

# SSL Plus

Version 4.5.1

## User's Guide

PUB-0300-0210  
November 26, 2003

© Certicom Corp. 2000-2003. All rights reserved.

Certicom, the Certicom logo, SSL Plus and Security Builder are trademarks or registered trademarks of Certicom Corp. All other trademarks or registered trademarks are property of their respective owners. This product is covered by one or more of the following U.S. Patents: US 6,195,433, 6,178,507, 6,141,420, 6,134,325, 6,122,736, 6,097,813, 6,078,667, 6,049,815, 5,999,626, 5,955,717, 5,933,504, 5,896,455, 5,889,865, 5,787,028, 5,761,305, 5,600,725, 4,745,568.

Other applications and corresponding foreign protection pending.

Certicom Corp.  
5520 Explorer Drive,  
4th Floor,  
Mississauga, Ontario,  
Canada, L4W 5L1  
905.507.4220



The *User's Guide* describes how to install SSL Plus and use the API.

Other documents provided with SSL Plus include the Programmer's Reference and the Update Guide. The Programmer's Reference document describes all the functions in the SSL Plus API. The Upgrade Guide describes the differences between this version of SSL Plus and SSL Plus (v4).

#### Copyright Notice

© Certicom Corp. 2000, 2001, 2002,2003. All rights reserved. This documentation contains Certicom's proprietary information and any use and distribution are limited to authorized licensees of Certicom. Any unauthorized use, reproduction, and distribution of this documentation is strictly prohibited by law.

#### Technical Support

You can reach Certicom's technical support department by telephone at 1-800-511-8011, by fax at 1-800-474-3877, or by email at [support@certicom.com](mailto:support@certicom.com).

# Table of Contents

---

<b>Introducing SSL Plus</b> .....	1
Welcome to SSL Plus: A Time-Saving Tool. ....	1
Installing SSL Plus .....	2
Distribution Types .....	2
<b>SSL Plus Components</b> .....	3
SSL Plus Architecture .....	3
Cryptographic Engine .....	3
SSL Plus API .....	3
Features of the SSL Protocol .....	4
Handshakes and Cipher Suites .....	4
Session Resumption .....	4
Certificates and Authentication .....	5
<b>Developing an SSL Plus Application</b> .....	7
Development Concepts .....	7
SSL Global Context Structure .....	7
SSL Connection Context Structure .....	7
Installable Objects .....	7
Callbacks and SSL Plus .....	8
SSL Plus RAD API .....	9
Cache Memory Manager .....	10
Developing an Application: Step-by-Step .....	11
Create the global context .....	11
Provide support for an RNG .....	12
Initialize the Cache Memory Manager .....	12
Configure the mandatory setting in the global context .....	12
Configure the optional settings in the global context .....	14
Client Authentication .....	15
Register the optional callbacks .....	16
Create the CMM (optional) .....	17
Create a connection context .....	17
Perform the handshake .....	18
Retrieve negotiation information (optional) .....	18
Transfer data .....	18
Close the connection .....	19
Get the CMM reference from the connection context .....	19
Destroy the connection context .....	19
Free the CMM resources in the connection context .....	20
Free the CMM resources in the global context .....	20
Destroy the global context .....	20
Restrictions on the use of the SSL Plus API .....	21
Managing Memory Allocation .....	25
Other Development Issues .....	26
Generating RSA export key pairs .....	26
Reducing memory overhead in idle connections .....	26

---

Renegotiating a connection with a peer . . . . .	26
<b>EAP Protocol . . . . .</b>	<b>27</b>
Support for the EAP Protocol . . . . .	27
EAP-TLS . . . . .	27
EAP-TTLS . . . . .	29
PEAP . . . . .	30
SCTP . . . . .	32
<b>Building and Running the Sample Applications . . . . .</b>	<b>35</b>
Building SSLSampleClient and SSLSampleServer . . . . .	35
Running SSLSampleClient and SSLSampleServer . . . . .	35
The Final Word: Know Your Makefiles! . . . . .	36
<b>Enabling WAP 2.0 Clients and Servers with SSL Plus . . . . .</b>	<b>37</b>
SSL Plus Global Context Functions . . . . .	37
SSL Plus Certificate Format Objects . . . . .	37
SSL Plus Protocol Objects . . . . .	38
SSL Plus Policy Objects . . . . .	38
SSL Plus Authentication Mode Objects . . . . .	38
SSL Plus Cipher Suite Objects . . . . .	39
SSL Plus Certificate Functions . . . . .	39
WAP TLS Profile and Tunneling . . . . .	42
TLS Profile . . . . .	42
Cipher Suites . . . . .	42
Session . . . . .	42
Server Authentication . . . . .	42
Client Authentication . . . . .	43
Certificate Chain Depth . . . . .	43
CA Practice Recommendation . . . . .	43
TLS Tunneling . . . . .	43
WAP Certificates and CRL Profiles . . . . .	44
General . . . . .	44
User Certificates for Authentication . . . . .	44
User Certificates for Digital Signatures . . . . .	45
X.509-Compliant Server Certificates . . . . .	45
Role Certificates . . . . .	46
Authority Certificates . . . . .	46
Other Certificates . . . . .	47
CRL Profiles . . . . .	47
References . . . . .	48
<b>Enabling MIDP 2.0 Clients and Servers with SSL Plus . . . . .</b>	<b>49</b>
SSL Plus Policy Objects . . . . .	49
MIDP X.509 Certificate Profile . . . . .	50
MIDP X.509 Certificate Profile . . . . .	50
Certificate Extensions . . . . .	50
Certificate Size . . . . .	50
Algorithm Support . . . . .	50
Certificate Processing for HTTPS . . . . .	50
MIDP X.509 Certificate Profile for Trusted MIDlet Suites . . . . .	51

---

Certificate Processing for OTA. . . . .	51
Certificate Expiration and Revocation . . . . .	51
References . . . . .	52
<b>Appendix D: SSL: A History . . . . .</b>	<b>53</b>
What is SSL?. . . . .	53
What is TLS? . . . . .	53
How Does SSL/TLS Work? . . . . .	54
<b>Appendix E: Further Reading . . . . .</b>	<b>55</b>
On SSL/TLS . . . . .	55
On Data Security and Encryption . . . . .	55
On Specific Algorithms. . . . .	55
On Standards and Other Protocols . . . . .	56
<b>Appendix F: Glossary of Terms . . . . .</b>	<b>57</b>
Common Encryption and SSL Related Terms. . . . .	57



# 1

## Introducing SSL Plus

---

### Welcome to SSL Plus: A Time-Saving Tool

SSL Plus is a simple and effective implementation of the Secure Sockets Layer (SSL) protocol that can be used to quickly inject security into a network application. Each configuration of SSL Plus contains a library of API functions. Each function has a clean and intuitive interface that will enable your application to perform the fundamentals of cryptography without bogging you down with minute details. With SSL Plus, you do not require an in-depth understanding of the SSL protocol and its related standards or cryptography. There is no need to consult the volumes of white papers and on-line references concerning SSL. By using SSL Plus, you can be sure that your application will be fully compliant with the SSL protocol. If rapid development is your objective then SSL Plus is the right choice.

SSL Plus is more than a simple and fast SSL development tool. Your application may be quite complex and may require more attention to cryptographic or protocol detail. SSL Plus is highly flexible in terms of how much control a developer wishes to exercise over the cryptographic process. Whether developing a simple SSL client or a complex system of applications, SSL Plus is the only SSL development tool you will need.

If you develop for constrained platforms, you need not look any further for your SSL solutions. Regardless of the physical constraints, complexity, or time limitations of your current SSL project, Certicom's SSL Plus and the SSL Plus family of products are the right choice. Congratulations!

# Installing SSL Plus

The SSL Plus distribution is contained in a **zip** file. In order to use SSL Plus you will have to extract the files. We suggest that you create a directory for your SSL Plus distribution (for example **sslplus4**) where you can copy the files. When the files are unzipped you will find the following sub-directories:

- **components/**  
contains component header files. These files contain the type definitions for the basic data types and functions used by SSL Plus.
- **include/**  
contains the header files.
- **lib/**  
contains the library files for this platform.
- **src/rad**  
contains the RAD (Rapid Application Development) source files. The RAD API speeds up development by providing your application with a default configuration. See “Developing an SSL Plus Application” on page 7.
- **samples/**  
contains sample code that demonstrates how to use SSL Plus.
- **src/**  
contains the source code for SSL Plus (source distributions only).
- **userdocs/**  
contains the documentation files.

The **components** library (libshared.a) is in **components/lib/<platform>**.

Platforms which support WAP 2.0 also have a **tools/** directory containing the Trustpoint toolkit.

In addition, your distribution contains a **README** file. This file includes the most up-to-date information on installation, configuration and technical support. Please refer to this document before using SSL Plus.

## Distribution Types

SSL Plus is available in two configurations: **binary** and **source**. The binary configuration contains all the SSL Plus library and header files for your platform. The source configuration, in addition, contains all the source code for the product. Both configurations include Security Builder.

If you have the source distribution you may wish to make changes to the library code. If so, be aware that the header files found in the **include** directory are duplicated in the **src** directory. Should you make changes, be sure to make them to both files. Failure to do so could cause compilation or linking errors when you try to build your application.



# 2

## SSL Plus Components

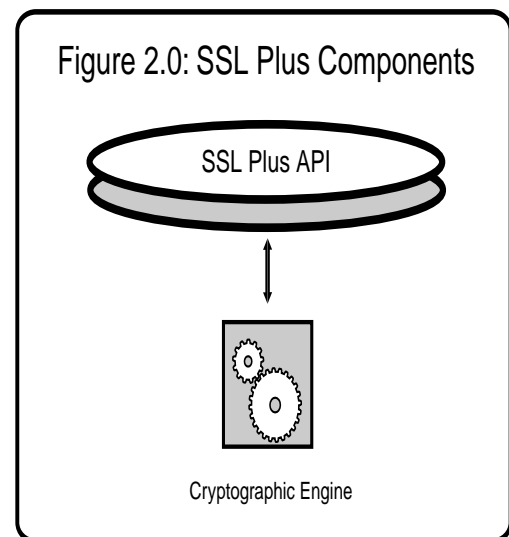
### SSL Plus Architecture

SSL Plus consists of two components: the SSL Plus API and the cryptographic engine used to perform the encryption operations used in the SSL protocol. See Figure 2.0.

#### Cryptographic Engine

A cryptographic engine is a module that allows SSL Plus to carry out tasks such as key generation and encryption. Your SSL Plus distribution already includes Certicom's Security Builder cryptographic engine. Security Builder can provide all the cryptographic features your application needs.

SSL Plus also supports the Cryptoswift RSA hardware accelerator (on the Win32, Linux-x86 and Solaris platforms).



#### SSL Plus API

The SSL Plus API is the interface you will need to learn about to develop SSL client and SSL server applications. Refer to the **SSL Plus Programmer's Reference** document in your distribution.

# Features of the SSL Protocol

The following sections describe some important features of the SSL protocol and how they are implemented in SSL Plus.

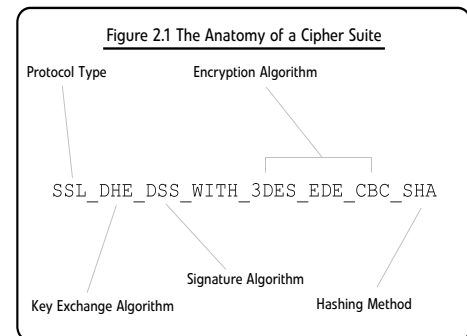
## Handshakes and Cipher Suites

A communication session between two SSL applications consists of two phases: a handshake phase and a data transfer phase. The handshake phase must be successfully concluded before the data transfer phase can begin.

The purpose of the handshake phase is for both applications to agree upon which algorithms they should use for secure data transfer. They do this by exchanging a *cipher suite*. A cipher suite is a value that represents a particular key exchange algorithm, data encryption algorithm and hashing algorithm (see Figure 2.1).

During the handshake the client application sends the server a list of cipher suites. This represents all the combinations of algorithms that it can support.

The first cipher suite in the list is its preferred choice of algorithms. The server then chooses its preferred cipher suite from this list and returns it to the client. Both applications have now agreed on which algorithms to use in the data transfer phase. Once they have generated the keys to use with those algorithms they can begin sending data. This marks the end of the handshake phase.



In SSL Plus, client and server applications can set their cipher suites by calling `ssl_SetCipherSuites()` before calling `ssl_Handshake()`. The library only includes support for those cipher suites that you select. This reduces the code size of your application. Refer to the **SSL Plus Programmer's Reference** for a list of all the supported cipher suites.

**Note 1:** If a server application cannot support any of a client application's cipher suites, it will terminate the handshake and the session will end.

**Note 2:** If your application sends small, repeated sections of plaintext, you should consider using a block cipher instead of the RC4 stream cipher. This is because statistical biases have been found in the data stream that RC4 generates [Mantin and Shamir, "A practical attack on broadcast RC4", Fast Software Encryption, 2001.] This vulnerability may allow an attacker to determine some bytes of a repeated section of plaintext from the ciphertext.

