

# SSL Plus

Version 4.5.1

## Programmer's Reference

PUB-0300-0211  
November 7, 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  
905.507.4200



The *Programmer's Reference* describes the functions in the SSL Plus API.

Other documents provided with SSL Plus include the User's Guide and the Upgrade Guide. The The User's Guide document describes how to use the SSL Plus API to develop an application. The Upgrade Guide describes what existing customers of SSL Plus need to know to upgrade their applications to version 4.0.

#### **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

---

## The SSL API 1

Introduction .....	1
SSL Plus Global Context Functions .....	2
ssl_CreateGlobalContext() .....	3
ssl_DestroyGlobalContext() .....	5
ssl_GetGlobalMemRef() .....	6
ssl_GetGlobalcbData() .....	7
ssl_SetPRNG() .....	8
ssl_SetPkiPolicy() .....	9
ssl_CreateProtocolPolicy() .....	11
ssl_SetProtocolPolicy() .....	12
ssl_DestroyProtocolPolicy() .....	13
ssl_SetProtocolPolicyChallengeLength() .....	14
ssl_SetProtocolPolicyPointCompression() .....	15
ssl_SetProtocolPolicyRollbackFlag() .....	16
ssl_SetProtocolPolicyTLS1Extension() .....	17
ssl_SetSessionIdentifierLength() .....	19
ssl_SetProtocol() .....	20
ssl_SetCryptographicStrength() .....	22
ssl_SetClientAuthModes() .....	24
ssl_SetCipherSuites() .....	26
ssl_SetSessionExpiryTime() .....	28
ssl_SetIOSemantics() .....	29
ssl_CreateCertList() .....	31
ssl_AddCertificate() .....	34
ssl_DestroyPkcs12Object() .....	37
ssl_AddPkcs12Pfx() .....	38
ssl_ImportPkcs12PFX() .....	41
ssl_ExportPkcs12PFX() .....	45
ssl_GenerateRSAExportKeyPair() .....	49
ssl_SetServerDHParams() .....	51
ssl_AddIdentity() .....	54
ssl_AddTrustedCerts() .....	56
ssl_DestroyCertList() .....	58
SSL Plus Connection Context Functions .....	59
ssl_CreateConnectionContext() .....	60
ssl_DestroyConnectionContext() .....	63
ssl_GetConnectionMemRef() .....	64
ssl_GetConnectioncbData() .....	65
SSL Plus Data Exchange Functions .....	66
ssl_Handshake() .....	67
ssl_RequestRenegotiation() .....	71
ssl_Read() .....	74
ssl_Write() .....	77
ssl_GetReadPendingSize() .....	79
ssl_GetWritePendingSize() .....	80
ssl_ServiceWriteQueue() .....	81
ssl_Close() .....	82
SSL Plus Connection Status Functions .....	85

ssl_GetNegotiatedProtocolVersion()	86
ssl_GetContextProtocolVersion()	87
ssl_GetNegotiatedCipher()	88
ssl_GetMasterSecret()	89
ssl_GetClientRandom()	90
ssl_GetServerRandom()	92
ssl_WasSessionResumed()	94
ssl_GetSessionID()	95
ssl_FreeRecordBuffers()	96
SSL Plus Callback Set Functions	98
ssl_SetIOFuncs()	99
ssl_SetAlertFunc()	100
ssl_SetSessionFuncs()	101
ssl_SetCheckCertificateChainFunc()	103
ssl_SetLogFunc()	104
SSL Plus Random Seed Function	105
ssl_GenerateRandomSeed()	106
SSL Plus Certificate Functions	108
ssl_ExtractCertificateNameItem()	109
ssl_ExtractRawCertData()	111
SSL Plus Miscellaneous Functions	112
ssl_GetVersion()	113
ssl_DecodeRecord()	114
ssl_TLSPRF()	115
ssl_SendAlert()	117
SSL Plus API Protocol Objects	118
SSL_PROTOCOL_SSLV2_SERVERSIDE	119
SSL_PROTOCOL_SSLV2_CLIENTSIDE	120
SSL_PROTOCOL_SSLV3_SERVERSIDE	121
SSL_PROTOCOL_SSLV3_CLIENTSIDE	122
SSL_PROTOCOL_TLSV1_SERVERSIDE	123
SSL_PROTOCOL_TLSV1_CLIENTSIDE	124
SSL_PROTOCOL_SSLV3_SSLV2_SERVERSIDE	125
SSL_PROTOCOL_SSLV3_SSLV2_CLIENTSIDE	127
SSL_PROTOCOL_TLSV1_SSLV3_SSLV2_SERVERSIDE	129
SSL_PROTOCOL_TLSV1_SSLV3_SSLV2_CLIENTSIDE	131
SSL_PROTOCOL_TLSV1_SSLV3_SERVERSIDE	133
SSL_PROTOCOL_TLSV1_SSLV3_CLIENTSIDE	135
SSL Plus API Authentication Mode Objects	137
SSL_ALG_CLIENT_AUTH_MODE_RSA_SIGN_SERVERSIDE()	138
SSL_ALG_CLIENT_AUTH_MODE_RSA_SIGN_CLIENTSIDE()	139
SSL_ALG_CLIENT_AUTH_MODE_ECDSA_SIGN_SECT163K1_SERVERSIDE()	140
SSL_ALG_CLIENT_AUTH_MODE_ECDSA_SIGN_SECT163K1_CLIENTSIDE()	141
SSL_ALG_CLIENT_AUTH_MODE_ECDSA_FIXED_ECDH_SECT163K1_SERVERSIDE()	142
SSL_ALG_CLIENT_AUTH_MODE_ECDSA_FIXED_ECDH_SECT163K1_CLIENTSIDE()	143
SSL_ALG_CLIENT_AUTH_MODE_DSS_SIGN_CLIENTSIDE()	144
SSL_ALG_CLIENT_AUTH_MODE_DSS_SIGN_SERVERSIDE()	145
SSL_ALG_CLIENT_AUTH_MODE_RSA_SIGN_CLIENTSIDE_CS()	146
SSL_ALG_CLIENT_AUTH_MODE_RSA_SIGN_SERVERSIDE_CS()	147
SSL Plus API Ciphersuite Objects	148
SSL_ALG_CIPHER_ECDH_ECDSA_SECT163K1_WITH_RC4_128_SHA_SERVERSIDE()	149

---

SSL_ALG_CIPHER_ECDH_ECDSA_SECT163K1_WITH_RC4_128_SHA_CLIENTSIDE()	150
SSL_ALG_CIPHER_ECDH_ECDSA_SECT163K1_NULL_SHA_SERVERSIDE()	151
SSL_ALG_CIPHER_ECDH_ECDSA_SECT163K1_NULL_SHA_CLIENTSIDE()	152
SSL_ALG_CIPHER_RSA_WITH_RC4_128_MD5_SERVERSIDE()	153
SSL_ALG_CIPHER_RSA_WITH_RC4_128_MD5_CLIENTSIDE()	154
SSL_ALG_CIPHER_RSA_WITH_RC4_128_SHA_SERVERSIDE()	155
SSL_ALG_CIPHER_RSA_WITH_RC4_128_SHA_CLIENTSIDE()	156
SSL_ALG_CIPHER_RSA_WITH_AES_128_CBC_SHA_SERVERSIDE()	157
SSL_ALG_CIPHER_RSA_WITH_AES_128_CBC_SHA_CLIENTSIDE()	158
SSL_ALG_CIPHER_RSA_WITH_AES_256_CBC_SHA_SERVERSIDE()	159
SSL_ALG_CIPHER_RSA_WITH_AES_256_CBC_SHA_CLIENTSIDE()	160
SSL_ALG_CIPHER_RSA_WITH_3DES_EDE_CBC_SHA_SERVERSIDE()	161
SSL_ALG_CIPHER_RSA_WITH_3DES_EDE_CBC_SHA_CLIENTSIDE()	162
SSL_ALG_CIPHER_RSA_EXPORT_WITH_RC4_40_MD5_SERVERSIDE()	163
SSL_ALG_CIPHER_RSA_EXPORT_WITH_RC4_40_MD5_CLIENTSIDE()	164
SSL_ALG_CIPHER_RSA_EXPORT_WITH_DES40_CBC_SHA_SERVERSIDE()	165
SSL_ALG_CIPHER_RSA_EXPORT_WITH_DES40_CBC_SHA_CLIENTSIDE()	166
SSL_ALG_CIPHER_RSA_WITH_DES_CBC_SHA_SERVERSIDE()	167
SSL_ALG_CIPHER_RSA_WITH_DES_CBC_SHA_CLIENTSIDE()	168
SSL_ALG_CIPHER_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA_CLIENTSIDE()	169
SSL_ALG_CIPHER_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA_SERVERSIDE()	170
SSL_ALG_CIPHER_DHE_DSS_WITH_DES_CBC_SHA_CLIENTSIDE()	171
SSL_ALG_CIPHER_DHE_DSS_WITH_DES_CBC_SHA_SERVERSIDE()	172
SSL_ALG_CIPHER_DHE_DSS_WITH_3DES_EDE_CBC_SHA_CLIENTSIDE()	173
SSL_ALG_CIPHER_DHE_DSS_WITH_3DES_EDE_CBC_SHA_SERVERSIDE()	174
SSL_ALG_CIPHER_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA_CLIENTSIDE()	175
SSL_ALG_CIPHER_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA_SERVERSIDE()	176
SSL_ALG_CIPHER_DHE_RSA_WITH_DES_CBC_SHA_CLIENTSIDE()	177
SSL_ALG_CIPHER_DHE_RSA_WITH_DES_CBC_SHA_SERVERSIDE()	178
SSL_ALG_CIPHER_DHE_RSA_WITH_3DES_EDE_CBC_SHA_CLIENTSIDE()	179
SSL_ALG_CIPHER_DHE_RSA_WITH_3DES_EDE_CBC_SHA_SERVERSIDE()	180
SSL_ALG_CIPHER_DH_ANON_EXPORT_WITH_DES40_CBC_SHA_CLIENTSIDE()	181
SSL_ALG_CIPHER_DH_ANON_EXPORT_WITH_DES40_CBC_SHA_SERVERSIDE()	182
SSL_ALG_CIPHER_DH_ANON_EXPORT_WITH_RC4_40_MD5_CLIENTSIDE()	183
SSL_ALG_CIPHER_DH_ANON_EXPORT_WITH_RC4_40_MD5_SERVERSIDE()	184
SSL_ALG_CIPHER_DH_ANON_WITH_RC4_128_MD5_CLIENTSIDE()	185
SSL_ALG_CIPHER_DH_ANON_WITH_RC4_128_MD5_SERVERSIDE()	186
SSL_ALG_CIPHER_DH_ANON_WITH_DES_CBC_SHA_CLIENTSIDE()	187
SSL_ALG_CIPHER_DH_ANON_WITH_DES_CBC_SHA_SERVERSIDE()	188
SSL_ALG_CIPHER_DH_ANON_WITH_3DES_EDE_CBC_SHA_CLIENTSIDE()	189
SSL_ALG_CIPHER_DH_ANON_WITH_3DES_EDE_CBC_SHA_SERVERSIDE()	190
SSL_ALG_CIPHER_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA_CLIENTSIDE()	191
SSL_ALG_CIPHER_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA_SERVERSIDE()	192
SSL_ALG_CIPHER_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA_CLIENTSIDE()	193
SSL_ALG_CIPHER_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA_SERVERSIDE()	194
SSL_ALG_CIPHER_DHE_DSS_WITH_RC4_128_SHA_CLIENTSIDE()	195
SSL_ALG_CIPHER_DHE_DSS_WITH_RC4_128_SHA_SERVERSIDE()	196
SSL_ALG_CIPHER_RSA_EXPORT_WITH_RC4_40_MD5_CLIENTSIDE_CS()	197
SSL_ALG_CIPHER_RSA_EXPORT_WITH_RC4_40_MD5_SERVERSIDE_CS()	198
SSL_ALG_CIPHER_RSA_WITH_3DES_EDE_CBC_SHA_CLIENTSIDE_CS()	199
SSL_ALG_CIPHER_RSA_WITH_3DES_EDE_CBC_SHA_SERVERSIDE_CS()	200
SSL_ALG_CIPHER_RSA_WITH_AES_128_CBC_SHA_CLIENTSIDE_CS()	201
SSL_ALG_CIPHER_RSA_WITH_AES_128_CBC_SHA_SERVERSIDE_CS()	202
SSL_ALG_CIPHER_RSA_WITH_AES_256_CBC_SHA_CLIENTSIDE_CS()	203

