Algorithms Used</th>
<th>Resulting Ciphers</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUITEB1 92</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</td>
<td>CALG_ECDHE CALG_ECDSA CALG_AES_256 CALG_SHA_384</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384</td>
</tr>
<tr>
<td>SUITEB1 28</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</td>
<td>CALG_ECDHE CALG_ECDSA CALG_AES_128 CALG_AES_256 CALG_SHA_384</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256</td>
</tr>
<tr>
<td>SSLMODE</td>
<td>TLS Cipher</td>
<td>Schannel Algorithms Used</td>
<td>Resulting Ciphers</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>--------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>SSLMODESP800131A</td>
<td>TLS_ECDHE_ECD SA_WITH_AES_256_GCM_SHA384</td>
<td>CALG_ECDHE, CALG_ECDSA, CALG_RSA_SSH</td>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_ECD SA_WITH_AES_128_GCM_SHA256</td>
<td>CALG_AES_128, CALG_SHA_256</td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384</td>
<td>CALG_AES_128, CALG_SHA_256</td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</td>
<td>CALG_AES_128, CALG_SHA_256</td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_GCM_SHA256</td>
<td>CALG_AES_128, CALG_SHA_256</td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_GCM_SHA256</td>
<td>CALG_AES_128, CALG_SHA_256</td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
<td>CALG_AES_256, CALG_SHA_384</td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSLMODE</td>
<td>TLS Cipher</td>
<td>Schannel Algorithms Used</td>
<td>Resulting Ciphers</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------</td>
<td>--------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>SSLMODE=DEPRECATED</td>
<td>TLS_ECDHE_ECD SA_WITH_AES_128_GCM_SHA256</td>
<td></td>
<td>TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_ECD SA_WITH_AES_256_GCM_SHA384</td>
<td></td>
<td>TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256</td>
<td></td>
<td>TLS_RSA_WITH_AES_128_GCM_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</td>
<td></td>
<td>TLS_RSA_WITH_AES_256_GCM_SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256</td>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384</td>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_GCM_SHA256</td>
<td></td>
<td>TLS_RSA_WITH_AES_128_GCM_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_GCM_SHA384</td>
<td></td>
<td>TLS_RSA_WITH_AES_256_GCM_SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA384</td>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td>TLS_DHE_RSA_WITH_AES_128_CBC_SHA</td>
<td></td>
<td>TLS_DHE_RSA_WITH_AES_128_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_DHE_RSA_WITH_AES_256_CBC_SHA</td>
<td></td>
<td>TLS_DHE_RSA_WITH_AES_256_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA</td>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA</td>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA</td>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA</td>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA</td>
</tr>
<tr>
<td>SSLMODE</td>
<td>TLS Cipher</td>
<td>Schannel Algorithms Used</td>
<td>Resulting Ciphers</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------</td>
<td>--------------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>SSLMODE_DEPRECATED</td>
<td>(continued)</td>
<td>(continued)</td>
<td>(continued)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TLS_DHE_DSS_WITH_AES_128_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TLS_DHE_DSS_WITH_AES_256_CBC_SHA256</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TLS_DHE_DSS_WITH_AES_128_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TLS_DHE_DSS_WITH_AES_256_CBC_SHA</td>
</tr>
</tbody>
</table>
Recommended Reading

Here is the recommended reading list for this title:

- SAS/CONNECT User’s Guide
- SAS/SHARE User’s Guide
- SAS DATA Step Statements: Reference
- SAS System Options: Reference
- Base SAS Procedures Guide
- SAS Language Reference: Concepts
- SAS XMLV2 and XML LIBNAME Engines: User’s Guide
- SAS 9.4 Intelligence Platform: Security Administration Guide
- SAS Companion that is specific to your operating environment
- 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**


**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

#### A
- AES (Advanced Encryption Standard) 19
- AES algorithm 10
- algorithms 18
  - for client/server data transfers 28
  - key length for data transfers 31
- SAS Proprietary Encryption 18
- SAS/SECURE 10, 18
- summary of 8
- authentication
  - client authentication by server 41
  - location of digital certificate for 38

#### B
- block cipher 18

#### C
- CAS_CLIENT_SSL_CA_LIST
  - environment variable 53
- certificate authorities (CAs)
  - digital certificate location 32
- certificate authorities (CAs)
  - trust lists 128
- Certificate authorities (CAs) 12
- Certificate Locations
  - TLS on UNIX 94
- Certificate Revocation List (CRL)
  - checking when digital certificate is validated 41
  - location of 43
  - client authentication by server 41
  - client/server connection outcomes 30
  - client/server data transfers
    - algorithm for 28
    - encrypting 28
    - key length for algorithm 31
- configuration
  - SAS Proprietary Encryption 7
  - SAS/SECURE 10
  - TLS 14