This vulnerability only affects the RC4 stream cipher and has no connection with SSL Plus or the SSL protocol.

## Session Resumption

The handshake phase can be a very lengthy process. This is because both the client and server applications have to use public-key cryptography to exchange a *pre-master secret*. The pre-master secret is used to generate the keys for the session. However, the SSL protocol allows two applications to resume a previous session and re-use its

associated pre-master secret. This feature is known as *session resumption*. The following is a brief description of how session resumption works.

When a client and a server application begin a new session, the server generates a session ID and returns it to the client. This indicates that the server is willing to resume this session. When the session has finished, both client and server store the negotiated algorithms and the pre-master secret. This will enable them to resume the session at a later time. When the client wishes to resume the session, it sends the same session ID to the server. The server then retrieves the negotiated algorithms and pre-master secret for that session. Since there is no need to exchange the pre-master secret, that part of the handshake is skipped. The keys that are generated for the resumed session are unique because the pre-master secret is combined with random numbers that are exchanged during the initial part of the handshake.

In SSL Plus, session resumption is handled internally by the library code. The application is only responsible for providing a database in which to store information about active sessions. Each record in the database contains enough information to resume that session. SSL Plus provides templates for callback functions which it calls periodically to add, remove and retrieve information in the database. These templates are called `ssl_AddSessionFunc`, `ssl_DeleteSessionFunc` and `ssl_GetSessionFunc`. These callbacks can be registered with the SSL Plus library by calling `ssl_SetSessionFuncs()`.

Each session in the database is identified by a session ID. When the client application connects to a server for the first time, it maps the session ID to the server's IP address and port number during the call to `ssl_CreateConnectionContext()`. This allows the library to find the session ID and the corresponding information in the database when the client attempts to connect to the server again. In the server application this mapping is unnecessary because the session ID is sent by the client.

For security reasons, you should consider deleting stored session data on a regular basis. If an attacker discovered the pre-master secret for a session, then they could read all the encrypted data that passed between the client and the server.

## Certificates and Authentication

In most situations a client will want to ensure that sensitive information is being sent to the correct server and not to an imposter. In data security terms, this is known as *server authentication*. Sometimes, the server will want to ensure that sensitive data is being sent to the correct client. This is known as *client authentication*. Server authentication is the most common type of authentication in practice. In SSL, both server and client authentication is achieved by means of electronic certificates. Certificates are documents issued by a CA (certificate authority) that contain information about the certificate holder, including a copy of the certificate holder's public key. When issued, a CA will sign the certificate with their private key. A CA is a trusted third party who makes it their business to provide authentication services to its clientele. Anyone who trusts that authority can trust that the public key belongs to whoever is named in the certificate.

When an SSL client application begins to handshake with an SSL server application, the server will send the client a copy of its certificate. If the client trusts the CA that signed the certificate then it can be certain of the identity of the server. The process of checking the certificate is called *verification*. The client can now negotiate its selected

key exchange and encryption algorithms with the server. If verification fails, then communications between the client and server will cease during the handshake stage.

In order to simplify the issuing of certificates, CAs can designate sub-authorities to issue certificates for them. For example, a CA "CertCo" could sign the certificate of a company, "Spacely Sprockets", which could sign the certificate of an employee, "George". If you receive this certificate, and you trust CertCo to only certify trustworthy sub-authorities, you can accept George's certificate as valid. This state of affairs is known as a certificate chain.

In SSL Plus, your application can interact with the verification process through a callback function called `ssl_CheckCertificateChainFunc`. The library calls this function during the handshake when it receives a certificate from the peer. Normally, a certificate would be rejected if it had expired or it was not signed by a trusted certificate authority. However `ssl_CheckCertificateChainFunc()` can change this behavior and either accept or reject the certificate based on your own criteria. You can set this callback with `ssl_SetCheckCertificateChainFunc()`.

In general, a responsible certificate authority will publish a set of policies, guidelines, and guarantees which should be examined before deciding whether that authority's certificates can be trusted. This will also help you to determine when a certificate should be considered to be valid.

# 3

## Developing an SSL Plus Application

---

### Development Concepts

There are four important concepts you should understand before you begin developing your SSL Plus application: the global context, the connection context, installable objects, certificates and the use of callback functions.

#### SSL Global Context Structure

Every SSL Plus application requires a single *global context structure*. The information stored in the global context structure remains unchanged between connections. This information includes the callback functions, certificate data, level of cryptographic strength, the authentication mode, the list of supported cipher suites and more. A global context is created by calling `ssl_CreateGlobalContext()`. A number of functions are used to configure the global context (see “Developing an Application: Step-by-Step” on page 11).

Before your application terminates you should destroy the global context by calling `ssl_DestroyGlobalContext()`.

#### SSL Connection Context Structure

SSL Plus represents the state of a connection with a remote peer in a *connection context structure*. Each time your application connects to a remote peer, you must create and initialize a connection context structure. Your application does not have to initialize the structure directly; the library provides API functions for this purpose. The library updates the structure during the handshake so your application can find out which algorithms were agreed upon.

In SSL, the terms *connection* and *session* do not mean the same thing. A connection refers to the underlying TCP/IP connection. A session refers to the exchange of messages that takes place during the SSL handshake and the data transfer phases. In SSL Plus, the state of a connection is represented by the connection context structure. The state of a session is represented by an entry in the session database. A connection context is created by calling `ssl_CreateConnectionContext()` and should be destroyed when the connection is terminated with `ssl_DestroyConnectionContext()`.

#### Installable Objects

The SSL Plus API has been redesigned to allow you to include only the library code you need in your application. By calling an installable object function, you can determine the level of support for cipher suites and protocols in your

application. For example, by calling the function

**SSL\_PROTOCOL\_SSLV3\_SERVERSIDE( )**, you can include support for version 3 of the SSL protocol in your server application. Support for other protocols is not included in your application. This results in a smaller code size for your application.

There are nine categories of installable objects:

- Cipher suite objects
- Protocol objects
- Client authentication mode objects
- Private key decryption objects
- Certificate format objects
- Key and certificate decoding objects
- Trusted root objects
- PKCS #12 certificate and private key decryption objects
- PRNG (Pseudo-Random Number Generator) objects

The SSL Plus API provides functions to install each of these objects. See the **SSL Plus Programmer's Reference** for more information.

Note that each cryptographic engine (Security Builder, Cryptoswift) has its own set of cipher suite, client authentication mode and private key decryption objects. The Cryptoswift objects have the suffix **\_CS**.

As a rule, we recommend that you do not install objects intended for different cryptographic engines in the same global context. Any exceptions are noted in “Developing an Application: Step-by-Step” on page 11.

## Callbacks and SSL Plus

SSL Plus makes extensive use of callback functions. A callback function provides a way for SSL Plus to call your code. The library provides the template for the function; you provide the function body. Callbacks are used for the following purposes:

- To maintain a database of active sessions.
- To allow your application to accept or reject a peer's certificate.
- To allow other tasks to proceed during lengthy cryptographic operations. Note that this feature is dependent on Security Builder support. Not all platforms support it.
- To provide seed data for the library. The library uses the seed data from your callback function to generate random data for your application. This callback is mandatory.
- To read and write data on a network connection. The I/O callbacks are mandatory.
- To ensure that your application is portable across different platforms. These system callback functions are: **cic\_MallocFunc**, **cic\_FreeFunc**, **cic\_ReallocFunc**, **cic\_MemsetFunc**, **cic\_MemcpyFunc**, **cic\_MemcmpFunc**, **cic\_TimeFunc**. These callbacks are mandatory.

Refer to the **SSL Plus Programmer's Reference** for a description of the templates for the callback functions. SSL Plus provides default implementations for some of the callback functions (see “SSL Plus RAD API” on page 9).

# SSL Plus RAD API

The SSL Plus RAD (Rapid Application Development) API provides a shortcut for many of the most common operations in an SSL Plus application. With the RAD API you can

- Initialize a global context structure.
- Create a session database.
- Initialize a socket layer.
- Create a socket.
- Make a connection using a socket.
- Accept a connection using a socket.
- Close a socket.

The RAD API also provides default implementations of all the mandatory callback functions, as well as functions to check the validity of a certificate and manipulate entries in the session database.

The RAD API is optional; you can still develop your application using the standard SSL Plus API alone.

**Note:** The RAD API is only available on platforms which support sockets.

See the **SSL Plus Programmer's Reference** for a complete list of the RAD API functions.

# Cache Memory Manager

If your SSL Plus application makes 100 or more connections, you could improve its performance by using the Cache Memory Manager (CMM). The CMM reduces the number of memory allocation requests your application makes by pre-allocating a buffer of memory from the OS heap for each connection. Any memory that is requested during the connection is allocated from this buffer (through your memory callback function). You can set the size of this buffer when you initialize the CMM in your application. The CMM also uses a cache to store recently freed blocks of memory. New requests for memory are satisfied from this cache.

To find out how to use the CMM in your application, see the section “Developing an Application: Step-by-Step” on page 11.



# Developing an Application: Step-by-Step

The following paragraphs describe the steps involved in developing an application that uses SSL Plus. Most of these steps are identical regardless of whether your application is a client or a server. Any differences are pointed out in the discussion.

We recommend that you consult the **SSL Plus Programmer's Reference** and the sample applications (sslsampleclient.c and sslsampleserver.c) contained in your distribution as you read through these sections.

Application development can be broken down into several steps:

- Create the global context
- Set a PRNG (mandatory)
- Initialize the Cache Memory Manager (optional)
- Configure the mandatory settings in the global context
- Configure the optional settings in the global context
- Configure client authentication options (client only)
- Configure the server identity (server only)
- Register the optional callbacks
- Create the Cache Memory Manager (optional)
- Create a connection context
- Perform the handshake
- Retrieve negotiation information (optional)
- Transfer data
- Close the connection
- Destroy the connection context
- Free CMM resources in the connection context
- Free CMM resources in the global context
- Destroy the global context

## Create the global context

You must create a global context in your application. The global context contains all the global data needed to maintain a connection with a peer. You can set all the memory callback functions in the **ssl\_Callbacks** parameter. You can also set the callback functions for generating the random seed data and getting the time. Note that the **psurrender** (yielding) callback function is optional. See the **SSL Plus Programmer's Reference** for more information on these functions.

```
ssl_GlobalContext* globalCtx;
ssl_Callbacks osCallbacks;

osCallbacks.pmalloc=my_mallocCallback;
osCallbacks.pfree=my_freeCallback;
osCallbacks.pmemset=my_memsetCallback;
osCallbacks.pmemcpy=my_memcpyCallback;
osCallbacks.pmemcmp=my_memcmpCallback;
osCallbacks.ptime=my_timeCallback;
osCallbacks.prandom=my_randomCallback;
osCallbacks.psurrender=my_surrenderCallback;

ssl_CreateGlobalContext (&osCallbacks,&globalCtx);
```

	<p>You can also pass your own data to the random and surrender callbacks by setting the <b>randomRef</b> and <b>surrenderRef</b> fields. This is optional.</p>
Provide support for an RNG	<p>Call <b>ssl_SetPRNG()</b> to include a PRNG (Pseudo-Random Number Generator) in your application. This function takes a PRNG object as a parameter. One object is available: <b>ssl_ANSIPRNG()</b> (for Security Builder).</p> <p>This function is mandatory; you must use a PRNG in your application.</p>
Initialize the Cache Memory Manager	<p>If you are going to use the Cache Memory Manager (CMM), you must add support for it in the global context. This step is optional.</p> <pre>ssl_MemoryManagerInit(globalCtx);</pre>
Configure the mandatory setting in the global context	<ol style="list-style-type: none"> <li>1. Call <b>ssl_SetProtocol()</b> to indicate whether your application acts as a client or a server and which versions of the SSL protocol it supports. This function takes an <b>ssl_ProtocolSide</b> object as a parameter. All the <b>ssl_ProtocolSide</b> objects are listed in <b>sslapiobj.h</b> and in the <b>SSL Plus Programmer's Reference</b>.</li> </ol> <p>In some cases, you may find that the site you are attempting to connect to has not implemented your chosen SSL protocol correctly. This may mean that you cannot establish a connection to the site. SSL Plus provides a set of functions that allow you to change the behavior of an SSL protocol. These functions are optional: you should only use them if you have trouble connecting to a particular site. See the section "Configure the optional settings in the global context" on page 14 for a description of how to use these functions.</p> <p>You can also call <b>ssl_SetPkiPolicy()</b> to change the default PKI policy for your chosen SSL protocol. This feature is optional.</p> <pre>/* Client application - supports SSLV3 */ ssl_SetProtocol(globalCtx, SSL_PROTOCOL_SSLV3_CLIENTSIDE);</pre> <ol style="list-style-type: none"> <li>2. Call <b>ssl_SetCipherSuites()</b> to indicate which cipher suites your application supports. This function takes an array of <b>ssl_CipherSuite</b> objects as a parameter. All the <b>ssl_CipherSuite</b> objects are listed in <b>sslapiobj.h</b> and in the <b>SSL Plus Programmer's Reference</b>.</li> </ol> <pre>/* Set two cipher suites */ ssl_CipherSuite suites[3]; suites[0]= SSL_ALG_CIPHER_RSA_WITH_RC4_128_MD5_CLIENTSIDE; suites[1]= SSL_ALG_CIPHER_RSA_WITH_AES_128_CBC_SHA_CLIENTSIDE; suites[2]= NULL; /* Last array element must be NULL */  ssl_SetCipherSuites(globalCtx,suites);</pre>

The order of the suites is important. Suites earlier in the list are given higher priority during the handshake. Your cipher suites must all have the same application type (server or client-side), and this must match the type set in `ssl_SetProtocol()`.

