

---

# Secure Transport Reference

Security



2004-08-31



Apple Inc.  
© 2003, 2004 Apple Computer, Inc.  
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.  
1 Infinite Loop  
Cupertino, CA 95014  
408-996-1010

Apple, the Apple logo, Carbon, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

**Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY**

**DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.**

**THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.**

**Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.**

# Contents

## Secure Transport Reference 5

---

Overview	5
Functions by Task	6
Creating and Disposing of a Session Context	6
Configuring an SSL Session	6
Managing an SSL Session	7
Managing Ciphers	7
Managing Root Certificates	8
Managing Certificates	8
Managing the Peer Domain Name	9
Deprecated Functions	9
Functions	9
SSLAddDistinguishedName	9
SSLClose	10
SSLDisposeContext	10
SSLGetAllowsAnyRoot	11
SSLGetAllowsExpiredCerts	11
SSLGetAllowsExpiredRoots	12
SSLGetBufferedReadSize	12
SSLGetClientCertificateState	13
SSLGetConnection	13
SSLGetDiffieHellmanParams	14
SSLGetEnableCertVerify	15
SSLGetEnabledCiphers	15
SSLGetNegotiatedCipher	16
SSLGetNegotiatedProtocolVersion	16
SSLGetNumberEnabledCiphers	17
SSLGetNumberSupportedCiphers	18
SSLGetPeerDomainName	18
SSLGetPeerDomainNameLength	19
SSLGetPeerID	19
SSLGetProtocolVersion	20
SSLGetProtocolVersionEnabled	21
SSLGetRsaBlinding	21
SSLGetSessionState	22
SSLGetSupportedCiphers	22
SSLHandshake	23
SSLNewContext	24
SSLRead	25
SSLSetAllowsAnyRoot	25
SSLSetAllowsExpiredCerts	26

- SSLSetAllowsExpiredRoots 27
- SSLSetCertificate 28
- SSLSetClientSideAuthenticate 29
- SSLSetConnection 29
- SSLSetDiffieHellmanParams 30
- SSLSetEnableCertVerify 31
- SSLSetEnabledCiphers 32
- SSLSetEncryptionCertificate 32
- SSLSetIOFuncs 33
- SSLSetPeerDomainName 34
- SSLSetPeerID 35
- SSLSetProtocolVersion 35
- SSLSetProtocolVersionEnabled 36
- SSLSetRsaBlinding 37
- SSLSetTrustedRoots 38
- SSLWrite 38
- Callbacks 39
  - SSLReadFunc 39
  - SSLWriteFunc 40
- Data Types 41
  - SSLConnectionRef 41
  - SSLContextRef 41
- Constants 42
  - SSL Authentication Constants 42
  - SSL Cipher Suite Constants 42
  - SSL Client Certificate State Constants 44
  - SSL Protocol Constants 45
  - SSL Session State Constants 46
- Result Codes 47

**Appendix A      [Deprecated Secure Transport Functions 51](#)**

---

- Deprecated in Mac OS X v10.5 51
  - SSLGetPeerCertificates 51
  - SSLGetTrustedRoots 52

**[Document Revision History 53](#)**

---

**[Index 55](#)**

---

# Secure Transport Reference

---

<b>Framework:</b>	Security/Security.h
<b>Declared in</b>	CipherSuite.h SecureTransport.h

## Overview

This document describes the public API for an implementation of the protocols Secure Sockets Layer version 3.0 and Transport Layer Security version 1.0.

There are no transport layer dependencies in this library; it can be used with BSD Sockets and Open Transport, among other protocols. To use this library, you must provide callback functions to perform the actual I/O on underlying network connections. You are also responsible for setting up raw network connections; you pass in an opaque reference to the underlying (connected) entity at the start of an SSL session in the form of an [SSLConnect ionRef](#) (page 41) object.

The following terms are used in this document:

- A *client* is the initiator of an SSL session. The canonical example of a client is a web browser communicating with an HTTPS URL.
- A *server* is an entity that accepts requests for SSL sessions made by clients. An example is a secure web server.
- An *SSL session* is bounded by calls to the functions [SSLHandshake](#) (page 23) and [SSLClose](#) (page 10). An active session is in some state between these two calls, inclusive.
- An *SSL session context*, or [SSLContextRef](#) (page 41), is an opaque reference to the state associated with one session. A session context cannot be reused for multiple sessions.

Most applications need only a few of the functions in this API, which are normally called in the following sequence:

1. Preparing for a session
  - a. Call [SSLNewContext](#) (page 24) to create a new SSL session context.
  - b. Write I/O functions and call [SSLSetIOFuncs](#) (page 33) to pass them to Secure Transport.
  - c. Establish a connection using CFNetwork, BSD Sockets, or Open Transport. Then call [SSLSetConnection](#) (page 29) to specify the connection to which the SSL session context applies.
  - d. Call [SSLSetPeerDomainName](#) (page 34) to specify the fully-qualified domain name of the peer to which you want to connect (optional but highly recommended).

- e. Call [SSLSetCertificate](#) (page 28) to specify the certificate to be used in authentication (required for server side, optional for client).
2. Starting a session
    - Call [SSLHandshake](#) (page 23) to perform the SSL handshake and establish a secure session.
  3. Maintaining the session
    - To transfer data over the secure session, call [SSLWrite](#) (page 38) and [SSLRead](#) (page 25) as needed.
  4. Ending a session
    - a. Call [SSLClose](#) (page 10) to close the secure session.
    - b. Close the connection and dispose of the connection reference ([SSLConnectionRef](#) (page 41)).
    - c. Call [SSLDisposeContext](#) (page 10) to dispose of the SSL session context.
    - d. If you have called [SSLGetPeerCertificates](#) (page 51) to obtain any certificates, call `CFRelease` to release the certificate reference objects.

In many cases, it is easier to use the CFNetwork API than Secure Transport to implement a simple connection to a secure (HTTPS) URL. See *CFNetwork Programming Guide* for documentation of the CFNetwork API and the *CFNetworkHTTPDownload* sample code for an example of code that downloads data from a URL. If you specify an HTTPS URL, this routine automatically uses Secure Transport to encrypt the data stream.

For functions to manage and evaluate certificates, see *Certificate, Key, and Trust Services Reference* and *Certificate, Key, and Trust Services Programming Guide*.

## Functions by Task

### Creating and Disposing of a Session Context

- [SSLNewContext](#) (page 24)  
Creates a new SSL session context.
- [SSLDisposeContext](#) (page 10)  
Disposes of an SSL session context.

### Configuring an SSL Session

- [SSLSetConnection](#) (page 29)  
Specifies an I/O connection for a specific session.
- [SSLGetConnection](#) (page 13)  
Retrieves an I/O connection—such as a socket or endpoint—for a specific session.

[SSLSetIOFuncs](#) (page 33)

Specifies callback functions that perform the network I/O operations.

[SSLSetProtocolVersionEnabled](#) (page 36)

Sets the allowed SSL protocol versions.

[SSLGetProtocolVersionEnabled](#) (page 21)

Retrieves the enabled status of a given protocol.

[SSLSetClientSideAuthenticate](#) (page 29)

Specifies the requirements for client-side authentication.

[SSLSetRsaBlinding](#) (page 37)

Enables or disables RSA blinding.

[SSLGetRsaBlinding](#) (page 21)

Obtains a value indicating whether RSA blinding is enabled.

## Managing an SSL Session

[SSLHandshake](#) (page 23)

Performs the SSL handshake.

[SSLGetSessionState](#) (page 22)

Retrieves the state of an SSL session.

[SSLGetNegotiatedProtocolVersion](#) (page 16)

Obtains the negotiated protocol version of the active session.

[SSLSetPeerID](#) (page 35)

Specifies data that is sufficient to uniquely identify the peer of the current session.

[SSLGetPeerID](#) (page 19)

Retrieves the current peer ID data.

[SSLGetBufferedReadSize](#) (page 12)

Determines how much data is available to be read.

[SSLRead](#) (page 25)

Performs a normal application-level read operation.

[SSLWrite](#) (page 38)

Performs a normal application-level write operation.

[SSLClose](#) (page 10)

Terminates the current SSL session.

## Managing Ciphers

[SSLGetNumberSupportedCiphers](#) (page 18)

Determines the number of cipher suites supported.

[SSLGetSupportedCiphers](#) (page 22)

Determines the values of the supported cipher suites.

[SSLSetEnabledCiphers](#) (page 32)

Specifies a restricted set of SSL cipher suites to be enabled by the current SSL session context.

- [SSLGetNumberEnabledCiphers](#) (page 17)  
Determines the number of cipher suites currently enabled.
- [SSLGetEnabledCiphers](#) (page 15)  
Determines which SSL cipher suites are currently enabled.
- [SSLGetNegotiatedCipher](#) (page 16)  
Retrieves the cipher suite negotiated for this session.
- [SSLSetDiffieHellmanParams](#) (page 30)  
Specifies Diffie-Hellman parameters.
- [SSLGetDiffieHellmanParams](#) (page 14)  
Retrieves the Diffie-Hellman parameters specified earlier.

## Managing Root Certificates

- [SSLSetAllowsAnyRoot](#) (page 25)  
Specifies whether root certificates from unrecognized certification authorities are allowed.
- [SSLGetAllowsAnyRoot](#) (page 11)  
Obtains a value specifying whether an unknown root is allowed.
- [SSLSetAllowsExpiredRoots](#) (page 27)  
Specifies whether expired root certificates are allowed.
- [SSLGetAllowsExpiredRoots](#) (page 12)  
Retrieves the value indicating whether expired roots are allowed.
- [SSLSetTrustedRoots](#) (page 38)  
Augments or replaces the default set of trusted root certificates for this session.
- [SSLGetTrustedRoots](#) (page 52) **Deprecated in Mac OS X v10.5**  
Retrieves the current list of trusted root certificates.

