IPv6/VSE

SSL Installation, Programming and User's Guide

Table of Contents

Preface

About this Publication

This is the IPv6/VSE SSL Installation, Programming and User's Guide manual. The manual provides a reference for the SSL facilities available within IPv6/VSE.

Trademarks

The following are lists of the trademark and products referenced in this manual. Symbols for trademarks and registered trademarks do not appear in subsequent references.

Barnard Software, Inc.

TCP/IP-TOOLS and IPv6/VSE are registered trademarks of Barnard Software, Inc.

International Business Machines Corporation

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both. Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Copyrights

Technical Support

IBM Customers

IBM IPv6/VSE customers should contact IBM for support.

BSI Customers

Technical Support is available from Barnard Software, Inc. by phone, mail or email:

Barnard Software, Inc.
806 Silk Oak Terrace
Lake Mary, FL 32746
Phone: 1-407-323-4773
Support: bsiopti@bsiopti.com
Sales: bsisales@bsiopti.com

Support is available from 9:00 a.m. through 5:00 p.m. EST, Monday through Friday.
If a TSR (Technical Support Representative) is not available at the time of your call, please leave a message and a TSR will return your call as soon as possible. Please provide the following information: name, company, phone number, product name, product release level, and a short description of the problem.

BSIUsers Announcement List Server

When new releases of IPv6/VSE are available BSI will post an announcement on its BSIUsers announcement list.

To subscribe to the BSIUsers announcement list send an email to this email address
BSIUsers-subscribe@yahoogroups.com

To unsubscribe to the BSIUsers announcement list send an email to this email address
BSIUsers-unsubscribe@yahoogroups.com

Problem Determination

If you have a problem using a IPv6/VSE application always check the SYSLST output for additional information and messages. Most messages are written to SYSLST and not to the VSE/ESA or z/VSE system console.

When contacting BSI for technical support always have the applications JCL/commands, console and SYSLST output available for problem determination. The SYSLST output is very important.

While a IPv6/VSE application is running, you can issue the AR CANCEL XX,PARTDUMP command to terminate IPv6/VSE application and dump the partition to SYSLST. Using the VSE/POWER Flush (F) command cancels the IPv6/VSE application partition without a dump.

If the IPv6/VSE application partition stops responding to its console interface, use the AR DUMP XX command to obtain a dump of the partition.

Installation

System Requirements

IPv6/VSE's Secure Sockets Layer (SSL) support uses the IBM z/VSE OpenSSL port. The OpenSSL port became available with z/VSE 5.1 and requires these APARs.

```
PM77065 / UK83637  = IPv6/VSE (253pre11+)
DY47397 / UD53864  = OpenSSL (requires UD53863)
DY47414 / UD53863  = VSE/AF Crypto support
```

Pre-z/VSE 5.1 users can contact BSI for special installation instructions.

CPACF

The z/VSE 5.1 OpenSSL port will take advantage of the System z's CP Assist for Cryptographic Function (CPACF). This CP Assist dramatically reduces CPU overhead involved in use the cryptographic functions. Users of the z890, z990, z9, z10, z114, z196 (and newer processors) should have the no-charge enablement feature installed. This feature must be ordered to enable CPACF. z/VSE system without CPACF will use system CPU resources to perform cryptographic functions.

Crypto Express Adapters

The z/VSE 5.1 OpenSSL port will take advantage of the System z Crypto Express adapters supported by z/VSE. Usage of Crypto Express adapters dramatically reduces the amount of CPU required to establish a secure socket connection. z/VSE systems without Crypto Express adapters will use system CPU resources to establish a secure socket connection.

Restrictions

The current OpenSSL based implementation can be used only for batch LE (and LE/C) applications. Non-LE based batch applications, CICS/VSE and CICS TS applications are currently not supported. This restriction will be removed in a future release.

External Security Managers

If you use an External Security Manager (and *not* the Basic Security Manager) the following implementation details of the Hardware Crypto support are important and must be observed.

The Hardware Crypto support is activated by the startup job SECSERV (Security Server) which is part of the *Basic Security Manager* and which runs in partition FB by default. If SECSERV is not started (because you are using an External Security Manager), the Hardware Crypto support is **not** available. However, the Hardware Crypto task can be started manually in any partition with a job stream such as the following:

```
* $$ JOB JNM=HWCRYPTO,DISP=D,CLASS=R
// JOB HWCRYPTO
// EXEC IJBCRYPT
/*
/&
* $$ EOJ

14.33.49 G1 0050 // JOB HWCRYPTO
14.33.49         DATE 05/07/2013, CLOCK 14/33/49
14.33.49 G1 0050 1J022I CPU CRYPTOGRAPHIC ASSIST FEATURE AVAILABLE.
14.33.49 G1 0050 1J017I CRYPTO HARDWARE NOT INSTALLED OR NOT DEFINED.
14.33.49 G1 0050 EOJ HWCRYPTO
14.33.49         DATE 05/07/2013,CLOCK 14/33/49,DURATION 00/00/00
```

SETPARM Values

There are two SETPARM variables that control the IJBSSL execution. SSL$DBG controls debug output and SSL$ICA controls use of the hardware CPACF instructions and Crypto Express adapter. When SSL$ICA is set to NO, all cryptographic functionality is performed using the system z CPU. No hardware assists are used. The SSL$ICA *must* be set to NO when running on pre-z/VSE 5.1 systems.

```
// SETPARM SSL$DBG='NO|YES'   Debug NO or YES. Default is NO.
// SETPARM SSL$ICA='NO|YES'   Hardware NO or YES. Default is YES.
```

UPSI Values

There are two UPSI variables that control BSTTPRXY/BSTTATLS execution.

```
// UPSI 1x   Display perform section entry/exit
// UPSI x1   Display non-error results from EZA calls
```

Installation Verification

You can verify your z/VSE install has SSL support by running a LIBR LD for the PRD1.BASE lib.slib. If your z/VSE supports SSL you will find an IJBSSL.PHASE in PRD1.BASE.

Pre-z/VSE 5.1 users can contact BSI for special installation instructions.

```
// EXEC LIBR,SIZE=256K
 LD L=PRD1
/*

DIRECTORY DISPLAY    SUBLIBRARY=PRD1.BASE           DATE: 2011-12-14
                                                    TIME: 18:16
-------------------------------------------------------------------
 M E M B E R      CREATION   LAST     BYTES    LIBR CONT SVA  A- R-
NAME     TYPE     DATE      UPDATE   RECORDS   BLKS STOR ELIG MODE
IJBSSL   PHASE    11-11-11 11-12-12 3999152 B   4048  NO  NO  31 ANY
```

User's Guide

Creating and Using SSL Certificates

This manual section describes how to establish yourself as a root certificate authority (root CA) using the OpenSSL toolset. As a root CA, you are able to sign and install certificates for use in your Internet server applications, such as CICS TS Web Services, BSTTVNET TN3270E Server, etc.

The information in this manual section was taken from Marcus Redivo's web page ...
http://www.eclectica.ca/howto/ssl-cert-howto.php
and it is also available here ...
http://www.debian-administration.org/articles/284

Researched and written by Marcus Redivo.
Permission to use this document for any purpose is hereby granted, providing that the copyright information and this disclaimer is retained. Author accepts no responsibility for any consequences arising from the use of this information.

Last modified on Wed May 4 08:52:05 2011

Warning
Please read the web page. The web page provides a detailed description of the certificate generation process. What follows in this manual is just a summary of the process. The web page listed above is also provided in the BSI IPv6/VSE download as a pdf file (IP-Creating and Using SSL Certificates.pdf).

Background

Why be our own root CA? So that we can take advantage of SSL encryption without spending unnecessary money on having our certificates signed.

A drawback is that browsers will still complain about our site not being trusted until our root certificate is imported. However, once this is done, we are no different from the commercial root CAs.

Clients will only import our root certificate if they trust us. This is where the commercial CAs come in: they purport to do extensive research into the people and organizations for whom they sign certificates. By importing (actually, by the browser vendors incorporating) their trusted root certificates, we are saying that we trust them when they guarantee that someone else is who they say they are. We can trust additional root CAs (like ourselves) by importing their CA certificates.

Note: *If you are in the business of running a commercial secure site, obtaining a commercially signed certificate is the only realistic choice.*

One-Time Setup

Set up, and create a root CA certificate.

Commands

```
# mkdir CA
# cd CA
# mkdir newcerts private
# echo '01' >serial
# touch index.txt
# (IMPORTANT: Install and edit the configuration file shown below.)
# openssl req -new -x509 -extensions v3_ca -keyout private/cakey.pem \
-out cacert.pem -days 365 -config ./openssl.cnf
```

Output

File	Purpose
cacert.pem	CA certificate
private/cakey.pem	CA private key

Distribute **cacert.pem** to your clients.

Per Certificate

Create certificate signing requests and sign them, supplying appropriate values for the Common Name and the Organizational Unit.

Commands

```
# openssl req -new -nodes -out req.pem -config ./openssl.cnf
# openssl ca -out cert.pem -config ./openssl.cnf -infiles req.pem
# cat key.pem cert.pem >key-cert.pem
```

Output

File	Purpose
key.pem	Private key
req.pem	Certificate signing request
cert.pem	Certificate
key-cert.pem	Combined private key and certificate

Install **key.pem** and **cert.pem**, or just **key-cert.pem** as appropriate for your server application.

Per Certificate - Renewal

Revoke the expired certificate, and re-sign the original request.

Commands

```
# openssl ca -revoke newcerts/<serial>.pem -config ./openssl.cnf
# openssl ca -out cert.pem -config ./openssl.cnf -infiles req.pem
```

Install the renewed certificates in the same manner as the original ones.

Configuration File

(This file is available for download) ftp://ftp.binarytool.com/pub/linux/ssl/openssl.cnf

```
---Begin---
#
# OpenSSL configuration file.
#

# Establish working directory.

dir                     = .

[ ca ]
default_ca              = CA_default

[ CA_default ]
serial                  = $dir/serial
database                = $dir/index.txt
new_certs_dir           = $dir/newcerts
certificate             = $dir/cacert.pem
private_key             = $dir/private/cakey.pem
default_days            = 365
default_md              = md5
preserve                = no
email_in_dn             = no
nameopt                 = default_ca
certopt                 = default_ca
policy                  = policy_match

[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ req ]
default_bits            = 1024                  # Size of keys
default_keyfile         = key.pem               # name of generated keys
```

```
default_md              = md5                   # message digest algorithm
string_mask             = nombstr               # permitted characters
distinguished_name      = req_distinguished_name
req_extensions          = v3_req

[ req_distinguished_name ]
# Variable name           Prompt string
#----------------------   ----------------------------------
0.organizationName      = Organization Name (company)
organizationalUnitName  = Organizational Unit Name (department, division)
emailAddress            = Email Address
emailAddress_max        = 40
localityName            = Locality Name (city, district)
stateOrProvinceName     = State or Province Name (full name)
countryName             = Country Name (2 letter code)
countryName_min         = 2
countryName_max         = 2
commonName              = Common Name (hostname, IP, or your name)
commonName_max          = 64

# Default values for the above, for consistency and less typing.
# Variable name                   Value
#------------------------------   ------------------------------
0.organizationName_default      = The Sample Company
localityName_default            = Metropolis
stateOrProvinceName_default     = New York
countryName_default             = US

[ v3_ca ]
basicConstraints        = CA:TRUE
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid:always,issuer:always

[ v3_req ]
basicConstraints        = CA:FALSE
subjectKeyIdentifier    = hash

----End----
```

References

More information is available at the following sites (opens in new window):

- OpenSSL Home Page
- OpenSSL Documentation
- OpenSSL FAQ
- Nick Burch's Certificate Management and Installation with OpenSSL
- Franck Martin's SSL Certificates HOWTO

PKCS12 / PEM Formats

Converting OpenSSL PEM Format Certificate To PKCS12 Format Certificate

Most browsers, including Internet Explorer, require that client certificates (which includes proxy certificates) be in the PKCS12 format rather than the X509 PEM format. Additionally, Java KeyStores require certificates to be in PKCS12 format. To convert a PEM formatted certificate to PKCS12 format, you need both the certificate and the private key for that certificate. Here's a typical openssl command and the resulting interactive session when converting PEM format to PKCS12 format:

openssl pkcs12 -export -in cert.pem -inkey key.pem -out cred.p12

Enter Export Password:
Verifying - Enter Export Password:

Command line options:
-export - generate a PKCS12 formatted file.
-in cert.pem - read in the X509 PEM formatted certificate from the file cert.pem
-inkey key.pem - read in the X509 PEM formatted key from the file key.pem
-out cred.p12 - write out the PKCS12 formatted 'credential' to the file cred.p12

PEM formatted certificates are fairly flexible. For example, both the certificate and the private key for that certificate can be contained in a single file. This is often the case with proxy certificates, which contain the proxy certificate, the proxy private key, and the user certificate (which was used to sign the proxy certificate). If you have a single file containing both the certificate and the key, you can specify the same filename for both the -in and -inkey command line options. OpenSSL will use the first certificate and first private key it finds in the file.

If the private key is encrypted, you will be prompted to enter the pass phrase for that key before entering the export password.

The export password does not have to be the same as the password you used for the PEM formatted private key. Whatever password you choose, you will need to enter that new password when importing the new PKCS12 credential into Windows.

Key Store Considerations

Before any SSL application can be run, a key store must be created, which contains the RSA public/private key pair and the SSL certificates.

The PEM (Privacy-enhanced mail) format is used by OpenSSL. PEM files may contain an RSA key pair, an SSL certificate or both. The PEM data may or may not be password protected. PEM file content is base-64 encoded.

In z/VSE the PEM data may be stored in a z/VSE VSAM ESDS file or a z/VSE library member. Using a z/VSE library member is recommended.

Creating a Key Store

OpenSSL allows specifying both the RSA key and certificate in one .pem file or using separate .pem files for RSA key and certificate. In a test environment it may be desirable to use self-signed certificates, in a production environment you should use certificates issued by a certificate authority (CA).

