THE POWER TO KNOW.

Encryption in SAS® 9.2

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2009. *Encryption in SAS® 9.2*. Cary, NC: SAS Institute Inc.

Encryption in SAS® 9.2

ISBN 978-1-59994-865-2

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, February 2009
2nd electronic book, May 2010

1st printing, February 2009

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

Contents

What's New

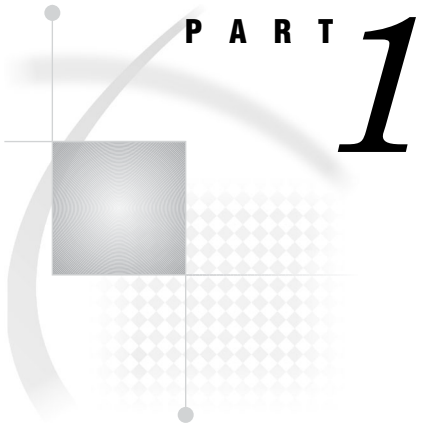
Overview

Here are the new and enhanced features supported for Encryption in SAS:

- encoded passwords for SAS data sets
- AES data encryption algorithm
- Secure Sockets Layer (SSL) for the z/OS operating environment
- Secure Shell (SSH) for the z/OS operating environment
- new system options, SSLPKCS12LOC= and SSLPKCS12PASS=, for the z/OS and UNIX operating environments
- new PDFSECURITY= system option

General Enhancements

- This release introduces support of encoded passwords for SAS data sets. The ability to accept encoded passwords is now supported by all areas of password handling within SAS.
- The AES data encryption algorithm is new to SAS/SECURE and SSL.
- This release introduces support for SSL under the z/OS and OpenVMS operating environments. Examples are provided for SAS/CONNECT and SAS/SHARE .
- This release introduces support for SSH under the z/OS operating environment.
- New system options introduced in this release are SSLPKCS12LOC= and SSLPKCS12PASS=.
- The PDFSECURITY= option is introduced to allow encrypting of ODS generated PDF files. This option along with other PDF system options can be used to restrict or access PDF files.



PART 1

Encryption in SAS 9.2

CHAPTER

1

Technologies for Encryption

Encryption: Overview

There is a great need to ensure the confidentiality of business transactions over a network between an enterprise and its consumers, between enterprises, and within an enterprise. SAS products and third-party strategies for protecting data and credentials (user IDs and passwords) are exchanged in a networked environment. This process of protecting data is called *encryption*. Encryption is the transformation of intelligible data (plaintext) into an unintelligible form (ciphertext) by means of a mathematical process. The ciphertext is translated back to plaintext when the appropriate key that is necessary for decrypting (unlocking) the ciphertext is applied.

SAS offers two classes of encryption strength:

- If you do not have SAS/SECURE, only the SASProprietary algorithm is available. SASProprietary uses 32-bit fixed encoding and is appropriate only for preventing

accidental exposure of information. SASProprietary is licensed with Base SAS software and is available in all deployments.

- If you have SAS/SECURE, you can use an industry standard encryption algorithm instead of the SASProprietary algorithm. SAS/SECURE is an add-on product that is licensed separately.

Encryption helps to protect information on-disk and in-transit as follows:

- *Over-the-wire* encryption protects SAS data and data while in transit. Passwords in transit to and from SAS servers are encrypted or encoded.
- *On-disk* encryption protects data at rest. Passwords in configuration files and the metadata are encrypted or encoded. Configuration files and metadata repository data sets are also host protected.

Providers of Encryption

SASProprietary

SASProprietary Overview

SASProprietary is a fixed encoding algorithm that is included with Base SAS software. It requires no additional SAS product licenses. The SAS proprietary algorithm is strong enough to protect your data from casual viewing. SASProprietary provides a medium level of security. SAS/SECURE and SSL provide a high level of security.

SASProprietary System Requirements

SAS 9.2 supports SASProprietary under these operating environments:

- OpenVMS
- UNIX
- Windows
- z/OS

SASProprietary Installation and Configuration

SASProprietary is part of Base SAS. Separate installation is not required.

For an example of configuring and using SASProprietary in your environment, see "SASProprietary for SAS/SHARE: Example" on page 32.

SAS/SECURE

SAS/SECURE Overview

SAS/SECURE software is an add-on product that provides industry standard encryption capabilities in addition to the SASProprietary algorithm. SAS/SECURE requires a license, and it must be installed on each computer that runs a Foundation SAS client and a server that will use the encryption algorithms.

Note: SAS/SECURE provides encryption of data in transit. It does not provide authentication or authorization capabilities. △

SAS/SECURE System Requirements

SAS 9.2 supports SAS/SECURE under these operating environments:

- UNIX
- Windows
- z/OS

Export Restrictions for SAS/SECURE

For software licensing and delivery purposes, SAS/SECURE is the product within the SAS System. For U.S. export licensing purposes, SAS designates each product based on the encryption algorithms and the product's functional capability. SAS/SECURE 9.2 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 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.

SAS/SECURE Installation and Configuration

SAS/SECURE must be installed on the SAS server computer, the client computer, and possibly other computers, depending on the SAS software that requires encryption. For installation details, see the SAS documentation for the software that uses encryption.

For examples of configuring and using SAS/SECURE in your environment, see Chapter 4, "Encryption Technologies: Examples," on page 31.

Secure Sockets Layer (SSL)

Secure Sockets Layer (SSL) Overview

SSL is an abbreviation for Secure Sockets Layer, which is a protocol that provides network data privacy, data integrity, and authentication. Developed by Netscape Communications, SSL uses encryption algorithms that include RC2, RC4, DES, TripleDES, AES and others.

In addition to providing encryption services, SSL performs client and server authentication, and it uses message authentication codes to ensure data integrity. SSL is supported by Netscape Navigator, Internet Explorer, and Mozilla Firefox. Many Web sites use the protocol to protect confidential user information, such as credit card numbers. The SSL protocol is application independent and allows protocols such as HTTP, FTP, and Telnet to be transparently layered above it. SSL is optimized for HTTP. SSL includes software that was developed by the OpenSSL Project for use in the OpenSSL Toolkit. For more information see **www.OpenSSL.org**.

Note: Transport Layer Security (TLS) is the successor to SSL V3.0. The Internet Engineering Task Force (IETF) took SSL V3.0, which was the *de facto* standard, modified it, renamed it TLS V1.0, and adopted it as a standard. △

SSL System Requirements

SAS 9 and later releases support SSL V2.0, SSL V3.0 and TLS V1.0.
SAS 9.2 supports SSL under these operating environments:

- UNIX
- Windows
- z/OS (new for SAS 9.2)
- OpenVMS

Note: The SAS/SECURE SSL software is included in the SAS installation software only for countries that allow the importation of encryption software. △

SSL Concepts

The following concepts are fundamental to understanding SSL:

Certification Authorities (CAs)
: Cryptography products provide security services by using digital certificates, public-key cryptography, private-key cryptography, and digital signatures. Certification 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. △

  From a trusted CA, members of an enterprise can obtain digital certificates to facilitate their e-business needs. The CA provides a variety of ongoing services to

the business client that include handling digital certificate requests, issuing digital certificates, and revoking digital certificates.

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.