#### D
- Data Encryption Standard (DES) 18
  - data transfers
    - algorithm for 28
    - encrypting 28
    - key length for algorithm 31
  - decrypting private keys 48, 49
  - DER format 104
    - Windows 121
  - DES (Data Encryption Standard) 18
  - DES algorithm 10
  - digital certificates 12
    - certificate trust list 102, 103, 104, 122, 129, 130
  - checking Certificate Revocation List when validating 41
  - converting between PEM and DER formats 104, 121
  - location for authentication 38
  - location for trusted certificate authorities 32
  - name of issuer 37
  - OpenSSL on UNIX 97
  - OpenSSL on z/OS 126
  - private key location 48
  - requesting from Microsoft Certificate Authority 117
  - serial number of 39
  - subject name of 40
  - viewing 102, 128
  - digital signatures 12
  - DSA 19

#### E
- encoded passwords 59, 63
  - encoding methods 62, 66, 67
  - in SAS programs 59, 63
  - saving to paste buffer 65
  - encoding methods 62, 66, 67
  - ENCRYPTFIPS= system option 25
  - encryption 3, 4
  - algorithms 4
  - comparison of technologies 20
contexts for coverage 4
data transfers 28
on-disk 4
over-the-wire 5
SAS/CONNECT client on UNIX example 70
SAS/SHARE client example 70
environment variables
   CAS_CLIENT_SSL_CA_LIST 53
   SAS_SSL_CIPHER_LIST 139
   SAS_SSL_MIN_PROTOCOL environment variable 54
   SAS_VIYA_TOKEN 56
   SSL_CERT_DIR 143
   SSL_USE_SNI 57
   SSLCACERTDIR 141
export restrictions for SAS/SECURE 10

F
FIPS 113, 122
   How SAS Implements FIPS 5
   SAS/SECURE 5
   TLS 5
   FIPS 140-2 algorithms 25
   FIPS 140-2 configuration SAS/SECURE 11
   FIPS 140-2 installation SAS/SECURE 11

I
implementation 20
installation
   SAS Proprietary Encryption 7
   SAS/SECURE 10
   TLS 14
tunneling 17

K
key length
   for data transfer algorithm 31
   keys
      private 12, 48, 49
      public 12

L
logging 21
logging security events 21

M
MD5 19
METHOD= option
   PROC PWENCODE statement 62
Microsoft Certificate Authority requesting digital certificate from 117
Microsoft CryptoAPI 10

N
NETENCRLALG
   NETENCYPTALGORITHM= system option 28
NETENCYPT system option 28
NETENCYPTALGORITHM= system option 28
NETENCYPTKEY= NETENCYPTKEYLEN= system option 31
NETENCYPTKEYLEN= system option 31

O
ODS generated PDF files 22
on-disk encryption 4
OpenSSL 113
   arguments and values 98, 100
certificate trust list 102
   converting between PEM and DER formats 104
creating digital certificates 126
digital certificates 97
   FIPS 113
   OpenSSL version support 13
   TLS on z/OS 126
   Verify Certificates 103, 129
OpenTLS
   converting between PEM and DER formats 121
OUT= option
   PROC PWENCODE statement 62
   over-the-wire encryption 5

P
passwords
   encoding 59, 63
   encoding methods 66, 67
   for decrypting private keys 48, 49
   paste buffer
   saving encoded passwords to 65
PDF files 22
PDF system options 22
PEM format 104
PKCS #12 DER encoding package file
   password for decrypting private keys 48
PKCS #12 encoding package file
location of 47
port forwarding 16
private keys 12
location of 48
password for decrypting 48, 49
PROC PWENCODE statement
PWENCODE procedure 61
providers
SAS Proprietary Encryption 6
SAS/SECURE 8
SSH 16
SSL 11
public keys 12
PWENCODE procedure 59
concepts 59
encoded passwords in SAS programs 63
encoding 60
encoding methods 66, 67
encoding passwords 63
saving encoded passwords to paste buffer 65
syntax 61

R
RC2 algorithm 10, 18
key length for data transfer algorithm 32
RC4 algorithm 10, 18
key length for data transfer algorithm 32
Request a Certificate
TLS on UNIX 95
RSA 19