SSL_ALG_CIPHER_RSA_WITH_AES_256_CBC_SHA_SERVERSIDE_CS()	204
SSL_ALG_CIPHER_RSA_WITH_RC4_128_MD5_CLIENTSIDE_CS()	205
SSL_ALG_CIPHER_RSA_WITH_RC4_128_MD5_SERVERSIDE_CS()	206
SSL_ALG_CIPHER_RSA_WITH_RC4_128_SHA_CLIENTSIDE_CS()	207
SSL_ALG_CIPHER_RSA_WITH_RC4_128_SHA_SERVERSIDE_CS()	208
SSL_ALG_CIPHER_RSA_EXPORT_WITH_DES40_CBC_SHA_SERVERSIDE_CS()	209
SSL_ALG_CIPHER_RSA_EXPORT_WITH_DES40_CBC_SHA_CLIENTSIDE_CS()	210
SSL_ALG_CIPHER_RSA_WITH_DES_CBC_SHA_SERVERSIDE_CS()	211
SSL_ALG_CIPHER_RSA_WITH_DES_CBC_SHA_CLIENTSIDE_CS()	212
SSL Plus API Private Key Decryption Suite Objects	213
SSL_ALG_PRV_KEY_DECRYPT_NULL	214
SSL_ALG_PRV_KEY_DECRYPT_PBE_MD5_DES	215
SSL Plus API Certificate Format Objects	216
SSL_CERT_FMT_X509()	217
SSL_CERT_FMT_X509_RSA()	218
SSL_CERT_FMT_X509_ECC()	219
SSL_CERT_FMT_X509_DSS()	220
SSL Plus API Key and Certificate Decoding Objects	221
SSL_ENC_PEM()	222
SSL_ENC_DER()	223
SSL Plus API PKCS #12 Certificate and Private Key Decryption Suite Objects	224
SSL_ALG_PKCS12_PBE_SHA_RC4_128()	225
SSL_ALG_PKCS12_PBE_SHA_RC4_40()	226
SSL_ALG_PKCS12_PBE_SHA_3DES()	227
SSL_ALG_PKCS12_PBE_SHA_2KEY_3DES()	228
SSL_ALG_PKCS12_PBE_SHA_RC2_128()	229
SSL_ALG_PKCS12_PBE_SHA_RC2_40()	230
SSL_ALG_PKCS12_PBE_MD2_DES()	231
SSL_ALG_PKCS12_PBE_MD2_RC2()	232
SSL_ALG_PKCS12_PBE_MD5_DES()	233
SSL_ALG_PKCS12_PBE_MD5_RC2()	234
SSL_ALG_PKCS12_PBE_SHA1_DES()	235
SSL_ALG_PKCS12_PBE_SHA1_RC2()	236
SSL Plus API RNG Objects	237
SSL_ALG_ANSIPRNG()	238
SSL Plus API Policy Objects	239
SSL_PKI_POLICY_SSLV2()	240
SSL_PKI_POLICY_SSLV3()	241
SSL_PKI_POLICY_TLSV1()	242
SSL_PKI_POLICY_WAPV2()	243
SSL_PKI_POLICY_MIDPV2()	244

## SSL Plus Callbacks 247

Mandatory Callbacks	247
cic_MallocFunc	248
cic_FreeFunc	249
cic_MemsetFunc	250
cic_MemcpyFunc	251
cic_MemcmpFunc	252
cic_TimeFunc	253
ssl_Callbacks	254

---

ssl_RandomFunc .....	255
ssl_IOFunc .....	256
Optional Callbacks .....	258
ssl_AlertFunc .....	259
ssl_AlertLevel .....	260
ssl_AlertCode .....	261
ssl_CheckCertificateChainFunc .....	265
ssl_SurrenderFunc .....	267
ssl_GetSessionFunc .....	268
ssl_AddSessionFunc .....	269
ssl_DeleteSessionFunc .....	270
ssl_LogFunc .....	271
ssl_CB_LogType .....	272
ssl_CB_LogSubType .....	273
Cipher Suite and Protocol Enumerated Types .....	275
ssl_CipherSuite .....	276
ssl_ProtocolVersion .....	277

## The SSL RAD API 279

Rapid Application Development .....	279
sslrad_MallocCallback() .....	280
sslrad_FreeCallback() .....	281
sslrad_MemsetCallback() .....	282
sslrad_MemcpyCallback() .....	283
sslrad_MemcmpCallback() .....	284
sslrad_TimeCallback() .....	285
sslrad_RandomCallback() .....	286
sslrad_CheckCertificateChainCallback() .....	287
sslrad_CreateSessionDB() .....	289
sslrad_DestroySessionDB() .....	290
sslrad_GetSessionCallback() .....	291
sslrad_AddSessionCallback() .....	292
sslrad_DeleteSessionCallback() .....	293
sslrad_SocketLayerInit() .....	294
sslrad_SocketLayerClose() .....	295
sslrad_SocketConnect() .....	296
sslrad_SocketCreateServer() .....	297
sslrad_SocketAccept() .....	298
sslrad_SocketClose() .....	299
sslrad_SocketReadCallback() .....	300
sslrad_SocketWriteCallback() .....	302
sslrad_InitializeCompleteServerGlobalContext() .....	303
sslrad_InitializeECCOnlyServerGlobalContext() .....	305
sslrad_InitializeRSAOnlyServerGlobalContext() .....	307
sslrad_InitializeCompleteNoClientAuthServerGlobalContext() .....	309
sslrad_InitializeECCOnlyNoClientAuthServerGlobalContext() .....	311
sslrad_InitializeRSAOnlyNoClientAuthServerGlobalContext() .....	313
sslrad_InitializeCompleteClientGlobalContext() .....	315
sslrad_InitializeECCOnlyClientGlobalContext() .....	317
sslrad_InitializeRSAOnlyClientGlobalContext() .....	319
sslrad_InitializeCompleteNoClientAuthClientGlobalContext() .....	321
sslrad_InitializeECCOnlyNoClientAuthClientGlobalContext() .....	323

---

`sslrad_initializeRSAOnlyNoClientAuthClientGlobalContext()` ..... 325



# 1

# The SSL API

---

## Introduction

This document describes the SSL Plus ( v4.0 ) API. It is split into three chapters. The first chapter contains a listing of all the SSL Plus API functions which can be used to secure an application, and the API objects which can be used in those functions. The second chapter contains the definitions for all the callback functions. The third chapter contains the SSL Plus RAD ( Rapid Application Development ) API functions. The SSL Plus RAD API is a simple " wrapper " interface that provides efficient implementations of all the callbacks required by SSL Plus. It also simplifies the process of creating a global context.

# SSL Plus Global Context Functions

This section describes the functions which create and configure a global context. The global context allows you to configure settings which apply to all the SSL connections you make. The configuration functions allow you to do the following:

- Indicate whether your application is an SSL server or an SSL client, and which version of the SSL protocol it supports.
- Set the cryptographic strength of a connection.
- Set the client authentication modes.
- Set the cipher suite list.
- Set the expiry period for SSL sessions.
- Set the behaviour of the **ssl\_Read()** function.
- Generate an exportable RSA key pair.
- Create a certificate list and add a certificate to it.
- Set the certificate list in the global context.
- Destroy a certificate list.
- Add trusted root certificates to a certificate list.

## ssl\_CreateGlobalContext( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_CreateGlobalContext(
    const ssl_Callbacks* sslCallbacks,
    ssl_GlobalContext** globalCtx
);
```

### Description

Creates a global context. A global context must be created and configured before a connection context is created ( see **ssl\_CreateConnectionContext( )** ).

This function is mandatory; you must call it in your application.

Notes:

( 1 ) The **sslCallbacks** parameter contains pointers to the application callbacks which will perform memory allocation, de-allocation, memory copy, memory compare, memory initialize, retrieve current time, generate a random seed, and yield/surrender during length crypto operations.

All these callback functions, with the exception of the surrender callback, are mandatory and must be initialized in the **sslCallbacks** structure.

( 2 ) Once a global context is configured you must not change it while connections exist that use this context. The only exception to this is the **ssl\_GenerateRSAExportKeyPair( )** function.

### Parameters

**sslCallbacks**

[ Input ] O/S callbacks.

**globalCtx**

[ Output ] Pointer to the global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SSL\_NULL\_CB**

NULL callback was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

## ssl\_DestroyGlobalContext( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_DestroyGlobalContext(  
    ssl_GlobalContext** globalCtx  
);
```

### Description

Releases the memory held by the global context. This should be the last SSL Plus function you call in your application.

This function is mandatory; you must call it in your application.

Notes:

( 1 ) The global context must not be destroyed until all the corresponding connection contexts have been destroyed.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

## ssl\_GetGlobalMemRef()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetGlobalMemRef(  
    const ssl_GlobalContext* globalCtx,  
    void** memRef  
);
```

### Description

Returns the memory reference stored in the global context.

### Parameters

**globalCtx**

[ Input ] Pointer to the global context.

**memRef**

[ Output ] Pointer to memory reference.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

## ssl\_GetGlobalcbData( )

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetGlobalcbData(  
    const ssl_GlobalContext* globalCtx,  
    void** cbData  
);
```

### Description

Returns the callback data contained in the global context. This data can then be passed to the TrustPoint C certificate parsing functions that take a **cbData** parameter.

### Parameters

**connCtx**

[ Input ] Pointer to the SSL connection context.

**cbData**

[ Output ] On output points to the callback data contained in the global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

## ssl\_SetPRNG( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetPRNG(  
    ssl_GlobalContext* globalCtx,  
    const ssl_PRNGProvider prng  
);
```

### Description

Sets the Pseudo-Random Number Generator provider in the global context. This function is mandatory.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**prng**

[ Input ] A PRNG object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_MEMORY**

Error allocating memory.



## ssl\_SetPkiPolicy()

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetPkiPolicy(
    ssl_GlobalContext* globalCtx,
    const ssl_PKIPolicyObject policy,
    ssl_ProtocolVersion protocolVersion
);
```

### Description

Overrides the default policy for the specified version of the SSL protocol.

See the description of the policy objects on page 239 to find out which PKI policies they each support.

See the description of the protocol objects on page 118 to find out which PKI policy they each install by default.

This function is useful, for example, in overriding the default TLS1 policy object with the WAP 2.0 policy object for applications which support WAP 2.0.

Notes:

( 1 ) This function should be called after **ssl\_SetProtocol**. Calling **ssl\_SetProtocol** after calling this function will overwrite the installed policy with the default policy.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**policy**

[ Input ] Pointer to the policy object to install.