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. Receipt of a document that contains a digital signature enables the receiver to verify the source of the document. Electronic documents can be verified if you know 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 document has not been changed since it was signed. A MAC is attached to a document to indicate the document's authenticity. Receipt of the document that contains a MAC enables the receiver (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 Certification 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.

SSL Installation and Configuration

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

- Appendix 1, "Installing and Configuring SSL under UNIX," on page 49
- Appendix 2, "Installing and Configuring SSL under Windows," on page 55
- Appendix 3, "Installing and Configuring SSL under z/OS," on page 61

For examples of configuring and using SSL in your environment, see Chapter 4, "Encryption Technologies: Examples," on page 31.

SSH (Secure Shell)

SSH (Secure Shell) Overview

SSH is an abbreviation for Secure Shell, which 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.

SSH System Requirements

OpenSSH supports SSH protocol versions 1.3, 1.5, and 2.0.
SAS 9.2 supports SSH under these operating environments:

- UNIX
- Windows
- z/OS

For additional resources, see

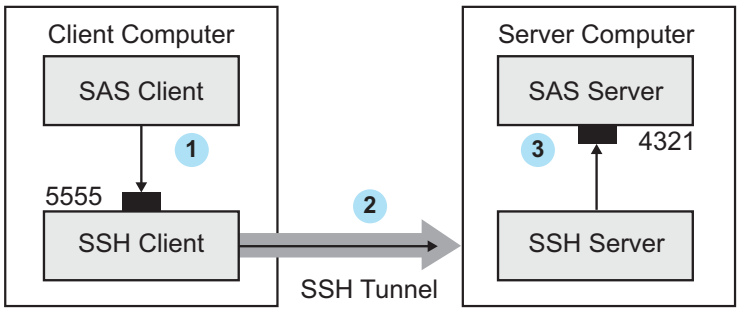
- **`www.openssh.com`**
- **`www.ssh.com`**
- ssh(1) UNIX manual page.

Under z/OS, the IBM Ported Tools for z/OS Program Product must be installed for OpenSSH support. See **`www-03.ibm.com/servers/eserver/zseries/zos/unix/port_tools.html`**.

SSH Tunneling Process

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

Figure 1.1 SSH Tunneling Process



❶ The SAS client passes its request to the SSH client's port 5555.
❷ The SSH client forwards the SAS client's request to the SSH server via an encrypted tunnel.
❸ 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:

- SSH tunneling software is installed on the client and server computers. Details about tunnel configuration depend on the specific SSH product that is used.
- The SSH client is started as an agent between the SAS client and the SAS server.
- The components of the tunnel are set up. The components are a "listen" port, a destination computer, and a destination port. The SAS client will access the listen port, which gets 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 40 and "SSH Tunnel for SAS/SHARE: Example" on page 41.

Encryption Algorithms

The following encryption algorithms are used by SASProprietary and SAS/SECURE:

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 a proprietary algorithm developed by RSA Data Security, Inc.

Note: RC2 is supported in SAS/SECURE and SSL. △

RC4
is a stream cipher. A *stream cipher* is an encryption algorithm that encrypts data 1 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 a proprietary algorithm developed by RSA Data Security, Inc.

Note: RC4 is supported in SAS/SECURE and SSL. △

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: DES is supported in SAS/SECURE and SSL. △

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 supported in SAS/SECURE and SSL. △

SASProprietary
is a cipher that provides basic fixed encoding encryption services under all operating environments that are supported by SAS. Included in Base SAS,

SASProprietary does not require additional SAS product licenses. The algorithm expands a single message to approximately one-third by using 32-bit encoding.

Note: SASProprietary is supported only by the SASProprietary encryption provider. △

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, and is one of the most popular algorithms that is used in symmetric key cryptography. AES is published as a U.S. Government Federal Information Processing Standard (FIPS 197).

Note: AES is supported in SAS/SECURE and SSL. △

Here is a summary of the encryption algorithms, by operating environment:

Table 1.1 Encryption Algorithms Supported by Operating Environments

	Operating Environments			
Encryption Algorithms	OpenVMS	UNIX	Windows	z/OS
SASProprietary	X	X	X	X
RC2		X	X	X
RC4		X	X	X
DES		X	X	X
TripleDES		X	X	X
SSL	X	X	X	X
AES		X	X	X

Encryption: Comparison

The following table compares the features of the encryption technologies:

Table 1.2 Summary of SASProprietary, SAS/SECURE, SSL, and SSH Features

Features	**SASProprietary**	**SAS/SECURE**	**SSL**	**SSH**
License required	No	Yes	No	No
Encryption and authentication	Encryption only	Encryption only	Encryption and authentication	Encryption only
Encryption level	Medium	High	High	High
Algorithms supported	SASProprietary fixed encoding	RC2, RC4, DES, TripleDES, AES	RC2, RC4, DES, TripleDES, AES	Product dependent
Installation required	No (part of Base SAS)	Yes	Yes	Yes

Features	SASProprietary	SAS/SECURE	SSL	SSH
Operating environments supported	UNIX Windows z/OS	UNIX Windows z/OS	UNIX Windows z/OS * OpenVMS*	UNIX Windows z/OS
SAS version support	8 and later	8 and later	9 and later	8.2 and later

* SAS 9.2 introduces support for SSL on z/OS and OpenVMS.

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. For an example, see "SAS/SECURE for the IOM Bridge: Examples" on page 39.

If you work in a SAS session on a client computer that exchanges data with a SAS server, you will specify SAS system options that implement encryption for the duration of the SAS session. If you connect a SAS/CONNECT client to a spawner, you will 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 13. For examples, see Chapter 4, "Encryption Technologies: Examples," on page 31.

Accessibility Features in SAS Products

For information about accessibility for any of the products mentioned in this book, see the documentation for that product. If you have questions or concerns about the accessibility of SAS products, send e-mail to accessibility@sas.com

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. In SAS 9.2, you can encrypt and password-protect your PDF output files by specifying the PDFSECURITY system option. Two levels of security are available: 40-bit (low) and 128-bit (high). With either of these settings, a password will be required to open a PDF file that has been generated with ODS.

You can find information on using the ODS PRINTER and PDF statements in the *SAS Output Delivery System: User's Guide*. The following table lists the PDF system options that are available to restrict or allow users' ability to access, assemble, copy, or modify ODS PDF files. Other SAS system options control whether the user can fill in forms and set the print resolution. These system options are documented in *SAS Language Reference: Dictionary*.

Table 1.3 PDF System Options

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

Note: The SAS/SECURE SSL software is included in the SAS installation software only for countries that allow the importation of encryption software. △

CHAPTER 2

SAS System Options for Encryption

NETENCRYPT System Option

Specifies whether client/server data transfers are encrypted.

Client: Optional

Server: Optional

Default: NONETENCRYPT

Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Operating Environments: OpenVMS, UNIX, Windows, z/OS

See also: NETENCRYPTALGORITHM

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

NETENCRYPT | NONETENCRYPT

Syntax Description

NETENCRYPT
specifies that encryption is required.

NONETENCRYPT
specifies that encryption is not required, but is optional.

Details

The default for this option specifies that encryption is used if the NETENCRYPTALGORITHM option is set and if both the client and the server are capable of encryption. If encryption algorithms are specified but either the client or the server is incapable of encryption, then encryption is not performed.