## Managing Certificates

- [SSLAddDistinguishedName](#) (page 9)  
Unsupported.
- [SSLSetAllowsExpiredCerts](#) (page 26)  
Specifies whether certificate expiration times are ignored.
- [SSLGetAllowsExpiredCerts](#) (page 11)  
Retrieves the value specifying whether expired certificates are allowed.
- [SSLSetCertificate](#) (page 28)  
Specifies this connection's certificate or certificates.
- [SSLGetClientCertificateState](#) (page 13)  
Retrieves the exchange status of the client certificate.
- [SSLSetEnableCertVerify](#) (page 31)  
Enables or disables peer certificate chain validation.
- [SSLGetEnableCertVerify](#) (page 15)  
Determines whether peer certificate chain validation is currently enabled.

[SSLSetEncryptionCertificate](#) (page 32)

Specifies the encryption certificates used for this connection.

[SSLGetPeerCertificates](#) (page 51) **Deprecated in Mac OS X v10.5**

Retrieves a peer certificate.

## Managing the Peer Domain Name

[SSLSetPeerDomainName](#) (page 34)

Specifies the fully qualified domain name of the peer.

[SSLGetPeerDomainNameLength](#) (page 19)

Determines the length of a previously set peer domain name.

[SSLGetPeerDomainName](#) (page 18)

Retrieves the peer domain name specified previously.

## Deprecated Functions

[SSLSetProtocolVersion](#) (page 35)

Sets the SSL protocol version. This function is deprecated.

[SSLGetProtocolVersion](#) (page 20)

Gets the SSL protocol version. This function is deprecated.

# Functions

## SSLAddDistinguishedName

Unsupported.

```
OSStatus SSLAddDistinguishedName (  
    SSLContextRef context,  
    const void *derDN,  
    size_t derDNLen  
);
```

### Parameters

*context*

An SSL session context reference.

*derDN*

A pointer to a buffer containing a DER-encoded distinguished name.

*derDNLen*

A value of type `size_t` representing the size of the buffer pointed to by the parameter *derDN*.

### Return Value

A result code. See “[Secure Transport Result Codes](#)” (page 47).

### Discussion

This function has not been implemented and is unsupported at this time.

**Availability**

Unsupported.

**Declared In**

SecureTransport.h

**SSLClose**

Terminates the current SSL session.

```
OSStatus SSLClose (  
    SSLContextRef context  
);
```

**Parameters**

*context*

The SSL session context reference of the session you want to terminate.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Availability**

Available in Mac OS X v10.2 and later.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLDisposeContext**

Disposes of an SSL session context.

```
OSStatus SSLDisposeContext (  
    SSLContextRef context  
);
```

**Parameters**

*context*

A reference to the SSL session context to dispose.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

When you are completely finished with a secure session, you should dispose of the SSL session context in order to release the memory associated with the session.

**Availability**

Available in Mac OS X v10.2 and later.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLGetAllowsAnyRoot**

Obtains a value specifying whether an unknown root is allowed.

```
OSStatus SSLGetAllowsAnyRoot (
    SSLContextRef context,
    Boolean *anyRoot
);
```

**Parameters***context*

An SSL session context reference.

*anyRoot*

On return, a Boolean indicating the current setting of the `anyRoot` flag.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

Use the [SSLSetAllowsAnyRoot](#) (page 25) function to set the value of the `anyRoot` flag. The effect and meaning of this flag is described in the discussion of the [SSLSetAllowsAnyRoot](#) (page 25) function.

**Availability**

Available in Mac OS X v10.2 and later.

**Declared In**

SecureTransport.h

**SSLGetAllowsExpiredCerts**

Retrieves the value specifying whether expired certificates are allowed.

```
OSStatus SSLGetAllowsExpiredCerts (
    SSLContextRef context,
    Boolean *allowsExpired
);
```

**Parameters***context*

An SSL session context reference.

*allowsExpired*

On return, this flag is set to the value of the Boolean flag that specifies whether expired certificates are ignored. If this value is `true`, then Secure Transport does not return an error if any certificates in the certificate chain are expired.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

You can set the `allowsExpired` flag to allow the handshake to succeed even if one or more certificates in the certificate chain have expired. This function returns the current setting of this flag. Use the [SSLSetAllowsExpiredCerts](#) (page 26) function to set the value of the `allowsExpired` flag.

**Availability**

Available in Mac OS X v10.2 and later.

**Declared In**

`SecureTransport.h`

**SSLGetAllowsExpiredRoots**

Retrieves the value indicating whether expired roots are allowed.

```
OSStatus SSLGetAllowsExpiredRoots (
    SSLContextRef context,
    Boolean *allowsExpired
);
```

**Parameters**

*context*

An SSL session context reference.

*allowsExpired*

On return, points to a Boolean value indicating whether expired roots are allowed. If this value is `true`, no errors are returned if the certificate chain ends in an expired root.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

Use the [SSLSetAllowsExpiredRoots](#) (page 27) function to change the setting of the `allowsExpired` flag.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`SecureTransport.h`

**SSLGetBufferedReadSize**

Determines how much data is available to be read.

```
OSStatus SSLGetBufferedReadSize (
    SSLContextRef context,
    size_t *bufSize
);
```

**Parameters**

*context*

An SSL session context reference.

*bufSize*

On return, the size of the data to be read.

#### Return Value

A result code. See “[Secure Transport Result Codes](#)” (page 47).

#### Discussion

This function determines how much data you can be guaranteed to obtain in a call to the [SSLRead](#) (page 25) function. This function does not block or cause any low-level read operations to occur.

#### Availability

Available in Mac OS X v10.2 and later.

#### Related Sample Code

SSLSample

#### Declared In

SecureTransport.h

## SSLGetClientCertificateState

Retrieves the exchange status of the client certificate.

```
OSStatus SSLGetClientCertificateState (
    SSLContextRef context,
    SSLClientCertificateState *clientState
);
```

#### Parameters

*context*

An SSL session context reference.

*clientState*

On return, a pointer to a value indicating the state of the client certificate exchange. See [SSL Client Certificate State Constants](#) (page 44) for a list of possible values.

#### Return Value

A result code. See “[Secure Transport Result Codes](#)” (page 47).

#### Discussion

The value returned reflects the latest change in the state of the client certificate exchange. If either peer initiates a renegotiation attempt, Secure Transport resets the state to `kSSLClientCertNone`.

#### Availability

Available in Mac OS X v10.3 and later.

#### Declared In

SecureTransport.h

## SSLGetConnection

Retrieves an I/O connection—such as a socket or endpoint—for a specific session.

```
OSStatus SSLGetConnection (
    SSLContextRef context,
    SSLConnectionRef *connection
);
```

**Parameters***context*

An SSL session context reference.

*connection*On return, a pointer to a session connection reference. If no connection has been set using the [SSLSetConnection](#) (page 29) function, then this parameter is NULL on return.**Return Value**A result code. See [“Secure Transport Result Codes”](#) (page 47).**Discussion**

You can use this function on either the client or server to retrieve the connection associated with a secure session.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

SecureTransport.h

**SSLGetDiffieHellmanParams**

Retrieves the Diffie-Hellman parameters specified earlier.

```
OSStatus SSLGetDiffieHellmanParams (
    SSLContextRef context,
    const void **dhParams,
    size_t *dhParamsLen
);
```

**Parameters***context*

An SSL session context reference.

*dhParams*

On return, points to a buffer containing the Diffie-Hellman parameter block in Open SSL DER format. The returned data is not copied and belongs to the SSL session context reference; therefore, you cannot modify the data and it is released automatically when you dispose of the context.

*dhParamsLen*On return, points to the length of the buffer pointed to by the *dhParams* parameter.**Return Value**A result code. See [“Secure Transport Result Codes”](#) (page 47).**Discussion**This function returns the parameter block specified in an earlier call to the function [SSLSetDiffieHellmanParams](#) (page 30). If [SSLSetDiffieHellmanParams](#) was never called, the *dhParams* parameter returns NULL and the *dhParamsLen* parameter returns 0.

### Availability

Available in Mac OS X v10.3 and later.

### Declared In

SecureTransport.h

## SSLGetEnableCertVerify

Determines whether peer certificate chain validation is currently enabled.

```
OSStatus SSLGetEnableCertVerify (
    SSLContextRef context,
    Boolean *enableVerify
);
```

### Parameters

*context*

An SSL session context reference.

*enableVerify*

On return, a pointer to a Boolean value specifying whether peer certificate chain validation is enabled. If this value is `true`, then Secure Transport automatically attempts to verify the certificate chain during exchange of peer certificates.

### Return Value

A result code. See “[Secure Transport Result Codes](#)” (page 47).

### Discussion

Use the [SSLSetEnableCertVerify](#) (page 31) function to set the value of the `enableVerify` flag.

### Availability

Available in Mac OS X v10.3 and later.

### Declared In

SecureTransport.h

## SSLGetEnabledCiphers

Determines which SSL cipher suites are currently enabled.

```
OSStatus SSLGetEnabledCiphers (
    SSLContextRef context,
    SSLCipherSuite *ciphers,
    size_t *numCiphers
);
```

### Parameters

*context*

An SSL session context reference.

*ciphers*

On return, points to the enabled cipher suites. Before calling, you must allocate this buffer using the number of enabled cipher suites retrieved from a call to the [SSLGetNumberEnabledCiphers](#) (page 17) function.

*numCiphers*

Pointer to the number of enabled cipher suites. Before calling, retrieve this value by calling the [SSLGetNumberEnabledCiphers](#) (page 17) function.

**Return Value**

A result code. See “[Secure Transport Result Codes](#)” (page 47). If the supplied buffer is too small, `errSSLBufferOverflow` is returned.