**protocolVersion**

[ Input ] Protocol version that will use the specified PKI policy. Must be one of **SSL\_ProtocolVersion\_SSLV2**, **SSL\_ProtocolVersion\_SSLV3** or **SSL\_ProtocolVersion\_TLSV1**.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

`CIC_ERR_ILLEGAL_PARAM`

Protocol version not recognized.

## ssl\_CreateProtocolPolicy()

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_CreateProtocolPolicy(
    ssl_GlobalContext* globalCtx,
    ssl_ProtocolVersion protocolVersion,
    ssl_ProtocolPolicy** policy
);
```

### Description

Creates and initializes a protocol policy object. You can use this object to change the behavior of the SSL protocol given by **protocolVersion**.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

**protocolVersion**  
[ Input ] The protocol to be changed. Must be one of **SSL\_ProtocolVersion\_SSLV2**, **SSL\_ProtocolVersion\_SSLV3** or **SSL\_ProtocolVersion\_TLSV1**.

**policy**  
[Input/Output] Pointer to the policy object.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_NO\_PTR**  
NULL pointer was passed.

**CIC\_ERR\_VER\_INVALID**  
Invalid protocol version.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

## ssl\_SetProtocolPolicy()

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetProtocolPolicy(  
    ssl_GlobalContext* globalCtx,  
    ssl_ProtocolPolicy** policy  
);
```

### Description

Sets a protocol policy object in the global context.

You must initialize the policy object with **ssl\_CreateProtocolPolicy()** before calling this function.

After you call this function, the changes that you specified with **ssl\_SetProtocolPolicyChallengeLength()**, **ssl\_SetProtocolPolicyPointCompression()**, **ssl\_SetProtocolPolicyRollbackFlag()** or **ssl\_SetProtocolPolicyTLS1Extension()** take effect.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**policy**

[Input/Output] Pointer to the policy object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SSL\_PROTOCOL\_NOT\_INSTALLED**

Invalid protocol version.

## ssl\_DestroyProtocolPolicy()

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_DestroyProtocolPolicy(  
    ssl_GlobalContext* globalCtx,  
    ssl_ProtocolPolicy** policy  
);
```

### Description

Destroys a protocol policy object and frees all the memory allocated to it.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**policy**

[Input/Output] Pointer to the policy object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

## ssl\_SetProtocolPolicyChallengeLength( )

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetProtocolPolicyChallengeLength(  
    ssl_ProtocolPolicy* policy,  
    cic_Uint16 challengeLen  
);
```

### Description

Sets the challenge length for the SSL2 protocol.

Be warned that calling this function could reduce the number of servers that you can inter-operate with; if the server only accepts 16 bytes then an error occurs. However, the higher the number, the higher the security; this is one of the security holes that was fixed in SSL3 and TLS1.

This function overcomes the inconsistencies in the different specifications. The SSL2 specification says "The CHALLENGE-LENGTH must be greater than or equal to 16 and less than or equal to 32". The SSL3 and TLS1 specification says the SSL2 protocol challenge length "must be 32".

The problem is that some sites only allow a 16 byte challenge length for the SSL2 protocol.

### Parameters

**policy**  
[ Input/Output ] Pointer to the policy object

**challengeLen**  
[Input] The new challenge length.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_NO\_PTR**  
NULL pointer was passed.

**CIC\_ERR\_VER\_INVALID**  
Policy is not SSL2.

**CIC\_ERR\_ILLEGAL\_PARM**  
Challenge length not in valid range.

## ssl\_SetProtocolPolicyPointCompression( )

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetProtocolPolicyPointCompression(
    ssl_GlobalContext* globalCtx,
    ssl_ProtocolPolicy* policy,
    ssl_ECCPointCompressionMode pcmode,
    ssl_ECCCompressionGuessing pcguess
);
```

### Description

Sets the point compression mode (TLS1 protocol only).

Point compression behaviour is specified using two parameters: the default mode and the guessing mode.

Regardless of the mode, SSL Plus interprets both compressed and uncompressed points. If the guessing is turned off, SSL Plus always output points as specified by the default mode.

If guessing is set, SSL Plus tries to guess whether or not the other side understands compressed points. If guessing is possible, the output format is adjusted accordingly. Otherwise, SSL Plus rolls back to the default mode.

The default value of **pcmode** is **SSL\_ECC\_UNCOMPRESSED**.

The default value of **pcguess** is **TRUE**.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**policy**

[Input/Output] Pointer to the policy object.

**pcmode**

[Input] Specifies the default mode of point compression. Set to either **SSL\_ECC\_UNCOMPRESSED** or **SSL\_ECC\_COMPRESSED**. The default value is **SSL\_ECC\_UNCOMPRESSED**.

**pcguess**

[Input] Specifies whether or not SSL Plus should attempt to find out the remote application's support for compression. The default value is **TRUE**.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_SSL\_PROTOCOL\_VERSION\_INVALID**

A policy object set with a protocol other than TLS1 was passed in.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

## ssl\_SetProtocolPolicyRollbackFlag( )

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetProtocolPolicyRollbackFlag(  
    ssl_ProtocolPolicy* policy,  
    cic_Bool rollbackFlag  
);
```

### Description

Enables or disables the rollback flag (SSL3 only). It is enabled by default.

Notes:

- (1) Currently it only makes sense to turn on the rollback flag for SSLV3.
- (2) On the server side, setting the rollback flag to TRUE turns off the check for the protocol version encoded in the CLIENT\_KEY\_EXCHANGE message. This allows the server to communicate with a greater number of clients which may not follow the standard and encode the protocol version contained in the client "hello" message. Currently this only affects SSLV3. This is the default behavior (no checking).
- (3) On the client side, if the rollback flag is set to TRUE, then the negotiated protocol version is encoded in the CLIENT\_KEY\_EXCHANGE; otherwise the protocol version sent in the client "hello" is encoded, which is what is specified in rfc 2246. Currently this only affects SSLV3. This is the default behavior.

### Parameters

**policy**

[ Input/Output ] Pointer to the policy object.

**rollbackFlag**

[Input] Specifies the status of the rollback flag. It is set to **TRUE** by default.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SSL\_PROTOCOL\_VERSION\_INVALID**

A policy object set with a protocol other than SSL3 was passed in.



## ssl\_SetProtocolPolicyTLS1Extension( )

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetProtocolPolicyTLS1Extension(
    ssl_GlobalContext* globalCtx,
    ssl_TLS1ExtensionFunc tls1ExtFunc,
    ssl_ProtocolPolicy* policy
);
```

### Description

Be warned that calling this function could reduce the number of servers that you can inter-operate with; if the server has not implemented forward compatibility properly (which is where the maximum fragment length negotiation data will be placed), then it returns an error. This is especially true for servers that support only the SSL2 protocol.

To understand what this gives you we have quoted the first paragraph in the Maximum Fragment Length Negotiation section of the TLS extension INTERNET-DRAFT document. For more details consult the specification.

"[TLS] specifies a fixed maximum plaintext fragment length of  $2^{14}$  bytes. It may be desirable for constrained clients to negotiate a smaller maximum fragment length due to memory limitations or bandwidth limitations."

Concerning forward compatibility, the TLS specification (rfc2246) states the following in the client hello section. This is also echoed in the SSL3 specification, but not the SSL2 specification.

"Forward compatibility note:

In the interests of forward compatibility, it is permitted for a client hello message to include extra data after the compression methods. This data must be included in the handshake hashes, but must otherwise be ignored. This is the only handshake message for which this is legal; for all other messages, the amount of data in the message must match the description of the message precisely."

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**ssl\_TLS1ExtensionFunc**

[Input] Pointer to the function for decoding/encoding the **max\_fragment\_length** extension.

**policy**

[Input/Output] Pointer to the policy object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

`CIC_ERR_SSL_PROTOCOL_VERSION_INVALID`

A policy object set with a protocol other than TLS1 was passed in.

`CIC_ERR_SSL_BAD_SIDE`

The protocol side of this mode different from that selected with **`ssl_CreateProtocolPolicy( )`** or another function.

## ssl\_SetSessionIdentifierLength()

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetSessionIdentifierLength(
    ssl_GlobalContext* globalCtx,
    const cic_Uint16 sessionIdentifierLength
);
```

### Description

Sets the maximum length for the session identifier on the server side when session resumption is enabled.

Notes:

(1) The default session identifier length is 32 bytes for SSLV3 and TLSV1 and 16 bytes for SSLV2.

(2) This function sets the default length of the session id generated by the server. This function has no effect on the client side. The client will still accept the default values for the session identifier.

(3) For TLSV1 and SSLV3, the accepted lengths are 1-32 bytes. For SSLV2 the accepted length is 16 bytes.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sessionIdentifierLength**

[ Input ] The identifier length (in bytes).

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_LENGTH**

The session ID length was not between 0 .. 32.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

## ssl\_SetProtocol()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetProtocol(  
    ssl_GlobalContext* globalCtx,  
    const ssl_ProtocolSide side  
);
```

### Description

Indicates whether your application behaves as an SSL server or an SSL client and the protocol versions it supports.

This function is mandatory; you must call it in your application.

The **ssl\_ProtocolSide** objects begin on page 118.

Notes:

- ( 1 ) If the protocol side differs from that previously set with a client authentication mode object in the function **ssl\_SetClientAuthModes()** , an error is returned.
- ( 2 ) If the protocol side differs from that previously set with a ciphersuite object in the function **ssl\_SetCiphersuites()** , an error is returned.
- ( 3 ) Once set, only the supported protocols can be changed. The protocol side ( server or client ) cannot be changed. However, you must not change the supported protocols once the connection contexts have been created or the results will be unpredictable.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**side**

[ Input ] Specifies whether your application is a client or server and which SSL protocol versions it supports.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side ( of this protocol ) is different from that selected with a previous protocol, or with **ssl\_SetCipherSuite ( )**, or with **ssl\_SetClientAuthModes( )**.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## ssl\_SetCryptographicStrength( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetCryptographicStrength(
    ssl_GlobalContext* globalCtx,
    const ssl_CryptoStrength strength
);
```

### Description

Allows you to set the cryptographic strength of an SSL connection. This function is optional.

The possible settings are:

**SSL\_CryptoStrength\_AllCrypto** -Both exportable and non-exportable cryptography

**SSL\_CryptoStrength\_StrongCryptoOnly** -Non-exportable cryptography

**SSL\_CryptoStrength\_ExportCryptoOnly** -Exportable cryptography

The strength of a connection is determined by the cipher suite chosen during the negotiation. The restrictions applied by this function only take effect after the cipher suites are set with **ssl\_SetCipherSuites( )**.

Strong cipher suites:

**TLS\_RSA\_WITH\_RC4\_128\_MD5**

**TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA**

**TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA**

**TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA**

**TLS\_ECDH\_ECDSA\_WITH\_RC4\_128\_SHA**

Exportable cipher suites:

**TLS\_RSA\_WITH\_RC4\_40\_MD5**

**TLS\_ECDH\_ECDSA\_NULL\_SHA**

The **SSL\_CryptoStrength\_AllCrypto** setting includes all of the cipher suites listed above.

Notes:

( 1 ) Calling this API on a SSL context which has already completed a handshake will have no effect unless the handshake is renegotiated.

( 2 ) The cryptographic strength must not be changed while any connection context created from this global context is being used in a handshake. Otherwise, the results will be unpredictable.

**Parameters****globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**strength**

[ Input ] Cryptographic strength to support.