Self-Signed

The following OpenSSL command can be used to create a self-signed RSA key/certificate file.

```
openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out
mycert.pem
```

The -nodes parameter stands for ”no DES encryption and causes the key store to be created without a password.

CA-Signed

The following OpenSSL command can be used to create a certificate request from a previously created key. The request can then be signed by a CA, like Verisign or Thawte. For testing you can also use Keyman/VSE.

```
openssl req -new -key mycert.pem -out myreq.pem
```

Transferring a Key Store to z/VSE

The z/VSE OpenSSL port expects the PEM data to be EBCDIC and stored in either a VSAM ESDS file or library member. If a library member is used it is a member with fixed format.

Since PEM files created by OpenSSL are ASCII with CRLF (Windows) or LF (Linux) used as line deliminators a simple FTP process can be used to transfer the PEM data to z/VSE

VSAM Transfer

```
jcb@dv9500t:~/vm/ijbssl> ftp vse42
Connected to vse42.
220 TCP/IP-TOOLS for VSE FTP Server Ready.
Name (vse42:jcb): jcb
331 User name OK, need password.
Password:
230 User JCB      logged in, proceed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> asc                ***(1)
200 Command OK.
ftp> quote site sbcs racoon   ***(2)
200 Command OK.
ftp> quote site output esds dlbl     ***(3)
250 Requested file action OK, completed.
ftp> put bsicert.pem    ***(4)
local: bsicert.pem remote: bsicert.pem
229 Entering Extended Passive Mode (|||04103|).
150 File status OK, about to open data connection.
100% |
**************************************************************************
**************************************************|  2028
9.12 MB/s    --:-- ETA
250 Requested file action OK, completed.
2028 bytes sent in 00:00 (21.05 KB/s)
ftp> quit
221 TCP/IP-TOOLS for VSE terminating connection.
jcb@dv9500t:~/vm/ijbssl>
```

***(1) specifies an ASCII transfer.
***(2) specifies the Single Byte Character Set (SBCS) table names racoon is to be used.
***(3) specifies a Change Directory (CD) command that mounts the library and sets the sublib.
***(4) specifies the member.type to be transferred. The type must be PEM.

Library Transfer

```
jcb@dv9500t:~/vm/ijbssl> ftp vse42
Connected to vse42.
220 TCP/IP-TOOLS for VSE FTP Server Ready.
Name (vse42:jcb): jcb
331 User name OK, need password.
Password:
230 User JCB      logged in, proceed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> asc                ***(1)
200 Command OK.
ftp> quote site lmode f       ***(2)
200 Command OK.
ftp> quote site sbcs racoon   ***(3)
200 Command OK.
ftp> cd smnt-library-prd2/config     ***(4)
250 Requested file action OK, completed.
ftp> put bsicert.pem    ***(5)
local: bsicert.pem remote: bsicert.pem
229 Entering Extended Passive Mode (|||04103|).
150 File status OK, about to open data connection.
100% |
**************************************************************************
**************************************************|  2028
9.12 MB/s    --:-- ETA
250 Requested file action OK, completed.
2028 bytes sent in 00:00 (21.05 KB/s)
ftp> quit
221 TCP/IP-TOOLS for VSE terminating connection.
jcb@dv9500t:~/vm/ijbssl>
```

***(1) specifies an ASCII transfer.
***(2) specifies the library member mode is F (fixed).
***(3) specifies the Single Byte Character Set (SBCS) table names racoon is to be used.
***(4) specifies a Change Directory (CD) command that mounts the library and sets the sublib.
***(5) specifies the member.type to be transferred. The type must be PEM.

PEM File Content

This is an example of what the contents of a PEM file looks like.

```
-----BEGIN PRIVATE KEY-----
MIICdgIBADANBgkqhkiG9w0BAQEFAASCAmAwggJcAgEAAoGBAObmalhanzS0fJbY
LH+smk24xV3JqdxaM/lGB6AkS+urivpakzNt9oDs/NeO2j8p6BG5hF6WrjMewHZm
J6saPvhszEIDUTKp2raHsHktK0heiB/d4iGtOlKo8eIzNnN34uPq/ko1HirDuRdK
tQPkouvsSQE8eJXQeURnL6WM24k1AgMBAAECgYEA1Wkv73vK3G7ZHJ6u+k5wy9c0
bEwA+E5S7VmBoM0LcsY5jM18cjN3e5AO3bkGFQ1JlmBObRJ97XfF0Wj3zxhSC47O
kY0WF1KznhoAcpPDFpduv3/O9d8Y4saKiNPgyhrrNWQjXP5R2ZH7Kr9OxRWx4X9s
DcTshYQFcw3AKhYWLn0CQQD9NlXdshMuRlohsUCFBpulR9RErdZUa/+cQ162OE91
Mm/cNNFpj18YAjEkfWjcHOH6nyBvFPJGHr/uMwbnZha7AkEA6XExwF6H1uxE17Um
DT38C4Kx8Lh9g01vcKdU4g7Y6h2/n2UyrpzMPOBgi5mQwuuzqaY0ZP1iv7cObC80
fj74zwJAXVuCLxnvznjr1E5I7oLwMfEYxzWct0WWWaPRMLf0O86bxYHHz/anZe+w
9LY0Sitv+fpfd/UuGkJOG2G+4uQV7wJAL/VpWzQQhD8WBZrZrm7FkTfQhrvgTVFj
S6mU7rxOJAroiz6nIuhjrT4Ll6J88w6pYnka2oST/P8Gn4MzLZySUQJARUwSbCnb
WXtLjDSVwjYI3dJrcag9lU30q9sPXBW7Nd8tw6WbeRjTasdXXMNIzOHvXkpP1VLw
KdMltG6aCZjs4w==
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIC7jCCAlegAwIBAgIJAPera+PTfMgzMA0GCSqGSIb3DQEBBQUAMIGPMQswCQYD
VQQGEwJVUzELMAkGA1UECAwCRkwxEjAQBgNVBAcMCUxha2UgTWFyeTEfMB0GA1UE
CgwWQmFybmFyZCBTb2Z0d2FyZSwgSW5jLjELMAkGA1UECwwCSVMxEDAOBgNVBAMM
B0Jhcm5hcmQxHzAdBgkqhkiG9w0BCQEWEGplZmZAYnNpb3B0aS5jb20wHhcNMTEx
MDMxMTgzMTQxWhcNMTIxMDMwMTgzMTQxWjCBjzELMAkGA1UEBhMCVVMxCzAJBgNV
BAgMAkZMMRIwEAYDVQQHDAlMYWtlIE1hcnkxHzAdBgNVBAoMFkJhcm5hcmQgU29m
dHdhcmUsIEluYy4xCzAJBgNVBAsMAklTMRAwDgYDVQQDDAdCYXJuYXJkMR8wHQYJ
KoZIhvcNAQkBFhBqZWZmQGJzaW9wdGkuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GN
ADCBiQKBgQDm5mpYWp80tHyW2Cx/rJpNuMVdyancWjP5RgegJEvrq4r6WpMzbfaA
7PzXjto/KegRuYRelq4zHsB2ZierGj74bMxCA1Eyqdq2h7B5LStIXogf3eIhrTpS
qPHiMzZzd+Lj6v5KNR4qw7kXSrUD5KLr7EkBPHiV0HlEZy+ljNuJNQIDAQABo1Aw
TjAdBgNVHQ4EFgQUECgQ4u5gjk6UE2j/RmdPB7JgerYwHwYDVR0jBBgwFoAUECgQ
4u5gjk6UE2j/RmdPB7JgerYwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQUFAAOB
gQDR+gDYhgC7fZAPuuYkHAuMf7azU0fEacQ657yFUk+ce2PxDYekr8yGKdYo3jbd
hS3LXANRwGEouK3MJmyToBUBJRGeYRzToOIPvp4iZTIBflVp9FbslQOBEWMbLfNp
Sk5M75zIr4G4ALEStKVAGAw+igIXoRaIDVqAg6zldiDI0g==
-----END CERTIFICATE-----
```

BSTTPRXY SSL Proxy Server

The BSTTPRXY SSL proxy server allows non-SSL server applications running in z/VSE to accept connections from SSL based client applications. Or, BSTTPRXY will allow non-SSL client applications to connect to SSL based server applications.

Proxy Types

Source	**Destination**
Accept clear text connection	Proxy to clear text connection
Accept SSL connection	Proxy to SSL connection
Accept clear text connection	Proxy to SSL connection
Accept SSL connection	Proxy to clear text connection
Accept IPv4 or IPv6 connection	Proxy to IPv4 or IPv6 connection

Proxy Examples

Accept a clear text connection from BSTTMTPC on port 25 and proxy the connection to an SMTP server listening on SSL port 465. This type of connection is commonly called ***smtps***.

Accept an SSL connection on port 443 and proxy the connection to CICS TS Web Services on clear text port 80. This type of connections commonly called ***https***.

Accept an SSL connection on port 992 and proxy the connection to the BSTTVNET TN3270E server on clear text port 23. This type of connection is commonly called ***telnets***.

Accept a clear text connection from BSTTFTPC on port 21 and proxy the connection to an FTP server listening on SSL port 990. This type of connection is commonly called ***ftps***.

Accept an SSL connection on port 990 and proxy the connection to the BSTTFTPS FTP server on clear text port 21. This type of connection is commonly called ***ftps***.

BSTTPRXY Commands

The following commands are available for using the BSTTPRXY application. BSTTPRXY commands can be continued on the next line if necessary. Continuation is indicated by a character in column 72.

Command	Description
ID nn	Stack ID
BUFFERSIZEK	Buffer size in K bytes The default is 16K. This is recommended.
KEYRING	KEYRING location Either VSAM -or- lib.slib
KEYFILE	Default RSA key and certificate location Either VSAM ESDS dlbl name -or library member name (member type is PEM)
SECTYPE	Security type Either SSL30 or TLSV1 TLSV1 is recommended
SESSION_TIMEOUT	Session Timeout value in seconds The default is 900.
HANDSHAKE_CLIENT	0, 3 The default is 3.
HANDSHAKE_SERVER	1, 2 The default is 1.
OPTION	FTP Indicates FTP PORT/PASV/EPRT/EPSV processing required
OPTION	SERVER|CLIENT Indicates server mode (default) or client mode
CA_AUTH	YES|NO NO is the default
CA_AUTH_TYPE	Authorization type value 0, 1, 2, 3. The default is 3. This parameter is only used when CA_AUTH is YES. 0 Validate client certificate using just the local database 1 Obtain CA certificates and certificate revocation lists not found in the local database from LDAP server 2 Obtain CA certificates and certificate revocation lists not found in the local database from LDAP server 3 Do not validate client certificate
PROXY	See next page ...

Command	Description
PROXY	PROXY command
Protocol	TCP
IP	V4 or V6
Port	Port number
Mode	TXT (clear) or SSL (encrypted)
KEYFILE	Member name or *
TO	
Protocol	TCP
IP	V4 or V6
Port	Port number
Mode	TXT (clear) or SSL (encrypted)
KEYFILE	Member name or *
IP Address	IPv4 or IPv6 IP address

Each OPTION specification must appear in separate OPTION commands.

Console Commands

Command	Description
MSG id,D=SHUTDOWN	Terminate the application Open sockets are closed SSL sockets are shutdown (abnormally closed)
MSG id,D=KILL	Terminate the application The application immediately goes to EOJ No sockets are closed

Handshake	Description
GSK_AS_CLIENT: (0)	Client receives server's public key and cert. Client authenticates server's public key and cert. Client checks its local CA-cert file to authenticate. KEYFILE = Dummy RSA key and CA-cert for authentication *(1)
GSK_AS_SERVER: (1)	Server sends its public key and cert to client. KEYFILE = Public RSA key and cert *(1)
GSK_AS_SERVER_WITH_CLIENT_AUTH: (2)	Server sends its public key and cert to client. Client authenticates server's public key and cert. Client checks its local CA-cert file to authenticate. KEYFILE = public RSA key and cert AND CA-cert for authentication Client sends its public key and cert to server. Server authenticates client's public key and cert. Server checks its local CA-cert file to authenticate. KEYFILE = public RSA key and cert AND CA-cert for authentication
GSK_AS_CLIENT_NO_AUTH: (3)	Client receives server's public key and cert. Client accepts server's pubic key and cert without authentication. KEYFILE not used! no RSA key or cert required.

* In the case of a server and client running on the same system they can use the same KEYFILE file (identical RSA key and cert - self-signed).

A typical web (HTTP) server is (1) and the browser is (0).

CICS TS web services with client authentication is (2).

SSL tunneling (SMTPS or TELNETS) is a (3) connecting to a (1).

BSTTPRXY JCL

The following shows sample JCL for using the BSTTPRXY proxy server.

```
// OPTION SYSPARM='66'
// SETPARM IPTRACE='NNNNNNN'
// SETPARM LRGBUF=YES
// LIBDEF *,SEARCH=(ssllib.slib,bsilib.slib)
// EXEC BSTTPRXY,SIZE=BSTTPRXY
ID 66
*
KEYRING PRD2.CONFIG
KEYFILE MYCERT
SECTYPE TLSV1
*
OPTION SERVER
*
PROXY TCP V4 1234 SSL * TO V4 23 TXT * LOCALHOST
/*
```

Partition Size

The BSTTPRXY server application requires a minimum 20M partition.

Partition Priority

The BSTTPRXY server is very CPU intensive due to SSL encryption and decryption processing. The BSTTPRXY partition priority should *always be lower* than the TCP/IP stack partition.

Operation

The BSTTPRXY application must be started *before* any of the applications it is servicing. And, the applications BSTTPRXY is servicing *should* be shutdown/terminated *before* terminating the BSTTPRXY application. If the BSTTPRXY application is terminated using the TERMINATE console command, all active sockets will be abnormally closed before BSTTPRXY goes to EOJ.