**Discussion**

Call the [SSLSetEnabledCiphers](#) (page 32) function to specify which SSL cipher suites are enabled.

**Availability**

Available in Mac OS X v10.2 and later.

**Declared In**

`SecureTransport.h`

## SSLGetNegotiatedCipher

Retrieves the cipher suite negotiated for this session.

```
OSStatus SSLGetNegotiatedCipher (
    SSLContextRef context,
    SSLCipherSuite *cipherSuite
);
```

**Parameters**

*context*

An SSL session context reference.

*cipherSuite*

On return, points to the cipher suite that was negotiated for this session.

**Return Value**

A result code. See “[Secure Transport Result Codes](#)” (page 47).

**Discussion**

You should call this function only when a session is active.

**Availability**

Available in Mac OS X v10.2 and later.

**Related Sample Code**

`SSLSample`

**Declared In**

`SecureTransport.h`

## SSLGetNegotiatedProtocolVersion

Obtains the negotiated protocol version of the active session.

```
OSStatus SSLGetNegotiatedProtocolVersion (
    SSLContextRef context,
    SSLProtocol *protocol
);
```

**Parameters***context*

An SSL session context reference.

*protocol*

On return, points to the negotiated protocol version of the active session.

**Return Value**A result code. See [“Secure Transport Result Codes”](#) (page 47). This function returns `kSSLProtocolUnknown` if no SSL session is in progress.**Discussion**

This function retrieves the version of SSL or TLS protocol negotiated for the session. Note that the negotiated protocol may not be the same as your preferred protocol, depending on which protocol versions you enabled with the [SSLSetProtocolVersionEnabled](#) (page 36) function. This function can return any of the following values:

- `kSSLProtocol2`
- `kSSLProtocol3`
- `kTLSProtocol1`
- `kSSLProtocolUnknown`

**Availability**

Available in Mac OS X v10.2 and later.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLGetNumberEnabledCiphers**

Determines the number of cipher suites currently enabled.

```
OSStatus SSLGetNumberEnabledCiphers (
    SSLContextRef context,
    size_t *numCiphers
);
```

**Parameters***context*

An SSL session context reference.

*numCiphers*

On return, points to the number of enabled cipher suites.

**Return Value**A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

You use the number of enabled cipher suites returned by this function when you call the [SSLGetEnabledCiphers](#) (page 15) function to retrieve the list of currently enabled cipher suites.

**Availability**

Available in Mac OS X v10.2 and later.

**Declared In**

SecureTransport.h

**SSLGetNumberSupportedCiphers**

Determines the number of cipher suites supported.

```
OSStatus SSLGetNumberSupportedCiphers (
    SSLContextRef context,
    size_t *numCiphers
);
```

**Parameters**

*context*

An SSL session context reference.

*numCiphers*

On return, points to the number of supported cipher suites.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

You use the number of enabled cipher suites returned by this function when you call the [SSLGetNumberSupportedCiphers](#) (page 18) function to retrieve the list of currently enabled cipher suites.

**Availability**

Available in Mac OS X v10.2 and later.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLGetPeerDomainName**

Retrieves the peer domain name specified previously.

```
OSStatus SSLGetPeerDomainName (
    SSLContextRef context,
    char *peerName,
    size_t *peerNameLen
);
```

**Parameters**

*context*

An SSL session context reference.

*peerName*

On return, points to the peer domain name.

*peerNameLen*

A pointer to the length of the peer domain name. Before calling this function, retrieve the peer domain name length by calling the function [SSLGetPeerDomainNameLength](#) (page 19).

#### Return Value

A result code. See [“Secure Transport Result Codes”](#) (page 47).

#### Discussion

If you previously called the [SSLSetPeerDomainName](#) (page 34) function to specify a fully qualified domain name for the peer certificate, you can use the [SSLGetPeerDomainName](#) function to retrieve the domain name.

#### Availability

Available in Mac OS X v10.2 and later.

#### Declared In

SecureTransport.h

## SSLGetPeerDomainNameLength

Determines the length of a previously set peer domain name.

```
OSStatus SSLGetPeerDomainNameLength (
    SSLContextRef context,
    size_t *peerNameLen
);
```

#### Parameters

*context*

An SSL session context reference.

*peerNameLen*

On return, points to the length of the peer domain name.

#### Return Value

A result code. See [“Secure Transport Result Codes”](#) (page 47).

#### Discussion

If you previously called the [SSLSetPeerDomainName](#) (page 34) function to specify a fully qualified domain name for the peer certificate, you can use the [SSLGetPeerDomainName](#) (page 18) function to retrieve the peer domain name. Before doing so, you must call the [SSLGetPeerDomainNameLength](#) function to retrieve the buffer size needed for the domain name.

#### Availability

Available in Mac OS X v10.2 and later.

#### Declared In

SecureTransport.h

## SSLGetPeerID

Retrieves the current peer ID data.

```
OSStatus SSLGetPeerID (
    SSLContextRef context,
    const void **peerID,
    size_t *peerIDLen
);
```

**Parameters***context*

An SSL session context reference.

*peerID*

On return, points to a buffer containing the peer ID data.

*peerIDLen*

On return, the length of the peer ID data buffer.

**Return Value**A result code. See [“Secure Transport Result Codes”](#) (page 47).**Discussion**

If the peer ID data for this context was not set by calling the [SSLSetPeerID](#) (page 35) function, this function returns a NULL pointer in the *peerID* parameter, and 0 in the *peerIDLen* parameter.

**Availability**

Available in Mac OS X v10.2 and later.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLGetProtocolVersion**

Gets the SSL protocol version. This function is deprecated.

```
OSStatus SSLGetProtocolVersion (
    SSLContextRef context,
    SSLProtocol *protocol
);
```

**Parameters***context*

An SSL session context reference.

*protocol*

On return, a pointer to the SSL protocol version.

**Return Value**A result code. See [“Secure Transport Result Codes”](#) (page 47).**Discussion**

Use the [SSLGetProtocolVersionEnabled](#) (page 21) function instead.

**Availability**

Available in Mac OS X v10.2.

**Declared In**

SecureTransport.h

**SSLGetProtocolVersionEnabled**

Retrieves the enabled status of a given protocol.

```
OSStatus SSLGetProtocolVersionEnabled (
    SSLContextRef context,
    SSLProtocol protocol,
    Boolean *enable
);
```

**Parameters***context*

An SSL session context reference.

*protocol*A value of type `SSLProtocol` representing an SSL protocol version.*enable*On return, points to a Boolean value indicating whether the specified protocol version is enabled. If this value is `true`, the protocol is enabled.**Return Value**A result code. See [“Secure Transport Result Codes”](#) (page 47).**Discussion**You can specify any one of the following values for the `protocol` parameter:

- `kSSLProtocol2`
- `kSSLProtocol3`
- `kTLSProtocol1`
- `kSSLProtocolAll` Specify this value to determine whether all protocols are enabled.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

SecureTransport.h

**SSLGetRsaBlinding**

Obtains a value indicating whether RSA blinding is enabled.

```
OSStatus SSLGetRsaBlinding (
    SSLContextRef context,
    Boolean *blinding
);
```

**Parameters***context*

An SSL session context reference.

*blinding*

On return, a pointer to a Boolean value indicating whether RSA blinding is enabled.

**Return Value**

A result code. See “[Secure Transport Result Codes](#)” (page 47).

**Discussion**

This function is used only on the server side of a connection.

Call the [SSLSetRsaBlinding](#) (page 37) function to enable or disable RSA blinding.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

SecureTransport.h

## SSLGetSessionState

Retrieves the state of an SSL session.

```
OSStatus SSLGetSessionState (
    SSLContextRef context,
    SSLSessionState *state
);
```

**Parameters**

*context*

An SSL session context reference.

*state*

On return, points to a constant that indicates the state of the SSL session. See “[SSL Session State Constants](#)” (page 46) for possible values.

**Return Value**

A result code. See “[Secure Transport Result Codes](#)” (page 47).

**Availability**

Available in Mac OS X v10.2 and later.

**Declared In**

SecureTransport.h

## SSLGetSupportedCiphers

Determines the values of the supported cipher suites.

```
OSStatus SSLGetSupportedCiphers (
    SSLContextRef context,
    SSLCipherSuite *ciphers,
    size_t *numCiphers
);
```

**Parameters***context*

An SSL session context reference.

*ciphers*

On return, points to the values of the supported cipher suites. Before calling, you must allocate this buffer using the number of supported cipher suites retrieved from a call to the [SSLGetNumberSupportedCiphers](#) (page 18) function.

*numCiphers*

Points to the number of supported cipher suites that you want returned. Before calling, retrieve this value by calling the [SSLGetNumberSupportedCiphers](#) (page 18) function.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47). If the supplied buffer is too small, `errSSLBufferOverflow` is returned.

**Discussion**

All the supported cipher suites are enabled by default. Use the [SSLSetEnabledCiphers](#) (page 32) function to enable a subset of the supported cipher suites. Use the [SSLGetEnabledCiphers](#) (page 15) function to determine which cipher suites are currently enabled.

**Availability**

Available in Mac OS X v10.2 and later.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLHandshake**

Performs the SSL handshake.

```
OSStatus SSLHandshake (
    SSLContextRef context
);
```

**Parameters***context*

An SSL session context reference.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

On successful return, the session is ready for normal secure communication using the functions [SSLRead](#) (page 25) and [SSLWrite](#) (page 38).