Encryption might not be supported at the client or at the server in these situations:

- You are using a release of SAS (before SAS 8) that does not support encryption.
- Your site (the client or the server) does not have a security software product installed.
- You specified encryption algorithms that are incompatible in SAS sessions on the client and the server.

NETENCRYPTALGORITHM= System Option

Specifies the algorithm or algorithms to be used for encrypted client/server data transfers.

Client: Optional

Server: Required

Alias: NETENCRALG=

Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Operating Environments: OpenVMS, UNIX, Windows, z/OS

See also: NETENCRYPT

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

NETENCRYPTALGORITHM=*algorithm | ("algorithm-1"... "algorithm-n")*

Syntax Description

algorithm | ("algorithm–1"... "algorithm-n")
specifies the algorithm or algorithms that can be used for encrypting data that is transferred between a client and a server across a network. When you specify two or more encryption algorithms, use a space or a comma to separate them, and enclose the algorithms in parentheses.

The following algorithms can be used:

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

Note: The SSL option is not applicable to the Integrated Object Model (IOM) metadata, OLAP, and table servers. △

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

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

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

Table 2.1 Client/Server Connection Outcomes

Server Settings	Client Settings	Connection Outcome
NONETENCRYPT NETENCRALG=*alg*	No settings	If the client is capable of encryption, the client/server connection will be encrypted. Otherwise, the connection will not be encrypted.
NETENCRYPT NETENCRALG=*alg*	No settings	If the client is capable of encryption, the client/server connection will be encrypted. Otherwise, the client/server connection will fail.
No settings	NONETENCRYPT NETENCRALG=*alg*	A client/server connection will not be encrypted.
No settings	NETENCRYPT NETENCRALG=*alg*	A client/server connection will fail.
NETENCRYPT or NONETENCRYPT NETENCRALG=*alg–1*	NETENCRALG=*alg-2*	Regardless of whether NETENCRYPT or NONETENCRYPT is specified, a client/server connection will fail.

Example

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

The client specifies two algorithms in the following OPTIONS statement:

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

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

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

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

NETENCRYPTKEYLEN= System Option

Specifies the key length that is used by the encryption algorithm for encrypted client/server data transfers.

Client: Optional

Server: Optional

Alias: NETENCRKEY=

Default: 0

Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Operating Environments: UNIX, Windows, z/OS

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

NETENCRYPTKEYLEN= 0 | 40 | 128

Syntax Description

0
specifies that the maximum key length that is supported at both the client and the server is used.

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

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

Details

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

By default, if you try to connect a computer that is capable of only a 40-bit key length to a computer that is capable of both a 40-bit and a 128-bit key length, the connection is

made using the lesser key length. If both computers are capable of 128-bit key lengths, a 128-bit key length is used.

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

SSLCALISTLOC= System Option

Specifies the location of digital certificates for trusted certification authorities (CA).

Client: Required

Server: Optional

Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Operating Environment: UNIX, z/OS

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCALISTLOC="*file–path*"

Syntax Description

"file-path"
specifies the location of a file that contains the digital certificates for the trusted certification authority (CA).

Details

The SSLCALISTLOC= option identifies the certification authority that SSL should trust. This option is required at the client because at least one CA must be trusted in order to validate a server's digital certificate. This option is required at the server only if the SSLCLIENTAUTH option is also specified at the server.

The CA list must be PEM-encoded (base64). Under z/OS, the file must be formatted as ASCII and must reside in a UNIX file system.

SSLCERTISS= System Option

Specifies the name of the issuer of the digital certificate that SSL should use.

Client: Optional

Server: Optional

Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Operating Environment: Windows

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

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 SSL.

Details

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

SSLCERTLOC= System Option

Specifies the location of the digital certificate that is used for authentication.

Client: Optional

Server: Required

Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Operating Environment: UNIX, z/OS

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCERTLOC="*file-path*"

Syntax Description

"file-path"
specifies the location of a file that contains a digital certificate.

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.

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.

SSLCERTSERIAL= System Option

Specifies the serial number of the digital certificate that SSL should use.

Client: Optional

Server: Optional

Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Operating Environment: Windows

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCERTSERIAL="*serial-number*"

Syntax Description

"serial-number"
specifies the serial number of the digital certificate that should be used by SSL.

Details

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

SSLCERTSUBJ= System Option

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

Client: Optional

Server: Optional

Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Operating Environment: Windows

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCERTSUBJ="*subject-name*"

Syntax Description

"subject-name"
specifies the subject name of the digital certificate that SSL should use.

Details

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

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

Operating Environments: UNIX, Windows, z/OS

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCLIENTAUTH | NOSSLCLIENTAUTH

Syntax Description

SSLCLIENTAUTH
specifies that the server should perform client authentication.

NOSSLCLIENTAUTH
specifies that the server should not perform client authentication.

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

Operating Environments: UNIX, Windows, z/OS

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

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 Certification Authority (CA) and contains a list of revoked digital certificates. The list contains only the revoked digital certificates that were issued by a specific CA.

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

SSLCRLLOC= System Option

Specifies the location of a Certificate Revocation List (CRL).

Client: Optional

Server: Optional

Operating Environment: UNIX, z/OS

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCRLLOC="*file-path*"

Syntax Description

"file-path"
specifies the location of a file that contains a Certificate Revocation List (CRL).

Details

The SSLCRLLOC= option is required only when the SSLCRLCHECK option is specified.

SSLPKCS12LOC= System Option

Specifies the location of the PKCS #12 encoding package file.

Client: Optional

Server: Optional

Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Operating Environment: UNIX, z/OS

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

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.

Note: If you run in a z/OS operating environment, this file must be in the UNIX file system. The OpenSSL library cannot read MVS data sets. △

Details

If the SSLPKCS12LOC= option is specified, the PKCS #12 DER encoding package must contain both the certificate and private key. The SSLCERTLOC= and SSLPVTKEYLOC= options will be ignored.

SSLPKCS12PASS= System Option

Specifies the password that SSL requires for decrypting the private key.

Client: Optional

Server: Optional

Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Operating Environment: UNIX, z/OS

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLPKCS12PASS=*password*

Syntax Description

password
specifies the password that SSL 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.

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

Operating Environment: UNIX, z/OS

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

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.

SSLPVTKEYPASS= System Option

Specifies the password that SSL requires for decrypting the private key.

Client: Optional

Server: Optional

Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Operating Environment: UNIX, z/OS

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLPVTKEYPASS="*password*"

Syntax Description

"password"
specifies the password that SSL requires in order to decrypt the private key. The private key is stored in the file that is specified by using the SSLPVTKEYLOC= option.

Details

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

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

CHAPTER

3

The PWENCODE Procedure

Overview: PWENCODE Procedure

The PWENCODE procedure enables you to encode passwords. Encoded passwords can be used in place of plaintext passwords in SAS programs that access relational database management systems (RDBMSs) and various servers, such as SAS/CONNECT servers, SAS/SHARE servers, and SAS Integrated Object Model (IOM) servers (such as the SAS Metadata Server).

Syntax: PWENCODE Procedure

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

PROC PWENCODE Statement

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. If the password contains embedded single or double quotation marks, use the standard SAS rules for quoting character constants (see "SAS Constants in Expressions" in *SAS Language Reference: Concepts* for details).

Featured in: Example 1 on page 27, Example 2 on page 28, and Example 3 on page 29

Options

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.