**Return Values****CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_ILLEGAL\_PARAM**Value of the **strength** parameter is invalid.

## ssl\_SetClientAuthModes( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetClientAuthModes(  
    ssl_GlobalContext* globalCtx,  
    const ssl_ClientAuthMode modes [ ]  
);
```

### Description

Sets the client authentication modes. This function is optional.

Use this in a server application to indicate that client authentication is required and to set the preferred order of the authentication modes. Use this in a client application to add support for client authentication and set the preferred order of the authentication modes.

If client authentication is not required there is no need to call this function.

Note that the library always enforces server authentication.

The **ssl\_ClientAuthMode** objects begin on page 137.

Notes:

( 1 ) The order of the modes is important. Modes earlier in the list take priority over later ones. The last element in the list must be **NULL**.

( 2 ) Each of the mode objects identifies the side of the protocol as either **CLIENTSIDE** or **SERVERSIDE** . All installed modes must have the same side.

( 3 ) If the protocol side of the mode differs from that set previously with **ssl\_SetProtocol()** or **ssl\_SetCipherSuites()** , an error is returned.

( 4 ) If **modes [ 0 ] == NULL** ( no modes to install ) this function does nothing and returns **CIC\_ERR\_NONE** .

( 5 ) The modes must not be changed while any connection context created from this global context is being used in a handshake. Otherwise, the results will be unpredictable.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**modes**

[ Input ] A ( **NULL**-terminated ) array of pointers to client authentication mode objects.



## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

`CIC_ERR_SSL_BAD_SIDE`

The protocol side of this mode is different from that selected with a previous mode, or with `ssl_SetProtocol()`, or with `ssl_SetCipherSuites()`.

## ssl\_SetCipherSuites()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetCipherSuites(
    ssl_GlobalContext* globalCtx,
    const ssl_CipherSuite ciphersuites [ ]
);
```

### Description

Adds a list of cipher suites to the global context.

The library uses these suites as negotiation options during the handshake stage of an SSL connection. When the handshake has finished ( without any errors ), one of the cipher suites will have been selected for the data transfer stage of the connection.

This function is mandatory; you must call it in your application.

The **ssl\_CipherSuite** objects begin on page 148.

Notes:

( 1 ) The order of the suites is important: suites earlier in the list take priority over later ones. The last element in a suites list must be **NULL**. Client applications with fewer suites will have a smaller code space than those with more suites.

( 2 ) Each of the cipher suite objects identifies the side of the protocol as either **CLIENTSIDE** or **SERVERSIDE** . All installed cipher suites must have the same protocol side.

( 3 ) If the protocol side of the ciphersuite differs from that set previously with **ssl\_SetProtocol()** or **ssl\_SetClientAuthModes()** , an error is returned.

( 4 ) If the list is empty ( **ciphersuites [ 0 ] == NULL** ), this function does nothing and simply returns **CIC\_ERR\_NONE** . If you attempt to set up a secure connection without any installed cipher suites the handshake process will fail and return the error **CIC\_ERR\_SSL\_NEGOTIATION** .

( 5 ) The installed ciphersuites must not be changed while any connection context created from the global context is being used in a handshake. Otherwise, the results will be unpredictable.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**ciphersuites**

[ Input ] A ( **NULL**-terminated ) array of pointers to ciphersuite objects.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

`CIC_ERR_SSL_BAD_SIDE`

The protocol side of this cipher suite is different from that selected with a previous cipher suite, or with `ssl_SetProtocol()`, or with `ssl_SetClientAuthModes()`.

## ssl\_SetSessionExpiryTime()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetSessionExpiryTime(  
    ssl_GlobalContext* globalCtx,  
    const cic_Uint32 sessionExpiryTime  
);
```

### Description

Sets the expiry period for SSL sessions. This function is optional.

Notes:

- ( 1 ) The default expiry period is 24 hours, set when the SSL global context is created.
- ( 2 ) RFC 2246 recommends that the expiry period be no greater than 24 hours.
- ( 3 ) The expiry period must not be changed while any connection context created from this global context is being used in a handshake. Otherwise, the results will be unpredictable.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sessionExpiryTime**

[ Input ] Expiry period ( in seconds ).

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

## ssl\_SetIOSemantics()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetIOSemantics(
    ssl_GlobalContext* globalCtx,
    const ssl_IOSemantics ioStyle
);
```

### Description

Sets the behavior of the **ssl\_Read()** function. This function is optional.

The **ssl\_Read()** function can return after the read request has either been partially or completely fulfilled. To use partial I/O, set **ioStyle** to **SSL\_IOSemantics\_PartialIO**. To use complete I/O, set **ioStyle** to **SSL\_IOSemantics\_CompleteIO**.

The default **ioStyle** is **SSL\_IOSemantics\_CompleteIO** ( set when the context is initialized with **ssl\_CreateGlobalContext()** ).

Notes:

- ( 1 ) The I/O semantics only apply to **ssl\_Read()** .
- ( 2 ) The default **ioStyle** is set to **SSL\_IOSemantics\_CompleteIO** when the context is initialized with **ssl\_CreateGlobalContext()** .
- ( 3 ) The I/O semantics must not be changed while any connection context is reading data ( with **ssl\_Read()** ). Otherwise, results will be unpredictable.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**ioStyle**

[ Input ] Specifies the I/O semantics. Set to either **SSL\_IOSemantics\_CompleteIO** or **SSL\_IOSemantics\_PartialIO** .

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

`CIC_ERR_ILLEGAL_PARAM`

Value of the **ioStyle** parameter is invalid.

## ssl\_CreateCertList( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_CreateCertList(
    ssl_GlobalContext* globalCtx,
    const cic_Uint32 prvKeyLength,
    const cic_Uint8* prvKeyData,
    const ssl_Encoding prvKeyEncoding,
    const ssl_PrivKeyDecryptionSuite prvKeyDecrSuite,
    const cic_Uint32 prvKeyPwdLength,
    const cic_Uint8* prvKeyPwdData,
    ssl_CertList** list
);
```

### Description

Creates an empty certificate list. You must call this function if you use **ssl\_AddCertificate()**.

If you supply the private key data, this function creates a certificate chain, otherwise it builds a list of trusted certificates.

Notes:

( 1 ) For Base64/PEM private keys, the key data must be enclosed between

"-----BEGIN ENCRYPTED PRIVATE KEY-----" and

"-----END ENCRYPTED PRIVATE KEY-----"

or

"-----BEGIN PRIVATE KEY-----" and

"-----END PRIVATE KEY-----".

( 2 ) If the **prvKeyData** contains an RSA private key, the global context must have been previously configured with at least one RSA ciphersuite ( see **ssl\_SetCiphersuites()** ) or the RSA client authentication suite ( see **ssl\_SetClientAuthModes()** ).

( 3 ) If the **prvKeyData** contains an ECC private key, the global context must have been previously configured with at least one ECC ciphersuite ( see **ssl\_SetCiphersuites()** ) or the ECC client authentication suite ( see **ssl\_SetClientAuthModes()** ).

### Parameters

**globalCtx**

[ Input ] Pointer to the SSL global context.

**prvKeyLength**

[ Input ] Size of the private key data.

**prvKeyData**

[ Input ] The private key to be used with this list. Pass NULL if not applicable.

**prvKeyEncoding**

[ Input ] Specifies the encoding of the private key.

The **ssl\_Encoding** objects begin on page 221.

**prvKeyDecrSuite**

[ Input ] Specifies the private key decryption used to decrypt the private key.

The **ssl\_PrivKeyDecryptionSuite** objects begin on page 213.

**prvKeyPwdLength**

[ Input ] Length of the private key password.

**prvKeyPwdData**

[ Input ] Password for the private key. Pass NULL if not applicable.

**list**

[ Output ] Pointer to a new, empty certificates list.

## Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_ILLEGAL\_PARAM**

Bad input, for example **prvKeyLength** is equal to zero and **prvKeyData** is not NULL.

**CIC\_ERR\_SSL\_NEEDS\_CIPHER\_OR\_CLIENTAUTH**

Before installing a private key, a ciphersuite or client authentication suite using the same key exchange algorithm ( RSA vs ECC ) must be installed first.



`CIC_ERR_SSL_NEEDS_PRNG`

PRNG not set yet. A PRNG must be installed first by calling `ssl_SetPRNG( )`.

## ssl\_AddCertificate()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_AddCertificate(
    ssl_CertList* list,
    const cic_Uint32 certLength,
    const cic_Uint8* certData,
    const ssl_Encoding certEncoding,
    const ssl_CertFormat certFormat,
    const cic_AllocationType certAllocationType
);
```

### Description

Adds one or more base64-encoded certificates to a certificate list.

You must call this function if you are creating a local identity or a trusted certificate list.

Notes:

- ( 1 ) The **list** must first be created with **ssl\_CreateCertList()**.
- ( 2 ) This function can be called multiple times; once for each certificate to be added. Alternatively, multiple base64/PEM or DER-encoded certificates may be concatenated together and supplied as the **certData** in a single call.
- ( 3 ) For base64/PEM certificates, the certificate data must be enclosed between "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----".
- ( 4 ) Multiple base64/PEM certificates can be concatenated and any data which is not enclosed between "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----" is ignored. This includes any leading or trailing data.
- ( 5 ) Two certificate encodings are supported:
  - SSL\_ENC\_PEM** -refers to base64 DER-encoded certificates with PEM headers.
  - SSL\_ENC\_DER** -refers to DER-encoded certificates.

### Parameters

**list**

[ Input/Output ] List of certificates to which to add the new certificate.

**certLength**

[ Input ] Contains the length ( in bytes ) of the certificate data.

**certData**

[ Input ] Pointer to the certificate data.

**certEncoding**

[ Input ] Certificate encoding.

The **ssl\_Encoding** objects begin on page 221.

**certFormat**

[ Input ] Certificate format.

The **ssl\_CertFormat** objects begin on page 216.

**certAllocationType**

[ Input ] If set to **CIC\_ALLOCATION\_POINTER\_COPY**, the data itself will not be copied. Instead, the function stores a pointer to the data. Your application is then responsible for keeping the pointer valid during the lifetime of the certificate list ( which is the lifetime of the global context ).

This feature is only supported when **certEncoding** is set to **SSL\_ENC\_DER**. In other words, it supports raw/DER certificate data, but not base64.

## Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_BAD\_LENGTH**

**certLength** is invalid.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_DER\_BAD\_ENCODING**

Certificate has incorrect DER encoding.

**CIC\_ERR\_BASE64\_BAD\_ENCODING**

Certificate has incorrect base64 encoding.

`CIC_ERR_BASE64_BAD_PEM`

Certificate has incorrect PEM encoding.

## ssl\_DestroyPkcs12Object( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_DestroyPkcs12Object(  
    ssl_GlobalContext* globalCtx,  
    cic_Uint32 objectsLength,  
    ssl_Pkcs12Object* objects  
);
```

### Description

Frees any memory associated with a Pkcs12Object array allocated by calling **ssl\_ImportPkcs12PFX( )**.

### Parameters

**globalCtx**

[ Input ] Pointer to the global context.

**objectsLength**

[ Input ] Length of the objects array.

**objects**

[ Input/Output ] Pointer to an array of ssl\_Pkcs12Objects.

### Return Values

**CIC\_ERR\_NONE**

Success.

## ssl\_AddPkcs12Pfx()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_AddPkcs12Pfx(
    ssl_CertList* list,
    const cic_Uint32 pfxlength,
    const cic_Uint8* const pfxdata,
    const cic_Uint32 pswdLength,
    const cic_Uint8* const pswdData,
    const ssl_P12EncryptionSuite suites [ ]
);
```

### Description

Parses a PKCS #12 PFX and adds the certificates and private key to a certificate list. The **list** must first be created with **ssl\_CreateCertList()**. The list can represent a local identity, or a set of trusted certificates.