BSTTPRXY Examples

BSTTVNET TN3270E Server

x3270

x3270 is an IBM 3270 terminal emulator for the X Window System and Windows. It runs on most Unix-like operating systems -- e.g., Linux, Mac OS X, Solaris and Cygwin. It also runs natively on Windows.

X3270 is distributed as source code, and can be used for free. See http://x3270.bgp.nu

x3270 runs over a TELNET connection, emulating either an IBM 3279 (color) or 3278 (monochrome). It supports:

- The full TN3270E protocol
- SSL/TLS (via the OpenSSL library) for encrypted sessions
- APL2 characters
- Non-English character sets, including Hebrew and DBCS Chinese and Japanese
- IND$FILE file transfer
- NVT mode (emulating a color xterm)
- A pop-up keypad for 3270-specific keys
- A scrollbar
- Printer session integration
- Extensive debugging and scripting facilities

```
Command format:
X3270 [prefix:]...[LUname@]hostname[:port]

E.g.,
x3270 L:192.168.1.238:1234
```

Prepending an L: onto hostname causes x3270 to first create an SSL tunnel to the host, and then create a TN3270 session inside the tunnel.

```
// EXEC BSTTPRXY,SIZE=BSTTPRXY
ID 66
KEYRING PRD2.CONFIG
KEYFILE BSICERT
SECTYPE TLSV1
OPTION SERVER
PROXY TCP V4 992 SSL * TO V4 23 TXT * LOCALHOST
/*
```

Options and Resources

SSL is controlled by a number of different command-line options and resources.

Specifying the Server's Root Certificate

If your host's certificate was not signed by a well-known certificate authority (CA), you can configure x3270 to accept it by specifying an alternate set of root certificate(s).

The **-cafile** option (or the **caFile** resource) can be used to specify a file containing one or more server root certificates. This file must be in PEM format.

If there are a large number of certificates, the **-cadir** option (or the **caDir** resource) can be used to specify a directory containing root certificates. This directory contains files that use the naming convention of *hash.seq*, where *hash* is the hash of the certificate value and *seq* is a sequence number (since multiple certificates could have the same hash) starting with 0.

Updating the Root Certificate Database

To avoid having to specify a root certificate for each invocation of x3270, one or more certificates can be added to the OpenSSL root certificate database. On Linux and Unix, this procedure is (unfortunately) an operating-system- and release- specific procedure. Howevever, the OpenSSL root certificate database is common to all OpenSSL applications on your workstation, so instructions on updating it should be easily found on the web.

On Windows, the root certificate database used by wc3270, ws3270 and wpr3287 is installed as part of the setup procedure and updating it is documented below.

The root certificate database is a PEM-format text file called **root-certs.txt**. The file is located in the wc3270 Application Data directory. (The Application Data directory varies between Windows releases, but it is easily found by selecting **wc3270 Explore AppData** in the **wc3270** program group.) You can add your root certificate (which must be in PEM format) to this file with a text editor such as Notepad.

Specifying a Certificate for the Client (the Emulator)

If you have been issued a certificate and a private key to allow your client (the emulator) to be authenticated by your host, there are several options that allow this to be specified to x3270.

The **-certfile** option (or the **certFile** resource) defines a file containing the client certificate. By default, this file is in PEM format, but it can also be in ASN1 format, which is specified by setting the **-certfiletype** option (or the **certFileType** resource) to the value **asn1**.

The **-chainfile** option (or the **chainFile** resource) defines a PEM-format file containing both the client certificate and any intermediate certificates that were used to sign it. If a chain file is specified, it is used instead of the certificate file.

Specifying the Private Key for the Client Certificate

The file containing the private key for the client certificate can be specified by the **-keyfile** option or the **keyFile** resource. This file is in PEM format by default, but can be in ASN1 format, by specifying the **-keyfiletype** option or the **keyFileType** resource with the value **asn1**.

If no explicit key file is specified, the default is to find the private key in the chain file or the client certificate file (whichever was specified).

If the private key is encrypted, then a password must be specified. The password is given with the **-keypasswd** option or the **keyPasswd** resource. The password can have one of two formats. The format **file:***filename* specifies a file containing the password. The format **string:***string*specifies the password as a string in the option or resource directly.

BSTTMTPC Mail Transport Client

The following sample JCL shows BSTTMTPC connecting to the BSTTPRXY server on port 25 (SMTP) and sending an email.

```
// EXEC BSTTMTPC,SIZE=BSTTMTPC
ID 00
OPEN 127.0.0.1 25
*
EHLO gmail.com
AUTH LOGIN jeffrey.webmail ????????
MAIL From: <jeffrey.webmail@gmail.com>
RCPT To: <jeff@bsiopti.com>
SUBJ Subject: Test Email
ORGA Organization: Barnard Software, Inc.
*
DATA
*
QUIT
/*
This is a test email text.
See http://www.bsiopti.com

Jeff
/*
```

The following sample BSTTPRXY JCL shows the proxy server accepting connections on port 25 (SMTP) and proxying the connection to smtp.gmail.com port 465. This configuration uses SMTPS or SMTP through an SSL tunnel. Note: The KEYFILE parameter must specify an RSA key/certificate member with a valid RSA key and certificate. However, the RSA key and certificate are not used.

```
// EXEC BSTTPRXY,SIZE=BSTTPRXY,PARM='TRAP(OFF)/'
ID 00
*
KEYRING PRD2.CONFIG
KEYFILE ZVSE51
SECTYPE TLSV1
*
OPTION CLIENT
* 74.125.157.108 IS SMTP.GMAIL.COM
PROXY TCP V4    25 TXT * TO V4   465 SSL * 74.125.157.108
/*
```

CICS TS Web Interface

The following sample BSTTPRXY JCL shows the proxy server accepting connections on port 443 (HTTPS) and proxying the connection to the CICS TS Web Server on port 8080.

```
// EXEC BSTTPRXY,SIZE=BSTTPRXY,PARM='TRAP(OFF)/'
ID 00
*
BUFFERSIZEK 16
*
KEYRING PRD2.CONFIG
KEYFILE CICSTS
SECTYPE TLSV1
*
OPTION SERVER
*
PROXY TCP V4   443 SSL * TO V4  8080 TXT * 127.0.0.1
/*
```

BSTTFTPC Batch FTP Client

The following sample JCL shows BSTTFTPC connecting to the BSTTPRXY server on port 2121 (FTP) and sending commands to a remote host FTP server on port 990(FTPS). BSTTPRXY supports Implicit-FTPS where the command connection is automatically encrypted using TLS.

```
// EXEC BSTTFTPC,SIZE=BSTTFTPC,OS390,TASKS=ANY
ID 00
OPEN 127.0.0.1 2121
PBSZ 0
PROT P
USER jcb
PASS bsi
PWD
STAT
QUIT
/*
```

The BSTTFTPC JCL/commands looks pretty typical but notice the PBSZ 0 and PROT P commands that immediately follow the OPEN command. These two commands, exactly as shown, must immediately follow the OPEN command. When using an Implicit-FTPS connection the command connection is automatically encrypted. The PBSZ 0 and PROT P commands tell the remote host FTP server that any data transfer done should also be encrypted. If these two commands are missing any data transfer will be done in clear text mode.

The following sample BSTTPRXY JCL shows the proxy server accepting connections on port 2121 (FTP) and proxying the connection to 192.168.1.60 port 990. This configuration uses FTPS or FTP through an implicit SSL tunnel. Note: The KEYFILE parameter must specify an RSA key/certificate member with a valid RSA key and certificate. Port mapping, for the FTP data connection, is handled automatically by BSTTPRXY.

```
// EXEC BSTTPRXY,SIZE=BSTTPRXY,PARM='TRAP(OFF)/'
ID 00
*
KEYRING PRD2.CONFIG
KEYFILE BSICERT
SECTYPE TLSV1
*
OPTION FTP
OPTION CLIENT
*
PROXY TCP V4  2121 TXT * TO V4   990 SSL * 192.168.1.60
/*
```

BSTTFTPS FTP Server

The following sample BSTTPRXY JCL shows the proxy server accepting connections on port 990 (Implicit-FTPS) and proxying the connection to 192.168.1.60 port 21 (FTP). This configuration uses FTPS or FTP through an implicit SSL tunnel. Note: The KEYFILE parameter must specify an RSA key/certificate member with a valid RSA key and certificate. Port mapping, for the FTP data connection, is handled automatically by BSTTPRXY.

```
// EXEC BSTTPRXY,SIZE=BSTTPRXY,PARM='TRAP(OFF)/'
ID 00
*
KEYRING PRD2.CONFIG
KEYFILE BSICERT
SECTYPE TLSV1
*
OPTION SERVER
*
PROXY TCP V4 990 SSL * TO V4 21 TXT * 127.0.0.1
/*
```

This example assumes you also are running the BSTTFTPS FTP server listening on port 21.

To test this function of BSTTPRXY we used the Linux LFTP FTP client. The LFTP client connected to BSTTPRXY using the 'lftp -d -u userid,password -p 990 ftps://192.168.1.238' command. The LFTP client connected to the BSTTPRXY server using an SSL connection on port 990. BSTTPRXY then proxied the connection to the BSTTFTPS FTP server on port 21.

ATLS Automatic Transport Layer Security

Automatic Transport Layer Security is a facility that is similar to the z/OS AT-TLS (Application Transparent - Transport Layer Security) facility. When a socket is established, the BSTTATLS application automatically determines if SSL/TLS is needed (based on your configuration) and converts the socket to SSL/TLS transparently to the application.

For example, the BSTTFTPS FTP server is running and listening for connections on port 21. If a clear text FTP client wishes to connect, they use port 21. The BSTTATLS application is also listening on port 990 (the FTP over SSL port) and when an SSL/TLS connection arrives on port 990 it is automatically converted to clear text and passed to the BSTTFTPS FTP server on port 21. This effectively this means that the BSTTFTPS FTP server is supporting both clear text and implicit-ftps connections.

At the same time, when BSTTMTPC batch Email client attempts to connect to SMTP.GMAIL.COM on port 25 (SMTP), the BSTTATLS application automatically converts the outbound socket to a connection to SMTP.GMAIL.COM on SSL/TLS port 465 (Implicit-SMTPS).

In both examples, no changes were made to the BSTTFTPS or BSTTMTPC applications or the JCL used to run these applications.

BSTTATLS works for both client and server applications and its operation is completely transparent to the application. The BSTTALTS application supports both batch and CICS applications written in any supported API including applications using the ASM SOCKET macro, EZASMI, EZASOKET and LE/C APIs.

Using BSTTATLS is very useful if you want to use SSL/TLS to securely transfer data but do not want to re-code or change your batch or CICS applications. Your batch or CICS socket applications continue to run unchanged even though they are now transferring data using SSL/TLS to remote hosts.

Design

The IPv6/VSE TCP/IP stacks run in separate partitions. The IPv4 stack (BSTTINET) runs in one partition and the IPv6 stack (BSTT6NET) runs in another. The TCP/IP stacks are then coupled together acting as a single stack. The BSTTATLS application is associated with a specific stack (BSTTINET or BSTT6NET). When running a typical dual stack configuration there will be two BSTTATLS application partitions. One for each TCP/IP stack.

The reasons for this design are performance, reliability and robustness. Since the BSTTATLS application is performing SSL/TLS functionality, the overhead of this functionality is offloaded from both the TCP/IP stack and the applications BSTTATLS is servicing.

Operation

The BSTTATLS application must be started *before* any of the applications it is servicing. And, the applications BSTTATLS is servicing *should* be shutdown/terminated *before* terminating the BSTTATLS application. If the BSTTATLS application is terminated using the TERMINATE console command, all active sockets will be abnormally closed before BSTTATLS goes to EOJ.

To allow jobs to wait for the BSTTATLS application to start up, use the BSTTWTLS application.

```
// LIBDEF PHASE,SEARCH=(bsilib.slib)
// OPTION SYSPARM='00'
// EXEC BSTTWTLS,SIZE=BSTTWTLS
/*
```

The BSTTWTLS application will attempt to verify the BSTTATLS application is available every 30 seconds. BSTTWTLS will go to EOJ when the BSTTATLS application is detected.

The BSTTWTLS application can be terminated early by issuing a MSG BSTTWTLS command on the console. No data needs to be entered with the MSG command.

ATLS Types

Source	Destination
Server	Accept SSL connection Convert to clear text
Client	Accept clear text connection Convert to SSL/TLS connection
IPv4 socket	To IPv4 socket
IPv6 socket	To IPv6 socket

ATLS Examples

Accept a clear text connection from BSTTMTPC and convert the connection to SSL/TLS on port 465. This type of connection is commonly called ***smtps***.

Accept an SSL connection on port 443 and convert the connection to CICS TS Web Services on clear text port 80. This type of connections commonly called ***https***.

Accept an SSL connection on port 992 and convert the connection to clear text on port 23. This type of connection is commonly called ***telnets***.

Accept a clear text connection from BSTTFTPC and convert the connection to SSL/TLS on port 990. This type of connection is commonly called ***ftps***.

Accept an SSL connection on port 990 and convert the connection to clear text on port 21. This type of connection is commonly called ***ftps***.

BSTTATLS Commands

The following commands are available for using the BSTTATLS application. BSTTATLS commands can be continued on the next line if necessary. Continuation is indicated by a character in column 72.