Featured in: Example 2 on page 28

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

Table 3.1 Supported Encoding Methods

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

* SAS/SECURE is an add-on product that requires a separate license. For details about SAS/SECURE, the SASProprietary algorithm, and the AES algorithm, see *Encryption in SAS*.

If the METHOD= option is omitted, the default encoding method, **sas002**, is used automatically.

Concepts: PWENCODE Procedure

Using Encoded Passwords in SAS Programs

When a password is encoded with PROC PWENCODE, the output string includes a *tag* that identifies the string as having been encoded. An example of a tag is **{sas001}**.

The tag indicates the encoding method. SAS servers and SAS/ACCESS engines recognize the tag and decode the string before using it. Encoding a password enables you to write SAS programs without having to specify a password in plaintext.

Encoding versus Encryption

PROC PWENCODE uses *encoding* to disguise passwords. With encoding, one character set is translated to another character set through some form of table lookup. *Encryption*, by contrast, involves the transformation of data from one form to another through the use of mathematical operations and, usually, a "key" value. Encryption is generally more difficult to break than encoding. PROC PWENCODE is intended to prevent casual, non-malicious viewing of passwords. You should not depend on PROC PWENCODE for all your data security needs; a determined and knowledgeable attacker can decode the encoded passwords.

Examples: PWENCODE Procedure

Example 1: Encoding a Password

Procedure features: IN= argument

This example shows a simple case of encoding a password and writing the encoded password to the SAS log.

Program

Encode the password.

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

Log

Output 3.1

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

{sas002}bXkgcGFzc3dvcmQ=

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.31 seconds
      cpu time            0.08 seconds
```

Example 2: Using an Encoded Password in a SAS Program

Procedure features:
IN= argument
OUT= option

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

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 will be written to.

```
proc pwencode in='mypass1' out=pwfile;
run;
```

Program 2: Using the Encoded Password

Declare a fileref for the encoded-password file.

```
filename pwfile 'external-filename';
```

Set the SYMBOLGEN SAS system option. The purpose of this step is to show 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. For more information about the SYMBOLGEN SAS system option, see **SAS Macro Language: Reference**.

```
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. For details about the INFILE and INPUT statements, the $VARYING. informat, and the CALL SYMPUT routine, see **SAS Language Reference: Dictionary**.

```
data _null_;
   infile pwfile obs=1 length=l;
```

```
   input @;
   input @1 line $varying1024. l;
   call symput('dbpass',substr(line,1,l));
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

```
28   data _null_;
29      infile pwfile obs=1 length=len;
30      input @;
31      input @1 line $varying1024. len;
32      call symput('dbpass',substr(line,1,len));
33   run;

NOTE: The infile PWFILE is:
      File Name=external-filename,
      RECFM=V,LRECL=256

NOTE: 1 record was read from the infile PWFILE.
      The minimum record length was 20.
      The maximum record length was 20.
NOTE: DATA statement used (Total process time):
      real time           3.94 seconds
      cpu time            0.03 seconds


34   libname x odbc
SYMBOLGEN:  Macro variable DBPASS resolves to {sas002}bXlwYXNzMQ==
34 !               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

Procedure features:

IN= argument

OUT= option

Other features:

FILENAME statement with CLIPBRD access method

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

Declare a fileref with the CLIPBRD access method. For more information about the FILENAME statement with the CLIPBRD access method, see **SAS Language Reference: Dictionary**.

```
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;
```

Example 4: Specifying an Encoding Method for a Password

Procedure features: METHOD= argument

This example shows a simple case of encoding a password using the **sas003** encoding method and writing the encoded password to the SAS log.

Program

Encode the password.

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

Log

Output 3.2

```
6    proc pwencode in='my password' encoding=sas003;
7    run;

{sas003}6EDB396015B96DBD9E80F0913A543819A8E5

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           0.14 seconds
      cpu time            0.09 seconds
```

CHAPTER

4

Encryption Technologies: Examples

SAS/SECURE for SAS/CONNECT: Example

SAS/CONNECT Client under UNIX

The following statements configure the client. The NETEN3CRYPTALGORITHM= option specifies the use of the RC4 algorithm.

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

SAS/CONNECT Server under UNIX

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

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

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

```
#!/bin/ksh
#__________________
# mystartup
#__________________
. ~/.profile
sas dmr -noterminal -comamid tcp $*
```

SASProprietary for SAS/SHARE: Example

SAS/SHARE Client

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

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

SAS/SHARE Server

This example shows how to set the options for encryption services on a SAS/SHARE server. The NETENCRYPT option specifies that encryption is required by any client

that accesses this server. The NETENCRYPTALGORITHM= option specifies that the SASProprietary algorithm be used for encryption of all data that is exchanged with connecting clients.

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

SSL for a SAS/CONNECT UNIX Spawner: Example

Startup 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.

For example:

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

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

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

SAS Commands and Arguments	Function
sastcpd	Starts the spawner
-service unxspawn	Specifies the spawner service (configured in the services file)
-netencryptalgorithm ssl	Specifies the SSL encryption algorithm
-sslcertloc /users/server/certificates/server.pem	Specifies the file path for the location of the server's certificate
-sslpvtkeyloc /users/server/certificates/ serverkey.pem	Specifies the file path for the location of the server's private key
-sslpvtkeypass password	Specifies the password to access the server's private key
-sslcalistloc /users/server/certificates/sas.pem	Specifies the CA trust list
-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

Here is an example of an executable file:

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

. ~/.profile
sas -dmr -noterminal $*
#-----------------------------
```

For complete information about starting a UNIX spawner, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Connection of a SAS/CONNECT Client to a UNIX Spawner

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

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

```
options netencryptalgorithm=ssl;
options sslcalistloc="/users/johndoe/certificates/cacerts.pem";
%let machine=apex.server.com;
signon machine.spawner user=_prompt_;
```

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

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

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

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

For complete information about connecting to a UNIX spawner, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

SSL for a SAS/CONNECT Windows Spawner: Example

Startup 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:

```
spawner -security -netencryptalgorithm ssl -sslcertsubj "apex.pc.com"
 -sascmd mysas.bat
```

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

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

SAS Command and Arguments	Function
spawner	Starts the spawner
-security	Specifies the requirement that a client provide a user name and password to access the spawner
-netencryptalgorithm ssl	Specifies the SSL encryption algorithm
-sslcertsubj "apex.pc.com"	Specifies the subject name that is used to search for a certificate from the Microsoft Certificate Store
-sascmd mysas.bat	Specifies the name of an executable file that starts a SAS session when you sign on without a script file

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 SSL should use. The subject that is assigned to the -SSLCERTSUBJ option and the computer that is specified in the client signon must be identical.

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 SSL to locate the appropriate CA digital certificate.

For complete information about starting a Windows spawner, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

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

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

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

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

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

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

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

SAS Options, Statements, and Arguments	Function
COMAMID=tcp	Specifies the TCP/IP access method
NETENCRYPTALGORITHM=ssl	Specifies the encryption algorithm
SIGNON=server-ID	Specifies which server to connect to
USER=_PROMPT_	Prompts for the user ID and password to be used for authenticating the client to the server

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

SSL for SAS/SHARE under UNIX: Example

Startup 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 authenticate=opt;
run;
```

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

Table 4.5 SAS Options and Statements for Server Start-Up Tasks

SAS Options and Statements	Server Start-Up Tasks
TCPSEC= _SECURE_	Secures the server
NETENCRALG=SSL	Specifies SSL as the encryption algorithm
SSLCERTLOC=server.pem	Specifies the filepath for the location of the server's certificate

SAS Options and Statements	Server Start-Up Tasks
SSLPVTKEYLOC=serverkey.pem	Specifies the filepath for the location of the server's private key
SSLPVTKEYPASS="password"	Specifies the password to access server's private key
PROC SERVER ID=shrserv	Starts the server
AUTHENTICATE=opt	Allow trusted users to access the server without authentication.