Note that **ECDH\_ECDSA** and **AES** cipher suites are only supported during TLSV1 connections.

A particular cipher suite can only be used in the handshake if your server application has a certificate containing a public key of the same type specified in the cipher suite. This also applies to client applications if you have set client authentication. However an SSL context can support multiple certificates, so you can use several different cipher suites.

Note that you must install the appropriate cipher suite objects for your cryptographic engine (Cryptoswift cipher suite objects have the suffix `_CS`).

As a rule, we recommend that you do not install cipher suite objects intended for different cryptographic engines in the same global context. The only exception to this is using Security Builder ECC or DH cipher suites with Cryptoswift RSA cipher suites.

3. Call `ssl_SetIOFuncs()` to set the I/O callback functions. The library uses your callback functions to read and write data on an SSL connection. The template for the `read()` and `write()` callback functions is the type definition `ssl_IOFunc`. Refer to `ssl_IOFunc` in the **SSL Plus Programmer's Reference** for a description of the behavior of the `read()` and `write()` functions.

```
/* Read and write callbacks created earlier */
ssl_SetIOFuncs(globalCtx,readCB,writeCB);
```

4. Set the server identity. The library enforces server authentication, so you must create a certificate list (or chain) that identifies your server application. If you are developing a client application, you must indicate which CAs (Certificate Authorities) you trust.

SSL Plus supports three types of identities: RSA, ECDSA and DSS. If necessary, you should repeat the following calls for each set of certificates and keys.

- `ssl_CreateCertList()` to create a new, empty certificate list.
- `ssl_AddCertificate()` to add each local certificate to the list. A certificate chain should be terminated with the root CA. The private key will be protected by a password.
- `ssl_AddIdentity()` to register the complete list with the global context.
- `ssl_DestroyCertList()` to ensure that the list is destroyed when the global context is destroyed.

To indicate which CAs you trust in your client application, call the following functions.

- `ssl_CreateCertList()` to create a new empty certificate list.
- `ssl_AddCertificate()` to add each trusted certificate to the list. If you have root certificates, add them to the list by calling `ssl_AddTrustedRoots()`.
- `ssl_AddTrustedCerts()` to register the complete list with the global context. This function also checks the expiry date of the certificates. If a certificate has expired an error is returned, however the certificate is still loaded into the context. You can replace it later on by calling the function with a valid certificate. Note that the expiry period is checked only when the certificates are loaded into the context. This might have an impact on long-running servers.
- `ssl_DestroyCertList()` to ensure that the list is destroyed when the global context is destroyed.

As an alternative to `ssl_AddCertificate()`, you can use

`ssl_AddPkcs12Pfx()` to add the certificates and private key from a PKCS#12 base64-encoded PFX to an identity or trusted certificate list. Note that this function places several restrictions on the PFX:

- The integrity mode of the PFX must be password integrity. If the integrity mode is public-key integrity, the error `CIC_ERR_PKCS_NEED_TRUSTED` is returned.
- The privacy mode of the PFX must be password privacy. If the privacy mode is public-key privacy, the error `CIC_ERR_PKCS_NEED_PRV_KEY` is returned.
- The PFX must use the same password for data encryption, private key decryption and MAC verification.
- The PFX can contain any number of certificates but must contain at most one private key.

**You must also ensure that you install all necessary cipher suites before you call this function.** If the PFX contains an RSA certificate or private key, an RSA cipher suite must be installed.

One of the parameters to the function is an array which identifies the encryption suites which are required to decrypt and parse the PFX. You should be able to identify which suites are required so that you include only the code you need in your application. All the encryption suite objects are listed in the **SSL Plus Programmer's Reference**.

## Configure the optional settings in the global context

1. Call `ssl_SetCryptographicStrength()` to set the cryptographic strength of an SSL connection. The strength of a connection is determined by the cipher suite chosen during the negotiation. The possible settings are `SSL_CryptoStrength_StrongCryptoOnly`, `SSL_CryptoStrength_ExportCryptoOnly` and `SSL_CryptoStrength_AllCrypto` (default value). The new strength setting takes effect after the connection is created with `ssl_CreateConnectionContext()`. For existing connections, the handshake must be renegotiated for the new strength setting to take effect.

```
/* Set strong crypto */
```

```
ssl_SetCryptographicStrength(global-
    Ctx,SSL_CryptoStrength_StrongCryptoOnly);
```

2. Call `ssl_SetSessionExpiryTime()` to set the expiry period for SSL sessions in the session database.

```
/* Set expiry period to sixty seconds) */
ssl_SetSessionExpiryTime(globalCtx,60UL)
```

3. Call `ssl_SetIOSemantics()` to set the behavior of the `ssl_Read()` function. You can set either partial or complete I/O. For example if you set partial I/O and attempt to read ten bytes, `ssl_Read()` returns before all ten bytes have been read. If you set complete I/O, `ssl_Read()` keeps calling your callback function until all ten bytes have been read. Note that if the callback function blocks, `ssl_Read()` will return less than ten bytes and return the error code `CIC_ERR_WOULD_BLOCK`.

```
/* Set partial IO (default is complete IO) */
ssl_SetIOSemantics(globalCtx,SSL_IOSemantics_PartialIO);
```

4. You can use the functions described below to change the behavior of your chosen SSL protocol. You may need to do this if the site you are connecting to does not conform completely to the protocol. You can change four aspects of a protocol's behaviour: the challenge length (SSL2 only), the point compression (TLS1 only), the rollback flag setting(SSL3 only) and the TLS1 extension setting. See the Programmer's Reference for more information.

```
ssl_ProtocolPolicy *protocolPolicy=NULL;

/* Create a policy object for the protocol */
ssl_CreateProtocolPolicy(global-
    Ctx,SSL_ProtocolVersion_SSLV2,&protocolPolicy);

/* Set the challenge length for the protocol */
ssl_SetProtocolPolicyChallengeLength(protocolPolicy,16);

/* Set the policy object for the protocol */
ssl_SetProtocolPolicy(globalCtx,protocolPolicy);

/* Destroy the policy object */
ssl_DestroyProtocolPolicy(globalCtx,&protocolPolicy);
```

## Client Authentication

If your client application has to authenticate itself to a server, you must load a certificate list (or certificate chain) that identifies your application.

First, call `ssl_SetClientAuthModes()` to set the client authentication mode(s). The client authentication modes indicate which identities (RSA, ECDSA, DSS) your application supports. The order of the client authentication mode objects is important: objects listed earlier in the list take precedence over later ones. See the **SSL Plus Programmer's Reference** for a list of all the client authentication mode objects.

Then call `ssl_CreateCertList()`, `ssl_AddCertificate()` (or `ssl_AddPkcs12Pfx()`), `ssl_AddIdentity()` and `ssl_DestroyCertList()`. The procedure is the same as that used to set the server identity.

To support client authentication in your server application, you must call `ssl_SetClientAuthModes()` and set the same (server-side) authentication objects as the client. Then call `ssl_CreateCertList()`, `ssl_AddCertificate()` (or `ssl_AddPkcs12Pfx()`), `ssl_AddTrustedCerts()` and `ssl_DestroyCertList()` to set the trusted CA list.

You must ensure that you install all necessary cipher suites and client authentication mode objects before you call `ssl_CreateCertList()`.

Note that you must install the appropriate client authentication mode objects for your cryptographic engine (Cryptoswift client authentication mode objects have the suffix `_CS`).

As a rule, we recommend that you do not install client authentication mode objects intended for different cryptographic engines in the same global context. The only exception to this is using Security Builder ECC or DH client authentication modes with Cryptoswift RSA client authentication modes.

## Register the optional callbacks

Use the following functions to set the optional callbacks:

- `ssl_SetAlertFunc()` to set the alert callback. A template for this callback, called `ssl_AlertFunc`, is defined in `ssltype.h`.
- `ssl_SetSessionFuncs()` to set the callbacks for session resumption database. Templates for these callbacks, named `ssl_AddSessionFunc`, `ssl_GetSessionFunc` and `ssl_DeleteSessionFunc`, are defined in `ssltype.h`.

The library calls these functions to maintain a database of active sessions. Your application is responsible for providing the database.

You can also set a parameter (called `sessionRef`) which the library passes to these callback functions. This parameter could be a pointer to your database. It is set when you call `ssl_CreateConnectionContext()`.

If your application is multi-threaded, you should ensure that access to the database is serialized. Session resumption is not a frequent occurrence, so this will not affect the performance of your application.

- `ssl_SetCheckCertificateChainFunc()` to set the callback which you use to override certificate chain validation errors. A template for this callback, called `ssl_CheckCertificateChainFunc`, is defined in `ssltype.h`. Note that TLS1 and SSL3 treat Basic Constraints differently. For TLS1, the certificate policy is to flag all Basic Constraints warnings, which are then passed to the certificate chain callback. However in SSL3 the Basic Constraints warnings are not flagged, so nothing is passed to the certificate chain callback. You can enforce the TLS1 certificate policy on SSL3 connections by calling the `ssl_SetPolicy()` function.
- `ssl_SetLogFunc()` to set the callback which logs information about a connection. A template for this callback, called `ssl_LogFunc`, is defined in `ssltype.h`.

**Create the CMM  
(optional)**

To create the CMM (Cache Memory Manager) and set the cache buffer size, call `ssl_MemoryManagerCreate()`.

```
ssl_MemoryManager *mm;
ssl_MemoryManagerCreate(globalCtx, memRef, 25000, 5000, 4000,
    256, &mm);
```

The `memRef` parameter can be used to pass data to the callback functions. If this parameter is NULL, the reference parameter stored in the global context is used.

Note that you can combine several of the steps listed above into a call to one of the RAD global context initialization functions. These functions allow you to

- Set RSA or ECC cipher suites, or both.
- Set the version of the SSL protocol you wish to use.
- Set the protocol side (client or server).
- Set client authentication (client applications still need to set their identity).
- Set partial or complete I/O.
- Create a session database.
- Provide default implementations for the session callback and certificate chain callback functions.

The initialization functions also provide default implementations for the mandatory callback functions (including the `read()` and `write()` callbacks). See the **SSL Plus Programmer's Reference** for more information on the RAD initialization functions.

**Create a  
connection  
context**

You must create and initialize a new connection context structure each time you connect to a peer. Any number of connection contexts can be created from the same global context. Call `ssl_CreateConnectionContext()` to allocate memory for a new connection context structure and set initial values. This function takes the following arguments:

- **defReadBufLen**  
Default size (in bytes) of the buffer used to read SSL records. The library allocates a buffer of this size for reading the records. If this parameter is zero, a buffer size of 4K is used.
- **maxReadBufLen**  
Maximum size (in bytes) of the buffer used to read SSL records. If the default buffer size specified by `defReadBufLen` is too small to hold the incoming record, the buffer temporarily increases to a size of `maxReadBufLen` bytes so that the record can be processed. If this parameter is zero, a maximum buffer size of 32K is used.
- **writeFragmentLen**  
Maximum size (in bytes) of the SSL records that are passed to `ssl_Write()`. Records that are above this limit are fragmented. If this parameter is zero, a default size of 4K is used.
- **peerID**



In client applications this parameter allows a client to set a unique identifier for each server it connects with. This allows the library to find the session ID and the corresponding information in the database when it attempts to resume a session with a server. Generally, this value is the server's IP address and port number. This parameter can be ignored by server applications.

- **ioRef**  
User-defined parameter that the library passes to your read/write callback functions.
- **randomRef**  
User-defined parameter that the library passes to your random callback function. This parameter is optional.
- **certRef**  
User-defined parameter that the library passes to your certificate chain validation callback function. This parameter is optional.
- **alertRef**  
User-defined parameter that the library passes to your alert callback function. This parameter is optional.
- **sessionRef**  
User-defined parameter that the library passes to your session callback functions.
- **surrenderRef**  
User-defined parameter that the library passes to your yield callback function.
- **logRef**  
User-defined parameter that the library passes to your log callback function.
- **memRef**  
User-defined parameter that the library passes to your memory callback functions.

## Perform the handshake

To begin the SSL handshake, call **ssl\_Handshake()**. This negotiates a cipher suite acceptable to both ends, verifies any certificates that are presented and creates keys for encrypting data.