If it finds any problems with the peer's certificate chain, Secure Transport aborts the handshake. You can use the [SSLGetPeerCertificates](#) (page 51) function to see the peer's certificate chain. This function can return a wide variety of result codes, including the following:

- `errSSLUnknownRootCert`—The peer has a valid certificate chain, but the root of the chain is not a known anchor certificate.
- `errSSLNoRootCert`—The peer's certificate chain was not verifiable to a root certificate.
- `errSSLCertExpired`—The peer's certificate chain has one or more expired certificates.
- `errSSLXCertChainInvalid`—The peer has an invalid certificate chain; for example, signature verification within the chain failed, or no certificates were found.

A return value of `errSSLWouldBlock` indicates that the `SSLHandshake` function must be called again until a different result code is returned.

#### Availability

Available in Mac OS X v10.2 and later.

#### Related Sample Code

SSLSample

#### Declared In

SecureTransport.h

## SSLNewContext

Creates a new SSL session context.

```
OSStatus SSLNewContext (
    Boolean isServer,
    SSLContextRef *contextPtr
);
```

#### Parameters

*isServer*

A Boolean value; True if the calling process is a server.

*contextPtr*

On return, points to a new SSL session context reference.

#### Return Value

A result code. See [“Secure Transport Result Codes”](#) (page 47).

#### Discussion

The SSL session context is an opaque data structure that identifies a session and stores session information. You must pass this object to every other function in the Secure Transport API.

#### Availability

Available in Mac OS X v10.2 and later.

#### Related Sample Code

SSLSample

**Declared In**

SecureTransport.h

**SSLRead**

Performs a normal application-level read operation.

```
OSStatus SSLRead (
    SSLContextRef context,
    void *data,
    size_t dataLength,
    size_t *processed
);
```

**Parameters***context*

An SSL session context reference.

*data*

On return, points to the data read. You must allocate this buffer before calling the function. The size of this buffer must be equal to or greater than the value in the `dataLength` parameter.

*dataLength*

The amount of data you would like to read.

*processed*

On return, points to the number of bytes actually read.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

The `SSLRead` function might call the `SSLReadFunc` (page 39) function that you provide (see [SSLSetIOFuncs](#) (page 33)). Because you may configure the underlying connection to operate in a nonblocking manner, a read operation might return `errSSLWouldBlock`, indicating that less data than requested was actually transferred. In this case, you should repeat the call to `SSLRead` until some other result is returned.

**Availability**

Available in Mac OS X v10.2 and later.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLSetAllowsAnyRoot**

Specifies whether root certificates from unrecognized certification authorities are allowed.

```
OSStatus SSLSetAllowsAnyRoot (
    SSLContextRef context,
    Boolean anyRoot
);
```

**Parameters***context*

An SSL session context reference.

*anyRoot*

A Boolean flag specifying whether root certificates from unrecognized certification authorities (CAs) are allowed. The default for this flag is `false`, specifying that roots from unrecognized CAs are not allowed.

**Return Value**

A result code. See “[Secure Transport Result Codes](#)” (page 47).

**Discussion**

The system maintains a set of root certificates signed by known, trusted root CAs. When the `anyRoot` flag is `true`, Secure Transport does not return an error if one of the following two conditions occurs:

- The peer returns a certificate chain with a root certificate, and the chain verifies to that root, but the CA for the root certificate is not one of the known, trusted root CAs. This results in an `errSSLUnknownRootCert` result code when the `anyRoot` flag is `false`.
- The peer returns a certificate chain that does not contain a root certificate, and the server can't verify the chain to one of the trusted root certificates. This results in an `errSSLNoRootCert` result code when the `anyRoot` flag is `false`.

Both of these error conditions are ignored when the `anyRoot` flag is `true`, allowing connection to a peer for which trust could not be established.

If you use this function to allow an untrusted root to be used for validation of a certificate—for example, after prompting the user for permission to do so—remember to set the `anyRoot` Boolean value back to `false`. If you don't, any random root certificate can be used for signing a certificate chain. To add a certificate to the list of trusted roots, use the `SecTrustSetAnchorCertificates` function.

**Availability**

Available in Mac OS X v10.2 and later.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLSetAllowsExpiredCerts**

Specifies whether certificate expiration times are ignored.

```
OSStatus SSLSetAllowsExpiredCerts (
    SSLContextRef context,
    Boolean allowsExpired
);
```

**Parameters***context*

An SSL session context reference.

*allowsExpired*A Boolean flag representing whether the certificate expiration times are ignored. The default for this flag is `false`, meaning expired certificates result in an `errSSLCertExpired` result code.**Return Value**A result code. See “[Secure Transport Result Codes](#)” (page 47).**Discussion**

You can use this function to allow the handshake to succeed even if one or more certificates in the certificate chain have expired. You can use the [SSLGetAllowsExpiredCerts](#) (page 11) function to determine the current setting of the `allowsExpired` flag.

Use the [SSLSetAllowsExpiredRoots](#) (page 27) function to set a flag specifying whether expired root certificates are allowed.

**Availability**

Available in Mac OS X v10.2 and later.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLSetAllowsExpiredRoots**

Specifies whether expired root certificates are allowed.

```
OSStatus SSLSetAllowsExpiredRoots (
    SSLContextRef context,
    Boolean allowsExpired
);
```

**Parameters***context*

An SSL session context reference.

*allowsExpired*A Boolean value indicating whether to allow expired root certificates. Pass `true` to allow expired roots.**Return Value**A result code. See “[Secure Transport Result Codes](#)” (page 47).**Discussion**

The default value for the `allowsExpired` flag is `false`. When this flag is `false`, Secure Transport returns an `errSSLCertExpired` result code during handshake if the root certificate is expired.

You can use the [SSLGetAllowsExpiredRoots](#) (page 12) function to determine the current setting of the `allowsExpired` flag.

Use the [SSLSetAllowsExpiredCerts](#) (page 26) function to set a value that determines whether expired non-root certificates are allowed.

#### Availability

Available in Mac OS X v10.3 and later.

#### Declared In

`SecureTransport.h`

## SSLSetCertificate

Specifies this connection's certificate or certificates.

```
OSStatus SSLSetCertificate (
    SSLContextRef context,
    CFArrayRef certRefs
);
```

#### Parameters

*context*

An SSL session context reference.

*certRefs*

The certificates to set. This array contains items of type `SecCertificateRef`, except for `certRefs[0]`, which is of type `SecIdentityRef`.

#### Return Value

A result code. See [“Secure Transport Result Codes”](#) (page 47).

#### Discussion

Setting the certificate or certificates is mandatory for server connections, but is optional for clients. Specifying a certificate for a client enables SSL client-side authentication. You must place in `certRefs[0]` a `SecIdentityRef` object that identifies the leaf certificate and its corresponding private key. Specifying a root certificate is optional; if it's not specified, the root certificate that verifies the certificate chain specified here must be present in the system wide set of trusted anchor certificates.

This function can be called only when no session is active.

Secure Transport assumes the following:

- The certificate references remain valid for the lifetime of the session.
- The identity specified in `certRefs[0]` is capable of signing.

The required capabilities of the identity specified in `certRefs[0]`, and of the optional certificate specified in the [SSLSetEncryptionCertificate](#) (page 32) function, are highly dependent on the application. For example, to work as a server with Netscape clients, the identity specified here must be capable of both signing and encrypting.

#### Availability

Available in Mac OS X v10.2 and later.

### Related Sample Code

SSLSample

### Declared In

SecureTransport.h

## SSLSetClientSideAuthenticate

Specifies the requirements for client-side authentication.

```
OSStatus SSLSetClientSideAuthenticate (
    SSLContextRef context,
    SSLAuthenticate auth
);
```

### Parameters

*context*

An SSL session context reference.

*auth*

A flag setting the requirements for client-side authentication. See [“SSL Authentication Constants”](#) (page 42) for possible values.

### Return Value

A result code. See [“Secure Transport Result Codes”](#) (page 47).

### Discussion

This function can be called only by servers. Use of this function is optional. The default authentication requirement is `kNeverAuthenticate`. This function may be called only when no session is active.

### Availability

Available in Mac OS X v10.2 and later.

### Related Sample Code

SSLSample

### Declared In

SecureTransport.h

## SSLSetConnection

Specifies an I/O connection for a specific session.

```
OSStatus SSLSetConnection (
    SSLContextRef context,
    SSLConnectionRef connection
);
```

### Parameters

*context*

An SSL session context reference.

*connection*

An SSL session connection reference. The connection data is opaque to Secure Transport; you can set it to any value that your application can use to uniquely identify the connection in the callback functions `SSLReadFunc` (page 39) and `SSLWriteFunc` (page 40).

#### Return Value

A result code. See “Secure Transport Result Codes” (page 47).

#### Discussion

You must establish a connection before creating a secure session. After calling the `SSLNewContext` (page 24) function to create an SSL session context, you call the `SSLSetConnection` function to specify the connection to which the context applies. You specify a value in the `connection` parameter that your callback routines can use to identify the connection. This value might be a pointer to a socket (if you are using the Sockets API) or an endpoint (if you are using Open Transport). For example, you might create a socket, start a connection on it, create a context reference, cast the socket to an `SSLConnectionRef`, and then pass both the context reference and connection reference to the `SSLSetConnection` function.

Note that the Sockets API is the preferred networking interface for new development.

On the client side, it’s assumed that communication has been established with the desired server on this connection. On the server side, it’s assumed that a connection has been established in response to an incoming client request .

This function must be called prior to the `SSLHandshake` (page 23) function; consequently, this function can be called only when no session is active.

#### Availability

Available in Mac OS X v10.2 and later.

#### Related Sample Code

SSLSample

#### Declared In

SecureTransport.h

## SSLSetDiffieHellmanParams

Specifies Diffie-Hellman parameters.

```
OSStatus SSLSetDiffieHellmanParams (
    SSLContextRef context,
    const void *dhParams,
    size_t dhParamsLen
);
```

#### Parameters

*context*

An SSL session context reference.