Note: As an alternative to using the SSLPVTKEYPASS= option to protect the private key, you might prefer that the private key remain unencrypted, and use the file system permissions to prevent read and write access to the file that contains the private key. To store the private key without encrypting it, use the –NODES option when requesting the certificate. △

SAS/SHARE Client Access of a SAS/SHARE Server

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

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

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

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

Table 4.6 SAS Options and Arguments Tasks for Accessing a SAS/SHARE Server from a Client

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

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

SSL for SAS/SHARE under Windows: Examples

Startup of a Multi-User SAS/SHARE Server

After certificates for the CA, the server, and the client have been generated, and imported into the appropriate certificate store, you can start a SAS/SHARE server.

Here is an example of how to start a secured SAS/SHARE server:

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

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

Table 4.7 SAS Options, Statements, and Arguments for Server Start-Up Tasks

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

SAS/SHARE Client Access of a SAS/SHARE Server

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

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

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

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

Table 4.8 SAS Options and Arguments for Accessing a SAS/SHARE Server from a Client

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

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

SAS/SECURE for the IOM Bridge: Examples

IOM Bridge Encryption Configuration

The IOM Bridge for SAS clients can use SAS/SECURE to encrypt network data between SAS and its clients.

SAS/SECURE must be installed at the SAS server and at the SAS client. SAS clients include COM clients and Java clients.

You can configure encryption properties in either metadata or in code.

- "IOM Bridge for SAS Clients: Metadata Configuration" on page 39
- "IOM Bridge for COM: Configuration in Code" on page 39
- "IOM Bridge for Java: Configuration in Code" on page 40

IOM Bridge for SAS Clients: Metadata Configuration

In order to connect a SAS client to a SAS server, the CreateObjectByLogicalName function must obtain encryption information from metadata that is stored in the metadata repository. SAS Management Console can be used to configure encryption properties into the metadata repository, as follows:

Required encryption level

In SAS Management Console, follow this path:
<Connection> ▸ Options ▸ Advanced Options ▸ Encryption ▸ Required Encryption Level

Valid values for required encryption levels are as follows:

None
: No encryption

Credentials
: Only user credentials (ID and password) are encrypted. This is the default.

Everything
: All client/server transfers, including credentials, are encrypted.

Server encryption algorithm

In SAS Management Console, follow this path:
<Connection> ▸ Options ▸ Advanced Options ▸ Encryption ▸ Server Encryption Algorithms

Valid values for server encryption algorithms are RC2, RC4, DES, TRIPLEDES, AES, and SASPROPRIETARY (the default).

For complete details about using SAS Management Console to configure the IOM Bridge, visit **`supportexp.unx.sas.com/rnd/itech/doc9/admin_oma/sasserver/iombridge`**.

IOM Bridge for COM: Configuration in Code

When using the CreateObjectByServer function to connect a Windows client to a SAS server, specify the following properties in your client code in the ServerDef object:

- BridgeEncryptionLevel

- BridgeEncryptionAlgorithm

Here is an example:

```
obServerDef.BridgeEncryptionLevel=EncryptAll;
obServerDef.BridgeEncryptionAlgorithm="TripleDes";
```

EncryptAll
causes all data, including credentials (user IDs and passwords), to be encrypted in client/server transfers.

TripleDes
is the specific encryption algorithm to be applied to data transfers.

For a complete list of encryption values, see the SAS Object Manager class reference (sasoman.chm).

IOM Bridge for Java: Configuration in Code

When using the BridgeServer object to connect a Java client to a SAS server, use the following functions to specify your encryption settings:

- setEncryptionContent
- setEncryptionAlgorithms
- setEncryptionPolicy

Here is an example:

```
obBridgeServer.setEncryptionContent(BridgeServer.ENCRYPTION_CONTENT_ALL);
obBridgeServer.setEncryptionAlgorithms(BridgeServer.ENCRYPTION_ALGORITHM_TRIPLEDES);
obBridgeServer.setEncryptionPolicy(BridgeServer.ENCRYPTION_POLICY_REQUIRED);
```

ENCRYPTION_CONTENT_ALL
causes all data, including credentials (user ID and password), to be encrypted in client/server transfers.

ENCRYPTION_ALGORITHM_TRIPLEDES
is the specific encryption algorithm to be applied to data transfers.

ENCRYPTION_POLICY_REQUIRED
specifies that encryption is required. If the server does not support encryption, the connection fails.

For a complete list of encryption values, see the Java reference for com.sas.services.connection at **www.support.sas.com**.

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:

```
sastcpd -service 4321
```

The UNIX spawner is started and is listening on destination port 4321. For complete details about starting a UNIX spawner, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

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

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

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

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

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

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

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

SSH Tunnel for SAS/SHARE: Example

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

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

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

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

SAS/SHARE Client Access of a SAS/SHARE Server

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

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

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

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

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

SSL for a SAS/CONNECT z/OS Spawner: Example

Startup 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.

For example:

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

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

Table 4.9 SAS Commands and Arguments for Spawner Start-Up Tasks

SAS Commands and Arguments	Function
sastcpd	Starts the spawner
-service 4321	Specifies the spawner service that is listening on part 4321
-netencryptalgorithm ssl	Specifies the SSL encryption algorithm
-sslpkcs12loc /users/server/certificates/serverkey.p12	Specifies the file path for the location of the server's PKCS #12 DER encoding package
-sslpkcs12pass password	Specifies the password to access the server's private key in the PKCS #12 package

SAS Commands and Arguments	Function
-sslcalistloc /users/server/certificates/sas.pem	Specifies the CA trust list
-sascmd /users/server/command.sh	Specifies the name of an executable file that starts a SAS session when you sign on without a script file

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

```
#!/bin/sh
args=$*
if [ -n "$INHERIT" ] ; then
   args="$args -inherit $INHERIT"
fi
if [ -n "$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 $args
```

For complete information about starting a z/OS spawner, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

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

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

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

```
options netencryptalgorithm=ssl;
options sslcalistloc="/users/johndoe/certificates/cacerts.pem";
%let machine=apex.server.com;
signon machine.spawner user=_prompt_;
```

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

Table 4.10 SAS Options and Arguments for Client Access to a SAS/CONNECT Server

SAS Options and Arguments	Client Access Tasks
NETENCRYPTALGORITHM=ssl	Specifies the encryption algorithm
SSLCALISTLOC=cacerts.pem	Specifies the CA trust list

SAS Options and Arguments	Client Access Tasks
SIGNON=*server-ID.service*	Specifies the server and service to connect to
USER=_PROMPT_	Prompts for the user ID and password to be used for authenticating the client to the server

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

For complete information about connecting to a z/OS spawner, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