**Note:** The **ssl\_Handshake()** function uses your I/O callback functions to read and write data on the network connection. As a result, it could return blocking errors such as **CIC\_ERR\_WOULD\_BLOCK**. If this happens, call **ssl\_Handshake()** repeatedly until it returns **CIC\_ERR\_NONE** or some other error code.

## Retrieve negotiation information (optional)

After **ssl\_Handshake()** has finished, you can call several functions to get information about the state of the SSL connection.

**ssl\_GetNegotiatedProtocolVersion()** returns the version of the SSL protocol that was agreed upon. **ssl\_GetNegotiatedCipher()** identifies which cipher suite was agreed upon. **ssl\_GetMasterSecret()** returns the master secret for the current session. If you attempted session resumption, you can call **ssl\_WasSessionResumed()** to find out if it actually occurred.

## Transfer data

To send data call **ssl\_Write()**. The library encrypts the data, passes it through a hashing function, and then uses your **write()** I/O callback function to send it over



the network. To read data call `ssl_Read()`. The library uses your `read()` I/O callback function to read the data from the network, then decrypts it and verifies that it has not been tampered with.

The library works with either blocking or non-blocking I/O. In blocking I/O, the read and write functions do not return until all the data has been read or written into the buffer. In non-blocking I/O, if your callback cannot fulfill a request then it should return `CIC_ERR_WOULD_BLOCK`. The library passes this value up to your application. If the error code came from a `read()` callback then your application should call the `ssl_Read()` function until it returns `CIC_ERR_NONE`. If the error code came from a `Write()` callback then you should call the `ssl_ServiceWriteQueue()` function until it returns `CIC_ERR_NONE`. This is because any data that is waiting to be sent is kept in an output queue.

If you receive the error code `CIC_ERR_SSL_HANDSHAKE_REQUESTED` while reading data, this indicates that the peer has requested a renegotiation of the handshake. To proceed with the renegotiation, call `ssl_Handshake()`. To refuse the request, call `ssl_Read()`, `ssl_Write()` or `ssl_Close()`.

To determine how much application data is waiting to be read, call `ssl_GetReadPendingSize()`. To determine how much data is waiting in the write queue, call `ssl_GetWritePendingSize()`.

Note that your `read()` callback function should contain an appropriate timeout interval for the socket, especially if you are developing a server application. See the sample callback function in the RAD API for an example of how to set the timeout interval.

### Close the connection

Once you have transferred all your data, close down the connection by calling `ssl_Close()`. This function writes any remaining data to the connection by calling `ssl_ServiceWriteQueue()` and then sends the peer a "close\_notify" message. This prevents an attacker from terminating a connection prematurely.

**Notes:** If you set non-blocking I/O, then `ssl_Close()` can return `CIC_ERR_WOULD_BLOCK`, in which case you should call `ssl_ServiceWriteQueue()` until it returns `CIC_ERR_NONE`. You do not need to call `ssl_Close()` if the library (or your callback function) returns a network error; in this case the library calls `ssl_Close()` for you. Improperly closing a session will result in the session data being deleted from the session database, preventing the session from being resumed at a later time.

### Get the CMM reference from the connection context

If you used the CMM in your application, you must remove it from memory before your application terminates. Call `ssl_GetConnectionMemRef()` to get a reference to the CMM stored in the connection context. You can then use this reference to free the resources allocated to the CMM.

### Destroy the connection context

Once the SSL connection has been closed, call `ssl_DestroyConnectionContext()` to delete the contents of the connection context structure.

Free the CMM resources in the connection context

Call **ssl\_MemoryManagerDestroy( )** to free the resources that were allocated to the CMM in the connection context. Use the reference parameter returned by **ssl\_GetConnectionMemRef( )** in this function.

Free the CMM resources in the global context

Call **ssl\_MemoryManagerShutdown( )** to free the resources that were allocated to the CMM in the global context.

Destroy the global context

If your application is finished with all its SSL connections, call **ssl\_DestroyGlobalContext( )** to delete the contents (and settings) of the global context structure. You should only do this if you are sure your application will not make any more SSL connections (i.e. your application is terminating).

# Restrictions on the use of the SSL Plus API

A number of restrictions have been placed on the API to prevent the context structures from being modified during an SSL connection. These restrictions are in place because SSL Plus does not carry out any form of locking (using mutexes or semaphores) in multi-threaded applications.

The points below contain guidelines on how to use the context structures in single and multi-threaded SSL Plus applications

- Most SSL Plus applications only need to create a single global context. Once you configure the global context, we recommend you do not change it (with the exception of calling `ssl_GenerateRSAExportKeyPair()`). This context can be used to create all future SSL connections. Each connection created from a global context has read-only access to that context. During the lifetime of the connection the library reads the global objects (protocol engine, cipher suites, local identities etc) configured in the global context. It is possible to change the global context configuration after the SSL connections have been created, however there are restrictions on when you can safely do so. Multi-threaded applications may also have to implement a mutex locking and unlocking mechanism to serialize access to the global context. Table 1 describes the restrictions that have been placed on the functions which modify the global context.
- Most SSL Plus applications should only create new SSL connections from a single thread. If two or more threads try to create a new SSL connection, you must ensure that calls to `ssl_CreateConnectionContext()` are serialized. This is usually done by enclosing the call to the function with a locking/unlocking mechanism. If only a single thread is used to create all connections, then no locking is needed.
- Most SSL Plus applications should only use a single thread per connection context. If you have multiple threads which access the same context, you must serialize access to the context. If only a single thread accesses the connection context, no locking is needed. Table 2 describes the restrictions that have been placed on the functions which modify the connection context.

Note that in multi-threaded applications your callback functions must also be thread-safe and re-entrant. For example, in the session database functions (which are used for session resumption) you must serialize access to the session database.

The column headings for Table 1 are described below.

- **Needs mutex** - If your application is multi-threaded, you must prevent the global context from being modified by two threads at the same time. All the functions which are marked with an "X" must be enclosed with a lock/unlock mechanism to prevent them from being called simultaneously. Functions that are not marked with an "X" do not need to be protected in this way.
- **Anytime** - Functions that are marked with an "X" can be called at any time.
- **No existing connections** - Functions that are marked with an "X" cannot be called while there exist connection contexts that have been created from this global context.
- **No connections handshaking** - Functions that are marked with an "X" cannot be called while any connection contexts created from this global context are being used to negotiate (or renegotiate) a handshake.
- **No connections reading data** - Functions that are marked with an "X" cannot be called while any connection contexts created from this global context are being used to read data with `ssl_Read()`.

**Table 1: Restrictions on functions that modify the global context**

	Needs mutex	Anytime	No existing connections	No connections h/dshaking	No connections reading data
<code>ssl_CreateGlobalContext()</code>		X			
<code>ssl_DestroyGlobalContext()</code>	X		X		
<code>ssl_SetProtocol()</code>	X		X		
<code>ssl_SetCryptographicStrength()</code>	X			X	
<code>ssl_SetClientAuthModes()</code>	X			X	
<code>ssl_SetCipherSuites()</code>	X			X	
<code>ssl_SetSessionExpiryTime()</code>	X			X	
<code>ssl_SetIOSemantics()</code>	X				X
<code>ssl_CreateCertList()</code>		X			
<code>ssl_AddCertificate()</code>		X			
<code>ssl_GenerateRSAExportKeyPair()</code>	X				
<code>ssl_AddIdentity()</code>	X			X	

**Table 1: Restrictions on functions that modify the global context**

	Needs mutex	Anytime	No existing connections	No connections h/dshaking	No connections reading data
ssl_AddTrustedCerts()	X			X	
ssl_AddTrustedRoots()		X			
ssl_DestroyCertList()		X			
ssl_CreateConnectionContext()	X				
ssl_SetIOFuncs()	X		X		
ssl_SetAlertFunc()	X		X		
ssl_SetSessionFuncs()	X		X		
ssl_SetCheckCertificateChainFunc()	X		X		
ssl_SetRandomFunc()	X		X		
ssl_SetSurrenderFunc()	X		X		
ssl_SetLogFunc()	X		X		
ssl_GenerateRandomSeed()		X			
ssl_ExtractCertificateNameItem()		X			
ssl_ExtractRawCertData()		X			
ssl_GetVersion()		X			
ssl_DecodeRecord()		X			

The column headings for Table 2 are described below.

- **Anytime** - The functions marked with an "X" can be called at any time.
- **Needs mutex** - If your application is multi-threaded, you must prevent the connection context from being modified by two threads at the same time. All the functions which are marked with an "X" must be enclosed with a lock/unlock mechanism to prevent them from being called simultaneously. Functions that are not marked with an "X" do not need to be protected in this way.

**Table 2: Restrictions on functions that modify the connection context**

	Anytime	Needs mutex
<code>ssl_DestroyConnectionContext()</code>		<b>X</b>
<code>ssl_Handshake()</code>		<b>X</b>
<code>ssl_RequestRenegotiation()</code>		<b>X</b>
<code>ssl_Read()</code>		<b>X</b>
<code>ssl_Write()</code>		<b>X</b>
<code>ssl_GetReadPendingSize()</code>		<b>X</b>
<code>ssl_GetWritePendingSize()</code>		<b>X</b>
<code>ssl_ServiceWriteQueue()</code>		<b>X</b>
<code>ssl_Close()</code>		<b>X</b>
<code>ssl_GetNegotiatedProtocolVersion()</code>	<b>X</b>	
<code>ssl_GetContextProtocolVersion()</code>	<b>X</b>	
<code>ssl_GetNegotiatedCipher()</code>	<b>X</b>	
<code>ssl_GetMasterSecret()</code>	<b>X</b>	
<code>ssl_WasSessionResumed()</code>	<b>X</b>	
<code>ssl_FreeRecordBuffers()</code>		<b>X</b>

# Managing Memory Allocation

In an effort to better control the memory requirements of SSL Plus, you might wish to consider the following in your application:

- Limiting the maximum record size used by SSL Plus
- Tracking the total memory allocation for each connection

To limit the maximum record size, call the `ssl_CreateConnectionContext()` function and set the `maxReadBufLen` and `writeFragmentLength` parameters to an appropriate value for your application. By setting these values, you can exercise control over the maximum size of any single allocation request by SSL Plus.

In general, any memory allocation granted by your `cic_MallocFunc()` callback function should be less than `MAX(maxReadBufLen and writeFragmentLength)` plus 100 bytes. Note that if the maximum memory allocation granted by `cic_MallocFunc()` is too small, SSL Plus may not be able to handle valid handshake messages.

If your application is a closed system involving only SSL Plus, even smaller allocation limits are permissible because SSL Plus supports the TLS extensions built into the "client\_hello" SSL message. If you are developing an open system (i.e. using SSL Plus with other products), you will be limited to either writing a sophisticated memory manager or making an arbitrary choice. You can also use the Cache Memory Manager in this case.

Setting the maximum record size and limiting the maximum size of single memory allocations addresses only part of SSL Plus's memory requirements. To achieve complete control over the memory requirements you must track all the memory allocations for each connection. There may be multiple memory allocations for each connection if the incoming messages are fragmented (fragmented messages are re-assembled before they are processed). You can limit the total memory allocation for a connection by tracking the allocation requests in the `cic_MallocFunc()` and `cic_FreeFunc()` callback functions. The `memRef` parameter can be used for this purpose.

In general it is a good idea to limit each memory allocation request to the record size appropriate for your application and to track the total memory allocations for each connection. In a desktop environment, this might mean setting the maximum record length to  $2^{15}$  bytes and limiting the sum of all allocations for a connection to less than  $2^{17}$  bytes. Note that the protocol permits handshake messages of up to  $2^{24}-1$  bytes, so determining the actual limit involves a little guesswork and experimentation.

Note: To save memory in idle connections, see the discussion on `ssl_FreeRecordBuffers()` in the section "Other Development Issues" on page 26.

## Other Development Issues

### Generating RSA export key pairs

US export regulations may require servers to use 512-bit or smaller RSA exchange keys. SSL has a feature that allows your application to pre-generate a 512-bit RSA key pair before any connections are made. The **ssl\_GenerateExportKeyPair()** function causes SSL Plus to generate an ephemeral 512-bit RSA key pair and store it in the global context.

You can only call **ssl\_GenerateExportKeyPair()** after calling **ssl\_CreateGlobalContext()**. When you subsequently call **ssl\_CreateConnectionContext()**, a copy of the exportable key-pair will be stored in the connection context and used during the handshake when negotiating exportable RSA cipher suites. Once the handshake is complete, the copy of the exportable key-pair is destroyed. If renegotiation is requested by either side (by calling **ssl\_RequestRenegotiation()**) the copy of the exportable key-pair is regenerated in the connection context (leaving the key pair in the global context untouched).

Note: Your server application is not required to call **ssl\_GenerateExportKeyPair()** in order to support exportable RSA key pairs. However, you may opt to call **ssl\_GenerateExportKeyPair()** before each call to **ssl\_CreateConnectionContext()** anyway, as this saves the connection context the cost of generating a new key pair.