*dhParams*

A pointer to a buffer containing the Diffie-Hellman parameters in Open SSL DER format.

*dhParamsLen*

A value representing the size of the buffer pointed to by the *dhParams* parameter.

**Return Value**

A result code. See “[Secure Transport Result Codes](#)” (page 47).

**Discussion**

You can use this function to specify a set of Diffie-Hellman parameters to be used by Secure Transport for a specific session. Use of this function is optional. If Diffie-Hellman ciphers are allowed, the server and client negotiate a Diffie-Hellman cipher, and this function has not been called, then Secure Transport calculates a set of process wide parameters. However, that process can take as long as 30 seconds. Diffie-Hellman ciphers are enabled by default; see [SSLSetEnabledCiphers](#) (page 32).

In SSL/TLS, Diffie-Hellman parameters are always specified by the server. Therefore, this function can be called only by the server side of the connection.

You can use the [SSLGetDiffieHellmanParams](#) (page 14) function to retrieve Diffie-Hellman parameters specified in an earlier call to [SSLSetDiffieHellmanParams](#).

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

SecureTransport.h

**SSLSetEnableCertVerify**

Enables or disables peer certificate chain validation.

```
OSStatus SSLSetEnableCertVerify (
    SSLContextRef context,
    Boolean enableVerify
);
```

**Parameters**

*context*

An SSL session context reference.

*enableVerify*

A Boolean value specifying whether peer certificate chain validation is enabled. Certificate chain validation is enabled by default. Specify `false` to disable validation.

**Return Value**

A result code. See “[Secure Transport Result Codes](#)” (page 47).

**Discussion**

By default, Secure Transport attempts to verify the certificate chain during an exchange of peer certificates. If you disable peer certificate chain validation, it is your responsibility to call [SSLGetPeerCertificates](#) (page 51) upon successful completion of the handshake and then to validate the peer certificate chain before transferring the data.

You can use the [SSLGetEnableCertVerify](#) (page 15) function to determine the current setting of the `enableVerify` flag.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

SecureTransport.h

## SSLSetEnabledCiphers

Specifies a restricted set of SSL cipher suites to be enabled by the current SSL session context.

```
OSStatus SSLSetEnabledCiphers (
    SSLContextRef context,
    const SSLCipherSuite *ciphers,
    size_t numCiphers
);
```

### Parameters

*context*

An SSL session context reference.

*ciphers*

A pointer to the cipher suites to enable.

*numCiphers*

The number of cipher suites to enable.

### Return Value

A result code. See [“Secure Transport Result Codes”](#) (page 47).

### Discussion

You can call this function, for example, to limit cipher suites to those that use exportable key sizes or to those supported by a particular protocol version.

This function can be called only when no session is active. The default set of enabled cipher suites is the complete set of supported cipher suites obtained by calling the [SSLGetSupportedCiphers](#) (page 22) function.

Call the [SSLGetEnabledCiphers](#) (page 15) function to determine which SSL cipher suites are currently enabled.

### Availability

Available in Mac OS X v10.2 and later.

### Related Sample Code

SSLSample

### Declared In

SecureTransport.h

## SSLSetEncryptionCertificate

Specifies the encryption certificates used for this connection.

```
OSStatus SSLSetEncryptionCertificate (
    SSLContextRef context,
    CFArrayRef certRefs
);
```

### Parameters

*context*

An SSL session context reference.

*certRefs*

A value of type `CFArrayRef` referring to an array of certificate references. The references are type `SecCertificateRef`, except for `certRefs[0]`, which is of type `SecIdentityRef`.

#### Return Value

A result code. See “[Secure Transport Result Codes](#)” (page 47).

#### Discussion

Use this function in one of the following cases:

- The leaf certificate specified in the [SSLSetCertificate](#) (page 28) function is not capable of encryption.
- The leaf certificate specified in the [SSLSetCertificate](#) (page 28) function contains a key that is too large or strong for legal encryption in this session. In this case, a weaker certificate is specified here and is used for server-initiated key exchange.

The following assumptions are made:

- The *certRefs* parameter’s references remain valid for the lifetime of the connection.
- The specified `certRefs[0]` value is capable of encryption.

This function can be called only when no session is active.

SSL servers that enforce the SSL3 or TLS1 specification to the letter do not accept encryption certificates with key sizes larger than 512 bits for exportable ciphers (that is, for SSL sessions with 40-bit session keys). Therefore, if you wish to support exportable ciphers and your certificate has a key larger than 512 bits, you must specify a separate encryption certificate.

#### Availability

Available in Mac OS X v10.2 and later.

#### Related Sample Code

SSLSample

#### Declared In

SecureTransport.h

## SSLSetIOFuncs

Specifies callback functions that perform the network I/O operations.

```
OSStatus SSLSetIOFuncs (
    SSLContextRef context,
    SSLReadFunc read,
    SSLWriteFunc write
);
```

#### Parameters

*context*

An SSL session context reference.

*read*

A pointer to your read callback function. See [SSLReadFunc](#) (page 39) for information on defining this function.

*write*

A pointer to your write callback function. See [SSLWriteFunc](#) (page 40) for information on defining this function.

#### Return Value

A result code. See “[Secure Transport Result Codes](#)” (page 47).

#### Discussion

Secure Transport calls your read and write callback functions to perform network I/O. You must define these functions before calling `SSLSetIOFuncs`.

You must call `SSLSetIOFuncs` prior to calling the [SSLHandshake](#) (page 23) function. `SSLSetIOFuncs` cannot be called while a session is active.

#### Availability

Available in Mac OS X v10.2 and later.

#### Related Sample Code

SSLSample

#### Declared In

SecureTransport.h

## SSLSetPeerDomainName

Specifies the fully qualified domain name of the peer.

```
OSStatus SSLSetPeerDomainName (
    SSLContextRef context,
    const char *peerName,
    size_t peerNameLen
);
```

#### Parameters

*context*

An SSL session context reference.

*peerName*

The fully qualified domain name of the peer—for example, `store.apple.com`. The name is in the form of a C string, except that NULL termination is optional.

*peerNameLen*

The number of bytes passed in the `peerName` parameter.

#### Return Value

A result code. See “[Secure Transport Result Codes](#)” (page 47).

#### Discussion

You can use this function to verify the common name field in the peer’s certificate. If you call this function and the common name in the certificate does not match the value you specify in the `peerName` parameter, then handshake fails and returns `errSSLXCertChainInvalid`. Use of this function is optional.

This function can be called only when no session is active.

#### Availability

Available in Mac OS X v10.2 and later.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLSetPeerID**

Specifies data that is sufficient to uniquely identify the peer of the current session.

```
OSStatus SSLSetPeerID (
    SSLContextRef context,
    const void *peerID,
    size_t peerIDLen
);
```

**Parameters***context*

An SSL session context reference.

*peerID*

A pointer to a buffer containing the peer ID data to set.

*peerIDLen*

The length of the peer ID data buffer.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

Secure Transport uses the peer ID to match the peer of an SSL session with the peer of a previous session in order to resume an interrupted session. If the peer IDs match, Secure Transport attempts to resume the session with the same parameters as used in the previous session with the same peer.

The data you provide to this function is treated as an opaque blob by Secure Transport but is compared byte by byte with previous peer ID data values set by the current application. An example of peer ID data is an IP address and port, stored in some caller-private manner. Calling this function is optional but is required if you want the session to be resumable. If you do call this function, you must call it prior to the handshake for the current session.

You can use the [SSLGetPeerID](#) (page 19) function to retrieve the peer ID data for the current session.

**Availability**

Available in Mac OS X v10.2 and later.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLSetProtocolVersion**

Sets the SSL protocol version. This function is deprecated.

```
OSStatus SSLSetProtocolVersion (
    SSLContextRef context,
    SSLProtocol version
);
```

**Parameters***context*

An SSL session context reference.

*version*

The SSL protocol version to negotiate.

**Return Value**A result code. See [“Secure Transport Result Codes”](#) (page 47).**Discussion**Use the [SSLSetProtocolVersionEnabled](#) (page 36) function instead.

This function cannot be called when a session is active.

**Availability**

Available in Mac OS X v10.2.

**Related Sample Code**

SSLSample

**Declared In**

SecureTransport.h

**SSLSetProtocolVersionEnabled**

Sets the allowed SSL protocol versions.

```
OSStatus SSLSetProtocolVersionEnabled (
    SSLContextRef context,
    SSLProtocol protocol,
    Boolean enable
);
```

**Parameters***context*

An SSL session context reference.

*protocol*The SSL protocol version to enable. Pass `kSSLProtocolAll` to enable all protocols.*enable*A Boolean value indicating whether to enable or disable the specified protocol. Specify `true` to enable the protocol.**Return Value**A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

Calling this function is optional. The default is that all supported protocols are enabled. When you call this function, only the specified protocol is affected. Therefore, if you call it once to disable SSL version 2 (for example), the other protocols all remain enabled. You may call this function as many times as you wish to enable and disable specific protocols. You can specify one of the following values for the `protocol` parameter:

- `kSSLProtocol2`
- `kSSLProtocol3`
- `kTLSProtocol1`
- `kSSLProtocolAll`

This function cannot be called when a session is active.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`SecureTransport.h`

**SSLSetRsaBlinding**

Enables or disables RSA blinding.

```
OSStatus SSLSetRsaBlinding (
    SSLContextRef context,
    Boolean blinding
);
```

**Parameters**

*context*

An SSL session context reference.

*blinding*

A Boolean value indicating whether to enable RSA blinding. Pass `true` to enable RSA blinding.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

This function is used only on the server side of a connection.