SSL for SAS/SHARE under z/OS: Example

Startup of a Multi-User SAS/SHARE Server

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

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

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

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

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

SAS Options, Statements, and Arguments	Server Start-Up Tasks
TCPSEC= _SECURE_	Secures the server
NETENCRALG=SSL	Specifies SSL as the encryption algorithm
SSLPKCS12LOC=server.p12	Specifies the filepath for the location of the server's private key
SSLPKCS12PASS="password"	Specifies the password to access server's private key
PROC SERVER ID=shrserv	Starts the server
AUTHENTICATE=opt	Allows trusted users to access the server without authentication

SAS/SHARE Client Access of a SAS/SHARE Server

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

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

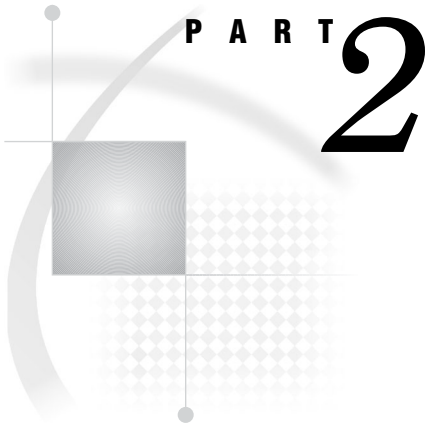
```
options sslcalistloc="/users/johndoe/certificates/cacerts.pem";
%let machine=apex.server.com;
libname a '.' server=machine.shrserv user=_prompt_;
```

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

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

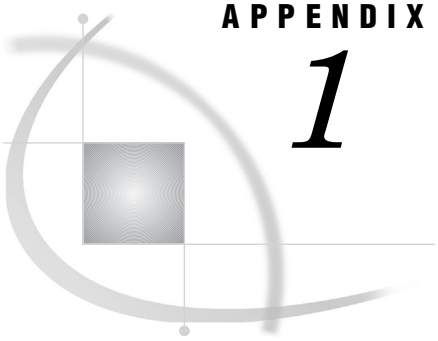
SAS Options and Arguments	Client Access Tasks
SSLCALISTLOC=cacerts.pem	Specifies the CA trust list
SERVER=machine.shrserv	Specifies the machine and server to connect to
USER=_PROMPT_	Prompts for the user ID and password to be used for authenticating the client to the server

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



PART 2

Installing and Configuring SSL



APPENDIX

1

Installing and Configuring SSL under UNIX

SSL under UNIX: System and Software Requirements

The system and software requirements for using SSL under UNIX operating environments are as follows:

- a computer that runs UNIX.
- Internet access and a Web browser such as Netscape Navigator or Internet Explorer.
- the TCP/IP communications access method.
- access to the OpenSSL utility at **`www.openssl.org/source`** if you plan to use the OpenSSL CA.
- knowledge of your site's security policy, practices, and technology. The properties of the digital certificates that you request are based on the security policies that have been adopted at your site.

Setting Up Digital Certificates for SSL under UNIX

Perform the following tasks to set up and use SSL:

1. Download and build SSL.
2. Create a digital certificate request.
3. Generate a digital certificate from the request.
4. View digital certificates.
5. End OpenSSL.
6. Create a CA trust list for the SSL client application.

Step 1. Download and Build SSL

If you want to use OpenSSL as your trusted Certification Authority (CA), follow the instructions for downloading and building OpenSSL that are given at **`www.openssl.org/source`**. For complete documentation about the OpenSSL utility, visit **`www.openssl.org/docs/apps/openssl.html`**.

The following sites provide information about alternative CA:

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

Step 2. Create a Digital Certificate Request

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

In this example, Proton, Inc. is the organization that is applying to become a CA by using OpenSSL. 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 the following tasks:

1. Select the **`apps`** subdirectory of the directory where OpenSSL was built.
2. Initialize OpenSSL.

```
$ openssl
```

3. Issue the appropriate command to request a digital certificate.

Table A1.1 Open SSL Commands for Requesting a Digital Certificate

Request Certificate for	OpenSSL Command
CA	req -config ./openssl.cnf -new -out sas.req -keyout saskey.pem -nodes
Server	req -config ./openssl.cnf -new -out server.req -keyout serverkey.pem
Client	req -config ./openssl.cnf -new -out client.req -keyout clientkey.pem

Table A1.2 Arguments and Values Used in OpenSSL Commands

OpenSSL Arguments and Values	Functions
req	Requests a certificate
-config ./openssl.cnf	Specifies the storage location for the configuration details for the OpenSSL program

OpenSSL Arguments and Values	Functions
-new	Identifies the request as new
-out sas.req	Specifies the storage location for the certificate request
-keyout saskey.pem	Specifies the storage location for the private key
-nodes	Prevents the private key from being encrypted

4 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 will prompt you for a password before allowing access to the private key. △

The following is an example of a request for a digital certificate:

```
OpenSSL> req -config ./openssl.cnf -new -out sas.req -keyout saskey.pem -nodes
Using configuration from ./openssl.cnf
Generating a 1024 bit RSA private key
.........................++++++
.........................................++++++
writing new private key to 'saskey.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [North Carolina]:
Locality Name (city) [Cary]:
Organization Name (company) [Proton Inc.]:
Organizational Unit Name (department) [IDB]:
Common Name (YOUR name) []: proton.com
Email Address []:Joe.Bass@proton.com
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
OpenSSL>
```

The request for a digital certificate is complete.

Note: For the server, the Common Name must be the name of the computer that the server runs on; for example, apex.serv.com. △

Step 3. Generate a Digital Certificate from the Request

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

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

Table A1.3 OpenSSL Commands for Generating Digital Certificates under UNIX

Generate Certificate for	OpenSSL Command
CA	x509 req -in sas.req -signkey saskey.pem -out sas.pem
Server	ca -config ./openssl.cnf -in server.req -out server.pem
Client	ca -config ./openssl.cnf -in client.req -out client.pem

Table A1.4 Arguments and Values Used in OpenSSL Commands under UNIX

OpenSSL Arguments and Values	Functions
x509	Identifies the certificate display and signing utility
req	Specifies that a certificate be generated from the request
ca	Identifies the Certification Authority utility
-config ./openssl.cnf	Specifies the storage location for the configuration details for the OpenSSL utility
-in filename.req	Specifies the storage location for the input for the certificate request
-out filename.pem	Specifies the storage location for the certificate
-signkey saskey.pem	Specifies the private key that will be used to sign the certificate that is generated by the certificate request

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 2003 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.

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.

The generation of a digital certificate is complete.

Step 4. View Digital Certificates

To view 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. End OpenSSL

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

Step 6. Create a CA Trust List for the SSL Client Application

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*.

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, create a new file and copy-and-paste into it the contents of all the digital certificates for CAs to be trusted by the client application.

Use the following template to create a CA trust list:

```
Certificate for OpenSSL CA

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

<PEM encoded certificate>

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


Certificate for Keon CA

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

<PEM encoded certificate>
```

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


Certificate for Microsoft CA

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


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

Because the digital certificate is encoded, it is unreadable. Therefore, the content of the digital certificate in this example is represented as **<PEM encoded certificate>** . The content of each digital certificate is delimited 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. In the preceding template, the file that is used contains the content of digital certificates for the CAs: OpenSSL, Keon, and Microsoft.