US export policy is not written in stone, and the requirements vary. If you do use the exportable RSA key, for the best security, you should occasionally generate a new one. Generating a new one every 24 hours or so should provide more than adequate security. To regenerate the copy of the exportable key pair stored in the connection context, call **ssl\_RequestRenegotiation()**.

### Reducing memory overhead in idle connections

Each connection context uses a read and write record buffer. The size of these buffers are configured when **ssl\_CreateConnectionContext()** is called. Normally, these buffers are allocated when **ssl\_CreateConnectionContext()** is called and destroyed when **ssl\_DestroyConnectionContext()** is called. This design helps prevent heap fragmentation by reducing the number of memory allocations required to read and write data. When a connection context has completed a handshake, but is idle, these buffers are not required and can be freed to reduce memory overhead. The buffers can be freed by calling **ssl\_FreeRecordBuffers()**. This function only frees the buffers if they are empty. The next time data is read or written these buffers will automatically be re-allocated.

### Renegotiating a connection with a peer

When you renegotiate a connection, a new handshake procedure is initiated. Both peers must be in agreement in order for renegotiation to take place. Any changes made to the connection context will take effect if renegotiation is successful. Renegotiation can occur anytime after **ssl\_Handshake()** has completed successfully, but before **ssl\_Close()** is called. You request renegotiation by calling **ssl\_RequestRenegotiation()**. To carry out the handshake, call **ssl\_Handshake()** (if the peer agrees to the renegotiation request).



# 4 EAP Protocol

---

## Support for the EAP Protocol

SSL Plus supports a number of extensions to the Extensible Authentication Protocol (EAP). The EAP is an extension of the PPP (Point to Point Protocol) that provides support for additional authentication methods within PPP.

SSL Plus supports the following extensions to EAP:

- EAP-TLS
- EAP-TTLS (EAP-Tunnel TLS)
- PEAP (Protected EAP)
- SCTP (Stream Control Transmission Protocol)

For more information on these extensions, see the IETF references in Appendix E: Further Reading.

The sections that follow describe the features of EAP and how SSL Plus supports them.

### EAP-TLS

EAP-TLS is an extension to EAP which has additional security features based on the TLS protocol. These features include ciphersuite negotiation, mutual authentication and key management.

Applications which wish to use EAP-TLS must fulfill certain criteria:

#### **Protocol Version**

Both client and server must support TLS V1.0.

#### **Server Authentication**

Both client and server must support server authentication.

#### **Data Fragmentation**

TLS records may be fragmented into multiple EAP packets. Each packet indicates whether it contains the first TLS record fragment, the  $n^{\text{th}}$  TLS record fragment or the last TLS record fragment. Both client and server must be able to read and write fragmented TLS records.

#### **Key Derivation**

New encryption keys (for use with PPP encryption) must be derived from the negotiated TLS master secret and the pseudo-random function defined in the TLS specification.

### Identity Verification

The EAP server should verify that the claimed identity corresponds to the certificate presented by the client. Similarly, the client must verify the validity of the EAP server certificate, and should also examine the EAP server name presented in the certificate, in order to determine whether the EAP server can be trusted.

**Note:** EAP-TLS also states that both client and server can support session resumption and client authentication.

To support EAP-TLS in your SSL Plus application, you must make the following changes:

- Install support for the TLS V1.0 protocol alone by calling `ssl_SetProtocol()` with either the `SSL_PROTOCOL_TLSV1_SERVERSIDE` or the `SSL_PROTOCOL_TLSV1_CLIENTSIDE` protocol object.
- Modify your I/O read callback function to parse EAP-TLS packets. Your application must strip all EAP-TLS packet data and pass only the TLS record data to SSL Plus. It must also handle all issues relating to lost packets (retries etc). Duplicate TLS records must be discarded or else the TLS handshake negotiation will fail.

The callback function must be able to return only part of a TLS record, as the SSL Plus library always reads incoming records in two steps. The library first requests the five byte header of a TLS record, then requests the record contents. Because of this two step process for reading records, the callback function may need to buffer TLS record data from the incoming EAP-TLS packets.

- Modify your I/O write callback function to construct EAP-TLS packets from TLS record data. SSL Plus sends these packets to the peer during the TLS negotiation phase of EAP-TLS. Your application is responsible for ensuring that packets are delivered in the right order and that lost packets are re-sent.

When SSL Plus calls your callback function it always attempts to write an entire TLS record. If the callback function could only send part of the record, SSL Plus will call it again and request that it sends the remainder. If the TLS record is fragmented because of EAP-TLS, your callback function should indicate when the complete record has been sent. If the callback function is unable to send the entire TLS record, it should buffer the record data and build fragmented packets as required by EAP-TLS.

- Derive new EAP encryption keys after the TLS negotiation phase has finished successfully. The functions `ssl_TLSPRF()`, `ssl_GetClientRandom()` and `ssl_GetServerRandom()` are available for this purpose.

## EAP-TTLS

EAP-TTLS is an extension to EAP-TLS which allows legacy protocols to be used (securely) in the authentication process.

Applications which wish to use EAP-TTLS must fulfill certain criteria:

### Protocol Version

Both client and server must support TLS V1.0.

### Server Authentication

Both client and server must support server authentication.

### Data Fragmentation

TLS records may be fragmented into multiple EAP packets. Each packet will indicate whether it contains the first TLS record fragment, the  $n^{\text{th}}$  TLS record fragment or the last TLS record fragment. Both client and server must be able to read and write fragmented records.

### Key Derivation

New encryption keys (for use with PPP encryption) must be derived from the negotiated TLS master secret and the pseudo-random function defined in the TLS specification.

### Session Resumption

EAP-TTLS states that the client and server may support session resumption. If it is supported, the server must have the option of not resuming a session based on its own information.

### Piggybacking

EAP-TTLS states that a single EAP-TTLS packet may contain both phase 1 (handshake) and phase 2 (tunnel) TLS messages. If the client has authentication or other AVPs (attribute-value pairs) to send to the server, it must tunnel them within the same packet immediately following its "Finished" message. If the client fails to do this, the server incorrectly assumes that the client has no AVPs to send, possibly affecting the outcome of the negotiation.

**Note:** EAP-TTLS also states that the client and server may support client authentication. EAP-TTLS suggests that servers may need to encode their own identifying information (e.g. IP address) in the session IDs that they generate.

To support EAP-TTLS in your SSL Plus application, you must make the following changes:

- Install support for the TLS V1.0 protocol alone by calling `ssl_SetProtocol()` with either the `SSL_PROTOCOL_TLSV1_SERVERSIDE` or the `SSL_PROTOCOL_TLSV1_CLIENTSIDE` protocol object.
- Modify your I/O read callback function to parse EAP-TTLS packets. Your application must strip all EAP-TTLS packet data and pass only the TLS record

data to SSL Plus. It must also handle all issues relating to lost packets (retries etc). Duplicate TLS records must be discarded or else the TLS handshake negotiation will fail.

The callback function must be able to return only part of a TLS record, as the SSL Plus library always reads incoming records in two steps. The library first requests the five byte header of a TLS record, then requests the record contents. Because of this two step process for reading records, the callback function may need to buffer TLS record data from the incoming EAP-TTLS packets.

- Modify your I/O write callback function to construct EAP-TTLS packets from TLS record data. SSL Plus sends these packets to the peer during the TLS negotiation phase of EAP-TTLS. Your application is responsible for ensuring that packets are delivered in the right order and that lost packets are re-sent.

When SSL Plus calls your callback function it always attempts to write an entire TLS record. If the callback function could only send part of the record, SSL Plus will call it again and request that it sends the remainder. If the TLS record is fragmented because of EAP-TTLS, your callback function should indicate when the complete record has been sent. If the callback function is unable to send the entire TLS record, it should buffer the record data and build fragmented packets as required by EAP-TTLS.

Note that in order to allow piggybacking of phase 1 and phase 2 TLS messages, an extra parameter has been added to the I/O write callback function. The SSL Plus library sets the **ssl\_IOState** enumerated type to indicate if the outgoing message is the TLS "Finished" message.

- Derive new EAP encryption keys after the TLS negotiation phase has finished successfully. The functions **ssl\_TLSPRF()**, **ssl\_GetClientRandom()** and **ssl\_GetServerRandom()** are available for this purpose.
- Modify the sample database callback functions to support EAP-TTLS. Server applications will need to store additional authentication information and may need to perform additional processing to determine if a session is valid for resumption. The callback functions contain a user-defined parameter for this purpose. The sessionID is available to server applications through the **ssl\_GetSessionID()** function (the sessionID is randomly generated by the SSL Plus library).

## PEAP

PEAP is an extension to EAP which provides a secure TLS channel between two peers. This prevents an attacker from injecting data into the data stream.

Applications which wish to use PEAP must fulfill certain criteria:

### Protocol Version

Both client and server must conform to TLS V1.0.

### Ciphersuites

Both the `TLS_RSA_WITH_RC4_128_MD5` and the `TLS_RSA_WITH_RC4_128_SHA` ciphersuites must be supported.

### Server Authentication

Both client and server must support server authentication.

### Data Fragmentation

TLS records may be fragmented into multiple EAP packets. Each packet will indicate whether it contains the first TLS record fragment, the  $n^{\text{th}}$  TLS record fragment or the last TLS record fragment. Both client and server must be able to read and write fragmented records.

### Key Derivation

New encryption keys (for use with PPP encryption) must be derived from the negotiated TLS master secret and the pseudo-random function defined in the TLS specification.

### Error Handling

PEAP does not have its own error capabilities, so the client and server must support sending TLS alert messages if an error occurs in the PEAP conversation.

### Re-negotiation

PEAP states that the client and server may support re-negotiation. If they do support it, the client must have the ability to refuse the re-negotiation request.

**Note:** PEAP also states that the client and server may support client authentication and TLS compression.

To support PEAP in your application, you must make the following changes:

- Install support for the TLS V1.0 protocol alone by calling `ssl_SetProtocol()` with either the `SSL_PROTOCOL_TLSV1_SERVERSIDE` or the `SSL_PROTOCOL_TLSV1_CLIENTSIDE` protocol object.
- Install either the `TLS_RSA_WITH_RC4_128_MD5` or the `TLS_RSA_WITH_RC4_128_SHA` ciphersuite.
- Modify your I/O read callback function to parse EAP packets. During the TLS negotiation phase, and during data transfer, your application must strip all incoming EAP packet data and pass only the TLS record data to SSL Plus. It must also handle all issues relating to lost packets (retries etc). Duplicate TLS records must be discarded or else the TLS handshake negotiation will fail.

The callback function must be able to return only part of a TLS record, as the SSL Plus library always reads incoming records in two steps. The library first requests the five byte header of a TLS record, then requests the record contents. Because of

this two step process for reading records, the callback function may need to buffer TLS record data from the incoming EAP packets.

- Modify your I/O write callback function to construct PEAP packets from TLS record data. SSL Plus sends these packets to the peer during phase 1 and phase 2 of PEAP. Your application is responsible for ensuring that packets are delivered in the right order and that lost packets are re-sent.

When SSL Plus calls your callback function it always attempts to write an entire TLS record. If the callback function could only send part of the record, SSL Plus will call it again and request that it sends the remainder. If the TLS record is fragmented because of PEAP, your callback function should indicate when the complete record has been sent. If the callback function is unable to send the entire TLS record, it should buffer the record data and build fragmented packets as required by PEAP.

- Derive new EAP encryption keys after the TLS negotiation phase has finished successfully. The `ssl_TLSPRF()`, `ssl_GetClientRandom()` and `ssl_GetServerRandom()` functions are available for this purpose.
- Send a TLS alert message to the peer if an error occurs during the PEAP conversation. SSL Plus provides the `ssl_SendAlert()` function for this purpose.
- Modify the sample database callback functions to support PEAP. Server applications will need to store additional authentication information and may need to perform additional processing to determine if a session is valid for resumption. The callback functions contain a user-defined parameter for this purpose.

## SCTP

SCTP is a reliable transport protocol operating on top of a connectionless packet network. SCTP allows the sequencing of user messages within multiple streams.

Applications which wish to use SCTP must fulfill certain criteria:

### Protocol Version

Both client and server must conform to TLS V1.0.

### Data Fragmentation

Both client and server must be able to handle fragmented SCTP user messages.

### Inbound and Outbound Streams

SCTP states that when establishing a TLS connection, a client or server must request the same number of inbound and outbound streams. A pair of streams with the same identifier is considered to be one bi-directional stream.

### Size of SCTP User Messages

SCTP states that an SCTP implementation must support transmitting all TLS records as SCTP messages. The maximum size of a TLS record is  $2^{14} + 2048 + 5 = 18437$  bytes.

**Note:** SCTP also states that the client may support session resumption.

To support SCTP in your application, you must make the following changes:

- Install support for the TLS V1.0 protocol alone by calling `ssl_SetProtocol()` with either the `SSL_PROTOCOL_TLSV1_SERVERSIDE` or the `SSL_PROTOCOL_TLSV1_CLIENTSIDE` protocol object.
- Pass the stream identifier to the I/O read/write callbacks by setting the `ioRef` parameter. Each stream identifier represents a pair of streams. You can set the `ioRef` parameter by calling `ssl_CreateConnectionContext()`. The SSL Plus library passes the `ioRef` parameter to your I/O read/write callback functions.
- Modify the I/O read callback function so that it parses incoming SCTP messages and extracts the TLS records. The SCTP stream is identified by the `ioRef` parameter. The SCTP data should be stripped so that only the TLS record data is passed to SSL Plus.