This feature thwarts a known attack to which RSA keys are vulnerable: It is possible to guess the RSA key by timing how long it takes the server to calculate the response to certain queries. RSA blinding adds a random calculation to each query response, thus making the attack impossible. Enabling RSA blinding is a trade-off between performance and security.

RSA blinding is enabled by default. Use the [SSLGetRsaBlinding](#) (page 21) function to determine the current setting.

**Availability**

Available in Mac OS X v10.3 and later.

**Declared In**

`SecureTransport.h`

## SSLSetTrustedRoots

Augments or replaces the default set of trusted root certificates for this session.

```

OSStatus SSLSetTrustedRoots (
    SSLContextRef context,
    CFArrayRef trustedRoots,
    Boolean replaceExisting
);

```

### Parameters

*context*

An SSL session context reference.

*trustedRoots*

A reference to an array of trusted root certificates of type `SecCertificateRef`.

*replaceExisting*

A Boolean value indicating whether to replace or append the current trusted root certificate set. If this value is `true`, the specified root certificates become the only roots that are trusted during this session. If this value is `false`, the specified root certificates are added to the current set of trusted root certificates.

### Return Value

A result code. See [“Secure Transport Result Codes”](#) (page 47).

### Discussion

Each successive call to this function with the *replaceExisting* parameter set to `false` results in accumulation of additional root certificates. To see the current set of trusted root certificates, call the [SSLGetTrustedRoots](#) (page 52) function.

### Availability

Available in Mac OS X v10.3 and later.

### Declared In

`SecureTransport.h`

## SSLWrite

Performs a normal application-level write operation.

```

OSStatus SSLWrite (
    SSLContextRef context,
    const void *data,
    size_t dataLength,
    size_t *processed
);

```

### Parameters

*context*

An SSL session context reference.

*data*

A pointer to the buffer of data to write.

*dataLength*

The amount, in bytes, of data to write.

*processed*

On return, the length, in bytes, of the data actually written.

#### Return Value

A result code. See “[Secure Transport Result Codes](#)” (page 47).

#### Discussion

The `SSLWrite` function might call the `SSLWriteFunc` (page 40) function that you provide (see `SSLSetIOFuncs` (page 33)). Because you may configure the underlying connection to operate in a no-blocking manner, a write operation might return `errSSLWouldBlock`, indicating that less data than requested was actually transferred. In this case, you should repeat the call to `SSLWrite` until some other result is returned.

#### Availability

Available in Mac OS X v10.2 and later.

#### Related Sample Code

SSLSample

#### Declared In

SecureTransport.h

## Callbacks

### SSLReadFunc

Defines a pointer to a customized read function that Secure Transport calls to read data from the connection.

```
typedef OSStatus (*SSLReadFunc) (
    SSLConnectionRef connection,
    void *data,
    size_t *dataLength
);
```

You would declare your callback function like this if you were to name it `MySSLReadFunction`:

```
OSStatus MySSLReadFunction (
    SSLConnectionRef connection,
    void *data,
    size_t *dataLength
);
```

#### Parameters

*connection*

A connection reference.

*data*

On return, points to the data read from the connection. You must allocate this memory before calling this function.

*dataLength*

On input, a pointer to an integer representing the length of the data in bytes. On return, points to the number of bytes actually transferred.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

Before using the Secure Transport API, you must write the functions `SSLReadFunc` and `SSLWriteFunc` (page 40) and provide them to the library by calling the `SSLSetIOFuncs` (page 33) function.

You may configure the underlying connection to operate in a nonblocking manner; in that case, a read operation may well return `errSSLWouldBlock`, indicating less data than requested was transferred and nothing is wrong except that the requested I/O hasn't completed. This result is returned to the caller from the functions `SSLRead` (page 25), `SSLWrite` (page 38), or `SSLHandshake` (page 23).

**Availability**

Available in Mac OS X v10.2 and later.

**Declared In**

`SecureTransport.h`

**SSLWriteFunc**

Defines a pointer to a customized write function that Secure Transport calls to write data to the connection.

```
typedef OSStatus (*SSLWriteFunc) (
    SSLConnectionRef connection,
    const void *data,
    size_t *dataLength
);
```

You would declare your callback function like this if you were to name it `MySSLWriteFunction`:

```
OSStatus MySSLWriteFunction (
    SSLConnectionRef connection,
    void *data,
    size_t *dataLength
);
```

**Parameters**

*connection*

The SSL session connection reference.

*data*

A pointer to the data to write to the connection. You must allocate this memory before calling this function.

*dataLength*

Before calling, an integer representing the length of the data in bytes. On return, this is the number of bytes actually transferred.

**Return Value**

A result code. See [“Secure Transport Result Codes”](#) (page 47).

**Discussion**

Before using the Secure Transport API, you must write the functions `SSLReadFunc` (page 39) and `SSLWriteFunc` and provide them to the library by calling the `SSLSetIOFuncs` (page 33) function.

You may configure the underlying connection to operate in a nonblocking manner. In that case, a write operation may well return `errSSLWouldBlock`, indicating less data than requested was transferred and nothing is wrong except that the requested I/O hasn't completed. This result is returned to the caller from the functions [SSLRead](#) (page 25), [SSLWrite](#) (page 38), or [SSLHandshake](#) (page 23).

**Availability**

Available in Mac OS X v10.2 and later.

**Declared In**

`SecureTransport.h`

## Data Types

**SSLConnectionRef**

Represents a pointer to an opaque I/O connection object.

```
typedef const void *SSLConnectionRef;
```

**Discussion**

The I/O connection object refers to data that identifies a connection. The connection data is opaque to Secure Transport; you can set it to any value that your application can use in the callback functions [SSLReadFunc](#) (page 39) and [SSLWriteFunc](#) (page 40) to uniquely identify the connection, such as a socket or endpoint. Use the [SSLSetConnection](#) (page 29) function to assign a value to the connection object.

**Availability**

Available in Mac OS X v10.2 and later.

**Declared In**

`SecureTransport.h`

**SSLContextRef**

Represents a pointer to an opaque SSL session context object.

```
struct SSLContext;
typedef struct SSLContext *SSLContextRef;
```

**Discussion**

The SSL session context object references the state associated with a session. You cannot reuse an SSL session context in multiple sessions.

**Availability**

Available in Mac OS X v10.2 and later.

**Declared In**

`SecureTransport.h`

## Constants

### SSL Authentication Constants

Represents the requirements for client-side authentication.

```
typedef enum {
    kNeverAuthenticate,
    kAlwaysAuthenticate,
    kTryAuthenticate
} SSLAuthenticate;
```

#### Constants

`kNeverAuthenticate`

Indicates that client-side authentication is not required. (Default.)

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kAlwaysAuthenticate`

Indicates that client-side authentication is required.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kTryAuthenticate`

Indicates that client-side authentication should be attempted. There is no error if the client doesn't have a certificate.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

### SSL Cipher Suite Constants

Represents the cipher suites available.

```

typedef UInt32 SSLCipherSuite;
enum
{
    SSL_NULL_WITH_NULL_NULL = 0x0000,
    SSL_RSA_WITH_NULL_MD5 = 0x0001,
    SSL_RSA_WITH_NULL_SHA = 0x0002,
    SSL_RSA_EXPORT_WITH_RC4_40_MD5 = 0x0003,
    SSL_RSA_WITH_RC4_128_MD5 = 0x0004,
    SSL_RSA_WITH_RC4_128_SHA = 0x0005,
    SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 = 0x0006,
    SSL_RSA_WITH_IDEA_CBC_SHA = 0x0007,
    SSL_RSA_EXPORT_WITH_DES40_CBC_SHA = 0x0008,
    SSL_RSA_WITH_DES_CBC_SHA = 0x0009,
    SSL_RSA_WITH_3DES_EDE_CBC_SHA = 0x000A,
    SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA = 0x000B,
    SSL_DH_DSS_WITH_DES_CBC_SHA = 0x000C,
    SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA = 0x000D,
    SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA = 0x000E,
    SSL_DH_RSA_WITH_DES_CBC_SHA = 0x000F,
    SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA = 0x0010,
    SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA = 0x0011,
    SSL_DHE_DSS_WITH_DES_CBC_SHA = 0x0012,
    SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA = 0x0013,
    SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA = 0x0014,
    SSL_DHE_RSA_WITH_DES_CBC_SHA = 0x0015,
    SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA = 0x0016,
    SSL_DH_anon_EXPORT_WITH_RC4_40_MD5 = 0x0017,
    SSL_DH_anon_WITH_RC4_128_MD5 = 0x0018,
    SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA = 0x0019,
    SSL_DH_anon_WITH_DES_CBC_SHA = 0x001A,
    SSL_DH_anon_WITH_3DES_EDE_CBC_SHA = 0x001B,
    SSL_FORTEZZA_DMS_WITH_NULL_SHA = 0x001C,
    SSL_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA = 0x001D,
    SSL_RSA_WITH_RC2_CBC_MD5 = 0xFF80,
    SSL_RSA_WITH_IDEA_CBC_MD5 = 0xFF81,
    SSL_RSA_WITH_DES_CBC_MD5 = 0xFF82,
    SSL_RSA_WITH_3DES_EDE_CBC_MD5 = 0xFF83,
    SSL_NO_SUCH_CIPHERSUITE = 0xFFFF
};

```

**Constants**

- SSL\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5**  
 Session key size conforms to pre-1998 US export restrictions.  
 Available in Mac OS X v10.2 and later.  
 Declared in `CipherSuite.h`.
- SSL\_RSA\_EXPORT\_WITH\_RC2\_CBC\_40\_MD5**  
 Session key size conforms to pre-1998 US export restrictions.  
 Available in Mac OS X v10.2 and later.  
 Declared in `CipherSuite.h`.
- SSL\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA**  
 Session key size conforms to pre-1998 US export restrictions.  
 Available in Mac OS X v10.2 and later.  
 Declared in `CipherSuite.h`.