Command	Description
ID nn	Stack ID
BUFFERSIZEK	Buffer size in K bytes The default is 16K. This is recommended.
KEYRING	KEYRING location Either VSAM -or- lib.slib
KEYFILE	Default RSA key and certificate location Either VSAM ESDS dlbl name -or library member name (member type is PEM)
SECTYPE	Security type Either SSL30 or TLSV1 TLSV1 is recommended
SESSION_TIMEOUT	Session Timeout value in seconds The default is 900.
HANDSHAKE_CLIENT	0, 3 The default is 3.
HANDSHAKE_SERVER	1, 2 The default is 1.
OPTION	FTP Indicates FTP PORT/PASV/EPRT/EPSV processing required
OPTION	SERVER|CLIENT Indicates server mode (default) or client mode
CA_AUTH	YES|NO NO is the default
CA_AUTH_TYPE	Authorization type value 0, 1, 2, 3. The default is 3. This parameter is only used when CA_AUTH is YES. 0 Validate client certificate using just the local database 1 Obtain CA certificates and certificate revocation lists not found in the local database from LDAP server 2 Obtain CA certificates and certificate revocation lists not found in the local database from LDAP server 3 Do not validate client certificate
ATTLS	See next page ...

Command	Description
ATTLS	ATTLS command
Port	Port number used by the local z/VSE application
TO	Required literal
Destination Host	IP Address or DNS Name
Mask Length	Length of subnet mask. E.g., 8, 16, 24
AS	Required literal
Port	Replacement Port number
Mode	TXT (clear) or SSL (encrypted/default)
KEYFILE	Member name or *

Sample ATTLS Commands

Command	Description
OPTION CLIENT ATTLS 25 TO SMTP.GOOGLE.COM AS 465 SSL	Intercept outbound CLIENT connections made to SMTP.GMAIL.COM on port 25, convert them to SSL connections on port 465 (Implicit-SMTPS).
OPTION CLIENT OPTION FTP ATTLS 21 TO FTPS.BSIOPTI.COM AS 990 SSL	Intercept outbound FTP CLIENT connections made to FTPS.BSIOPTI.COM on port 21, convert them to SSL connections on port 990 (Implicit-FTPS).
OPTION SERVER ATTLS 23 AS 992 SSL	Intercept inbound SERVER connections made on SSL port 992 (Implicit-TELNETS), convert them to clear text connections on port 23.
OPTION SERVER ATTLS 80 AS 443 SSL	Intercept inbound SERVER connections made on SSL port 443 (Implicit-HTTPS), convert them to clear text connections on port 80.
OPTION SERVER OPTION FTP ATTLS 21 AS 990 SSL	Intercept inbound SERVER connections made on SSL port 990 (Implicit-FTPS), convert them to clear text connections on port 21.

Handshake	Description
GSK_AS_CLIENT: (0)	Client receives server's public key and cert. Client authenticates server's public key and cert. Client checks its local CA-cert file to authenticate. KEYFILE = Dummy RSA key and CA-cert for authentication *(1)
GSK_AS_SERVER: (1)	Server sends its public key and cert to client. KEYFILE = Public RSA key and cert *(1)
GSK_AS_SERVER_WITH_CLIENT_AUTH: (2)	Server sends its public key and cert to client. Client authenticates server's public key and cert. Client checks its local CA-cert file to authenticate. KEYFILE = public RSA key and cert AND CA-cert for authentication Client sends its public key and cert to server. Server authenticates client's public key and cert. Server checks its local CA-cert file to authenticate. KEYFILE = public RSA key and cert AND CA-cert for authentication
GSK_AS_CLIENT_NO_AUTH: (3)	Client receives server's public key and cert. Client accepts server's pubic key and cert without authentication. KEYFILE not used! no RSA key or cert required.

* In the case of a server and client running on the same system they can use the same KEYFILE file (identical RSA key and cert - self-signed).

A typical web (HTTP) server is (1) and the browser is (0).

CICS TS web services with client authentication is (2).

SSL tunneling (SMTPS or TELNETS) is a (3) connecting to a (1).

Console Commands

Command	Description
MSG id,D=SHUTDOWN	Terminate the application Open sockets are closed SSL sockets are shutdown (abnormally closed)
MSG id,D=KILL	Terminate the application The application immediately goes to EOJ No sockets are closed

BSTTATLS JCL

The following shows sample JCL for using the BSTTATLS proxy server.

```
// OPTION SYSPARM='66'
// SETPARM IPTRACE='NNNNNNN'
// SETPARM LRGBUF=YES
// LIBDEF *,SEARCH=(ssllib.slib,bsilib.slib)
// EXEC BSTTATLS,SIZE=BSTTATLS
ID 66
*
KEYRING PRD2.CONFIG
KEYFILE MYCERT
SECTYPE TLSV1
*
OPTION SERVER
ATTLS 23 AS 992 SSL
/*
```

Partition Size

The BSTTATLS server application requires a minimum 20M partition plus 72K for each possible socket to be handled by the BSTTATLS server.

For example, to support 600 TELNETS sessions, 4 FTPS sessions and 100 HTTPS sessions you will need a 20MB + 51MB = 71MB partition.

Partition Priority

The BSTTATLS server is very CPU intensive due to SSL encryption and decryption processing. The BSTTATLS partition priority should *always be lower* than the TCP/IP stack partition.

BSTTATLS Sample Configuration

```
// OPTION SYSPARM='00'
// SETPARM IPTRACE='NNNNNNN'
// SETPARM LRGBUF=YES
// LIBDEF *,SEARCH=(ssllib.slib,bsilib.slib)
// EXEC BSTTWAIT,SIZE=BSTTWAIT
/*
// EXEC BSTTATLS,SIZE=BSTTATLS
ID 00
*
KEYRING PRD2.CONFIG
KEYFILE MYCERT
SECTYPE TLSV1
*
* Convert outbound SMTP connections to Implicit-SMTPS
* but only connections to SMTP.GOOGLE.COM
OPTION CLIENT
ATTLS 25 TO SMTP.GOOGLE.COM AS 465 SSL
*
* Convert Implicit-TELNETS connections to TELNET
OPTION SERVER
ATTLS 23 AS 992 SSL
*
* Convert Implicit-HTTPS connections for CICS TS CWI
OPTION SERVER
ATTLS 80 AS 443 SSL
*
* Convert outbound FTP connections to Implicit-FTPS
OPTION CLIENT
OPTION FTP
ATTLS 21 TO FTPS.BSIOPTI.COM AS 990 SSL
*
* Convert inbound Implicit-FTPS connections to FTP
OPTION SERVER
OPTION FTP
ATTLS 21 AS 990 SSL
/*
```

Programming

How System SSL Works

Before you start writing your application, let's look at how System SSL works.

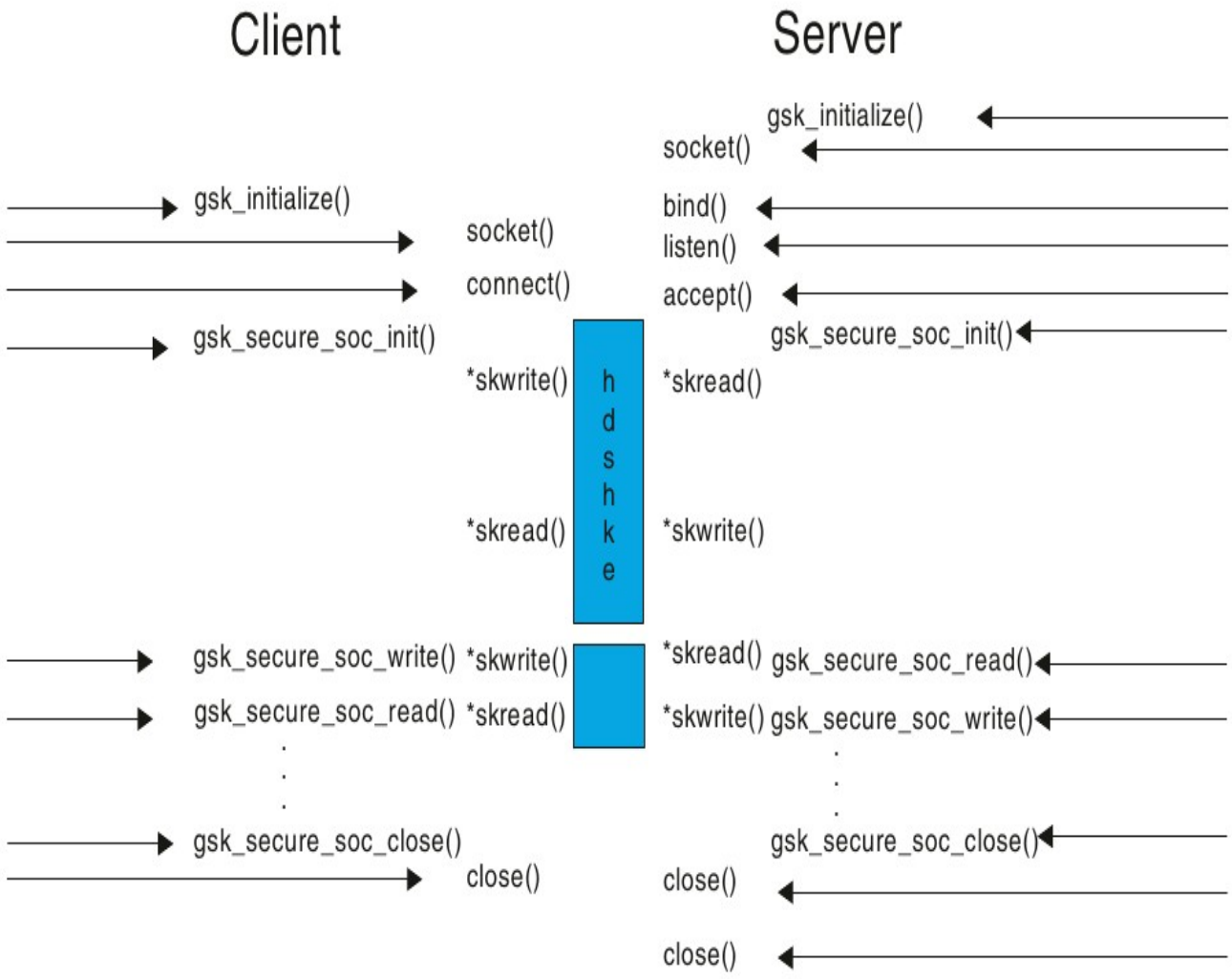
The SSL protocol begins with a “handshake.” During the handshake, the client authenticates the server, the server optionally authenticates the client, and the client and server agree on how to encrypt and decrypt information.

X.509 certificates are used by both the client and server when securing communications using System SSL. The client must verify the server's certificate based on the certificate of the Certificate Authority (CA) that signed the certificate or based on a self-signed certificate from the server. The server must verify the client's certificate (if requested) using the certificate of the CA that signed the client's certificate. The client and the server then use the session keys and begin encrypted communications.

Using SSL on z/VSE

The figure on the next page describes the basic structure of the elements needed in your System SSL source program. These elements are:

1. A socket() call to obtain a socket descriptor.
2. A gsk_initialize() call to set up the application's environment for secure communications. In most cases, this means extracting and verifying the certificate from the specified key database file to be used by the application (to eventually authenticate itself to the peer system in later processing).
3. Socket calls to activate a connection (a connect() call for a client program or bind(), listen(), and accept() calls for a server program).
4. A gsk_secure_soc_init() call to initiate the SSL handshake negotiation of the cryptographic parameters. gsk_secure_soc_init() initiates the handshaking process. The z/VSE application programmer does not have to write the "handshaking" function. This is a function of the SSL support. gsk_secure_soc_init() makes calls to the user supplied *skwrite() and *skread() routines.
5. A gsk_secure_soc_write() call to encrypt data and pass it to the user supplied *skwrite() routine for transmission.
6. A gsk_secure_soc_read() call to read data by calling the user supplied *skread() routine, to decrypt the data, and to return the decrypted data.
7. A gsk_secure_soc_close() call to disable System SSL support for the socket.
8. A close() call to destroy the connected sockets.


Client
Server
gsk_initialize()
socket()
bind()
listen()
accept()
gsk_secure_soc_init()
*skwrite()
*skread()
*skread()
*skwrite()
h
d
s
h
k
e
gsk_secure_soc_write()
*skwrite()
*skread()
gsk_secure_soc_read()
gsk_secure_soc_read()
*skread()
*skwrite()
gsk_secure_soc_write()
gsk_secure_soc_close()
close()
gsk_secure_soc_close()
close()
close()

Handshake	Description
GSK_AS_CLIENT: (0)	Client receives server's public key and cert. Client authenticates server's public key and cert. Client checks its local CA-cert file to authenticate. DNAME = Dummy RSA key and CA-cert for authentication *(1)
GSK_AS_SERVER: (1)	Server sends its public key and cert to client. DNAME = Public RSA key and cert *(1)
GSK_AS_SERVER_WITH_CLIENT_AUTH: (2)	Server sends its public key and cert to client. Client authenticates server's public key and cert. Client checks its local CA-cert file to authenticate. DNAME = public RSA key and cert AND CA-cert for authentication Client sends its public key and cert to server. Server authenticates client's public key and cert. Server checks its local CA-cert file to authenticate. DNAME = public RSA key and cert AND CA-cert for authentication
GSK_AS_CLIENT_NO_AUTH: (3)	Client receives server's public key and cert. Client accepts server's pubic key and cert without authentication. DNAME not used! no RSA key or cert required.

*(1) In the case of a server and client running on the same system they can use the same DNAME file (identical RSA key and cert - self-signed).

A typical web (HTTP) server is (1) and the browser is (0).

CICS TS web services with client authentication is (2).

SSL tunneling (SMTPS or TELNETS) is a (3) connecting to a (1).

IPv6/VSE provides access to secure sockets (SSL) through the LE/C, EZASMI and EZASOKET APIs. The various GSK function calls provide simple access to OpenSSL secure socket routines.

For information about calling OpenSSL routines directly please refer to the IBM OpenSSL programming manual. Directly calling OpenSSL routines is available only to C language applications.

For detailed information about the EZASMI GSK macro calls and HLL EZASOKET GSK calls, refer to the IBM z/OS System Secure Sockets Layer Programming SC24-5901-00 manual.