S
SAS Deployment Manager
Add Your Certificates to the SAS
Private JRE 110
certificate trust list 104, 122
Certificates 88
SAS programs
encoded passwords in 59, 63
SAS Proprietary 6
SAS Proprietary algorithm 18
SAS Proprietary Encryption 4
algorithms 18
comparison of technologies 20
configuration 7
installation 7
SAS/SHARE example 70
software availability 7
system requirements 7
SAS_SSL_CIPHER_LIST environment variable 139
SAS_SSL_MIN_PROTOCOL 54
SAS_VIYA_TOKEN environment variable 56
SAS/CONNECT
client on UNIX example 70
SAS/SECURE example 70
server on UNIX example 70
SSH tunnel example 81
TLS UNIX spawner example 71
TLS Windows spawner example 73
TLS z/OS spawner example 75
SAS/Secure
encryption 25
SAS/SECURE 8
algorithms 10, 18
comparison of technologies 20
configuration 10
export restrictions 10
FIPS 5
FIPS 140-2 configuration 11
FIPS 140-2 installation 11
installation 10
SAS/CONNECT example 70
software availability 10
system requirements 10
Windows and 10
SAS/SHARE
client example 70
SAS Proprietary Encryption example 70
server example 70
SSH tunnel example 82
TLS on UNIX example 77
TLS on Windows examples 79
TLS on z/OS example 80
sasproprietary algorithm 18
SDM
certificate trust list 130
Secure Shell
See SSH (Secure Shell)
Secure Sockets Layer
See SSL (Secure Sockets Layer)
security logging 21
serial number of digital certificate 39
servers
client authentication by 41
SAS/CONNECT on UNIX example 70
SAS/SHARE server example 70
Setting up digital certificates
TLS on UNIX 95
software requirements
TLS on UNIX 93
TLS on Windows 116
TLS on z/OS 125
spawners
   TLS SAS/CONNECT UNIX example 71
   TLS SAS/CONNECT Windows example 73
   TLS SAS/CONNECT z/OS example 75
SSL (Secure Sockets Layer) 11, 122
   See also OpenSSL
   SSL CERT_DIR environment variable 143
   SSL_USE_SNI environment variable 57
   SSLCACERTDATA system option 36
   SSLCACERTDIR environment variable 141
   SSLCACERTDIR system option 34
   SSLCALISTLOC= system option 32
   SSLCERTISS= system option 37
   SSLCERTLOC= system option 38
   SSLCERTSERIAL= system option 39
   SSLCERTSUBJ= system option 40
   SSLCIPHERLIST system option 42
   SSLCLIENTAUTH system option 41
   SSLCRLCHECK system option 41
   SSLCRLLOC= system option 43
   SSLLMINPROTOCOL 45
   SSLMODE= system option 44
   SSSLPKCS12LOC= system option 47
   SSSLPKCS12PASS= system option 48
   SSLPVTKEYLOC= system option 48
   SSLPVTKEYPASS= system option 49
   SSLREQCERT= system option 50
   SSLSNIIHOSTNAME system option 51
   stream cipher 18
   subject name of digital certificate 40
   system option
      SSLCACERTDATA 36
      SSLCACERTDIR 34
      SSLCIPHERLIST 42
      SSLLMINPROTOCOL System Option 45
      SSLSNIIHOSTNAME 51
system options
   PDF 22
   SSLMODE 44
system requirements
   SAS Proprietary Encryption 7
   SAS/SECURE 10
   SSH 16
   TLS on UNIX 93
   TLS on Windows 116
   TLS on z/OS 125

T
   TLS 113
   TLS (Transport Layer Security)
      installation 14
   TLS (Secure Sockets Layer)
      SAS/CONNECT z/OS spawner example 75
   TLS (Transport Layer Security) 11
      comparison of technologies 20
      concepts 12
      configuration 14
      name of issuer of digital certificate 37
      overview 11
      password for decrypting private key 48, 49
   SAS/CONNECT UNIX spawner example 71
   SAS/CONNECT Windows spawner example 73
   SAS/SHARE on UNIX example 77
   SAS/SHARE on Windows examples 79
   SAS/SHARE on z/OS example 80
   serial number of digital certificate 39
   SSLREQCERT= 50
   subject name of digital certificate 40
   system and software requirements on Windows 116
   system and software requirements on z/OS 125
   TLS version support 13
   trusted certificate authorities 32
   TLS (Transport Security Layer)
      system and software requirements on UNIX 93
   Transport Layer Security
      See TLS (Transport Layer Security)
   TripleDES algorithm 10, 19
      trust lists 128
      trusted certificate authorities (CAs)
         digital certificate location 32
         tunneling 16, 17
         installation and setup 17
      SSH for SAS/CONNECT example 81
      SSH for SAS/SHARE example 82

U
   UNIX
     converting between PEM and DER formats 104
     Creating a certificate trustlist 102, 104
     creating a digital certificate request 97
OpenSSL on  97
SAS Deployment Wizard  110
SAS/CONNECT client example  70
SAS/CONNECT server example  70
TLS on  93
TLS SAS/CONNECT spawner example  71
TLS SAS/SHARE example  77
TLS system and software requirements  93
Validate Certificates  112
Verify certificates in trustlist  103

W
Windows
converting between PEM and DER formats  121
Creating a certificate trustlist  122

Z
z/OS
Creating a certificate trustlist  130
creating digital certificates  126
TLS on  125
TLS SAS/CONNECT spawner example  75
TLS SAS/SHARE example  80
TLS system and software requirements  125
Verify certificates in trustlist  129
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.