- `SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA`  
Session key size conforms to pre-1998 US export restrictions.  
Available in Mac OS X v10.2 and later.  
Declared in `CipherSuite.h`.
- `SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`  
Session key size conforms to pre-1998 US export restrictions.  
Available in Mac OS X v10.2 and later.  
Declared in `CipherSuite.h`.
- `SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`  
Session key size conforms to pre-1998 US export restrictions.  
Available in Mac OS X v10.2 and later.  
Declared in `CipherSuite.h`.
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`  
Session key size conforms to pre-1998 US export restrictions.  
Available in Mac OS X v10.2 and later.  
Declared in `CipherSuite.h`.
- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`  
Session key size conforms to pre-1998 US export restrictions.  
Available in Mac OS X v10.2 and later.  
Declared in `CipherSuite.h`.
- `SSL_RSA_WITH_RC2_CBC_MD5`  
This value can be specified for SSL 2 but not SSL 3.  
Available in Mac OS X v10.2 and later.  
Declared in `CipherSuite.h`.
- `SSL_RSA_WITH_IDEA_CBC_MD5`  
This value can be specified for SSL 2 but not SSL 3.  
Available in Mac OS X v10.2 and later.  
Declared in `CipherSuite.h`.
- `SSL_RSA_WITH_DES_CBC_MD5`  
This value can be specified for SSL 2 but not SSL 3.  
Available in Mac OS X v10.2 and later.  
Declared in `CipherSuite.h`.
- `SSL_RSA_WITH_3DES_EDE_CBC_MD5`  
This value can be specified for SSL 2 but not SSL 3.  
Available in Mac OS X v10.2 and later.  
Declared in `CipherSuite.h`.

## SSL Client Certificate State Constants

Represents the status of client certificate exchange.

```
typedef enum {  
    kSSLClientCertNone,  
    kSSLClientCertRequested,  
    kSSLClientCertSent,  
    kSSLClientCertRejected  
} SSLClientCertificateState;
```

### Constants

kSSLClientCertNone

Indicates that the server hasn't asked for a certificate and that the client hasn't sent one.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

kSSLClientCertRequested

Indicates that the server has asked for a certificate, but the client has not sent it.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

kSSLClientCertSent

Indicates that the server asked for a certificate, the client sent one, and the server validated it. The application can inspect the certificate using the function [SSLGetPeerCertificates](#) (page 51).

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

kSSLClientCertRejected

Indicates that the client sent a certificate but the certificate failed validation. This value is seen only on the server side. The server application can inspect the certificate using the function [SSLGetPeerCertificates](#) (page 51).

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

## SSL Protocol Constants

Represents the SSL protocol version.

```
typedef enum {  
    kSSLProtocolUnknown,  
    kSSLProtocol2,  
    kSSLProtocol3,  
    kSSLProtocol30nly,  
    kTLSProtocol1,  
    kTLSProtocol10nly,  
    kSSLProtocolAll  
} SSLProtocol;
```

### Constants

kSSLProtocolUnknown

Specifies that no protocol has been or should be negotiated or specified; use default.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kSSLProtocol2`

Specifies that only the SSL 2.0 protocol may be negotiated.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kSSLProtocol3`

Specifies that the SSL 3.0 protocol is preferred; the SSL 2.0 protocol may be negotiated if the peer cannot use the SSL 3.0 protocol.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kSSLProtocol30nly`

Specifies that only the SSL 3.0 protocol may be negotiated; fails if the peer tries to negotiate the SSL 2.0 protocol.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kTLSProtocol1`

Specifies that the TLS 1.0 protocol is preferred but lower versions may be negotiated.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kTLSProtocol10nly`

Specifies that only the TLS 1.0 protocol may be negotiated.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kSSLProtocolAll`

Specifies all supported versions.

Available in Mac OS X v10.3 and later.

Declared in `SecureTransport.h`.

### Discussion

The descriptions given here apply to the functions [SSLSetProtocolVersion](#) (page 35) and [SSLGetProtocolVersion](#) (page 20). For the functions [SSLSetProtocolVersionEnabled](#) (page 36) and [SSLGetProtocolVersionEnabled](#) (page 21), only the following values are used. For these functions, each constant except `kSSLProtocolAll` specifies a single protocol version.

- `kSSLProtocol2`
- `kSSLProtocol3`
- `kTLSProtocol1`
- `kSSLProtocolAll`

## SSL Session State Constants

Represents the state of an SSL session.

```
typedef enum {
    kSSLIdle,
    kSSLHandshake,
    kSSLConnected,
    kSSLClosed,
    kSSLAborted
} SSLSessionState;
```

**Constants**

kSSLIdle

No I/O has been performed yet.

Available in Mac OS X v10.2 and later.

Declared in SecureTransport.h.

kSSLHandshake

The SSL handshake is in progress.

Available in Mac OS X v10.2 and later.

Declared in SecureTransport.h.

kSSLConnected

The SSL handshake is complete; the connection is ready for normal I/O.

Available in Mac OS X v10.2 and later.

Declared in SecureTransport.h.

kSSLClosed

The connection closed normally.

Available in Mac OS X v10.2 and later.

Declared in SecureTransport.h.

kSSLAborted

The connection aborted.

Available in Mac OS X v10.2 and later.

Declared in SecureTransport.h.

## Result Codes

The most common result codes returned by Secure Transport functions are listed in the table below.

Errors in the range of -9819 through -9840 are fatal errors that are detected by the peer.

Result Code	Value	Description
errSSLProtocol	-9800	SSL protocol error. Available in Mac OS X v10.2 and later.
errSSLNegotiation	-9801	The cipher suite negotiation failed. Available in Mac OS X v10.2 and later.
errSSLFatalAlert	-9802	A fatal alert was encountered. Available in Mac OS X v10.2 and later.

Result Code	Value	Description
errSSLWouldBlock	-9803	Function is blocked; waiting for I/O. This is not fatal. Available in Mac OS X v10.2 and later.
errSSLSessionNotFound	-9804	An attempt to restore an unknown session failed. Available in Mac OS X v10.2 and later.
errSSLClosedGraceful	-9805	The connection closed gracefully. Available in Mac OS X v10.2 and later.
errSSLClosedAbort	-9806	The connection closed due to an error. Available in Mac OS X v10.2 and later.
errSSLXCertChainInvalid	-9807	Invalid certificate chain. Available in Mac OS X v10.2 and later.
errSSLBadCert	-9808	Bad certificate format. Available in Mac OS X v10.2 and later.
errSSLCrypto	-9809	An underlying cryptographic error was encountered. Available in Mac OS X v10.2 and later.
errSSLInternal	-9810	Internal error. Available in Mac OS X v10.2 and later.
errSSLModuleAttach	-9811	Module attach failure. Available in Mac OS X v10.2 and later.
errSSLUnknownRootCert	-9812	Certificate chain is valid, but root is not trusted. Available in Mac OS X v10.2 and later.
errSSLNoRootCert	-9813	No root certificate for the certificate chain. Available in Mac OS X v10.2 and later.
errSSLCertExpired	-9814	The certificate chain had an expired certificate. Available in Mac OS X v10.2 and later.
errSSLCertNotYetValid	-9815	The certificate chain had a certificate that is not yet valid. Available in Mac OS X v10.2 and later.
errSSLClosedNoNotify	-9816	The server closed the session with no notification. Available in Mac OS X v10.2 and later.
errSSLBufferOverflow	-9817	An insufficient buffer was provided. Available in Mac OS X v10.2 and later.

Result Code	Value	Description
errSSLBADCipherSuite	-9818	A bad SSL cipher suite was encountered. Available in Mac OS X v10.2 and later.
errSSLPeerUnexpectedMsg	-9819	An unexpected message was received. Available in Mac OS X v10.3 and later.
errSSLPeerBadRecordMac	-9820	A bad record MAC was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerDecryptionFail	-9821	Decryption failed. Available in Mac OS X v10.3 and later.
errSSLPeerRecordOverflow	-9822	A record overflow occurred. Available in Mac OS X v10.3 and later.
errSSLPeerDecompressFail	-9823	Decompression failed. Available in Mac OS X v10.3 and later.
errSSLPeerHandshakeFail	-9824	The handshake failed. Available in Mac OS X v10.3 and later.
errSSLPeerBadCert	-9825	A bad certificate was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerUnsupportedCert	-9826	An unsupported certificate format was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerCertRevoked	-9827	The certificate was revoked. Available in Mac OS X v10.3 and later.
errSSLPeerCertExpired	-9828	The certificate expired. Available in Mac OS X v10.3 and later.
errSSLPeerCertUnknown	-9829	The certificate is unknown. Available in Mac OS X v10.3 and later.
errSSLIllegalParam	-9830	An illegal parameter was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerUnknownCA	-9831	An unknown certificate authority was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerAccessDenied	-9832	Access was denied. Available in Mac OS X v10.3 and later.

Result Code	Value	Description
errSSLPeerDecodeError	-9833	A decoding error occurred. Available in Mac OS X v10.3 and later.
errSSLPeerDecryptError	-9834	A decryption error occurred. Available in Mac OS X v10.3 and later.
errSSLPeerExportRestriction	-9835	An export restriction occurred. Available in Mac OS X v10.3 and later.
errSSLPeerProtocolVersion	-9836	A bad protocol version was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerInsufficientSecurity	-9837	There is insufficient security for this operation. Available in Mac OS X v10.3 and later.
errSSLPeerInternalError	-9838	An internal error occurred. Available in Mac OS X v10.3 and later.
errSSLPeerUserCancelled	-9839	The user canceled the operation. Available in Mac OS X v10.3 and later.
errSSLPeerNoRenegotiation	-9840	No renegotiation is allowed. Available in Mac OS X v10.3 and later.
errSSLConnectionRefused	-9844	The peer dropped the connection before responding. Available in Mac OS X v10.4 and later.
errSSLDecryptionFail	-9845	Decryption failed. Available in Mac OS X v10.3 and later.
errSSLBadRecordMac	-9846	A bad record MAC was encountered. Available in Mac OS X v10.3 and later.
errSSLRecordOverflow	-9847	A record overflow occurred. Available in Mac OS X v10.3 and later.
errSSLBadConfiguration	-9848	A configuration error occurred. Available in Mac OS X v10.3 and later.