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 "Converting between PEM and DER File Formats for SSL" on page 59. △

Converting between PEM and DER File Formats for SSL

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

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
```

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

```
OpenSSL> x509 -inform DER -outform PEM -in server.der -out server.pem
```

APPENDIX

2

Installing and Configuring SSL under Windows

SSL under Windows: System and Software Requirements

The system and software requirements for using SSL under 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 such as Netscape Navigator or Internet Explorer.
- the TCP/IP communications access method.
- Microsoft Certificate Services add-on software.
- if you will run your own CA, the Microsoft Certification 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 SSL encryption.

  The Windows spawner must run on a computer 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 that you request will depend on the security policies that have been adopted at your site.

Setting Up Digital Certificates for SSL under Windows

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

- Configure SSL.
- Request a digital certificate.

Step 1. Configure SSL

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

The following keywords might be helpful when searching the Microsoft Web site:

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

Step 2. Request a Digital Certificate

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

- the Microsoft Certification Authority
- a certification authority that is not Microsoft

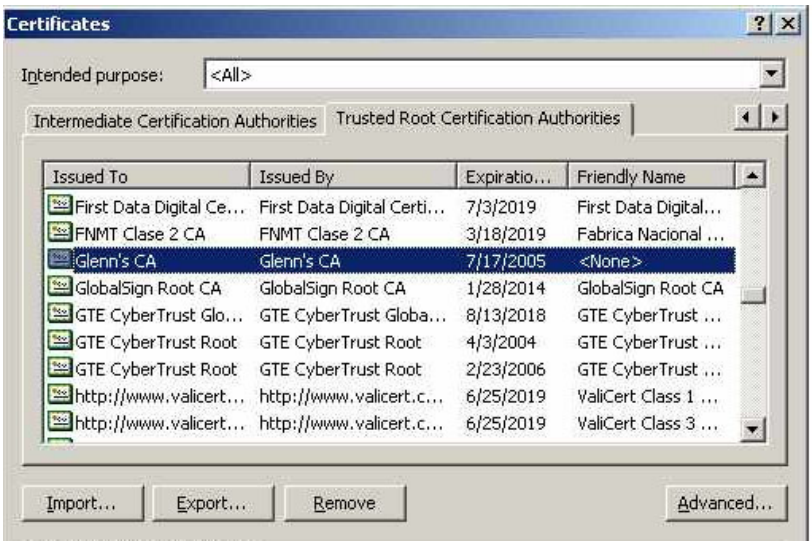
Request a Digital Certificate from the Microsoft Certification Authority

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

1. *System administrator*: If you are running your own CA, use Microsoft Certificate Services to create an active Certification Authority (CA).
2. *User*:
   a. Use the Certificate Request wizard to request a digital certificate from an active enterprise CA. The Certificate Request wizard lists all digital certificate types that the user can install.
   b. Select a digital certificate type.
   c. Select security options.
   d. Submit the request to an active CA that is configured to issue the digital certificate.

      After the CA issues the requested digital certificate, the digital certificate is automatically installed in the Certificate Store. The installed digital certificate is highlighted, as shown in Display A2.1 on page 57.

Display A2.1 Digital Certificate Installation in the Certificate Store



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

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

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

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

   Note: The Windows operating environment can import digital certificates that were generated in the UNIX operating environment. To convert from UNIX (PEM format) to Windows (DER format) before importing, see "Converting between PEM and DER File Formats for SSL" on page 59. △

For details about importing existing digital certificates, see "Import a Digital Certificate to a Certificate Store" on page 57.

Import a Digital Certificate to a Certificate Store

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

Certificate Type	Certificate Storage Location
Client	Personal Certificate Store
Server	Personal Certificate Store
CA (self-signed)	Trusted Root Certification Authorities

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

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

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

Specify the digital certificate to import to a Certificate Store by selecting the **Personal** tab in the **Certificates** window, as shown in Display A2.2 on page 58.

Display A2.2 Digital Certificate Selections for a Personal Certificate Store

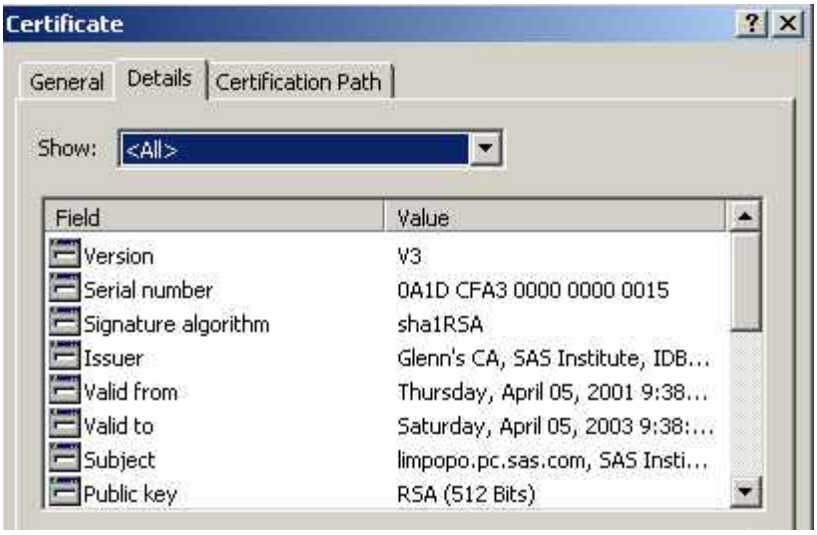
2 Click **Import** and follow the instructions to import digital certificates.

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

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

4 To view the details about a digital certificate, select the digital certificate and click **View**. Typical results are shown in Display A2.3 on page 59.

Display A2.3 Digital Certificate Details Tab



Converting between PEM and DER File Formats for SSL

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

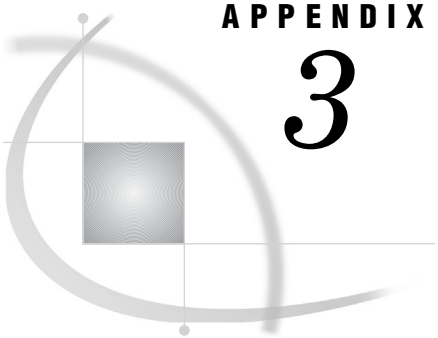
Under Windows, you can import a file that is created in either PEM or DER format. However, a digital certificate that is created in DER format must be converted to PEM format before it can be included in a trust list under UNIX.

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
```

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

```
OpenSSL> x509 -inform DER -outform PEM -in server.der -out server.pem
```



APPENDIX

3

Installing and Configuring SSL under z/OS

SSL under z/OS: System and Software Requirements

The system and software requirements for using SSL under 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.

Setting Up Digital Certificates for SSL under z/OS

Perform these tasks to set up and use SSL:

1. Authorize access to the RACDCERT command.
2. Create a CA digital certificate.
3. Create server and client digital certificates.
4. View digital certificates.
5. Create a CA trust list for the SSL client application.

Step 1. Authorize Access to the RACDCERT Command

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

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

The following sites provide information about alternative CAs:

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

Step 2. Create the Digital Certificate for the CA

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

In this example, Proton, Inc. is the organization that is applying to become a CA by using RACDCERT. After Proton, Inc. becomes a CA, it can serve as a CA for issuing digital certificates to clients (users) and servers on its network.