The callback function must be able to return only part of a TLS record, as the SSL Plus library always reads incoming records in two steps. The library first requests the five byte header of a TLS record, then requests the record contents. Because of this two step process for reading records, the callback function may need to buffer TLS record data from the incoming SCTP packets.

- Modify the IO write callback so that it constructs SCTP messages from TLS record data. The SCTP stream is identified by the `ioRef` parameter.





# A

## Building and Running the Sample Applications

### Building SSLSampleClient and SSLSampleServer

On Unix platforms, run **gmake** with the **makefile** located in the **samples** directory.

On the Win32, PocketPC and Palm platforms, use the project files located in the **samples** directory.

### Running SSLSampleClient and SSLSampleServer

The client and server applications work together to demonstrate a secure SSL connection. This section shows you how to run the applications.

The command-line syntax for the client application is shown below.

```
sslsampleclient [host] [port]
```

The default host is *localhost* and the default port number is 4433.

Although the client and server applications are intended to work together, you can also test the client by connecting to any secure HTTP server. If you have an internet connection, you might like to try the following:

```
sslsampleclient www.microsoft.com 443
```

You should see something like the following appear on the screen:

```
SSL Plus library version: V4.0
Peer certificate chain information:
Error: Certificate chain not trusted
```

```
SSL connection information:
  Protocol      : TLSV1
  Cipher Suite  : TLS_RSA_WITH_RC4_128_MD5
  Resumed       : NO
  Duration      : 541 ms
```

```
Response from server:
HTTP/1.1 404
Server: Microsoft-IIS/5.0
Date: Mon, 11 Mar 2002 14:13:15 GMT
P3P: CP='ALL IND DSP COR ADM CONo CUR CUSo IVAo IVDo PSA PSD
      TAI TELo OUR SAMo CNT COM INT NAV ONL PHY PRE PUR UNI'
Pragma: no-cache
cache-control: no-store
Connection: Keep-Alive
```

Content-Length

The command-line syntax for the server application is shown below.

```
sslsampleserver [port]
```

The server application sends some host information to the client and displays some information about the client.

When the sample server application connects with the sample client, you should see something like the following appear on the screen:

```
SSL Plus library version:V4.0
```

```
SSL connection information:
```

```
Protocol      : TLSV1
Cipher Suite  : TLS_ECDH_ECDSA_WITH_RC4_128_SHA
Resumed       : NO
Duration      : 40 ms
```

```
Request from client:
```

```
GET /index.html HTTP/1.0
```

### The Final Word: Know Your Makefiles!

Even if you are not using the provided makefiles to build your SSL Plus application, they are still useful sources of information. The makefiles list all the directories, lib files, object files and preprocessor directives you need to build the sample applications on their respective platforms. If you run into trouble, consult the makefiles before calling customer support. In most cases, linker errors are the result of missing library or header files in your project

# B

## Enabling WAP 2.0 Clients and Servers with SSL Plus

This appendix describes the changes that have been made to SSL Plus v4.3 owing to the introduction of WAP 2.0. It describes what you have to do to create a WAP 2.0-compliant application (client or server) using SSL Plus.

To create a WAP 2.0-compliant application you must call the functions required to create an SSL connection as well the functions described in the first section of this appendix. The API functions are described in more detail in the SSL Plus Programmer's Reference and in this User's Guide.

The remaining two sections of this document ("WAP TLS Profile and Tunneling" on page 42 and "WAP Certificates and CRL Profiles" on page 44) mirror the format of the WAP references listed below. You may find these sections useful in integrating WAP 2.0 into your SSL Plus application.

The first reference [WAPCert] describes WAP 2.0 certificate profiles. The second reference [WAPTLS] describes the WAP profile for using TLS 1.0 as the transport layer security protocol.

[WAPCert] WAP Forum, *WAP Certificate and CRL Profiles* (WAP-211-WAPCert), 22-May-2001. See <http://www.wapforum.org>.

[WAPTLS] WAP Forum, *Wireless Application Protocol TLS Profile and Tunneling* (WAP-219-TLS), 11-April-2001. See <http://www.wapforum.org>.

### SSL Plus Global Context Functions

This group of functions configure a global context. The settings in the global context apply to all SSL connections.

**ssl\_SetSessionExpiryTime()** is a function which sets the expiry period for SSL sessions. The default expiry period is 24 hours. [WAPTLS] recommends an expiry period of 12 hours.

**ssl\_SetSessionIdentifierLength()** is a new function for setting the maximum length of a session identifier. The default length is 32 bytes. [WAPTLS] recommends a length of 8 bytes or less.

### SSL Plus Certificate Format Objects

These objects provide support for specific certificate formats. Applications must install one of the following certificate format objects .

**SSL\_CERT\_FORMAT\_X509()** is an **ssl\_CertFormat** object which contains the features that parse X.509 certificates. This object supports RSA and ECC certificates.

**SSL\_CERT\_FORMAT\_X509\_ECC()** is an **ssl\_CertFormat** object which contains the features that parse X.509 certificates<sup>1</sup>. This object supports ECC certificates only.

**SSL\_CERT\_FORMAT\_X509\_RSA()** is an **ssl\_CertFormat** object which contains the features that parse X.509 certificates. This object supports RSA certificates only.

## SSL Plus Protocol Objects

These objects provide support for specific versions of the SSL protocol. No new modes have been added for [WAPTLS]. Applications must install one of the following protocol objects.

**SSL\_PROTOCOL\_TLSV1\_CLIENTSIDE()** is an **ssl\_ProtocolSide** object which provides support for TLS 1.0 in a client application. It also enforces the PKI policy for TLS 1.0. It negotiates TLS 1.0 (client-side only).

**SSL\_PROTOCOL\_TLSV1\_SERVERSIDE()** is an **ssl\_ProtocolSide** object which provides support for TLS 1.0 in a server application. It also enforces the PKI policy for TLS 1.0. It negotiates TLS 1.0 (server-side only).

## SSL Plus Policy Objects

These objects define the PKI policies that govern how the certificate format objects are used. At least one of these policy objects is installed by each protocol object, but this behavior can be overridden. To override the default policy for a protocol, use the function **ssl\_SetPolicy()**. The following policy objects are provided.

**SSL\_PKI\_POLICY\_SSLV2()** is a new **ssl\_Policy** object which enforces the PKI policy required by SSL2. This policy includes only the requirements for SSL 2.0.

**SSL\_PKI\_POLICY\_SSLV3()** is a new **ssl\_Policy** object which enforces the PKI policy required by SSL3. This policy includes only the requirements for SSL 3.0.

**SSL\_PKI\_POLICY\_TLSV1()** is a new **ssl\_Policy** object which enforces the PKI policy required by [RFC2246]. This policy includes only the requirements for TLS 1.0.

**SSL\_PKI\_POLICY\_WAPV2()** is a new **ssl\_Policy** object which enforces the PKI policy required by [WAPCert,WAPTLS]. This policy includes only the requirements for TLS 1.0 and WAP 2.0.

The **ssl\_SetPolicy()** function can be used to change the policies associated with a protocol object, i.e. change the default policy associated with a protocol.

## SSL Plus Authentication Mode Objects

These objects provide support for specific client authentication modes. No new modes have been introduced as a result of [WAPTLS]. An application that requires client authentication must install one of the following modes.

---

1. Note that ECC is not required by [WAPTLS], but it is described in [WAPCert].

**SSL\_CLIENT\_AUTH\_MODE\_RSA\_SIGN\_CLIENTSIDE()** is an **ssl\_ClientAuthMode** object which provides support for client authentication using RSA signatures (on the client side).

**SSL\_CLIENT\_AUTH\_MODE\_RSA\_SIGN\_SERVERSIDE()** is an **ssl\_ClientAuthMode** object which provides support for client authentication using RSA signatures (on the server side).

By default, server authentication is enforced whenever any of the WAP 2.0 cipher suites are installed.

## SSL Plus Cipher Suite Objects

These objects provide support for specific cipher suites.

Client applications must install one of the following client-side cipher suites [WAPTLS]. Server applications must install all of the following server-side cipher suites [WAPTLS].

**SSL\_ALG\_CIPHER\_RSA\_WITH\_RC4\_128\_SHA\_CLIENTSIDE()** is an **ssl\_CipherSuite** object which provides support for the cipher suite **TLS\_RSA\_WITH\_RC4\_128\_SHA** on the client side.

**SSL\_ALG\_CIPHER\_RSA\_WITH\_RC4\_128\_SHA\_SERVERSIDE()** is an **ssl\_CipherSuite** object which provides support for the cipher suite **TLS\_RSA\_WITH\_RC4\_128\_SHA** on the server side.

**SSL\_ALG\_CIPHER\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA\_CLIENTSIDE()** is an **ssl\_CipherSuite** object which provides support for the cipher suite **TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA** on the client side.

**SSL\_ALG\_CIPHER\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA\_SERVERSIDE()** is an **ssl\_CipherSuite** object which provides support for the cipher suite **TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA** on the server side.

Applications (both client and server) which use **SSL\_CERT\_FORMAT\_X509\_WAP\_20\_ECC()** must install at least one ECC cipher suite object.

## SSL Plus Certificate Functions

These functions can be used to access fields in a certificate. They provide access to each entity at the level identified below. In some cases, the fields extracted by these functions contain nested components that are not required by WAP 2.0. The extraction of these nested components is beyond the scope of SSL Plus. The following certificate functions are provided by SSL Plus<sup>1</sup>.

**ssl\_ExtractRawCertData()** converts certificate objects to DER format. All other certificate functions use this DER format.

**ssl\_GetConnectioncbData()** returns a pointer to the callback data contained within a connection context. The certificate functions use this data.

---

1. The **ssl\_ExtractCertificateNameItem()** function is deprecated.

`ssl_GetGlobalcbData()` returns a pointer to the callback data contained within a connection context. The certificate functions use this data.

The header and library files for the Trustpoint (`tp_`) functions are included in your SSL Plus distribution.

### **X.509 Certificate API**

- `tp_X509CertDecodeBegin()` allocates and initializes an API context for parsing certificates.
- `tp_X509CertDecodeGetSerialNumber()` extracts a certificate's serial number (see [RFC2459]).
- `tp_X509CertDecodeGetSignatureAlgorithmOid()` extracts the signature algorithm OID from a certificate.
- `tp_X509CertDecodeGetSubject()` extracts the subject from a certificate.
- `tp_X509CertDecodeGetIssuer()` extracts the issuer from a certificate.
- `tp_X509CertDecodeGetExtByOid()` extracts the extension from a certificate using the given object identifier. This function provides access to the following extensions: authority key identifier, basic constraints, certificate policies, domain information (see [WAPCert]), extended key usage, key usage, name constraints, policy constraints, subject alternative name and subject key identifier.
- `tp_X509CertDecodeGetExtByIndex()` extracts extensions from a certificate using the given index.
- `tp_X509CertDecodeEnd()` cleans up and de-allocates the certificate parsing context.

### **X.509v3 Certificate Extension API**

- `tp_X509ExtDecode()` decodes the object identifier, criticality flag and value of an extension.
- `tp_X509ExtDecodeAuthorityKeyIdentifier()` extracts a component from the authority key identifier.
- `tp_X509ExtDecodeExtKeyUsage()` extracts an object identifier from an extended key usage extension.
- `tp_X509ExtDecodeSubjectKeyIdentifier()` extracts a component from the subject key identifier.
- `tp_X509ExtDecodeSubjectAltName()` extracts the **dNSName** and **iPAddress** components from the subject alternative name.

### **X.509 Name API**

- `tp_X509NameDecodeBegin()` allocates and initializes an API context for parsing a name.
- `tp_X509NameDecodeGetStringByOid()` extracts a string component of the name by OID. It performs the operations previously carried out by `ssl_ExtractCertificateNameItem()` and extends them to include the

extraction of the following additional names (see [RFC2459]): distinguished name qualifier, domain component, generation qualifier, given name, initials, surname and title.

- **tp X509NameDecodeGetAttrByOid( )** extracts an attribute component of the name by OID.
- **tp X509NameDecodeEnd( )** cleans up and de-allocates the name parsing context.

# WAP TLS Profile and Tunneling

**TLS Profile** To enable TLS 1.0, install one of the protocol objects (see SSL Plus Protocol Objects).

**Cipher Suites** To enable ciphersuites, install one of the cipher suite objects (see SSL Plus Cipher Suite Objects).

**Session** **Session Resume**  
To enable session resumption, create and install callbacks for the functions listed below. These functions provide SSL Plus with an entry point for the session database.