GSK Functions Provided

Function	**Description**
GSKINIT	GSK Initialize
GSKSSOCINIT	GSK Secure Socket Initialize
GSKSSOCREAD	GSK Secure Socket Read
GSKSSOCWRITE	GSK Secure Socket Write
GSKSSOCCLOSE	GSK Secure Socket Close
GSKSSOCRESET	GSK Secure Socket Reset
GSKUNINIT	GSK Uninitialize
GSKFREEMEM	GSK Free Memory
GSKGETCIPHINF	GSK Get Cypher Information
GSKGETDNBYLAB	GSK Get DName By Label

GSKINIT

```
        EZASMI TYPE=GSKINIT,
              SECTYPE=
              KEYRING=
              V3TIMEOUT=
              CAROOTS=
              AUTHTYPE=
              TASK=
              ERRNO=
              RETCODE=
```

The GSK Initialize function is issued once for a partition after the first INITAPI call has been made.

GSKINIT Parameters

The **SECTYPE** parameter specifies the security type. This field is a null (0x00) terminated string. Values of 'SSL30' or 'SSLV3' select the OpenSSL SSLv3_method(). Values of 'TLS31', 'TLSV1' or 'ALL' select the OpenSSL TLSv1_method(). 'TLSV1' or 'ALL' is recommended.

The **KEYRING** parameter specifies the location of the RSA key and certificate data. This data can be located in a VSAM ESDS file or a z/VSE library member. When a VSAM ESDS file is used the KEYRING parameter should refer to a null string (single byte of 0x00). Otherwise the KEYRING parameter specifies a null terminated string containing the lib.slib. The library member name or VSAM ESDS DLBL name is specified in the GSKSSOCINIT call's DNAME parameter. The library member type is always '.PEM'.

The **V3TIMEOUT** parameter specified a fullword containing the number of seconds that must elapse before the server's session data will time out.

The **CAROOTS** parameter specifies the address of a fullword containing the certificate authorization value. If GSK_CA_ROOTS_LOCAL_AND_X500 is specified then the AUTHTYPE parameter is used to determine the certificate validation level.

TYPE	**Description**
0	GSK_CA_ROOTS_LOCAL_ONLY
1	GSK_CA_ROOTS_LOCAL_AND_X500

The **AUTHTYPE** parameter specifies the address of a fullword containing the authorization type value. This parameter is ignored unless the CAROOTS parameter specifies GSK_CA_ROOTS_LOCAL_AND_X500.

TYPE	**Description**
0	GSK_CLIENT_AUTH_LOCAL Validate client certificate using just the local database
1	GSK_CLIENT_AUTH_STRONG_OVER_SSL Obtain CA certificates and certificate revocation lists not found in the local database from LDAP server
2	GSK_CLIENT_AUTH_STRONG Obtain CA certificates and certificate revocation lists not found in the local database from LDAP server
3	GSK_CLIENT_AUTH_PASSTHRU Do not validate client certificate

GSKSSOCINIT

```
EZASMI TYPE=GSKSSOCINIT,
      S=
      HANDSHAKE=
      DNAME=
      SECTYPE=
      V3CIPHER=
      SKREAD=
      SKWRITE=
      V3CIPHSEL=
      CERTINFO=
      REASCODE=
      TASK=
      ERRNO=
      RETCODE=
```

The GSKSSOCINIT is called immediately after accept() has completed (server mode) or immediately after a connect() has completed (client mode).

GSKSSOCINIT Parameters

The **S** (socket) parameter specifies the address of a halfword containing the socket number.

The **HANDSHAKE** parameter specifies the address of a fullword containing the type of connection.

Value	Connection Type	Description
0	Connect() to remote host	GSK_AS_CLIENT
1	Accept() from remote host	GSK_AS_SERVER
2	Accept() from remote host	GSK_AS_SERVER_WITH_CLIENT_AUTH
3	Connect() to remote host	GSK_AS_CLIENT_NO_AUTH

Values of 0 and 3 are used by SSL client applications. Values of 1 and 2 are used by SSL server applications. The value 3 (GSK_AS_CLIENT_NO_AUTH) causes the GSK API to perform a GSK_AS_SERVER handshake. The value of 2 (GSK_AS_SERVER_WITH_CLIENT_AUTH) causes the GSK API to perform a GSK_AS_CLIENT handshake.

The value of 3 (GSK_AS_CLIENT_NO_AUTH) is useful for creating an SSL tunnel.

The **DNAME** parameter specifies the location of the RSA key and certificate data. This parameter is the address of a NULL terminated string that contains either the VSAM ESDS DLBL name or the z/VSE library member name. The member type is always PEM. See the GSKINIT KEYRING parameter for more information.

The **SECTYPE** parameter is the same as the GSKINIT SECTYPE parameter. See the GSKINIT SECTYPE parameter comments for details.

The **SKREAD** parameter specifies the address of an EZASMI TYPE=RECV routine that uses standard linkage. If this parameter is omitted or the address of the SKREAD routine is NULL the API will provide a routine for this function. Using the provided routine is recommended.

The **SKWRITE** parameter specifies the address of an EZASMI TYPE=SEND routine that uses standard linkage. If this parameter is omitted or the address of the SKREAD routine is NULL the API will provide a routine for this function. Using the provided routine is recommended.

The **V3CIPHSEL** parameter is the address of a 2 byte field where the GSK_SSOCOPEN routines will return the cipher that was selected for use in the secure socket connection.

The **CERTINFO** parameter is the address of a fullword where the GSK_SSOCOPEN routines will return the address of the certificate information table.

The **REASCODE** parameter is the address of a fullword that contains the error reason code when an error occurs. The field has no meaning and is undefined unless the RETCODE field contains -1.

The **V3CIPHER** parameter specifies the cypher options that application wishes to use. This NULL terminated string specifies the type of cyphers the application wishes to use from strongest to weakest. E.g., '2F350A09'

Hex Value	**Cypher Type**
01	NULL-MD5
02	NULL-SHA
03	EXP-RC4-MD5
04	RC4-MD5
05	RC4-SHA
06	EXP-RC2-CBC-MD5
08	EXP-DES-CBC-SHA
09	DES-CBC-SHA
0A	DES-CBC3-SHA
11	EXP-EDH-DSS-DES-CBC-SHA
12	EDH-DSS-CBC-SHA
13	EDH-DSS-DES-CBC3-SHA
14	EXP-EDH-RSA-DES-CBC-SHA
15	EDH-RSA-DES-CBC-SHA

16	EDH-RSA-DES-CBC3-SHA
17	EXP-ADH-RC4-MD5
18	ADH-RC4-MD5
19	EXP-ADH-DES-CBC-SHA
1A	ADH-DES-CBC-SHA
1B	ADH-DES-CBC3-SHA
2F	AES128-SHA
32	DHE-DSS-AES128-SHA
33	DHE-RSA-AES128-SHA
34	ADH-AES128-SHA
35	AES256-SHA
38	DHE-DSS-AES256-SHA
39	DHE-RSA-AES256-SHA
3A	ADH-AES256-SHA
62	EXP1024-DES-CBC-SHA
63	EXP1024-DHE-DSS-DES-CBC-SHA
64	EXP1024-RC4-SHA
65	EXP1024-DHE-DSS-RC4-SHA
66	DHE-DSS-RC4-SHA

The **RETCODE** field is a fullword that contain the return code upon completion. If the value in this field is -1, an error occurred. If the value is greater than 0, the field contains the address of a control block referred to as the SSOCDATA. This value is used in other GSK secure socket requests.

GSKSSOCREAD

The GSKSSOCREAD request is used to read data from a secure socket connection.

```
         EZASMI TYPE=GSKSSOCREAD,
               SSOCDATA=
               BUF=
               NBYTE=
               ERRNO=
               RETCODE=
```

SSL_read() works based on the SSL/TLS records. Data is received in records (with a maximum record size of 16kB for SSLv3/TLSv1). Only when a record has been completely received, it can be processed (decryption and check of integrity). Therefore data that was not retrieved at the last call of SSL_read() can still be buffered inside the SSL layer and will be retrieved on the next call to SSL_read(). If num is higher than the number of bytes buffered, SSL_read() will return with the bytes buffered. If no more bytes are in the buffer, SSL_read() will trigger the processing of the next record. Only when the record has been received and processed completely, SSL_read() will return reporting success. At most the contents of the record will be returned. As the size of an SSL/TLS record may exceed the maximum packet size of the underlying transport (e.g. TCP), it may be necessary to read several packets from the transport layer before the record is complete and SSL_read() can succeed.

Therefore, 16K is the largest amount of data that can be read using a single GSKSSOCREAD request.

The gsk_secure_soc_read() has a special call. When a gsk_secure_soc_read() is issued with a buffer length of zero (NBYTE=0) the SSL_pending() call is issued. SSL_pending() returns the number of bytes which are available inside SSL for immediate read. Since SSL data are received in blocks from the peer, data can be buffered inside SSL and are ready for immediate retrieval with gsk_secure_soc_read().

GSKSSOCREAD Parameters

Comments about GSKSSOCREAD parameters.

The **SSOCDATA** parameter specifies the address of a fullword that contains the address of the control block returned after successful completion of the GSKSSOCOPEN request.

The **BUF** parameter specifies the address of the buffer.

The **NBYTE** parameter specifies the address of a fullword containing the length of the data.

The **RETCODE** parameter specifies the address of a fullword when the length the actual data read is placed.

RETCODE	Meaning
> 0	Amount of data returned
Equal to 0	Socket terminated normally
< 0	Error

GSKSSOCWRITE

The GSKSSOCWRITE request is used to write data to a secure socket connection.

```
         EZASMI TYPE=GSKSSOCWRITE,
               SSOCDATA=
               BUF=
               NBYTE=
               ERRNO=
               RETCODE=
```

SSL works based on SSL/TLS records. Data is received in records (with a maximum record size of 16kB for SSLv3/TLSv1). Therefore, while the length of a GSKSSOCWRITE request can be of any size, the GSKSSOCWRITE routine will only transfer a block of 16kB into the SSL_write() function in a single call. GSKSSOCWRITE will loop issuing SSL_write() requests until all data has been transferred.

GSKSSOCWRITE Parameters

Comments about GSKSSOCWRITE parameters.

The **SSOCDATA** parameter specifies the address of a fullword that contains the address of the control block returned after successful completion of the GSKSSOCOPEN request.

The **BUF** parameter specifies the address of the buffer.

The **NBYTE** parameter specifies the address of a fullword containing the length of the data.

The **RETCODE** parameter specifies the address of a fullword when the length the actual data read is placed.

RETCODE	Meaning
> 0	Amount of data returned
Equal to 0	Socket terminated normally
< 0	Error

The ERRNO parameter specifies the address of a fullword that contain the error number if and only if the RETCODE field contains a value less than zero (< 0).

GSKSSOCCLOSE

```
         EZASMI TYPE=GSKSSOCCLOSE,
               SSOCDATA=
               ERRNO=
               RETCODE=
```

The GSKSSOCCLOSE function is called immediately before a socket close() function is used. It is very important that a socket be closed immediately after GSKSSOCCLOSE is done.

The GSKSSOCCLOSE function invokes the SSL_shutdown() function to perform a full 2-step bidirectional shutdown of a secure socket connection.

When the application is the first party to send the "close notify" alert, SSL_shutdown() will only send the alert and then set the SSL_SENT_SHUTDOWN flag (so that the session is considered good and will be kept in cache). The GSKSSOCCLOSE routine will complete the bidirectional shutdown handshake by calling SSL_shutdown() again. The second call will make SSL_shutdown() wait for the peer's "close notify" shutdown alert.

GSKSSOCCLOSE Parameters

The **SSOCDATA** parameter specifies the address of a fullword that contains the address of the control block returned after successful completion of the GSKSSOCOPEN request.

GSKSSOCRESET

```
         EZASMI TYPE=GSKSSOCRESET,
                SSOCDATA=
                ERRNO=
                RETCODE=
```

The GSKSSOCRESET force OpenSSL to re-negotiate the SSL parameters for the connection. This function should not be used unless necessary because it can create a security exposure for the socket.

GSKSSOCRESET Parameters

The SSOCDATA parameter specifies the address of a fullword that contains the address of the control block returned after successful completion of the GSKSSOCOPEN request.

GSKUNINIT

```
         EZASMI TYPE=GSKUNINIT,
                ERRNO=
                RETCODE=
```

The GSKUNINIT function terminates the GSK SSL environment. This function should be called only once for a partition before an application goes to EOJ.

GSKUNINIT Parameters

GSKFREEMEM

```
EZASMI TYPE=GSKSSOCRESET,
      ADDRINFO=
      ERRNO=
      RETCODE=
```

The GSKFREEMEM function releases storage allocated by the GSK OpenSSL interface to return information.

GSKFREEMEM Parameters

The ADDRINFO= parameter specifies the address of the storage returned by the GSKGETDNBYLAB call.

GSKGETCIPHINF

```
         EZASMI TYPE=GSKSSOCRESET,
               CIPHLEVEL=
               SECLEVEL=
               ERRNO=
               RETCODE=
```

The GSKGETCIPHINF function determines the encryption level that the system can support and returns a list of cipher specifications that SSL can use. This allows an application to determine, at run time, the level of SSL encryption that the installed application can request.

GSKGETCIPHINF Parameters

The **CIPHLEVEL** parameter specifies the address of a fullword that contains the cipher level.

Level	**Description**
1	Low (Approved for export) "09151203060201"
2	High "05043538392F32330A161309151203060201"

The **SECLEVEL** parameter specifies the address of a control block (structure) where the security level information is returned.

```
VERSION    DS       F              SSL Version
V3CIPHERS  DS       XL64           SSLV3 Ciphers
V2CIPHERS  DS       XL32           SSLV2 Cuphers
SECLEVEL   DS       F              Security LEVEL
*                                  1 = USA
*                                  2 = USA approved for export
*                                  3 = France approved for export
```

GSKGETDNBYLAB

```
EZASMI TYPE=GSKGETDNBYLAB,
       KEYLABEL=
       ERRNO=
       RETCODE=
```

The GSKGETDNBYLAB function returns the name of the file that contains the RSA key and certificate.