Notes:

- ( 1 ) This function assumes that the PFX contains at most a single private key. If more than one private key is found, the error **CIC\_ERR\_SSL\_INVALID\_PFX** is returned.
- ( 2 ) This function assumes that the PFX uses the same password for data encryption, private key decryption, and MAC verification.
- ( 3 ) This function assumes that the integrity mode of the PFX is Password integrity. If the integrity mode is Public-key integrity, then the error **CIC\_ERR\_PKCS\_NEED\_TRUSTED** is returned.
- ( 4 ) This function assumes that the privacy mode of the PFX is Password privacy. If the privacy mode is Public-Key privacy, then the error **CIC\_ERR\_PKCS\_NEED\_PRV\_KEY** is returned.
- ( 5 ) If the **pfxData** contains an RSA private key, the global context must have been previously configured with at least one RSA ciphersuite ( see **ssl\_SetCipherSuites()** ) or the RSA client authentication suite ( see **ssl\_SetClientAuthModes()** ).
- ( 6 ) If the **pfxData** contains an ECC private key, the global context must have been previously configured with at least one ECC ciphersuite ( see **ssl\_SetCipherSuites()** ) or the ECC client authentication suite ( see **ssl\_SetClientAuthModes()** ).
- ( 7 ) The **suites** parameter is used to identify the algorithms used for certificate and private key encryption in the **pfxData**. You must identify each of the algorithms used by the **pfxData** in order to parse and decrypt the PFX. To minimize code size, only include the algorithms used by the PFX. If a required algorithm is not present the error **CIC\_ERR\_PKCS12\_MISSING\_ALG** is returned.

## Parameters

<b>list</b>	[ Input/Output ] List of certificates to which to add the certificate ( s ) and/or private key contained in the PKCS #12 PFX.
<b>pfxLength</b>	[ Input ] Contains the length ( in bytes ) of the the PKCS #12 PFX data.
<b>pfxData</b>	[ Input ] Pointer to the PKCS #12 PFX data.
<b>pswdLength</b>	[ Input ] Contains the length ( in bytes ) of the password.
<b>pswdData</b>	[ Input ] Pointer to the password data.
<b>suites</b>	[ Input ] A ( NULL-terminated ) array of pointers to PKCS #12 PFX suite decryption objects.

## Return Values

<b>CIC_ERR_NONE</b>	Success.
<b>CIC_ERR_NO_PTR</b>	NULL pointer was passed.
<b>CIC_ERR_MEMORY</b>	Error allocating memory.
<b>CIC_ERR_BAD_LENGTH</b>	<b>pfxLength</b> or <b>pswdLength</b> is invalid.
<b>CIC_ERR_DER_BAD_ENCODING</b>	PFX has incorrect DER encoding.
<b>CIC_ERR_PKCS_BAD_INPUT</b>	<b>pfxData</b> specifies an invalid PKCS #12 PFX.

`CIC_ERR_PKCS_BAD_VERSION`

Bad version

`CIC_ERR_PKCS_AUTH_FAILED`

Authentication of the PKCS #12 PFX failed.

`CIC_ERR_PKCS_NEED_TRUSTED`

The PFX is invalid for this operation because the PFX is protected using Public-key integrity mode, requiring a user to provide the public key to parse the PFX.

`CIC_ERR_PKCS_NEED_PRV_KEY`

The PFX is invalid for this operation because the PFX is protected using Public-key privacy mode, requiring a user to provide the private key to parse the PFX.

`CIC_ERR_SSL_INVALID_PFX`

The PFX is invalid for this operation, containing either no certificates and no private key, or multiple private keys.

`CIC_ERR_PKCS12_MISSING_ALG`

The PFX uses algorithm ( s ) which have not been supported through the **suites** parameter.

`CIC_ERR_SSL_NEEDS_PRNG`

PRNG not set yet. A PRNG must be installed first by calling **ssl\_SetPRNG( )**.



## ssl\_ImportPkcs12PFX( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_ImportPkcs12PFX(
    ssl_GlobalContext* globalCtx,
    const cic_Uint32 pfxLength,
    const cic_Uint8* pfx,
    const cic_Uint32 decryptPswdLength,
    const cic_Uint8* decryptPswd,
    const ssl_Pl2EncryptionSuite suites [ ],
    cic_Uint32* objectsLength,
    ssl_Pkcs12Object* objects
);
```

### Description

Parses a PKCS #12 PFX, returning all certificates and any private keys in an array.

( 3 ) This function assumes that the PFX uses the same password for data encryption, private key decryption, and MAC verification.

( 4 ) This function assumes that the integrity mode of the PFX is Password integrity mode. If the integrity mode is Public-key integrity mode, then the error **CIC\_ERR\_PKCS\_NEED\_TRUSTED** is returned.

( 5 ) This function assumes that the privacy mode of the PFX is Password privacy mode. If the privacy mode is Public-key privacy mode, then the error **CIC\_ERR\_PKCS\_NEED\_PRV\_KEY** is returned.

( 6 ) If the **pfxData** contains an RSA private key, the global context must have been previously configured with at least one RSA ciphersuite ( see **ssl\_SetCipherSuites( )** ) or the RSA client authentication suite ( see **ssl\_SetClientAuthModes( )** ).

( 7 ) If the **pfxData** contains an ECC private key, the global context must have been previously configured with at least one ECC ciphersuite ( see **ssl\_SetCipherSuites( )** ) or the ECC client authentication suite ( see **ssl\_SetClientAuthModes( )** ).

( 7 ) If the **pfxData** contains an ECC private key, the global context must have been previously configured with at least one ECC ciphersuite ( see **ssl\_SetCipherSuites( )** ) or the ECC client authentication suite ( see **ssl\_SetClientAuthModes( )** ).

( 8 ) The **suites** parameter is used to identify the algorithms used for Certificate and Private key encryption in the **pfxData** . It is necessary to identify each of the algorithms used by the **pfxData** in order to parse and decrypt the PFX. To minimize application code size, only include the algorithms used by the PFX. If a required algorithm is not present the error **CIC\_ERR\_PKCS12\_MISSING\_ALG** is returned.

## Parameters

**globalCtx**

[ Input ] Pointer to the global context.

**pfxLength**

[ Input ] Length of the PFX buffer.

**pfx**

[ Input ] Pointer to the PFX to parse.

**decryptPswdLength**

[ Input ] Length of the password to decrypt with.

**decryptPswd**

[ Input ] Pointer to the password to be used for data decryption.

**suites**

[ Input ] A ( NULL-terminated ) array of pointers to PKCS #12 PFX suite decryption objects.

**objectsLength**

[ Input/Output ] On input, this value defines the length of the objects array. On output, this value defines the number of objects in the array.

**objects**

[ Output ] Pointer to an array of **ssl\_Pkcs12Objects**. ( Any private keys in the pfx will be returned in a BASE64/PEM encoded PKCS #8 structure. Any certificates in the pfx structure will be returned as a BASE64/PEM encoded certificate within the array of ssl\_Pkcs12Objects. A secret is simply a string of octets. ) If this objects is null, then objects length will contain the required size of the objects array. Memory is allocated for the different parameters within the structure. To free this memory call **ssl\_DestroyPkcs12Object()**.

## Return Values

**CIC\_ERR\_NONE**

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

`CIC_ERR_BAD_LENGTH`

`pfxLength` or `decryptPswdLength` is invalid.

`CIC_ERR_DER_BAD_ENCODING`

PFX has incorrect DER encoding.

`CIC_ERR_PKCS_BAD_INPUT`

`pfx` specifies an invalid PKCS #12 PFX.

`CIC_ERR_PKCS_BAD_VERSION`

Bad version

`CIC_ERR_PKCS_AUTH_FAILED`

Authentication of the PKCS #12 PFX failed.

`CIC_ERR_PKCS_NEED_TRUSTED`

The PFX is invalid for this operation because the PFX is protected using Public-key integrity mode, requiring a user provided public key to parse the PFX.

`CIC_ERR_PKCS_NEED_PRV_KEY`

The PFX is invalid for this operation because the PFX is protected using Public-key privacy mode, requiring a user provided private key to parse the PFX.

`CIC_ERR_PKCS12_MISSING_ALG`

The PFX uses algorithm ( s ) which have not been supported through the `suites` parameter.



## ssl\_ExportPkcs12PFX( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_ExportPkcs12PFX(
    ssl_GlobalContext* globalCtx,
    const cic_Uint32 certificateLength,
    const cic_Uint8* certificate,
    const ssl_CertFormat certFormat,
    const cic_Uint8* friendlyName,
    const cic_Uint32 prvKeyLength,
    const cic_Uint8* prvKey,
    const ssl_Encoding encoding,
    const ssl_PrivKeyDecryptionSuite prvKeyDecrSuite,
    const cic_Uint32 prvKeyPwdLength,
    const cic_Uint8* prvKeyPwdData,
    const cic_Uint32 pswdLength,
    const cic_Uint8* pswdData,
    const cic_Uint16 iterations,
    const ssl_Pl2EncryptionSuite certificateAlg,
    const ssl_Pl2EncryptionSuite keyAlg,
    cic_Uint32* pfxLength,
    cic_Uint8** pfxData
);
```

### Description

Exports a certificate and a private key into a PKCS 12 PFX.

This is a high level function for exporting certificate and public key into a PFX. This function assumes that the PFX uses the same password for data encryption, private key decryption, and MAC verification.

Notes:

- ( 1 ) Memory is allocated for the PFX buffer and should be freed by the caller.
- ( 2 ) The PFX will use the same password for data encryption, private key decryption, and MAC verification.
- ( 3 ) The resulting integrity mode of the PFX is Password integrity mode.
- ( 4 ) This function assumes that the privacy mode of the PFX is Password privacy mode.
- ( 5 ) If the **prvKey** is an RSA private key, the global context must have been previously configured with at least one RSA ciphersuite ( see **ssl\_SetCipherSuites( )** ) or the RSA client authentication suite ( see **ssl\_SetClientAuthModes( )** ).
- ( 6 ) If the **prvKey** is an ECC private key, the global context must have been previously configured with at least one ECC ciphersuite ( see

**ssl\_SetCipherSuites()** ) or the ECC client authentication suite ( see **ssl\_SetClientAuthModes()** ).

( 7 ) The **certificateAlg** parameter is used to identify the algorithm used for Certificate encryption in the **pfxDat**a . It is necessary to identify the algorithm in order to encrypt the PFX. If a required algorithm is not present the error **CIC\_ERR\_PKCS12\_MISSING\_ALG** is returned.

( 8 ) The **keyAlg** parameter is used to identify the algorithm used for Private Key encryption in the **pfxDat**a . It is necessary to identify the algorithm in order to encrypt the PFX. If a required algorithm is not present the error **CIC\_ERR\_PKCS12\_MISSING\_ALG** is returned.

( 9 ) For a base64/PEM certificate, the certificate data must be enclosed between "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----".

( 10 ) Two certificate encodings are supported: **SSL\_ENCODING\_PEM** -refers to base64 DER-encoded certificates with PEM headers. **SSL\_ENCODING\_DER** -refers to DER-encoded certificates.

## Parameters

**globalCtx**

[ Input ] Pointer to the global context.

**certificateLength**

[ Input ] Certificate length.

**certificate**

[ Input ] Pointer to the certificate data.