- `ssl_AddSessionFunc()`
- `ssl_DeleteSessionFunc()`
- `ssl_GetSessionFunc()`

Use `ssl_SetSessionExpiryTime()` to set the session expiration time (see “SSL Plus Global Context Functions” on page 37).

## **Session Identifier**

To allow smaller session identifiers, use `ssl_SetSessionIdentifierLength()` to set the session identifier length (see “SSL Plus Global Context Functions” on page 37).

**Server Authentication** By default, SSL Plus supports server authentication on both the client and the server.

## **Enabling client-side X.509 certificate processing**

To enable the client to support X.509 server certificates, install a certificate format object (see SSL Plus Certificate Format Objects).

## **Enabling client-side endpoint identification (RFC2818)**

[WAPTLS] recommends that the client respect the guidelines for endpoint identification as applied to the server (see [RFC2818])<sup>1</sup>. To allow endpoint identification, SSL Plus provides functions for retrieving the `subjectAltName` and the `subject` (“SSL Plus Certificate Functions” on page 39). These functions can be used with the certificate callback (see `ssl_CheckCertificateChainFunc()` [ProgRef]) and combined with other application code to allow endpoint identification.

The retrieval of a DNS name from the end entity’s certificate and its subsequent validation via DNS is an application issue.

## **Client guidelines for processing X.509 certificates**

---

1. This is different from server authentication [RFC2246]. Server authentication occurs by default



To allow clients to process X.509 certificates, install a certificate format object (see SSL Plus Certificate Format Objects). See (B) to obtain a better understanding of how these objects meet these guidelines.

#### **Client guidelines for processing unknown attributes and extensions**

To allow clients to process unknown attributes and extensions, install a certificate format object (see “SSL Plus Certificate Format Objects” on page 37) and review the certificate functions. See (B) to obtain a better understanding of how these objects meet these guidelines.

#### **Server guidelines for processing X.509 certificates**

To allow servers to process X.509 certificates, install a certificate format object (see SSL Plus Certificate Format Objects). See (B) to obtain a better understanding of how these objects meet these guidelines.

### **Client Authentication**

To enable client authentication in a client or server, install a certificate format object (see “SSL Plus Certificate Format Objects” on page 37) and install an authentication mode object (see “SSL Plus Authentication Mode Objects” on page 38).

#### **Enabling server-side endpoint identification (RFC2818)**

[WAPTLS] recommends that the server respects the guidelines for endpoint identification as applied to the client (see [RFC2818])<sup>1</sup>. To allow endpoint authentication, SSL Plus provides functions for retrieving the `subjectAltName` and the `subject` (see “SSL Plus Certificate Functions” on page 39). These functions can be used with the certificate callback (see `ssl_CheckCertificateChainFunc()` in the SSL Plus Programmer’s Reference) and combined with other application code to allow endpoint identification.

### **Certificate Chain Depth**

SSL Plus places no limits on the depth of a certificate chain.

### **CA Practice Recommendation**

Not applicable to SSL Plus.

### **TLS Tunneling**

The use of a proxy server with SSL Plus is an application issue.

---

1. This is different from client authentication [RFC2246]. Client authentication is handled by the SSL Plus Authentication Mode Objects.

# WAP Certificates and CRL Profiles

## General

### Recognition of certificate profiles

To allow recognition of attributes and extensions, install a certificate object (see “SSL Plus Certificate Format Objects” on page 37) and use the certificate functions to access them (see “SSL Plus Certificate Functions” on page 39). To enable the WAP 2.0 PKI policy, install the WAP policy object (see “SSL Plus Policy Objects” on page 38). Attributes and extensions are processed in the certificate callback (see `ssl_CheckCertificateChainFunc()`).

In some cases, a substantial amount of effort might be needed to process these attributes and extensions. Subsequent sections of this document describe how SSL Plus can help with this.

### Minimum size restrictions on certificates

There are no limits placed on the size of a certificate by SSL Plus.

### Processing nonconforming certificates

Certificate processing in TLS 1.0 uses the certificate path validation algorithm (see [RFC2459]). SSL Plus handles all certificates using this algorithm. All validation failures are passed to the certificate callback (see `ssl_CheckCertificateChainFunc()`).

## User Certificates for Authentication

To enable client authentication, see “Client Authentication” on page 43.

### Certificate serial number

Not applicable to SSL Plus.

### Signature (Algorithm)

To allow verification of certificates signed with **sha1WithRSAEncryption** or **ecdsa-with-SHA1** keys, install a certificate format object. To obtain the key type used to sign a certificate, use a certificate function.

### Issuer (Name)

To access a distinguished name attribute defined by [RFC2459], use a certificate function.

### Subject (Name)

To access the subject field defined by [RFC2459], use a certificate function.

### Subject Public Key

There are no limits placed upon the size of an RSA key by SSL Plus. Only ECC key sizes of 163 bits (on curve sect163k1 [SEC2]) can be used with SSL Plus. To enable a key type, install a cipher suite and certificate format object.

### Certificate Extensions

SSL Plus recognizes all certificate extensions required and recommended in [WAPCert]. To enable recognition of attributes and extensions, see “Recognition of certificate profiles” on page 44.

Note 1: The **keyEncipherment** and **keyAgreement** bits are enforced according to [WAPCert] using the certificate callback (see **sslCheckCertificateChainFunc()** [ProgRef]).

Note 2: SSL Plus assumes that the **certificatePolicies** extension contains one **CertPolicyId**.

## User Certificates for Digital Signatures

The use of digital signatures as described in [WAPTLS] lies outside the scope of SSL Plus (for example, there is no way to use **signText()** with SSL Plus).

SSL Plus meets all of the certificate requirements for user certificates (see “User Certificates for Authentication” on page 44). The functions needed to access the key usage extensions are described in the section “SSL Plus Certificate Functions” on page 39.

## X.509-Compliant Server Certificates

To enable server authentication, see “Server Authentication” on page 42.

### Scope

SSL Plus supports certificate profiles which can be sent over the air. These profiles are described in [RFC2246].

### Certificate Serial Number

The functions needed to access the serial number are described in the section SSL Plus Certificate Functions.

### Signature (Algorithm)

SSL Plus supports the signature algorithms required by [WAPCert] (see “Certificate serial number” on page 44). Key limits are described in “Subject Public Key” on page 44.

### Issuer (Name)

SSL Plus supports chain building with unknown name attributes<sup>1</sup>. To enable chain building, install a certificate format object (see “SSL Plus Certificate Format Objects” on page 37). To access a distinguished name attribute, see “Issuer (Name)” on page 44.

### Subject (Name)

See “Recognition of certificate profiles” on page 44 and “Subject (Name)” on page 44.

### Subject Public Key

See “Subject Public Key” on page 44.

---

1. A name attribute is an attribute matching the issuer name (see [RFC2459], section 4.1.2.4).

### Certificate Extensions

SSL Plus recognizes all of the extensions required or recommended in [WAPCert] (see “Certificate Extensions” on page 44). To enable recognition and processing of attributes and extensions, see “Recognition of certificate profiles” on page 44. These extensions, whether critical or non-critical, are available through the certificate callback.

The following artifacts are available through the certificate functions (see “SSL Plus Certificate Functions” on page 39).

- **dNSName** component in the subject alternative name
- **keyIdentifier** field in the authority key identifier
- **id-kp-serverAuth** object identifier, if it exists
- **iPAddress** in the subject alternative name

The **CPSuri** and the **UserNotice** qualifiers contained within the certificate policies extension are not available to the application.

Server certificates used with [RFC2246] correctly process the keyUsage bits whenever the protocol object is installed (see “SSL Plus Protocol Objects” on page 38).

### Role Certificates

Not applicable to SSL Plus.

### Authority Certificates

TLS 1.0 does not require the handling of WTLS hybrid certificate chains<sup>1</sup>.

#### Certificate Serial Number

See “Certificate serial number” on page 44.

#### Signature Algorithm

See “Signature (Algorithm)” on page 44.

#### Issuer (Name)

See “Issuer (Name)” on page 44.

#### Subject (Name)

SSL Plus checks that self-signed certificates use the distinguished name as required by [RFC2459].

#### Subject Public Key

See “Subject Public Key” on page 44.

### Certificate Extensions

To enable the policy requirements for the **keyUsage** and the **basicConstraint** extensions, install a certificate object (see “SSL Plus Certificate Format Objects” on page 37). To enable recognition and processing of these attributes and extensions, see

---

1. A WTLS hybrid certificate chain contains an X.509 certificate which is used to sign an end-entity WTLS certificate.

“Recognition of certificate profiles” on page 44. These extensions, whether critical or non-critical, are available through the certificate callback.

To access the **subjectKeyIdentifier**, use a certificate function (see “SSL Plus Certificate Functions” on page 39).

#### Other Certificates

Only X.509 certificates corresponding to the profiles required by [RFC2246, WAPCert, WAPTLS] are supported.

#### CRL Profiles

Not applicable to SSL Plus.

## References

[ProgRef] Certicom Corp., *SSL Plus Version 4.0 Programmer's Reference (PUB-0300-0211)*, 7-March-2002.

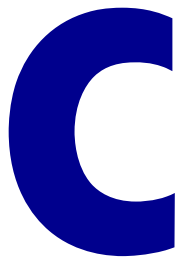
[SEC2] Certicom Corp., *SEC2: Recommended Elliptic Curve Domain Parameters*, Version 1.0, 20-September-2000.

[RFC2246] Internet Engineering Task Force, *The TLS V1.0 Protocol (RFC 2246)*, January-1999. See <http://www.ietf.org>.

[RFC2247] Internet Engineering Task Force, *Using Domains in LDAP/X.500 Distinguished Names (RFC 2247)*, January-1998. See <http://www.ietf.org>.

[RFC2459] Internet Engineering Task Force, *Internet X.509 Public Key Infrastructure Certificate and CRL Profile (RFC 2459)* January-1999. See <http://www.ietf.org>.

[RFC2818] Internet Engineering Task Force, *HTTP over TLS (RFC 2818)*, May-2000. See <http://www.ietf.org>.



# Enabling MIDP 2.0 Clients and Servers with SSL Plus

---

This appendix describes the changes that have been made to SSL Plus v4.3 owing to the introduction of MIDP 2.0. It describes what you have to do to create a MIDP 2.0-compliant application (client or server) using SSL Plus.

To create a WAP 2.0-compliant application you must call the functions required to create an SSL connection as well the functions described in the first section of this appendix. The API functions are described in more detail in the SSL Plus Programmer's Reference and in this User's Guide.

The remaining sections of this document ("MIDP X.509 Certificate Profile" on page 50 and "MIDP X.509 Certificate Profile for Trusted MIDlet Suites" on page 51) mirror the format of the MIDP reference listed below.

[MIDP] Java Community Process, *Mobile Information Device Profile 2.0* (JSR 118), 18-April-2002.

## SSL Plus Policy Objects

These objects define the PKI policies that govern how the certificate format objects are used (see [ProgRef]). The following policy object must be installed to enable MIDP 2.0.

**SSL\_PKI\_POLICY\_MIDPV2()** is a new **ssl\_Policy** object which enforces the PKI policy required by MIDP 2.0. This policy includes only the requirements for MIDP 2.0.

# MIDP X.509 Certificate Profile

## MIDP X.509 Certificate Profile

SSL Plus fulfills the requirements in [WAPCert, WAPTLS] as described in [SSL+]. To disable user certificates for authentication, do not install any authentication mode objects (see [SSL+]). SSL Plus does not support user certificates for digital signatures.

To disable support for standard key extensions, do not use the certificate extraction functions (see [SSL+]). SSL Plus delegates extended path validation to the application.

To enable the MIDP 2.0 PKI policy, install the MIDP policy object (see “SSL Plus Policy Objects” on page 49).

## Certificate Extensions

SSL Plus recognizes the basic constraints and key usage extensions. They are provided to the application through the certificate callback.

To enable support for multiple CA certificates that have the same distinguished name but different public keys, use the certificate functions to obtain the authority key identity or the subject key identity (see [SSL+]). If the application does not use either of these extensions, then some other mechanism can be added to the certificate callback. This other mechanism is beyond the scope of SSL Plus.

SSL Plus can process certificates with unknown distinguished name attributes and it can process certificates with unknown non-critical extensions.

To recognize the **serialNumber** attribute defined in [WAPCert], use the certificate functions (see [SSL+]).

## Certificate Size

There are no limits placed on the size of a certificate by SSL Plus.

## Algorithm Support

To enable support for any of the required signature algorithms, install the appropriate certificate format object (see [SSL+]). There are no limits placed on the size of an RSA key by SSL Plus.

## Certificate Processing for HTTPS

See [SSL+] for a discussion on handling endpoint identification using SSL Plus. The **id-kp-serverAuth** object identifier can be obtained by using the appropriate certificate function to extract it from the certificate (if it exists) (see [SSL+]).



# MIDP X.509 Certificate Profile for Trusted MIDlet Suites

## Certificate Processing for OTA

To recognize the key usage, see [SSL+] for a discussion on recognition of certificate profiles using SSL Plus.

The following artifacts are available through the certificate functions (see [SSL+]).