GSKGETDNBYLAB Parameters

The KEYLABEL parameter specifies the address of a null terminated string containing the name of the key or certificate.

GSK Sample Applications

Basic Design

Non-SSL Simple Server

```
     EZASMI TYPE=INITAPI
     EZASMI TYPE=SOCKET
     EZASMI TYPE=BIND

ACPT EZASMI TYPE=ACCEPT

READ EZASMI TYPE=READ
     EZASMI TYPE=WRITE
     B      READ

EOD  EZASMI TYPE=CLOSE (Accept socket)
     B      ACPT

EOJ  EZASMI TYPE=CLOSE (Listen socket)
     EZASMI TYPE=TERMAPI
     EOJ
```

Non-SSL Client

```
     EZASMI TYPE=INITAPI
     EZASMI TYPE=SOCKET


     EZASMI TYPE=CONNECT

     EZASMI TYPE=WRITE
     EZASMI TYPE=READ




     EZASMI TYPE=CLOSE
     EZASMI TYPE=TERMAPI
     EOJ
```

SSL Simple Server

```
     EZASMI TYPE=INITAPI
     EZASMI TYPE=GSKINIT
     EZASMI TYPE=SOCKET
     EZASMI TYPE=BIND

ACPT EZASMI TYPE=ACCEPT
     EZASMI TYPE=GSKSSOCOPEN

READ EZASMI TYPE=GSKSSOCREAD
     EZASMI TYPE=GSKSSOCWRITE
     B      READ

EOD  EZASMI TYPE=GSKSSOCCLOSE
     EZASMI TYPE=CLOSE (Accept socket)
     B      ACPT

EOJ  EZASMI TYPE=CLOSE (Listen socket)
     EZASMI TYPE=GSKUNINIT
     EZASMI TYPE=TERMAPI
     EOJ
```

SSL Simple Client

```
     EZASMI TYPE=INITAPI
     EZASMI TYPE=GSKINIT
     EZASMI TYPE=SOCKET


     EZASMI TYPE=CONNECT
     EZASMI TYPE=GSKSSOCOPEN

     EZASMI TYPE=GSKSSOCWRITE
     EZASMI TYPE=GSKSSOCREAD


     EZASMI TYPE=GSKSSOCCLOSE



     EZASMI TYPE=CLOSE
     EZASMI TYPE=GSKUNINIT
     EZASMI TYPE=TERMAPI
     EOJ
```

The GSK Server Sample

The GSK Client Sample

GSK Error Codes

```
   /* General definitions */

#define GSK_VERSION2         2
#define GSK_VERSION3         3

#define GSK_VERSION          GSK_VERSION3

#define GSK_SEC_LEVEL_US          1
#define GSK_SEC_LEVEL_EXPORT      2
#define GSK_SEC_LEVEL_EXPORT_FR   3
#define GSK_LOW_SECURITY          1
#define GSK_HIGH_SECURITY         2

  /* Return codes. These are the return values */
  /* for gsk_initialize                       */
#define GSK_INITIALIZE_OK         0 /* Successful completion         */
#define GSK_KEYFILE_IO_ERROR      1 /* Error: keyring io error       */
#define GSK_KEYFILE_OPEN_FAILED   2 /* Error: keyring open error     */
#define GSK_KEYFILE_BAD_FORMAT    3 /* Error: keyring format problem */
#define GSK_KEYFILE_BAD_PASSWORD  4 /* Error: Keyring password bad   */
#define GSK_KEYFILE_BAD_MALLOC    5 /* Error: Malloc failed          */
#define GSK_KEYFILE_NOTHING_TO_WRITE   6
#define GSK_KEYFILE_WRITE_FAILED       7
#define GSK_KEYFILE_NOT_FOUND          8
#define GSK_KEYFILE_BAD_DNAME   9 /* Error: Invalid distinquish name */
#define GSK_KEYFILE_BAD_KEY    10 /* Error: Certificate has expired  */
#define GSK_KEYFILE_KEY_EXISTS        11
#define GSK_KEYFILE_BAD_LABEL         12
#define GSK_KEYFILE_DUPLICATE_NAME    13
#define GSK_KEYFILE_DUPLICATE_KEY     14
#define GSK_KEYFILE_DUPLICATE_LABEL   15
#define GSK_BAD_FORMAT_OR_INVALID_PW  16

#define GSK_WARNING_INVALID_SERVER_CERT      98
#define GSK_WARNING_INVALID_SERVER_PRIV_KEY  99

#define GSK_ERR_INIT_PARM_NOT_VALID   100 /* invalid cipher spec     */
 /* #define GSK_INIT_HARD_RT          101 */ /* no keyring file or pw*/
#define GSK_INIT_SEC_TYPE_NOT_VALID   102 /* invalid security type   */
#define GSK_INIT_V2_TIMEOUT_NOT_VALID 103 /* invalid V2 timeout value*/
#define GSK_INIT_V3_TIMEOUT_NOT_VALID 104 /* invalid V3 timeout value*/
#define GSK_KEYFILE_CERT_EXPIRED      105

  /* Return codes. These are the return values */
  /* for gsk_secure_soc_init                   */
#define GSK_SOC_BAD_V2_CIPHER                  -40
#define GSK_SOC_BAD_V3_CIPHER                  -41
#define GSK_SOC_BAD_SEC_TYPE                   -42
#define GSK_SOC_BAD_SEC_TYPE_COMBINATION       -102
#define GSK_SOC_NO_READ_FUNCTION               -43
#define GSK_SOC_NO_WRITE_FUNCTION              -44
```

```
#define GSK_ERROR_NO_CIPHERS                    -1
#define GSK_ERROR_NO_CERTIFICATE                -2
#define GSK_ERROR_BAD_CERTIFICATE               -4
#define GSK_ERROR_UNSUPPORTED_CERTIFICATE_TYPE  -6
#define GSK_ERROR_IO                            -10
#define GSK_ERROR_BAD_MESSAGE                   -11
#define GSK_ERROR_BAD_MAC                       -12
#define GSK_ERROR_UNSUPPORTED                   -13
#define GSK_ERROR_BAD_CERT_SIG                  -14
#define GSK_ERROR_BAD_CERT                      -15
#define GSK_ERROR_BAD_PEER                      -16
#define GSK_ERROR_PERMISSION_DENIED             -17
#define GSK_ERROR_SELF_SIGNED                   -18
#define GSK_ERROR_BAD_MALLOC                    -20
#define GSK_ERROR_BAD_STATE                     -21 /* V3 */
#define GSK_ERROR_SOCKET_CLOSED                 -22
#define GSK_ERROR_GSK_INITIALIZATION_FAILED     -23
#define GSK_ERROR_HANDLE_CREATION_FAILED        -24
#define GSK_ERROR_BAD_DATE                      -25
#define GSK_ERROR_BAD_KEY_LEN_FOR_EXPORT        -26
#define GSK_ERROR_NO_PRIVATE_KEY                -27
#define GSK_BAD_PARAMETER                       -28
#define GSK_ERROR_INTERNAL                      -29
#define GSK_ERROR_WOULD_BLOCK                   -30
#define GSK_ERROR_LOAD_GSKLIB                   -31
#define GSK_ERROR_NO_INITIALIZE                 -32
#define GSK_ERROR_SRB                           -33
#define GSK_ERROR_UNKNOWN_ERROR                 -99

  /* Misc functions */
#define GSK_FUNCTION_RESET_SID                  1

  /* return codes. These are the return values        */
  /* for gsk_secure_soc_write and gsk_secure_soc_read */
#define GSK_ERROR_BAD_BUFFER_SIZE               -100
#define GSK_ERROR_BAD_BUFFER                    -103

  /* return codes. These are the return values        */
  /* for gsk_secure_soc_write and gsk_secure_soc_read */
#define GSK_ERROR_BAD_SSL_HANDLE                -101
#define GSK_ERROR_TIMEOUT                       -102

  /* Return codes. */
#define GSK_ERROR_NOT_SERVER                    -50
#define GSK_ERROR_NOT_SSLV3                     -51
#define GSK_ERROR_NOT_SSLV3_CLIENT              -52
```