Perform these tasks:

1 Request a digital CA certificate. Here is an example of a request:

```
RACDCERT GENCERT CERTAUTH +
SUBJECTSDN( +
  CN('proton.com') +
  C('US') +
  SP('North Carolina') +
  L('Cary') +
  O('Proton Inc.') +
  OU('IDB') +
) +
ALTNAME( +
  EMAIL('Joe.Bass@proton.com') +
) +
WITHLABEL('Proton CA')
```

2 Export the CA certificate in PEM format:

```
RACDCERT CERTAUTH EXPORT(LABEL('Proton CA')) +
DSN(CA.CERT)
```

3 Copy the certificate to the UNIX file system.

Note: SSL certificate and key files must reside in the z/OS UNIX file system. The OpenSSL library cannot read MVS data sets. △

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

4 Convert the certificate file to ASCII format

Note: SSL 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.

You will specify the CA digital certificate using the SSLCALISTLOC= system option.

Step 3. Create the Server and Client Digital Certificates

Perform these tasks to create a digital certificate for a server and a client. The steps are identical for the server and the client. This example shows the tasks for the server.

1 Request a signed server certificate.

Here is an example of a request for a signed server certificate for user SERVER that runs on **proton.zos.com**.

```
RACDCERT GENCERT ID(SERVER) +
SUBJECTSDN( +
  CN('proton.zos.com') +
  C('US') +
  SP('North Carolina') +
  L('Cary') +
  O('Proton Inc.') +
  OU('IDB') +
) +
ALTNAME( +
  EMAIL('Joe.Bass@proton.com') +
) +
WITHLABEL('Proton Server') +
SIGNWITH(CERTAUTH LABEL('Proton CA'))
```

2 Export the server certificate and key that are specified in PKCS #12 DER encoding package format.

Note: The PKCS #12 DER encoding package is the format used by the RACDCERT utility to encode the exported certificate and private key for an entity, such as a server. It is a binary format. △

```
RACDCERT ID(SERVER) EXPORT(LABEL('Proton Server')) +
DSN(SERVER.P12) +
PASSWORD('abcd')
```

3 Copy the certificate to the UNIX file system.

Note: The PKCS #12 DER encoding package file must reside in the z/OS UNIX file system. The OpenSSL library cannot read MVS data sets. Because the file is already in binary format, its conversion to ASCII is unnecessary. △

```
cp //server.p12 server.p12
```

The creation of the server digital certificate and key is complete.

A PKCS #12 DER encoding package is the format that RACDCERT uses to export a certificate and a key for an entity. The exported package file contains both the certificate and the key. The content of the package file is secure by using the password that is specified in the RACDCERT EXPORT command.

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

Note: For the server, the Common Name must be the name of the computer that the server runs on (for example, **proton.zos.com**.) △

Step 4. View Digital Certificates

To view a digital certificate, issue these commands:

```
RACDCERT CERTAUTH LIST(LABEL('Proton CA'))
RACDCERT ID(SERVER) LIST(LABEL('Proton Server'))
```

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

To read the certificate files, issue these commands:

```
RACDCERT CHECKCERT(CA.CERT)
RACDCERT CHECKCERT(SERVER.P12) PASS('abcd')
```

Step 5. Create a CA Trust List for the SSL Client Application

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*.

If there is only one CA to trust (for example, Proton CA), in the client application, use the SSLCALISTLOC= option to specify the name of the file that contains the CA digital certificate, which was created in Step 2.

If multiple CAs are to be trusted by the client application, use the UNIX **cat** command to concatenate the contents of all the digital certificates for CAs. All the certificates must be encoded in PEM format and in ASCII format.

As an alternative method for creating a CA trust list, use this template to copy and paste the digital certificates into one file:

```
Certificate for Proton CA

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

<PEM encoded certificate>

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


Certificate for Keon CA

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

<PEM encoded certificate>

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

```
Certificate for Microsoft CA

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


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

Because the digital certificate is encoded, it is unreadable. Therefore, the content of the digital certificate in this example is represented as **<PEM encoded certificate>** . The content of each digital certificate is delimited using a **-----BEGIN CERTIFICATE-----** and **-----END CERTIFICATE-----** pair. All text outside the delimiters is ignored. Therefore, you might not want to use delimited lines for descriptive comments. In the preceding template, the file that is used contains the content of digital certificates for the CAs: Proton, Keon, and Microsoft.

Glossary

authentication
the process of verifying the identity of a person or process within the guidelines of a specific security policy.

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

Certificate Revocation List
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. Short form: CRL.

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.

CRL
See Certificate Revocation List.

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

data security technologies
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
a format that is used for creating SSL files in Windows operating environments. Short form: DER.

encryption
the act of transforming intelligible data (plaintext) into an unintelligible form (ciphertext) by means of a mathematical process.

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

PKCS #12
See Public Key Cryptography Standard #12.

plaintext
intelligible data. See also encryption and ciphertext.

port forwarding
See SSH tunnel.

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 and 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
a personal information exchange syntax standard. It defines a file format that is used to store private keys with accompanying public-key certificates. Short form: PKCS #12. See also SSL (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.

SAS/SECURE
an add-on product that uses the RC2, RC4, DES, and TripleDES encryption algorithms. SAS/SECURE requires a license, and it must be installed on each computer that runs a client and a server that will use the encryption algorithms. SAS/SECURE provides a high level of security.

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
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. Short form: SSH. See also SSH tunnel.

Secure Sockets Layer
See SSL (Secure Sockets Layer).

SSH
See Secure Shell.

SSH tunnel
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. Port forwarding is another term for tunneling. See also Secure Shell.

SSL (Secure Sockets Layer)
a protocol that provides network security and privacy. SSL uses encryption algorithms RC2, RC4, DES, TripleDES, and AES. SSL provides a high level of security. It was developed by Netscape Communications.

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

TLS (Transport Layer Security)
the successor to Secure Sockets Layer (SSL) V3.0. The Internet Engineering Task Force (IETF) adopted SSL V3.0 as the de facto standard, made some modifications, and renamed it TLS. TLS is virtually SSLV3.1. See also SSL (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.

Index

I

J

K

M

N

O

P

R

S

T

U

W

Z

Your Turn

We want your feedback.

- If you have comments about this book, please send them to **yourturn@sas.com**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **suggest@sas.com**.

SAS® Publishing Delivers!

Whether you are new to the work force or an experienced professional, you need to distinguish yourself in this rapidly changing and competitive job market. SAS® Publishing provides you with a wide range of resources to help you set yourself apart. Visit us online at support.sas.com/bookstore.

SAS® Press

Need to learn the basics? Struggling with a programming problem? You'll find the expert answers that you need in example-rich books from SAS Press. Written by experienced SAS professionals from around the world, SAS Press books deliver real-world insights on a broad range of topics for all skill levels.

support.sas.com/saspress

SAS® Documentation

To successfully implement applications using SAS software, companies in every industry and on every continent all turn to the one source for accurate, timely, and reliable information: SAS documentation. We currently produce the following types of reference documentation to improve your work experience:

- Online help that is built into the software.
- Tutorials that are integrated into the product.
- Reference documentation delivered in HTML and PDF – **free** on the Web.
- Hard-copy books.

support.sas.com/publishing

SAS® Publishing News

Subscribe to SAS Publishing News to receive up-to-date information about all new SAS titles, author podcasts, and new Web site features via e-mail. Complete instructions on how to subscribe, as well as access to past issues, are available at our Web site.

support.sas.com/spn

sas® THE POWER TO KNOW®