**certFormat**

[ Input ] Specifies the certificate format. See **ssl\_CertFormat** .

**friendlyName**

[ Input ] Pointer to the friendly name attribute.

**prvKeyLength**

[ Input ] Length of the private key to be exported.

**prvKey**

[ Input ] Pointer to a PEM encoded Private Key to be exported.

**encoding**

[ Input ] Specifies the encoding of the private key and certificate

**prvKeyDecrSuite**

[ Input ] Specifies the private key decryption used to decrypt the private key.

**prvKeyPwdLength**

[ Input ] Length of the private key password.

**prvKeyPwdData**

[ Input ] Password for the private key. Pass NULL if not applicable.

**pswdLength**

[ Input ] Length of the password.

**pswdData**

[ Input ] Pointer to the password for data encryption.

**iterations**

[ Input ] Number of iterations to use for key derivation

**certificateAlg**

[ Input ] Encryption algorithm to use on the certificate.

**keyAlg**

[ Input ] Encryption algorithm to use on the private key.

**pfxLength**

[ Output ] Length of the resulting PFX.

**pfxData**

[ Output ] Pointer to the calculated PFX. Memory is allocated for this buffer.  
( PFX is a BASE64 DER encoded object. )

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

`CIC_ERR_PKCS12_MISSING_ALG`

An encryption algorithm was not specified.

`CIC_ERR_ILLEGAL_PARAM`

Bad input, for example `certificateLength` is zero but `certificate` is not NULL.

`CIC_ERR_DER_BAD_ENCODING`

PFX has incorrect DER encoding.

`CIC_ERR_PKCS_BAD_INPUT`

`pfx` specifies an invalid PKCS #12 PFX.

`CIC_ERR_SSL_NEEDS_PRNG`

PRNG not set yet. A PRNG must be installed first by calling `ssl_SetPRNG()`.



## ssl\_GenerateRSAExportKeyPair( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GenerateRSAExportKeyPair(
    ssl_GlobalContext* globalCtx
);
```

### Description

Generates and adds an RSA export key pair to a global context.

Notes:

( 1 ) Before calling this function to generate the RSA export keys, the ciphersuite **SSL\_ALG\_CIPHER\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5\_SERVERSIDE( )** must be installed in the global context ( see **ssl\_SetCipherSuites( )** ).

( 2 ) Only servers using RSA export ciphersuites need an RSA export key pair.

( 3 ) The key pair generated will be used in all new connections created from this global context. This function will not change the RSA key pair of any existing connection.

( 4 ) This function is most useful in servers that want to pre-generate the RSA export key pair before creating a number of connections, so that each connection handshake does not have to incur the cost of generating an RSA key pair.

( 5 ) If a connection context is created prior to calling this function then it will not have a pre-generated RSA export key pair, and a key pair will be generated automatically during the handshake.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_MEMORY**

Error allocating memory.

`CIC_ERR_SSL_BAD_SIDE`

Protocol side must be server.

`CIC_ERR_SSL_NEEDS_CIPHER_OR_CLIENTAUTH`

The ciphersuite

`SSL_ALG_CIPHER_RSA_EXPORT_WITH_RC4_40_MD5_SERVERSIDE()` must be installed in the global context first.

`CIC_ERR_SSL_NEEDS_PRNG`

PRNG not set yet. A PRNG must be installed first by calling `ssl_SetPRNG()`.

## ssl\_SetServerDHParams ( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetServerDHParams(
    ssl_GlobalContext* globalCtx,
    const cic_Uint32 pLength,
    const cic_Uint8* pData,
    const cic_Uint32 gLength,
    const cic_Uint8* gData,
    const cic_AllocationType allocationType
);
```

### Description

Sets the Diffie-Hellman parameters necessary to negotiate Diffie-Hellman key exchanges.

Notes:

( 1 ) Only server side applications need to call this function.

( 2 ) Diffie-Hellman parameters can be set with this function or with **ssl\_SetServerDHParamsFromCert** , and are required to negotiate any of the following cipher suites:

**TLS\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA**

**TLS\_DHE\_DSS\_WITH\_DES\_CBC\_SHA**

**TLS\_DHE\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA**

**TLS\_DH\_anon\_EXPORT\_WITH\_RC4\_40\_MD5**

**TLS\_DH\_anon\_WITH\_RC4\_128\_MD5**

**TLS\_DH\_anon\_EXPORT\_WITH\_DES40\_CBC\_SHA**

**TLS\_DH\_anon\_WITH\_DES\_CBC\_SHA**

**TLS\_DH\_anon\_WITH\_3DES\_EDE\_CBC\_SHA**

**TLS\_DHE\_DSS\_EXPORT1024\_WITH\_DES\_CBC\_SHA**

**TLS\_DHE\_DSS\_EXPORT1024\_WITH\_RC4\_56\_SHA**

**TLS\_DHE\_DSS\_WITH\_RC4\_128\_SHA**

( 3 ) For the following cipher suites, the SSL and TLS protocols specify that the length of the prime ( **pLength** ) can be at most 512 bits ( 64 bytes ).

**TLS\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA**

**TLS\_DH\_anon\_EXPORT\_WITH\_RC4\_40\_MD5**

**TLS\_DH\_anon\_EXPORT\_WITH\_DES40\_CBC\_SHA**

( 4 ) For the following cipher suites, the SSL and TLS protocols specify that the length of the prime ( **pLength** ) can be at most 1024 bits ( 128 bytes ).

`TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA`

`TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA`

## Parameters

`globalCtx`

[ Input/Output ] Pointer to the SSL global context.

`pLength`

[ Input ] Size ( in bytes ) of prime `pData` .

`pData`

[ Input ] Prime.

`gLength`

[ Input ] Size ( in bytes ) of generator `gData` .

`gData`

[ Input ] Generator.

`allocationType`

[ Input ] If set to `CIC_ALLOCATION_POINTER_COPY` , the data itself will not be copied. Instead, the function stores a pointer to the data. Your application is then responsible for keeping the pointer valid during the lifetime of the global context.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this context is different than the side selected with **ssl\_SetCipherSuites()**, or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

**CIC\_ERR\_SSL\_NEEDS\_PRNG**

PRNG not set yet. A PRNG must be installed first by calling **ssl\_SetPRNG()**.

## ssl\_AddIdentity()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_AddIdentity(  
    ssl_GlobalContext* globalCtx,  
    ssl_CertList* list  
);
```

### Description

Adds a certificate list as a local identity to a global context. Server applications must call this function. Client applications only have to call this function if the client authentication modes have been set ( see **ssl\_SetClientAuthModes()** ).

Notes:

- ( 1 ) The identity must contain both a certificate and a private key.
- ( 2 ) The private key will NOT be checked against the public key in the leaf certificate to make sure they match.
- ( 3 ) **ssl\_CertList** objects are reference counted objects. When you add the object to a global context the reference count of the object is increased. When you call **ssl\_DestroyCertList()** the reference count is decremented, and if the reference count becomes zero the memory for the object is released. So, after adding the **ssl\_CertList** object to a global context you can safely call **ssl\_DestroyCertList()** and when the global context is destroyed it will automatically release the memory for the object.
- ( 4 ) The certificates must have a specific ordering in the certificate list or client/server authentication and certificate chain validation will not function correctly. The first certificates in the list must be the entity certificate and the last certificate must be the CA certificate.
- ( 5 ) The installed identities must not be changed while any connection context created from this global context is being used in a handshake. Otherwise, the results will be unpredictable.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**list**

[ Input/Output ] List of certificates to add.

### Return Values

**CIC\_ERR\_NONE**

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

`CIC_ERR_SSL_INCOMPLETE_IDENTITY`

Certificate list does not contain both private key and certificate.

`CIC_ERR_SSL_UNSUPPORTED_PUBKEY_TYPE`

The public key type in the first certificate is unsupported.

## ssl\_AddTrustedCerts()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_AddTrustedCerts(  
    ssl_GlobalContext* globalCtx,  
    ssl_CertList* trusted  
);
```

### Description

Adds a certificate list as a set of trusted certificates to a global context. This function is optional.

Notes:

- ( 1 ) **ssl\_CertList** objects are reference-counted objects. When you add the object to a global context the reference count of the object is incremented. When you call **ssl\_DestroyCertList()** the reference count is decremented, and if the reference count becomes zero the memory for the object is released. So, after adding the **ssl\_CertList** object to a global context you can safely call **ssl\_DestroyCertList()** and when the global context is destroyed it automatically releases the memory for the object.
- ( 2 ) If the **trusted** parameter is NULL the error **CIC\_ERR\_NO\_PTR** is returned.
- ( 3 ) The trusted certificates must not be changed while any connection context created from this global context is being used in a handshake. Otherwise the results will be unpredictable.
- (4) If any one of the trusted certificates has expired, the trusted list is still installed in the global context, and the return code **CIC\_ERR\_SSL\_WARN\_TRUSTED\_EXPIRED** is returned.
- (5) It is your responsibility to ensure that the trusted certificates meet the necessary requirements for certificate verification. The validation date of the trusted certificates is checked when the certificates are loaded. The validation date is not checked again during certificate validation. The basic constraints for the trusted certificates are also not checked during certificate validation or loading of the trusted certificates.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**trusted**

[ Input/Output ] List of trusted certificates to add.

### Return Values

**CIC\_ERR\_NONE**

Success.



`CIC_ERR_NO_PTR`

`globalCtx` or `trusted` parameter is NULL.

`CIC_ERR_SSL_WARN_TRUSTED_EXPIRED`

One or more of the trusted certificates has expired. Note that if this warning is returned the trusted list is still loaded into the global context. It is up to you to reload the list after removing any expired certificates.

`CIC_ERR_INTERNAL`

An error occurred while extracting the certificate validity date.

## ssl\_DestroyCertList()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_DestroyCertList(  
    ssl_CertList** list  
);
```

### Description

Destroys a list of certificates. You must call this function if your application called **ssl\_CreateCertList()**.

Notes:

( 1 ) If the certificate list contains a decrypted private key, the key buffer will be zeroed before the list is destroyed.

### Parameters

**list**  
[ Input/Output ] Certificate list.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_NO\_PTR**  
NULL pointer was passed.

# SSL Plus Connection Context Functions

This section describes the functions which create and destroy a connection context. You must create a connection context in order to establish an SSL connection with a peer.

## ssl\_CreateConnectionContext( )

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_CreateConnectionContext(
    ssl_GlobalContext* globalCtx,
    const cic_Uint16 defReadBufLen,
    const cic_Uint16 maxReadBufLen,
    const cic_Uint16 writeFragmentLen,
    const cic_Buffer peerID,
    void* const ioRef,
    void* const randomRef,
    void* const certRef,
    void* const alertRef,
    void* const sessionRef,
    void* const surrenderRef,
    void* const logRef,
    void* const memRef,
    ssl_ConnectionContext** connCtx
);
```

### Description

Initializes a connection context.

This function is mandatory; you must call it in your application.