# Deprecated Secure Transport Functions

---

A function identified as deprecated has been superseded and may become unsupported in the future.

## Deprecated in Mac OS X v10.5

### SSLGetPeerCertificates

Retrieves a peer certificate. (Deprecated in Mac OS X v10.5.)

```
OSStatus SSLGetPeerCertificates (
    SSLContextRef context,
    CFArrayRef *certs
);
```

#### Parameters

*context*

An SSL session context reference.

*certs*

On return, a pointer to an array of values of type `SecCertificateRef` representing the peer certificate and the certificate chain used to validate it. The certificate at index 0 of the returned array is the peer certificate; the root certificate (or the closest certificate to it) is at the end of the returned array. The entire array is created by the Secure Transport library; you must release it when you are finished with it.

#### Return Value

A result code. See “[Secure Transport Result Codes](#)” (page 47).

#### Discussion

This function is valid any time after a handshake attempt. You can use it to examine a peer certificate, to examine a certificate chain to determine why a handshake attempt failed, or to retrieve the certificate chain in order to validate the certificate yourself (see [SSLSetEnableCertVerify](#) (page 31)).

#### Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.5.

#### Related Sample Code

SSLSample

#### Declared In

SecureTransport.h

## SSLGetTrustedRoots

Retrieves the current list of trusted root certificates. (Deprecated in Mac OS X v10.5.)

```
OSStatus SSLGetTrustedRoots (  
    SSLContextRef context,  
    CFArrayRef *trustedRoots  
);
```

### Parameters

*context*

An SSL session context reference.

*trustedRoots*

On return, a pointer to a value of type `CFArrayRef`. This array contains values of type `SecCertificateRef` representing the current set of trusted roots.

### Return Value

A result code. See “[Secure Transport Result Codes](#)” (page 47).

### Discussion

You can use the [SSLSetTrustedRoots](#) (page 38) function to replace or add to the set of trusted root certificates. If [SSLSetTrustedRoots](#) (page 38) has never been called for this session, the `SSLGetTrustedRoots` function returns the system’s default set of trusted root certificates.

### Availability

Available in Mac OS X v10.3 and later.

Deprecated in Mac OS X v10.5.

### Declared In

`SecureTransport.h`

# Document Revision History

---

This table describes the changes to *Secure Transport Reference*.

Date	Notes
2004-08-31	Clarified and expanded some function descriptions.
2003-05-15	New document that provides a reference to the Secure Transport API, which implements the SSL and TLS protocols for secure communication over a network.

## REVISION HISTORY

### Document Revision History

# Index

---

## E

---

errSSLBadCert **constant** 48  
errSSLBadCipherSuite **constant** 49  
errSSLBadConfiguration **constant** 50  
errSSLBadRecordMac **constant** 50  
errSSLBufferOverflow **constant** 48  
errSSLCertExpired **constant** 48  
errSSLCertNotYetValid **constant** 48  
errSSLClosedAbort **constant** 48  
errSSLClosedGraceful **constant** 48  
errSSLClosedNoNotify **constant** 48  
errSSLConnectionRefused **constant** 50  
errSSLCrypto **constant** 48  
errSSLDecryptionFail **constant** 50  
errSSLFatalAlert **constant** 47  
errSSLIllegalParam **constant** 49  
errSSLInternal **constant** 48  
errSSLModuleAttach **constant** 48  
errSSLNegotiation **constant** 47  
errSSLNoRootCert **constant** 48  
errSSLPeerAccessDenied **constant** 49  
errSSLPeerBadCert **constant** 49  
errSSLPeerBadRecordMac **constant** 49  
errSSLPeerCertExpired **constant** 49  
errSSLPeerCertRevoked **constant** 49  
errSSLPeerCertUnknown **constant** 49  
errSSLPeerDecodeError **constant** 50  
errSSLPeerDecompressFail **constant** 49  
errSSLPeerDecryptError **constant** 50  
errSSLPeerDecryptionFail **constant** 49  
errSSLPeerExportRestriction **constant** 50  
errSSLPeerHandshakeFail **constant** 49  
errSSLPeerInsufficientSecurity **constant** 50  
errSSLPeerInternalError **constant** 50  
errSSLPeerNoRenegotiation **constant** 50  
errSSLPeerProtocolVersion **constant** 50  
errSSLPeerRecordOverflow **constant** 49  
errSSLPeerUnexpectedMsg **constant** 49  
errSSLPeerUnknownCA **constant** 49  
errSSLPeerUnsupportedCert **constant** 49

errSSLPeerUserCancelled **constant** 50  
errSSLProtocol **constant** 47  
errSSLRecordOverflow **constant** 50  
errSSLSessionNotFound **constant** 48  
errSSLUnknownRootCert **constant** 48  
errSSLWouldBlock **constant** 48  
errSSLXCertChainInvalid **constant** 48

## K

---

kAlwaysAuthenticate **constant** 42  
kNeverAuthenticate **constant** 42  
kSSLAborted **constant** 47  
kSSLClientCertNone **constant** 45  
kSSLClientCertRejected **constant** 45  
kSSLClientCertRequested **constant** 45  
kSSLClientCertSent **constant** 45  
kSSLClosed **constant** 47  
kSSLConnected **constant** 47  
kSSLHandshake **constant** 47  
kSSLIdle **constant** 47  
kSSLProtocol2 **constant** 46  
kSSLProtocol3 **constant** 46  
kSSLProtocol3Only **constant** 46  
kSSLProtocolAll **constant** 46  
kSSLProtocolUnknown **constant** 45  
kTLSProtocol1 **constant** 46  
kTLSProtocol1Only **constant** 46  
kTryAuthenticate **constant** 42

## S

---

SSL Authentication Constants 42  
SSL Cipher Suite Constants 42  
SSL Client Certificate State Constants 44  
SSL Protocol Constants 45  
SSL Session State Constants 46  
SSLAddDistinguishedName **function** 9  
SSLClose **function** 10

- SSLConnectionRef **data type** 41
- SSLContextRef **structure** 41
- SSLDisposeContext **function** 10
- SSLGetAllowsAnyRoot **function** 11
- SSLGetAllowsExpiredCerts **function** 11
- SSLGetAllowsExpiredRoots **function** 12
- SSLGetBufferedReadSize **function** 12
- SSLGetClientCertificateState **function** 13
- SSLGetConnection **function** 13
- SSLGetDiffieHellmanParams **function** 14
- SSLGetEnableCertVerify **function** 15
- SSLGetEnabledCiphers **function** 15
- SSLGetNegotiatedCipher **function** 16
- SSLGetNegotiatedProtocolVersion **function** 16
- SSLGetNumberEnabledCiphers **function** 17
- SSLGetNumberSupportedCiphers **function** 18
- SSLGetPeerCertificates **function** (Deprecated in Mac OS X v10.5) 51
- SSLGetPeerDomainName **function** 18
- SSLGetPeerDomainNameLength **function** 19
- SSLGetPeerID **function** 19
- SSLGetProtocolVersion **function** 20
- SSLGetProtocolVersionEnabled **function** 21
- SSLGetRsaBlinding **function** 21
- SSLGetSessionState **function** 22
- SSLGetSupportedCiphers **function** 22
- SSLGetTrustedRoots **function** (Deprecated in Mac OS X v10.5) 52
- SSLHandshake **function** 23
- SSLNewContext **function** 24
- SSLRead **function** 25
- SSLReadFunc **callback** 39
- SSLSetAllowsAnyRoot **function** 25
- SSLSetAllowsExpiredCerts **function** 26
- SSLSetAllowsExpiredRoots **function** 27
- SSLSetCertificate **function** 28
- SSLSetClientSideAuthenticate **function** 29
- SSLSetConnection **function** 29
- SSLSetDiffieHellmanParams **function** 30
- SSLSetEnableCertVerify **function** 31
- SSLSetEnabledCiphers **function** 32
- SSLSetEncryptionCertificate **function** 32
- SSLSetIOFuncs **function** 33
- SSLSetPeerDomainName **function** 34
- SSLSetPeerID **function** 35
- SSLSetProtocolVersion **function** 35
- SSLSetProtocolVersionEnabled **function** 36
- SSLSetRsaBlinding **function** 37
- SSLSetTrustedRoots **function** 38
- SSLWrite **function** 38
- SSLWriteFunc **callback** 40
- SSL\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA **constant** 44
- SSL\_DHE\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA **constant** 44
- SSL\_DH\_anon\_EXPORT\_WITH\_DES40\_CBC\_SHA **constant** 44
- SSL\_DH\_anon\_EXPORT\_WITH\_RC4\_40\_MD5 **constant** 44
- SSL\_DH\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA **constant** 44
- SSL\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA **constant** 43
- SSL\_RSA\_EXPORT\_WITH\_RC2\_CBC\_40\_MD5 **constant** 43
- SSL\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5 **constant** 43
- SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_MD5 **constant** 44
- SSL\_RSA\_WITH\_DES\_CBC\_MD5 **constant** 44
- SSL\_RSA\_WITH\_IDEA\_CBC\_MD5 **constant** 44
- SSL\_RSA\_WITH\_RC2\_CBC\_MD5 **constant** 44