SSL Functions and Error Codes

```
/* Function codes. */
#define SSL_F_CLIENT_CERTIFICATE                100
#define SSL_F_CLIENT_FINISHED                   167
#define SSL_F_CLIENT_HELLO                      101
#define SSL_F_CLIENT_MASTER_KEY                         102
#define SSL_F_D2I_SSL_SESSION                   103
#define SSL_F_DO_DTLS1_WRITE                    245
#define SSL_F_DO_SSL3_WRITE                     104
#define SSL_F_DTLS1_ACCEPT                      246
#define SSL_F_DTLS1_ADD_CERT_TO_BUF             295
#define SSL_F_DTLS1_BUFFER_RECORD               247
#define SSL_F_DTLS1_CLIENT_HELLO                248
#define SSL_F_DTLS1_CONNECT                     249
#define SSL_F_DTLS1_ENC                         250
#define SSL_F_DTLS1_GET_HELLO_VERIFY                    251
#define SSL_F_DTLS1_GET_MESSAGE                         252
#define SSL_F_DTLS1_GET_MESSAGE_FRAGMENT        253
#define SSL_F_DTLS1_GET_RECORD                          254
#define SSL_F_DTLS1_OUTPUT_CERT_CHAIN                   255
#define SSL_F_DTLS1_PREPROCESS_FRAGMENT                 288
#define SSL_F_DTLS1_PROCESS_OUT_OF_SEQ_MESSAGE          256
#define SSL_F_DTLS1_PROCESS_RECORD              257
#define SSL_F_DTLS1_READ_BYTES                          258
#define SSL_F_DTLS1_READ_FAILED                         259
#define SSL_F_DTLS1_SEND_CERTIFICATE_REQUEST            260
#define SSL_F_DTLS1_SEND_CLIENT_CERTIFICATE             261
#define SSL_F_DTLS1_SEND_CLIENT_KEY_EXCHANGE            262
#define SSL_F_DTLS1_SEND_CLIENT_VERIFY                  263
#define SSL_F_DTLS1_SEND_HELLO_VERIFY_REQUEST           264
#define SSL_F_DTLS1_SEND_SERVER_CERTIFICATE             265
#define SSL_F_DTLS1_SEND_SERVER_HELLO                   266
#define SSL_F_DTLS1_SEND_SERVER_KEY_EXCHANGE            267
#define SSL_F_DTLS1_WRITE_APP_DATA_BYTES        268
#define SSL_F_GET_CLIENT_FINISHED               105
#define SSL_F_GET_CLIENT_HELLO                          106
#define SSL_F_GET_CLIENT_MASTER_KEY             107
#define SSL_F_GET_SERVER_FINISHED               108
#define SSL_F_GET_SERVER_HELLO                          109
#define SSL_F_GET_SERVER_VERIFY                         110
#define SSL_F_I2D_SSL_SESSION                   111
#define SSL_F_READ_N                            112
#define SSL_F_REQUEST_CERTIFICATE               113
#define SSL_F_SERVER_FINISH                     239
#define SSL_F_SERVER_HELLO                      114
#define SSL_F_SERVER_VERIFY                     240
#define SSL_F_SSL23_ACCEPT                      115
#define SSL_F_SSL23_CLIENT_HELLO                116
#define SSL_F_SSL23_CONNECT                     117
#define SSL_F_SSL23_GET_CLIENT_HELLO                    118
#define SSL_F_SSL23_GET_SERVER_HELLO                    119
```

```
#define SSL_F_SSL23_PEEK                        237
#define SSL_F_SSL23_READ                        120
#define SSL_F_SSL23_WRITE                       121
#define SSL_F_SSL2_ACCEPT                       122
#define SSL_F_SSL2_CONNECT                      123
#define SSL_F_SSL2_ENC_INIT                     124
#define SSL_F_SSL2_GENERATE_KEY_MATERIAL        241
#define SSL_F_SSL2_PEEK                         234
#define SSL_F_SSL2_READ                         125
#define SSL_F_SSL2_READ_INTERNAL                236
#define SSL_F_SSL2_SET_CERTIFICATE              126
#define SSL_F_SSL2_WRITE                        127
#define SSL_F_SSL3_ACCEPT                       128
#define SSL_F_SSL3_ADD_CERT_TO_BUF              296
#define SSL_F_SSL3_CALLBACK_CTRL                233
#define SSL_F_SSL3_CHANGE_CIPHER_STATE                  129
#define SSL_F_SSL3_CHECK_CERT_AND_ALGORITHM             130
#define SSL_F_SSL3_CLIENT_HELLO                         131
#define SSL_F_SSL3_CONNECT                      132
#define SSL_F_SSL3_CTRL                         213
#define SSL_F_SSL3_CTX_CTRL                     133
#define SSL_F_SSL3_DIGEST_CACHED_RECORDS        293
#define SSL_F_SSL3_DO_CHANGE_CIPHER_SPEC        292
#define SSL_F_SSL3_ENC                          134
#define SSL_F_SSL3_GENERATE_KEY_BLOCK                   238
#define SSL_F_SSL3_GET_CERTIFICATE_REQUEST              135
#define SSL_F_SSL3_GET_CERT_STATUS              289
#define SSL_F_SSL3_GET_CERT_VERIFY              136
#define SSL_F_SSL3_GET_CLIENT_CERTIFICATE       137
#define SSL_F_SSL3_GET_CLIENT_HELLO             138
#define SSL_F_SSL3_GET_CLIENT_KEY_EXCHANGE              139
#define SSL_F_SSL3_GET_FINISHED                         140
#define SSL_F_SSL3_GET_KEY_EXCHANGE             141
#define SSL_F_SSL3_GET_MESSAGE                          142
#define SSL_F_SSL3_GET_NEW_SESSION_TICKET       283
#define SSL_F_SSL3_GET_RECORD                   143
#define SSL_F_SSL3_GET_SERVER_CERTIFICATE       144
#define SSL_F_SSL3_GET_SERVER_DONE              145
#define SSL_F_SSL3_GET_SERVER_HELLO             146
#define SSL_F_SSL3_HANDSHAKE_MAC                285
#define SSL_F_SSL3_NEW_SESSION_TICKET                   287
#define SSL_F_SSL3_OUTPUT_CERT_CHAIN                    147
#define SSL_F_SSL3_PEEK                         235
#define SSL_F_SSL3_READ_BYTES                   148
#define SSL_F_SSL3_READ_N                       149
#define SSL_F_SSL3_SEND_CERTIFICATE_REQUEST             150
#define SSL_F_SSL3_SEND_CLIENT_CERTIFICATE              151
#define SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE             152
#define SSL_F_SSL3_SEND_CLIENT_VERIFY                   153
#define SSL_F_SSL3_SEND_SERVER_CERTIFICATE              154
#define SSL_F_SSL3_SEND_SERVER_HELLO                    242
#define SSL_F_SSL3_SEND_SERVER_KEY_EXCHANGE             155
#define SSL_F_SSL3_SETUP_KEY_BLOCK              157
#define SSL_F_SSL3_SETUP_READ_BUFFER                    156
```

```
#define SSL_F_SSL3_SETUP_WRITE_BUFFER                   291
#define SSL_F_SSL3_WRITE_BYTES                          158
#define SSL_F_SSL3_WRITE_PENDING                159
#define SSL_F_SSL_ADD_CLIENTHELLO_TLSEXT        277
#define SSL_F_SSL_ADD_DIR_CERT_SUBJECTS_TO_STACK        215
#define SSL_F_SSL_ADD_FILE_CERT_SUBJECTS_TO_STACK       216
#define SSL_F_SSL_ADD_SERVERHELLO_TLSEXT        278
#define SSL_F_SSL_BAD_METHOD                    160
#define SSL_F_SSL_BYTES_TO_CIPHER_LIST                  161
#define SSL_F_SSL_CERT_DUP                      221
#define SSL_F_SSL_CERT_INST                     222
#define SSL_F_SSL_CERT_INSTANTIATE              214
#define SSL_F_SSL_CERT_NEW                      162
#define SSL_F_SSL_CHECK_PRIVATE_KEY             163
#define SSL_F_SSL_CHECK_SERVERHELLO_TLSEXT              280
#define SSL_F_SSL_CHECK_SRVR_ECC_CERT_AND_ALG           279
#define SSL_F_SSL_CIPHER_PROCESS_RULESTR        230
#define SSL_F_SSL_CIPHER_STRENGTH_SORT                  231
#define SSL_F_SSL_CLEAR                         164
#define SSL_F_SSL_COMP_ADD_COMPRESSION_METHOD           165
#define SSL_F_SSL_CREATE_CIPHER_LIST                    166
#define SSL_F_SSL_CTRL                          232
#define SSL_F_SSL_CTX_CHECK_PRIVATE_KEY                 168
#define SSL_F_SSL_CTX_NEW                       169
#define SSL_F_SSL_CTX_SET_CIPHER_LIST                   269
#define SSL_F_SSL_CTX_SET_CLIENT_CERT_ENGINE            290
#define SSL_F_SSL_CTX_SET_PURPOSE               226
#define SSL_F_SSL_CTX_SET_SESSION_ID_CONTEXT            219
#define SSL_F_SSL_CTX_SET_SSL_VERSION                   170
#define SSL_F_SSL_CTX_SET_TRUST                         229
#define SSL_F_SSL_CTX_USE_CERTIFICATE                   171
#define SSL_F_SSL_CTX_USE_CERTIFICATE_ASN1              172
#define SSL_F_SSL_CTX_USE_CERTIFICATE_CHAIN_FILE        220
#define SSL_F_SSL_CTX_USE_CERTIFICATE_FILE              173
#define SSL_F_SSL_CTX_USE_PRIVATEKEY                    174
#define SSL_F_SSL_CTX_USE_PRIVATEKEY_ASN1       175
#define SSL_F_SSL_CTX_USE_PRIVATEKEY_FILE       176
#define SSL_F_SSL_CTX_USE_PSK_IDENTITY_HINT             272
#define SSL_F_SSL_CTX_USE_RSAPRIVATEKEY                 177
#define SSL_F_SSL_CTX_USE_RSAPRIVATEKEY_ASN1            178
#define SSL_F_SSL_CTX_USE_RSAPRIVATEKEY_FILE            179
#define SSL_F_SSL_DO_HANDSHAKE                          180
#define SSL_F_SSL_GET_NEW_SESSION               181
#define SSL_F_SSL_GET_PREV_SESSION              217
#define SSL_F_SSL_GET_SERVER_SEND_CERT                  182
#define SSL_F_SSL_GET_SIGN_PKEY                         183
#define SSL_F_SSL_INIT_WBIO_BUFFER              184
#define SSL_F_SSL_LOAD_CLIENT_CA_FILE                   185
#define SSL_F_SSL_NEW                           186
#define SSL_F_SSL_PEEK                          270
#define SSL_F_SSL_PREPARE_CLIENTHELLO_TLSEXT            281
#define SSL_F_SSL_PREPARE_SERVERHELLO_TLSEXT            282
#define SSL_F_SSL_READ                          223
#define SSL_F_SSL_RSA_PRIVATE_DECRYPT                   187
```

```
#define SSL_F_SSL_RSA_PUBLIC_ENCRYPT                    188
#define SSL_F_SSL_SESSION_NEW                   189
#define SSL_F_SSL_SESSION_PRINT_FP              190
#define SSL_F_SSL_SESS_CERT_NEW                         225
#define SSL_F_SSL_SET_CERT                      191
#define SSL_F_SSL_SET_CIPHER_LIST               271
#define SSL_F_SSL_SET_FD                        192
#define SSL_F_SSL_SET_PKEY                      193
#define SSL_F_SSL_SET_PURPOSE                   227
#define SSL_F_SSL_SET_RFD                       194
#define SSL_F_SSL_SET_SESSION                   195
#define SSL_F_SSL_SET_SESSION_ID_CONTEXT        218
#define SSL_F_SSL_SET_SESSION_TICKET_EXT        294
#define SSL_F_SSL_SET_TRUST                     228
#define SSL_F_SSL_SET_WFD                       196
#define SSL_F_SSL_SHUTDOWN                      224
#define SSL_F_SSL_UNDEFINED_CONST_FUNCTION              243
#define SSL_F_SSL_UNDEFINED_FUNCTION                    197
#define SSL_F_SSL_UNDEFINED_VOID_FUNCTION       244
#define SSL_F_SSL_USE_CERTIFICATE               198
#define SSL_F_SSL_USE_CERTIFICATE_ASN1                  199
#define SSL_F_SSL_USE_CERTIFICATE_FILE                  200
#define SSL_F_SSL_USE_PRIVATEKEY                201
#define SSL_F_SSL_USE_PRIVATEKEY_ASN1                   202
#define SSL_F_SSL_USE_PRIVATEKEY_FILE                   203
#define SSL_F_SSL_USE_PSK_IDENTITY_HINT                 273
#define SSL_F_SSL_USE_RSAPRIVATEKEY             204
#define SSL_F_SSL_USE_RSAPRIVATEKEY_ASN1        205
#define SSL_F_SSL_USE_RSAPRIVATEKEY_FILE        206
#define SSL_F_SSL_VERIFY_CERT_CHAIN             207
#define SSL_F_SSL_WRITE                         208
#define SSL_F_TLS1_CERT_VERIFY_MAC              286
#define SSL_F_TLS1_CHANGE_CIPHER_STATE                  209
#define SSL_F_TLS1_CHECK_SERVERHELLO_TLSEXT             274
#define SSL_F_TLS1_ENC                          210
#define SSL_F_TLS1_PREPARE_CLIENTHELLO_TLSEXT           275
#define SSL_F_TLS1_PREPARE_SERVERHELLO_TLSEXT           276
#define SSL_F_TLS1_PRF                          284
#define SSL_F_TLS1_SETUP_KEY_BLOCK              211
#define SSL_F_WRITE_PENDING                     212

/* Reason codes. */
#define SSL_R_APP_DATA_IN_HANDSHAKE             100
#define SSL_R_ATTEMPT_TO_REUSE_SESSION_IN_DIFFERENT_CONTEXT 272
#define SSL_R_BAD_ALERT_RECORD                          101
#define SSL_R_BAD_AUTHENTICATION_TYPE                   102
#define SSL_R_BAD_CHANGE_CIPHER_SPEC                    103
#define SSL_R_BAD_CHECKSUM                      104
#define SSL_R_BAD_DATA_RETURNED_BY_CALLBACK             106
#define SSL_R_BAD_DECOMPRESSION                         107
#define SSL_R_BAD_DH_G_LENGTH                   108
#define SSL_R_BAD_DH_PUB_KEY_LENGTH             109
#define SSL_R_BAD_DH_P_LENGTH                   110
#define SSL_R_BAD_DIGEST_LENGTH                         111
```

```
#define SSL_R_BAD_DSA_SIGNATURE                          112
#define SSL_R_BAD_ECC_CERT                       304
#define SSL_R_BAD_ECDSA_SIGNATURE                305
#define SSL_R_BAD_ECPOINT                        306
#define SSL_R_BAD_HANDSHAKE_LENGTH               332
#define SSL_R_BAD_HELLO_REQUEST                          105
#define SSL_R_BAD_LENGTH                         271
#define SSL_R_BAD_MAC_DECODE                     113
#define SSL_R_BAD_MAC_LENGTH                     333
#define SSL_R_BAD_MESSAGE_TYPE                           114
#define SSL_R_BAD_PACKET_LENGTH                          115
#define SSL_R_BAD_PROTOCOL_VERSION_NUMBER        116
#define SSL_R_BAD_PSK_IDENTITY_HINT_LENGTH               316
#define SSL_R_BAD_RESPONSE_ARGUMENT              117
#define SSL_R_BAD_RSA_DECRYPT                    118
#define SSL_R_BAD_RSA_ENCRYPT                    119
#define SSL_R_BAD_RSA_E_LENGTH                           120
#define SSL_R_BAD_RSA_MODULUS_LENGTH                     121
#define SSL_R_BAD_RSA_SIGNATURE                          122
#define SSL_R_BAD_SIGNATURE                      123
#define SSL_R_BAD_SSL_FILETYPE                           124
#define SSL_R_BAD_SSL_SESSION_ID_LENGTH                  125