Notes:

- ( 1 ) The default session expiry time is set to 24 hours. You can change the expiry time by calling **ssl\_SetSessionExpiryTime( )**.
- ( 2 ) The default I/O style is set to **SSL\_IOSemantics\_CompleteIO**. You can change the I/O style by calling **ssl\_SetIOSemantics( )**.
- ( 3 ) If the value of **writeFragmentLen** is either 512, 1024, 2048 or 4096, then this value is used when negotiating a maximum record fragment length with the server, as specified in [www.ietf.org/internet-drafts/draft-ietf-tls-extensions-03.txt](http://www.ietf.org/internet-drafts/draft-ietf-tls-extensions-03.txt). If the server supports the extension `max_fragment_length`, then both client and server limit records to one of these sizes ( 512, 1024, 2048 or 4096 bytes ), so you must ensure that **maxReadBufLen** is large enough to handle records of this size. However, since most SSL implementations do not support the extensions listed above, you can use the **maxReadBufLen** parameter to process larger records from the server.
- ( 4 ) You can fine-tune the memory management for each connection by setting the **memRef** parameter. If you set this parameter to NULL, the library uses the reference parameter set in **ssl\_CreateGlobalContext( )**.
- ( 5 ) You can use the reference parameters ( such as **ioRef** and **randomRef** ) to pass information to the callback functions.

## Parameters

**globalCtx**

[ Input/Output ] Pointer to the global context.

**defReadBufLen**

[ Input ] Default size ( in bytes ) of the buffer used to read SSL records. The context allocates a buffer of this size for reading SSL records. If **defReadBufLen** is zero, a default of 4K ( 4096 bytes ) is used.

**maxReadBufLen**

[ Input ] Maximum size ( in bytes ) of the buffer used to read SSL records.

This defines the largest supported size for an SSL record. If the default record buffer size specified by **defReadBufLen** is too small to hold the incoming record, the record buffer temporarily increases to **maxReadBufLen** bytes so that the record can be processed. If the incoming record exceeds **maxReadBufLen** bytes the error **CIC\_ERR\_SSL\_OVERFLOW** is returned.

The maximum size of an SSL 2.0 record is  $2^{15}$  ( 32768 ) bytes. The maximum size of an SSL 3.0 or TLS 1.0 record is  $2^{14} + 2048$  ( 18432 ) bytes.

If **maxReadBufLen** is zero, a default size of  $2^{15}$  ( 32768 ) bytes is used.

Desktop systems usually have sufficient memory to allocate a 18432 byte read buffer. However embedded systems often have limited memory and allocating this much memory may not be feasible. In this case you should specify a smaller value for **maxReadBufLen**.

**writeFragmentLength**

[ Input ] Sets the maximum size of the data records that are passed to **ssl\_Write()**. Records that are above the limit are fragmented. If **writeFragmentLength** is equal to zero, a default write fragment length of 4K ( 4096 bytes ) is used.

**peerID**

[ Input ] For client side applications, this optional parameter contains the session key passed to **ssl\_GetSessionFunc()**. For server side applications, this optional parameter contains the session id to be used for non-resumed sessions. For SSLV2 connections the buffer must be at least 16 bytes long and the first 16 bytes will be used as the session id. For SSLV3 and TLSV1 connections the first 32 bytes will be used.

**ioRef**

[ Input ] Optional reference parameter for the I/O callback functions.

**randomRef**

[ Input ] Optional reference parameter for the random callback function.

**certRef**

[ Input ] Optional reference parameter for the check cert chain callback function.

**alertRef**

[ Input ] Optional reference parameter for the alert callback function.

**sessionRef**

[ Input ] Optional reference parameter for the session callback functions.

**surrenderRef**

[ Input ] Optional reference parameter for the surrender callback function.

**logRef**

[ Input ] Optional reference parameter for the logging callback function.

**memRef**

[ Input ] Optional reference parameter for the memory callback functions. If this parameter is NULL, then the reference parameter used by **globalCtx** will be used.

**connCtx**

[ Output ] Pointer to the SSL connection context.

## Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_NEEDS\_PRNG**

PRNG not set yet. A PRNG must be installed first by calling **ssl\_SetPRNG()**.

## ssl\_DestroyConnectionContext( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_DestroyConnectionContext(  
    ssl_ConnectionContext** connCtx  
);
```

### Description

Destroys a connection context and frees all the resources held by the structure. The context is zeroed before the memory is released.

This function is mandatory; you must call it in your application.

Note:

( 1 ) You should call **ssl\_Close( )** to close the connection before calling this function.

### Parameters

**connCtx**

[ Input/Output ] Pointer to the SSL connection context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

## ssl\_GetConnectionMemRef( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetConnectionMemRef(  
    const ssl_ConnectionContext* connCtx,  
    void** memRef  
);
```

### Description

Returns a memory reference stored in the connection context.

### Parameters

**connCtx**

[ Input ] Pointer to the connection context.

**memRef**

[ Output ] Pointer to memory reference.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.



## ssl\_GetConnectioncbData( )

```
CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetConnectioncbData(  
    const ssl_ConnectionContext* connCtx,  
    void** cbData  
);
```

### Description

Returns the callback data contained in the connection context. This data can then be passed to the TrustPoint C certificate parsing functions that take a **cbData** parameter.

### Parameters

**connCtx**

[ Input ] Pointer to the SSL connection context.

**cbData**

[ Output ] On output points to the callback data contained in the connection context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

# SSL Plus Data Exchange Functions

The functions in this section perform the following tasks:

- Handshake with a peer application.
- Request a renegotiation of the handshake.
- Read data from an SSL connection.
- Write data to an SSL connection.
- Gets the number of bytes that are waiting to be read in the library's internal read queue.
- Gets the number of bytes that are waiting to be sent in the library's internal write queue.
- Send all the remaining bytes in the internal write queue.
- Close an SSL connection.

## ssl\_Handshake( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_Handshake(
    ssl_ConnectionContext* connCtx
);
```

### Description

Performs an SSL handshake with a peer.

This function is mandatory; you must call it in your application.

When the handshake has finished, you can use **ssl\_Read( )** and **ssl\_Write( )** to exchange data with the peer.

In the absence of any errors, **ssl\_Handshake( )** will not return until the handshake has finished. Note that the handshake process will cause data to be transferred across the connection using the I/O callbacks.

If you use non-blocking I/O then **ssl\_Handshake( )** could return **CIC\_ERR\_WOULD\_BLOCK**. If this happens, call **ssl\_Handshake( )** repeatedly until it returns **CIC\_ERR\_NONE** or some other error code. Calls to all the SSL callbacks could occur during the handshake. Error codes from these callbacks will be passed back up to the caller.

You should also call this function to renegotiate a handshake with the peer, after making a request with **ssl\_RequestRenegotiation( )**.

Notes on renegotiation:

( 1 ) If the peer refuses to renegotiate the handshake

**CIC\_ERR\_SSL\_RENEGOTIATION\_REFUSED** is returned. This is only a warning- the connection context is still valid and can still be used to read and write data.

( 2 ) The peer may ignore your renegotiation request and send application data instead. If this occurs **CIC\_ERR\_SSL\_READ\_REQUIRED** is returned. This indicates that the renegotiation is still pending and there is application data which must be processed first by calling **ssl\_Read( )**. You can also call **ssl\_Write( )**, however we recommend that you only call **ssl\_Read** or **ssl\_Close** so as not to confuse the peer. When the renegotiation request is accepted by the peer, **ssl\_Read( )** returns **CIC\_ERR\_SSL\_HANDSHAKE\_REQUIRED** to indicate that you must call **ssl\_Handshake( )** to continue with the renegotiation or **ssl\_Close( )** to close the connection.

For more information on renegotiating a handshake, see **ssl\_RequestRenegotiation( )**.

### Parameters

**connCtx**

[ Input/Output ] Pointer to the SSL connection context.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

`CIC_ERR_SSL_OVERFLOW`

Handshake record caused read buffer to overflow.

`CIC_ERR_SSL_BAD_RECORD_LENGTH`

Handshake record length exceeds maximum defined by SSL specification.

`CIC_ERR_WOULD_BLOCK`

I/O is blocking.

`CIC_ERR_SSL_UNEXPECTED_MSG`

Unexpected message.

`CIC_ERR_SSL_BAD_MAC`

Verification of SSL Record Message Authentication Code failed.

`CIC_ERR_SSL_DECRYPT_FAILED`

SSL record decryption failed.

`CIC_ERR_SSL_UNKNOWN_RECORD`

Unknown record type.

`CIC_ERR_SSL_NEGOTIATION`

SSL negotiation error.

`CIC_ERR_SSL_IO`

I/O error from callbacks.

`CIC_ERR_SSL_FATAL_ALERT`

Fatal alert was received or sent.

`CIC_ERR_SSL_PROTOCOL`

General protocol error. May be caused by incorrectly formatted messages.

`CIC_ERR_SSL_RESUMABLE_SESSION`

Server is trying to resume a session with different session parameters.

`CIC_ERR_SSL_BAD_FINISHED_MESSAGE`

Finished message is incorrect.

`CIC_ERR_SSL_CONNECTION_CLOSED`

Session was closed.

`CIC_ERR_SSL_CONNECTION_CLOSED_GRACEFUL`

Session was closed gracefully.

`CIC_ERR_SSL_BAD_CONTEXT`

The global context has not been configured with a protocol.

`CIC_ERR_SSL_HANDSHAKE_ALREADY_COMPLETED`

Handshake is already completed.

`CIC_ERR_SSL_RENEGOTIATION_REFUSED`

A renegotiated handshake has been refused by the peer.

`CIC_ERR_SSL_READ_REQUIRED`

Renegotiation with the peer has not started yet. There is application data which must be processed with `ssl_Read()` before the handshake can proceed.



## ssl\_RequestRenegotiation( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_RequestRenegotiation(
    ssl_ConnectionContext* connCtx
);
```

### Description

Requests a renegotiation of the handshake. This function is optional.

After calling this function, you must call **ssl\_Handshake( )** to carry out the renegotiation. There is no guarantee that renegotiation will take place: the peer may perform the renegotiation, refuse it by sending an alert ( TLS1 connections only ), ignore it, or send application data which must be processed first.

If the peer refuses the request, **ssl\_Handshake( )** returns **CIC\_ERR\_SSL\_RENEGOTIATION\_REFUSED** . If the peer sends application data, it returns **CIC\_ERR\_SSL\_READ\_REQUIRED** .

If the configuration of the context has changed since the initial handshake, the new parameters are used in the renegotiation. New keys are generated regardless of whether the context changed.

**ssl\_Close( )** can be called to close the connection without renegotiating.

A problem could occur if blocking sockets are used and the peer ignores the renegotiation request. When the client makes a renegotiation request ( by sending a **client\_hello** message ) it will be in one of the following states:

- waiting for an alert
- waiting for a handshake message
- waiting for application data.

If the server ignores the handshake and sends no alert then the client could be placed in a permanent blocking state while it waits for data from the server. If the server sends application data or an alert then the client knows that its request was ignored and will not be placed in a blocking state. However, if the server sends nothing, the client will wait indefinitely for the next record. The only solution to this is to use non-blocking sockets.

Notes:

( 1 ) Renegotiation cannot be requested unless a secure connection has already been established. This is indicated by the error **CIC\_ERR\_SSL\_HANDSHAKE\_REQUIRED** .

( 2 ) Renegotiation cannot be requested for SSL2 connections. This is indicated by the error **CIC\_ERR\_SSL\_NOT\_SUPPORTED** .

( 3 ) A renegotiation request cannot be made again until the renegotiation is is completed or refused by the peer. This is indicated by the error **CIC\_ERR\_SSL\_RENEGOTIATION\_ALREADY\_REQUESTED** . Since SSL3 does

not support the " No Renegotiation " alert, if the peer refuses your renegotiation request you may never be able to make another one on this connection.

( 4 ) We recommend that you do not request a renegotiation while there is data to be serviced in the write queue, or unprocessed data in the read buffer ( or waiting in the connection socket ).