- **id-kp-codeSigning** object identifier, if it exists

## Certificate Expiration and Revocation

Use the function `ssl CheckCertificateChainFunc( ) [ProgRef])` and check for the expired certificate warning.

Applications which need to check the revocation status of certificates can add this to the certificate callback in SSL Plus. It is not a feature supported by SSL Plus.

## References

[ProgRef] Certicom Corp., *SSL Plus Version 4.0 Programmer's Reference* (PUB-0300-0211), 7-March-2002.

[SSL+] Certicom Corp., *Enabling WAP 2.0 Clients and Servers with SSL Plus 4.3*, Revision 1.41.

[WAPCert] WAP Forum, *WAP Certificate and CRL Profiles (WAP-211-WAPCert)*, 22-May-2001. See <http://www.wapforum.org>.

[WAPTLS] WAP Forum, *Wireless Application Protocol TLS Profile and Tunneling (WAP-219-TLS)*, 11-April-2001. See <http://www.wapforum.org>.

# D

## Appendix D: SSL: A History

---

### What is SSL?

The Secure Sockets Layer (SSL) protocol is a set of programming rules and guidelines that describe how to provide network client/server applications with fast and reliable application layer security. The latest version of the SSL protocol, version 3.0, provides three security primitives: secrecy, data integrity and authentication.

The original SSL protocol was developed by Netscape Communications corp. in 1994. The first version to ship, version 2.0, supported an RSA algorithm for encryption and X.509 certificates for authentication. In 1996 Netscape released SSL 3.0 which included new features and security improvements. SSL 3.0 supports more cryptographic algorithms, including ECC, DSS, Diffie-Hellman and Fortezza. In addition, SSL 3.0 supports more advanced authentication techniques including multiple client and server certificates and CA chains.

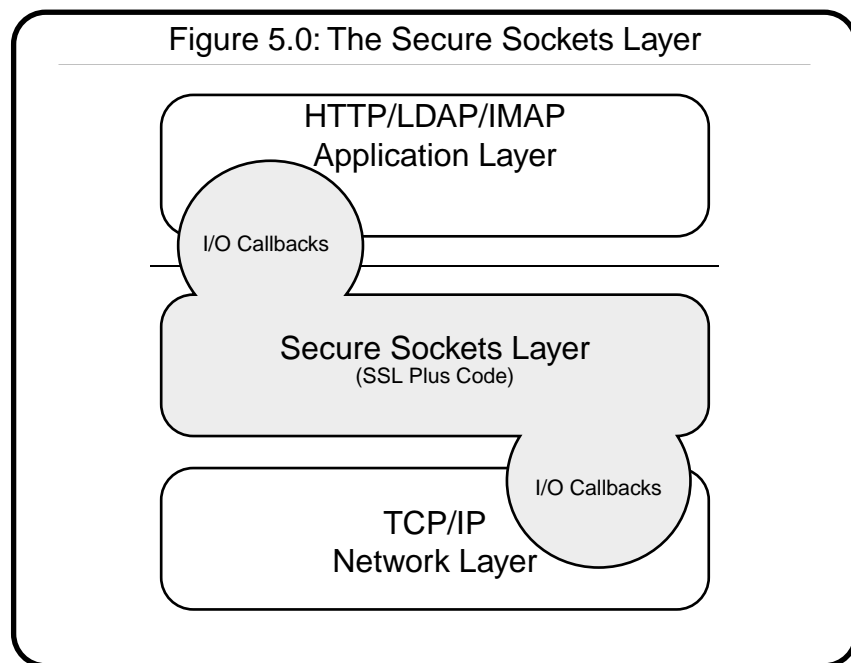
The SSL protocol, in one form or another, has become the world-wide accepted standard in web or hypertext transfer protocol (HTTP) communications.

### What is TLS?

In response to Netscape's revolutionizing security protocol, Microsoft developed their own web security standard called PCT that was roughly based on SSL 2.0. PCT offered a number of features that were later incorporated into SSL 3.0. The need for standardization became apparent. In 1997, the Internet Engineering Task Force (IETF) formed a working group that included both Netscape and Microsoft, to develop a stream security protocol based on SSL. TLS 1.0 is the latest version of this new protocol.

## How Does SSL/TLS Work?

The SSL protocol runs on top of Network Layer which is governed by the Transmission Control Protocol/Internet Protocol (TCP/IP) and below the Application Layer which may be governed by the Hypertext Transport Protocol (HTTP), the Lightweight Directory Access Protocol (LDAP), the Internet Messaging Access Protocol (IMAP), or the File Transfer Protocol (FTP). Your application interacts with both the TCP/IP and SSL protocols via the SSL I/O callbacks. Figure 5.0 illustrates this relationship:



The SSL Protocol is comprised of two protocol layers: the SSL Record Protocol (SSLRP) and the SSL Handshake Protocol (SSLHP). The SSLHP acts as a higher-level interface over the SSLRP. The SSLHP allows the SSL protocol to be “protocol independent.” This means that a higher application-level protocol can work above SSL without any regard for SSL. The application level is “plugged in” to **SSL Plus** through the use of I/O callbacks.

# E

## Appendix E: Further Reading

---

### On SSL/TLS

Freier, Alan., et al. *The SSL Protocol Version 3.0*. <http://www.netscape.com/eng/ssl3/>, 1996

### On Data Security and Encryption

Diffie, W., Hellman, M. E.. "New Directions in Cryptography". *IEEE Transactions on Information Theory*, V.IT-22, n. 6, Jun 1977, pp 74-84.

Ford, Warwick, Baum, Michael S. *Secure Electronic Commerce*. New York: Prentice Hall, 1997.

Menezes, Alfred J., et al. *Handbook of Applied Cryptography*. New York: CRC Press, 1997.

Schneier, Bruce. *Applied Cryptography*. Second Edition. New York: John Wiley & Sons, 1996.

### On Specific Algorithms

ANSI X3.106. "American National Standard for Information Systems-Data Link Encryption," American National Standards Institute, 1983.

FIPS PUB 186, "Digital Signature Standard," National Institute of Standards and Technology, U.S. Department of Commerce, 18 May 1994.

RFC 1321: The MD5 Message Digest Algorithm. April 1992.

FIPS PUB 180-1, "Secure Hash Standard," National Institute of Standards and Technology, U.S. Department of Commerce, DRAFT, 31 May 1994.

Rivest, R., Shamir, A., & Adleman, L.A. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, v. 21, n. 2, Feb 1978, pp. 120-126.

Tuchman, W. "Hellman Presents No Shortcut Solutions To DES," *IEEE Spectrum*, v. 16, n.7, July 1979, pp. 40-41.

## On Standards and Other Protocols

CCITT. *The Directory-Authentication Framework*. CCITT, 1988.

Certicom Corp. *ECC in Standards*. <http://www.certicom.com/resources/standards/index.html>.

The Internet Engineering Task Force Website. <http://www.ietf.org>.

The Directory-Authentication Framework, CCITT, 1988.

RSA Data Security. *RSA Cryptography Standard*, <http://www.rsasecurity.com/rsalabs/PKCS/pkcs-1/index.html>.

RSA Data Security. *Password-Based Cryptography Standard*, <http://www.rsasecurity.com/rsalabs/PKCS/pkcs-5/index.html>.

RSA Data Security. *Private-Key Information Syntax Standard*, <http://www.rsasecurity.com/rsalabs/pubs/PKCS/pkcs-8/index.html>.

RSA Data Security. *Certification Request Syntax Standard*, <http://www.rsasecurity.com/rsalabs/pubs/PKCS/pkcs-10/index.html>.

Internet Engineering Task Force, Network Working Group, *PPP EAP TLS Authentication Protocol*, RFC2716, October 1999. See <http://www.ietf.org>.

Internet Engineering Task Force, Network Working Group, *EAP Tunneled TLS Authentication Protocol (EAP-TTLS)*, February 2002. See <http://www.ietf.org/internet-drafts/draft-ietf-pppext-eap-ttls-01.txt>.

Internet Engineering Task Force, PPPEXT Working Group, *Protected EAP Protocol (PEAP)*, August 2001. See <http://www.ietf.org>.

Internet Engineering Task Force, Network Working Group, *Stream Control Transmission Protocol*, RFC2960, November 2000. See <http://www.ietf.org>.

# F

## Appendix F: Glossary of Terms

---

### Common Encryption and SSL Related Terms

asymmetric algorithm	An encryption algorithm which has two keys: a public key and private key, where the public key can be distributed openly while the private key is kept secret. Asymmetric algorithms may be capable of a number of operations, including encryption, digital signatures, and key agreement. Also known as a public key algorithm.
cipher suite	An SSL encryption method which includes the key exchange algorithm, the symmetric encryption algorithm, and the secure hash algorithm used to protect the integrity of the communication.
DES	A symmetric encryption algorithm with a 56-bit key. Also exists in a variant called 3DES or triple DES that involves using DES with three 56-bit keys. While this is 168 bits of key, an attack called the "meet in the middle" attack means that 3DES can be broken with the same effort as a 112-bit algorithm.
Diffie-Hellman	An asymmetric algorithm that allows key agreement: the two parties exchange public keys and use them in conjunction with their private keys to generate a shared secret. An eavesdropper who sees the public keys but does not have access to the private key of either party can not determine this secret.
DSA	The Digital Signature Algorithm-an asymmetric algorithm which allows you to create digital signatures.
DSS	The Digital Signature Standard, a US Government standard which combines DSA and SHA-1 to specify a format for signing data.
Elliptic Curve Cryptography (ECC)	A very efficient method of providing strong asymmetric cryptography through an elliptic curve analogue of the discrete log problem. Elliptic Curve key sizes are quite small compared to RSA key sizes, but provide the same or superior amounts of security. For example: the RSA equivalent of a 160-bit Elliptic Curve key is 1024-bits. ECC has the highest strength-per-bit of any known cryptosystem.
ECDSA	An Elliptic Curve analog of the Digital Signature Algorithm.
ECDH	An Elliptic Curve analog of the Diffie-Hellman key exchange algorithm.

IETF	The Internet Engineering Task Force, a group responsible for developing standards for use on the Internet.
MAC	A message authentication code: an algorithm that uses a secure hash algorithm and a secret key to assure the integrity of a message. The sender creates the MAC using the message and the secret key; the recipient verifies the message with the same secret key. No one who does not know the secret key can modify the message, because they cannot produce a MAC that can be verified with that key.
MD2	A hash algorithm created by Ron Rivest. This is an older implementation of MD5. (see below)
MD5	A secure hash algorithm created by Ron Rivest. MD5 takes an arbitrary input and hashes it into a 16-byte digest.
message digest algorithm	See secure hash algorithm.
PCT	Private Communications Technology, a protocol developed by Microsoft as a successor to SSL 2.0. Microsoft is now supporting the TLS effort to create an Internet standard for transport-level encrypted communication.
PKCS	The Public Key Cryptography Standards, a suite of specifications created by RSA to standardize cryptographic formats and operations.
PKCS#1: RSA Encryption Standard	A specification of data formats for the RSA protocol, including the formatting for RSA encryption and RSA signatures and a format for storing public and private keys.
PKCS#5: Password-Based Encryption Standard	A specification of a format for using password-based encryption with DES to protect data.
PKCS#8: Private-Key Information Syntax Standard	A specification of a format for storing private keys, including the ability to encrypt them with PKCS#5.
PKCS#10: Certification Request Syntax Standard	A specification of a standard format for encoding certificate requests, including the name of the person requesting the certificate and their public key.
public key algorithm	See asymmetric algorithm.
RC4	A symmetric algorithm designed by Ron Rivest which can use keys of variable length; it is usually used with a 40-bit or 128-bit key.
RSA	An asymmetric algorithm which provides the ability to encrypt data and create and verify digital signatures.



secret key algorithm	See symmetric algorithm.
secure hash algorithm	An algorithm which creates a unique hash value for each possible input. Cryptographically secure hash algorithms make it infeasible to find two messages that hash to the same value; this means that the hash of a message is as unique as the message itself. Thus, signing the hash is equivalent to signing the document, since the hash is uniquely tied to the document that produced it.
SHA-1	A secure hash algorithm specified by the NIST and used in the DSS. SHA-1 takes an arbitrary input and hashes it into a 20-byte digest.
SSL context	A structure that specifies all of the information associated with a particular SSL connection; <b>SSL Plus</b> uses an SSL context as a part of all its interfaces.
SSL	Secure Sockets Layer: a protocol that provides an encrypted and authenticated communications stream over a data transport. <b>SSL Plus</b> implements the SSL protocol, version 2.0 and 3.0.
symmetric algorithm	An encryption algorithm which only has one key, which is used for both encrypting and decrypting. This key must be kept secret to keep the encrypted messages secure. Also known as a secret key algorithm.
TLS	Transport Layer Security: an IETF working group which is standardizing a protocol based on SSL as an Internet standard for providing encrypted and authenticated communications. Also, the protocol the working group is specifying.
X.509	A standard which specifies the format of certificates, which provide a way to securely tie a name to a public key, allowing strong authentication.