#define SSL_R_BAD_STATE                          126
#define SSL_R_BAD_WRITE_RETRY                    127
#define SSL_R_BIO_NOT_SET                        128
#define SSL_R_BLOCK_CIPHER_PAD_IS_WRONG                  129
#define SSL_R_BN_LIB                             130
#define SSL_R_CA_DN_LENGTH_MISMATCH              131
#define SSL_R_CA_DN_TOO_LONG                     132
#define SSL_R_CCS_RECEIVED_EARLY                 133
#define SSL_R_CERTIFICATE_VERIFY_FAILED                  134
#define SSL_R_CERT_LENGTH_MISMATCH               135
#define SSL_R_CHALLENGE_IS_DIFFERENT                     136
#define SSL_R_CIPHER_CODE_WRONG_LENGTH                   137
#define SSL_R_CIPHER_OR_HASH_UNAVAILABLE         138
#define SSL_R_CIPHER_TABLE_SRC_ERROR                     139
#define SSL_R_CLIENTHELLO_TLSEXT                 226
#define SSL_R_COMPRESSED_LENGTH_TOO_LONG         140
#define SSL_R_COMPRESSION_FAILURE                141
#define SSL_R_COMPRESSION_ID_NOT_WITHIN_PRIVATE_RANGE 307
#define SSL_R_COMPRESSION_LIBRARY_ERROR                  142
#define SSL_R_CONNECTION_ID_IS_DIFFERENT         143
#define SSL_R_CONNECTION_TYPE_NOT_SET                    144
#define SSL_R_COOKIE_MISMATCH                    308
#define SSL_R_DATA_BETWEEN_CCS_AND_FINISHED              145
#define SSL_R_DATA_LENGTH_TOO_LONG               146
#define SSL_R_DECRYPTION_FAILED                          147
#define SSL_R_DECRYPTION_FAILED_OR_BAD_RECORD_MAC        281
#define SSL_R_DH_PUBLIC_VALUE_LENGTH_IS_WRONG            148
#define SSL_R_DIGEST_CHECK_FAILED                149
#define SSL_R_DUPLICATE_COMPRESSION_ID                   309
#define SSL_R_ECC_CERT_NOT_FOR_KEY_AGREEMENT             317
#define SSL_R_ECC_CERT_NOT_FOR_SIGNING                   318
#define SSL_R_ECC_CERT_SHOULD_HAVE_RSA_SIGNATURE         322
```

```
#define SSL_R_ECC_CERT_SHOULD_HAVE_SHA1_SIGNATURE     323
#define SSL_R_ECGROUP_TOO_LARGE_FOR_CIPHER            310
#define SSL_R_ENCRYPTED_LENGTH_TOO_LONG               150
#define SSL_R_ERROR_GENERATING_TMP_RSA_KEY            282
#define SSL_R_ERROR_IN_RECEIVED_CIPHER_LIST           151
#define SSL_R_EXCESSIVE_MESSAGE_SIZE                  152
#define SSL_R_EXTRA_DATA_IN_MESSAGE             153
#define SSL_R_GOT_A_FIN_BEFORE_A_CCS                  154
#define SSL_R_HTTPS_PROXY_REQUEST               155
#define SSL_R_HTTP_REQUEST                      156
#define SSL_R_ILLEGAL_PADDING                   283
#define SSL_R_INVALID_CHALLENGE_LENGTH                158
#define SSL_R_INVALID_COMMAND                   280
#define SSL_R_INVALID_PURPOSE                   278
#define SSL_R_INVALID_STATUS_RESPONSE                 328
#define SSL_R_INVALID_TICKET_KEYS_LENGTH        325
#define SSL_R_INVALID_TRUST                     279
#define SSL_R_KEY_ARG_TOO_LONG                        284
#define SSL_R_KRB5                              285
#define SSL_R_KRB5_C_CC_PRINC                   286
#define SSL_R_KRB5_C_GET_CRED                   287
#define SSL_R_KRB5_C_INIT                       288
#define SSL_R_KRB5_C_MK_REQ                     289
#define SSL_R_KRB5_S_BAD_TICKET                       290
#define SSL_R_KRB5_S_INIT                       291
#define SSL_R_KRB5_S_RD_REQ                     292
#define SSL_R_KRB5_S_TKT_EXPIRED                293
#define SSL_R_KRB5_S_TKT_NYV                    294
#define SSL_R_KRB5_S_TKT_SKEW                   295
#define SSL_R_LENGTH_MISMATCH                   159
#define SSL_R_LENGTH_TOO_SHORT                        160
#define SSL_R_LIBRARY_BUG                       274
#define SSL_R_LIBRARY_HAS_NO_CIPHERS                  161
#define SSL_R_MESSAGE_TOO_LONG                        296
#define SSL_R_MISSING_DH_DSA_CERT               162
#define SSL_R_MISSING_DH_KEY                    163
#define SSL_R_MISSING_DH_RSA_CERT               164
#define SSL_R_MISSING_DSA_SIGNING_CERT                165
#define SSL_R_MISSING_EXPORT_TMP_DH_KEY               166
#define SSL_R_MISSING_EXPORT_TMP_RSA_KEY        167
#define SSL_R_MISSING_RSA_CERTIFICATE                 168
#define SSL_R_MISSING_RSA_ENCRYPTING_CERT       169
#define SSL_R_MISSING_RSA_SIGNING_CERT                170
#define SSL_R_MISSING_TMP_DH_KEY                171
#define SSL_R_MISSING_TMP_ECDH_KEY              311
#define SSL_R_MISSING_TMP_RSA_KEY               172
#define SSL_R_MISSING_TMP_RSA_PKEY              173
#define SSL_R_MISSING_VERIFY_MESSAGE                  174
#define SSL_R_NON_SSLV2_INITIAL_PACKET                175
#define SSL_R_NO_CERTIFICATES_RETURNED                176
#define SSL_R_NO_CERTIFICATE_ASSIGNED                 177
#define SSL_R_NO_CERTIFICATE_RETURNED                 178
#define SSL_R_NO_CERTIFICATE_SET                179
#define SSL_R_NO_CERTIFICATE_SPECIFIED                180
```

```
#define SSL_R_NO_CIPHERS_AVAILABLE               181
#define SSL_R_NO_CIPHERS_PASSED                  182
#define SSL_R_NO_CIPHERS_SPECIFIED               183
#define SSL_R_NO_CIPHER_LIST                     184
#define SSL_R_NO_CIPHER_MATCH                    185
#define SSL_R_NO_CLIENT_CERT_METHOD              331
#define SSL_R_NO_CLIENT_CERT_RECEIVED            186
#define SSL_R_NO_COMPRESSION_SPECIFIED           187
#define SSL_R_NO_GOST_CERTIFICATE_SENT_BY_PEER   330
#define SSL_R_NO_METHOD_SPECIFIED                188
#define SSL_R_NO_PRIVATEKEY                      189
#define SSL_R_NO_PRIVATE_KEY_ASSIGNED            190
#define SSL_R_NO_PROTOCOLS_AVAILABLE             191
#define SSL_R_NO_PUBLICKEY                       192
#define SSL_R_NO_REQUIRED_DIGEST                 324
#define SSL_R_NO_SHARED_CIPHER                   193
#define SSL_R_NO_VERIFY_CALLBACK                 194
#define SSL_R_NULL_SSL_CTX                       195
#define SSL_R_NULL_SSL_METHOD_PASSED             196
#define SSL_R_OLD_SESSION_CIPHER_NOT_RETURNED    197
#define SSL_R_ONLY_TLS_ALLOWED_IN_FIPS_MODE      297
#define SSL_R_OPAQUE_PRF_INPUT_TOO_LONG          327
#define SSL_R_PACKET_LENGTH_TOO_LONG             198
#define SSL_R_PARSE_TLSEXT                       227
#define SSL_R_PATH_TOO_LONG                      270
#define SSL_R_PEER_DID_NOT_RETURN_A_CERTIFICATE  199
#define SSL_R_PEER_ERROR                         200
#define SSL_R_PEER_ERROR_CERTIFICATE             201
#define SSL_R_PEER_ERROR_NO_CERTIFICATE          202
#define SSL_R_PEER_ERROR_NO_CIPHER               203
#define SSL_R_PEER_ERROR_UNSUPPORTED_CERTIFICATE_TYPE 204
#define SSL_R_PRE_MAC_LENGTH_TOO_LONG            205
#define SSL_R_PROBLEMS_MAPPING_CIPHER_FUNCTIONS  206
#define SSL_R_PROTOCOL_IS_SHUTDOWN               207
#define SSL_R_PSK_IDENTITY_NOT_FOUND             223
#define SSL_R_PSK_NO_CLIENT_CB                   224
#define SSL_R_PSK_NO_SERVER_CB                   225
#define SSL_R_PUBLIC_KEY_ENCRYPT_ERROR           208
#define SSL_R_PUBLIC_KEY_IS_NOT_RSA              209
#define SSL_R_PUBLIC_KEY_NOT_RSA                 210
#define SSL_R_READ_BIO_NOT_SET                   211
#define SSL_R_READ_TIMEOUT_EXPIRED               312
#define SSL_R_READ_WRONG_PACKET_TYPE             212
#define SSL_R_RECORD_LENGTH_MISMATCH             213
#define SSL_R_RECORD_TOO_LARGE                   214
#define SSL_R_RECORD_TOO_SMALL                   298
#define SSL_R_REQUIRED_CIPHER_MISSING            215
#define SSL_R_REUSE_CERT_LENGTH_NOT_ZERO         216
#define SSL_R_REUSE_CERT_TYPE_NOT_ZERO           217
#define SSL_R_REUSE_CIPHER_LIST_NOT_ZERO         218
#define SSL_R_SERVERHELLO_TLSEXT                 275
#define SSL_R_SESSION_ID_CONTEXT_UNINITIALIZED   277
#define SSL_R_SHORT_READ                         219
#define SSL_R_SIGNATURE_FOR_NON_SIGNING_CERTIFICATE   220
```

```
#define SSL_R_SSL23_DOING_SESSION_ID_REUSE              221
#define SSL_R_SSL2_CONNECTION_ID_TOO_LONG       299
#define SSL_R_SSL3_EXT_INVALID_ECPOINTFORMAT            321
#define SSL_R_SSL3_EXT_INVALID_SERVERNAME       319
#define SSL_R_SSL3_EXT_INVALID_SERVERNAME_TYPE          320
#define SSL_R_SSL3_SESSION_ID_TOO_LONG                  300
#define SSL_R_SSL3_SESSION_ID_TOO_SHORT                 222
#define SSL_R_SSLV3_ALERT_BAD_CERTIFICATE       1042
#define SSL_R_SSLV3_ALERT_BAD_RECORD_MAC        1020
#define SSL_R_SSLV3_ALERT_CERTIFICATE_EXPIRED           1045
#define SSL_R_SSLV3_ALERT_CERTIFICATE_REVOKED           1044
#define SSL_R_SSLV3_ALERT_CERTIFICATE_UNKNOWN           1046
#define SSL_R_SSLV3_ALERT_DECOMPRESSION_FAILURE         1030
#define SSL_R_SSLV3_ALERT_HANDSHAKE_FAILURE             1040
#define SSL_R_SSLV3_ALERT_ILLEGAL_PARAMETER             1047
#define SSL_R_SSLV3_ALERT_NO_CERTIFICATE        1041
#define SSL_R_SSLV3_ALERT_UNEXPECTED_MESSAGE            1010
#define SSL_R_SSLV3_ALERT_UNSUPPORTED_CERTIFICATE       1043
#define SSL_R_SSL_CTX_HAS_NO_DEFAULT_SSL_VERSION        228
#define SSL_R_SSL_HANDSHAKE_FAILURE             229
#define SSL_R_SSL_LIBRARY_HAS_NO_CIPHERS        230
#define SSL_R_SSL_SESSION_ID_CALLBACK_FAILED            301
#define SSL_R_SSL_SESSION_ID_CONFLICT                   302
#define SSL_R_SSL_SESSION_ID_CONTEXT_TOO_LONG           273
#define SSL_R_SSL_SESSION_ID_HAS_BAD_LENGTH             303
#define SSL_R_SSL_SESSION_ID_IS_DIFFERENT               231
#define SSL_R_TLSV1_ALERT_ACCESS_DENIED                 1049
#define SSL_R_TLSV1_ALERT_DECODE_ERROR                  1050
#define SSL_R_TLSV1_ALERT_DECRYPTION_FAILED             1021
#define SSL_R_TLSV1_ALERT_DECRYPT_ERROR                 1051
#define SSL_R_TLSV1_ALERT_EXPORT_RESTRICTION            1060
#define SSL_R_TLSV1_ALERT_INSUFFICIENT_SECURITY         1071
#define SSL_R_TLSV1_ALERT_INTERNAL_ERROR                1080
#define SSL_R_TLSV1_ALERT_NO_RENEGOTIATION              1100
#define SSL_R_TLSV1_ALERT_PROTOCOL_VERSION              1070
#define SSL_R_TLSV1_ALERT_RECORD_OVERFLOW               1022
#define SSL_R_TLSV1_ALERT_UNKNOWN_CA                    1048
#define SSL_R_TLSV1_ALERT_USER_CANCELLED                1090
#define SSL_R_TLSV1_BAD_CERTIFICATE_HASH_VALUE          1114
#define SSL_R_TLSV1_BAD_CERTIFICATE_STATUS_RESPONSE     1113
#define SSL_R_TLSV1_CERTIFICATE_UNOBTAINABLE            1111
#define SSL_R_TLSV1_UNRECOGNIZED_NAME                   1112
#define SSL_R_TLSV1_UNSUPPORTED_EXTENSION               1110
#define SSL_R_TLS_CLIENT_CERT_REQ_WITH_ANON_CIPHER      232
#define SSL_R_TLS_INVALID_ECPOINTFORMAT_LIST            157
#define SSL_R_TLS_PEER_DID_NOT_RESPOND_WITH_CERTIFICATE_LIST 233
#define SSL_R_TLS_RSA_ENCRYPTED_VALUE_LENGTH_IS_WRONG 234
#define SSL_R_TRIED_TO_USE_UNSUPPORTED_CIPHER           235
#define SSL_R_UNABLE_TO_DECODE_DH_CERTS                 236
#define SSL_R_UNABLE_TO_DECODE_ECDH_CERTS               313
#define SSL_R_UNABLE_TO_EXTRACT_PUBLIC_KEY              237
#define SSL_R_UNABLE_TO_FIND_DH_PARAMETERS              238
#define SSL_R_UNABLE_TO_FIND_ECDH_PARAMETERS            314
#define SSL_R_UNABLE_TO_FIND_PUBLIC_KEY_PARAMETERS      239
```

```
#define SSL_R_UNABLE_TO_FIND_SSL_METHOD                 240
#define SSL_R_UNABLE_TO_LOAD_SSL2_MD5_ROUTINES          241
#define SSL_R_UNABLE_TO_LOAD_SSL3_MD5_ROUTINES          242
#define SSL_R_UNABLE_TO_LOAD_SSL3_SHA1_ROUTINES         243
#define SSL_R_UNEXPECTED_MESSAGE                        244
#define SSL_R_UNEXPECTED_RECORD                         245
#define SSL_R_UNINITIALIZED                             276
#define SSL_R_UNKNOWN_ALERT_TYPE                        246
#define SSL_R_UNKNOWN_CERTIFICATE_TYPE                  247
#define SSL_R_UNKNOWN_CIPHER_RETURNED                   248
#define SSL_R_UNKNOWN_CIPHER_TYPE                       249
#define SSL_R_UNKNOWN_KEY_EXCHANGE_TYPE                 250
#define SSL_R_UNKNOWN_PKEY_TYPE                         251
#define SSL_R_UNKNOWN_PROTOCOL                          252
#define SSL_R_UNKNOWN_REMOTE_ERROR_TYPE                 253
#define SSL_R_UNKNOWN_SSL_VERSION                       254
#define SSL_R_UNKNOWN_STATE                             255
#define SSL_R_UNSUPPORTED_CIPHER                        256
#define SSL_R_UNSUPPORTED_COMPRESSION_ALGORITHM         257
#define SSL_R_UNSUPPORTED_DIGEST_TYPE                   326
#define SSL_R_UNSUPPORTED_ELLIPTIC_CURVE                315
#define SSL_R_UNSUPPORTED_PROTOCOL                      258
#define SSL_R_UNSUPPORTED_SSL_VERSION                   259
#define SSL_R_UNSUPPORTED_STATUS_TYPE                   329
#define SSL_R_WRITE_BIO_NOT_SET                         260
#define SSL_R_WRONG_CIPHER_RETURNED                     261
#define SSL_R_WRONG_MESSAGE_TYPE                        262
#define SSL_R_WRONG_NUMBER_OF_KEY_BITS                  263
#define SSL_R_WRONG_SIGNATURE_LENGTH                    264
#define SSL_R_WRONG_SIGNATURE_SIZE                      265
#define SSL_R_WRONG_SSL_VERSION                         266
#define SSL_R_WRONG_VERSION_NUMBER                      267
#define SSL_R_X509_LIB                                  268
#define SSL_R_X509_VERIFICATION_SETUP_PROBLEMS          269
```