( 5 ) If the peer requests renegotiation, you will be notified through the error code `CIC_ERR_SSL_HANDSHAKE_REQUESTED` . If you wish to proceed with the renegotiation you must call `ssl_Handshake()` . To refuse the renegotiation call either `ssl_Read()` or `ssl_Write()` . You can also close the connection with `ssl_Close()` .

( 6 ) In server applications, a renegotiation request causes data to be transferred across the connection using the `write ( )` callback. In client connections, the request does not cause data to be transferred.

( 7 ) After the handshake has completed, all ephemeral key pairs ( ie. RSA export key pair ) are destroyed, to reduce the memory overhead of the connection context. If renegotiation takes place these ephemeral keys will be regenerated.

## Parameters

`connCtx`

[ Input/Output ] Pointer to the SSL connection context.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

`CIC_ERR_SSL_CONNECTION_CLOSED`

Session was closed.

`CIC_ERR_SSL_CONNECTION_CLOSED_GRACEFUL`

Session was closed gracefully.



`CIC_ERR_SSL_IO`

I/O error from callback.

`CIC_ERR_SSL_HANDSHAKE_REQUIRED`

Renegotiation cannot proceed before handshake is completed.

`CIC_ERR_SSL_NOT_SUPPORTED`

Renegotiation with SSL2 is not supported.

`CIC_ERR_SSL_RENEGOTIATION_ALREADY_REQUESTED`

Renegotiation has already been requested.

## ssl\_Read( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_Read(
    ssl_ConnectionContext* connCtx,
    cic_Uint32* length,
    cic_Uint8* data
);
```

### Description

Reads data from a peer on an SSL connection. This function is mandatory; you must call it in your application.

Notes:

- ( 1 ) Data cannot be read unless a secure connection has been established. This is indicated by the error **CIC\_ERR\_SSL\_HANDSHAKE\_REQUIRED**.
- ( 2 ) The data is read using the **read ( )** callback function. Any error codes that are returned from the callback function are passed back up to your application.
- ( 3 ) In the case of non-blocking I/O, if the **read ( )** callback function returns **CIC\_ERR\_WOULD\_BLOCK**, the **length** parameter will be set to the number of bytes that were decrypted and placed in the output buffer.
- ( 4 ) In the case of blocking I/O, requests can be completely or partially fulfilled depending upon the I/O method selected with **ssl\_SetIOSemantics( )**.
- ( 5 ) If the peer sends more data than is requested, the additional bytes are stored in a read buffer. You can call **ssl\_GetReadPendingSize( )** to determine how many bytes remain in this buffer. Subsequent calls to **ssl\_Read( )** will read bytes from this buffer.
- ( 6 ) If this function is called immediately after receiving the error **CIC\_ERR\_SSL\_HANDSHAKE\_REQUESTED( )** from **ssl\_Handshake( )**, renegotiation will be refused by this side of the connection.

### Parameters

**connCtx**

[ Input/Output ] Pointer to the SSL connection context.

**length**

[ Input/Output ] On input, indicates the number of bytes to read ( must be greater than zero ).

On output, contains the actual number of bytes read ( can be zero ).

**data**

[ Output ] Pointer to data buffer.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_ILLEGAL_PARAM`

**Length** parameter invalid.

`CIC_ERR_MEMORY`

Error allocating memory.

`CIC_ERR_SSL_OVERFLOW`

Handshake record caused read buffer to overflow.

`CIC_ERR_SSL_BAD_RECORD_LENGTH`

Handshake record length exceeds maximum defined by SSL specification.

`CIC_ERR_WOULD_BLOCK`

I/O is blocking.

`CIC_ERR_SSL_IO`

I/O error from callbacks.

`CIC_ERR_SSL_CONNECTION_CLOSED_GRACEFUL`

Session was closed gracefully.

`CIC_ERR_SSL_CONNECTION_CLOSED`

Session was closed.

`CIC_ERR_SSL_HANDSHAKE_REQUIRED`

Handshake is required.

**CIC\_ERR\_SSL\_RENEGOTIATION\_REFUSED**

A renegotiated handshake has been refused by the peer. This is only a warning, and can only be returned if **ssl\_RequestRenegotiation()** was called. The connection is still valid.

**CIC\_ERR\_SSL\_HANDSHAKE\_REQUESTED**

A renegotiation has been requested by the peer. To proceed with renegotiation call **ssl\_Handshake()**. If you call **ssl\_Read()**, **ssl\_Write()**, or **ssl\_Close()** the renegotiation will be refused.

## ssl\_Write()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_Write(
    ssl_ConnectionContext* connCtx,
    const cic_Uint8* data,
    cic_Uint32* length
);
```

### Description

Writes data to a peer on an SSL connection. This function is mandatory; you must call it in your application.

Notes:

- ( 1 ) Data can not be written unless a secure connection has been established. This is indicated by the error **CIC\_ERR\_SSL\_HANDSHAKE\_REQUIRED**.
  - ( 2 ) The data is sent using the **write ( )** callback function. Any error codes that are returned from the callback function are passed back up to your application.
  - ( 3 ) In the case of non-blocking I/O, the **write ( )** callback returns **CIC\_ERR\_WOULD\_BLOCK** if it can only send some of the data. The remaining data is stored in a write queue. The **length** parameter indicates how many bytes are in this queue. You should only call this function when there is no data in the write queue. To send the remaining data, either call **ssl\_Write()** again, or call **ssl\_ServiceWriteQueue()**.
- You can also call **ssl\_GetWritePendingSize()** to determine how much data is waiting in the write queue.
- ( 4 ) Note that **ssl\_Close()** sends all the remaining data in the write queue before closing the connection.
  - ( 5 ) If this function is called immediately after receiving the error **CIC\_ERR\_SSL\_HANDSHAKE\_REQUESTED()** from **ssl\_Handshake()**, it indicates that renegotiation has been refused by this side of the connection.

### Parameters

**connCtx**

[ Input/Output ] Pointer to the SSL connection context.

**data**

[ Input ] Data to be written.

**length**

[ Input/Output ] On input, the number of bytes to send ( must be greater than zero ).

On output, contains the number of bytes ready for sending in the write queue ( can be zero ). In the case of non-blocking I/O, some data in the write queue may not have been sent.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_ILLEGAL_PARAM`

**Length** parameter is invalid.

`CIC_ERR_MEMORY`

Error allocating memory.

`CIC_ERR_WOULD_BLOCK`

I/O is blocking.

`CIC_ERR_SSL_IO`

I/O error from callbacks.

`CIC_ERR_SSL_CONNECTION_CLOSED_GRACEFUL`

Session was closed gracefully.

`CIC_ERR_SSL_CONNECTION_CLOSED`

Session was closed.

`CIC_ERR_SSL_HANDSHAKE_REQUIRED`

Handshake is required.

## ssl\_GetReadPendingSize()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetReadPendingSize(  
    const ssl_ConnectionContext* connCtx,  
    cic_Uint32* waitingBytes  
);
```

### Description

Gets the number of bytes that remain in the library's internal read queue after a call to **ssl\_Read()**. This function is optional.

### Parameters

**connCtx**

[ Input ] Pointer to the SSL connection context.

**waitingBytes**

[ Output ] Pointer to the number of bytes waiting to be read.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

## ssl\_GetWritePendingSize()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetWritePendingSize(  
    const ssl_ConnectionContext* connCtx,  
    cic_Uint32* waitingBytes  
);
```

### Description

Gets the number of bytes waiting to be sent in the library's internal write queue. This function is optional.

You can use this function together with **ssl\_ServiceWriteQueue()** to send the data in the write queue. When all the bytes have been sent, this function sets **waitingBytes** to zero.

See **ssl\_Write()** and **ssl\_ServiceWriteQueue()** for more information about the internal write queue.

### Parameters

**connCtx**

[ Input ] Pointer to the SSL connection context.

**waitingBytes**

[ Output ] Pointer to the number of bytes waiting to be sent.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.



## ssl\_ServiceWriteQueue()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_ServiceWriteQueue(
    ssl_ConnectionContext* connCtx
);
```

### Description

Attempts to send all the data in the write queue. This function is optional.

You should call this function if **ssl\_Write()** or **ssl\_Close()** return **CIC\_ERR\_WOULD\_BLOCK** and you have no additional data to send. You should call **ssl\_ServiceWriteQueue()** repeatedly until it returns **CIC\_ERR\_NONE** or a fatal error.

If **ssl\_Handshake()** or **ssl\_Read()** return **CIC\_ERR\_WOULD\_BLOCK**, you should call them again, instead of using **ssl\_ServiceWriteQueue()**.

You can call **ssl\_GetWritePendingSize()** to find out how many bytes remain to be sent in the internal write queue.

### Parameters

**connCtx**  
[ Input/Output ] Pointer to the SSL connection context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_NO\_PTR**  
NULL pointer was passed.

**CIC\_ERR\_WOULD\_BLOCK**  
I/O is blocking.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_IO**  
I/O error from callbacks.

## ssl\_Close()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_Close(  
    ssl_ConnectionContext* connCtx  
);
```

### Description

Closes the SSL session with the peer by sending a "close\_notify" message. This function is mandatory; you must call it to close the session. If you do not close a session, the library may not be able to resume it at a later time.

A "close\_notify" message indicates to the peer that it should not expect any further communications. It should only be called when no errors have occurred on the connection. If the peer closes the connection first, **ssl\_Read()** returns **CIC\_ERR\_SSL\_CONNECTION\_CLOSED\_GRACEFUL**. You do not need to call **ssl\_Close** in this case.

**ssl\_Close()** has no effect on the underlying network connection; it must be closed manually with its native API. If the network connection is closed without first calling **ssl\_Close()**, the peer may treat this session as unreliable and refuse to cache the information necessary to resume it.

### Parameters

**connCtx**  
[ Input/Output ] Pointer to the SSL connection context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_NO\_PTR**  
NULL pointer was passed.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_OVERFLOW**  
Handshake record caused read buffer to overflow.

**CIC\_ERR\_WOULD\_BLOCK**  
Blocking error.

`CIC_ERR_SSL_UNEXPECTED_MSG`

Unexpected message.

`CIC_ERR_SSL_BAD_MAC`

Verification of SSL record message failed.

`CIC_ERR_SSL_DECRYPT_FAILED`

SSL record decryption failed.

`CIC_ERR_SSL_UNKNOWN_RECORD`

Unknown record type.

`CIC_ERR_SSL_NEGOTIATION`

SSL negotiation error.

`CIC_ERR_SSL_IO`

I/O error from callbacks.

`CIC_ERR_SSL_FATAL_ALERT`

Fatal alert was received or sent.

`CIC_ERR_SSL_PROTOCOL`

General protocol error. May be caused by incorrectly formatted messages.

`CIC_ERR_SSL_RESUMABLE_SESSION`

Server tried to resume a session with incorrect session parameters.

`CIC_ERR_SSL_BAD_FINISHED_MESSAGE`

The finished message is incorrect.

`CIC_ERR_SSL_CONNECTION_CLOSED_GRACEFUL`

Session was closed gracefully.

`CIC_ERR_SSL_CONNECTION_CLOSED`

Session was closed.

# SSL Plus Connection Status Functions

You can use the functions in this section to get the connection parameters that were agreed upon during the handshake. Specifically, they perform the following tasks:

- Get the version of the SSL protocol that was agreed upon during the handshake.
- Get the version of the SSL protocol that is currently being negotiated in the handshake.
- Get the ID of the cipher suite that was agreed upon during the handshake.
- Get the master secret that was agreed upon during the handshake.
- Extracts random data from a client\_hello or server\_hello message. This data can be used to generate key material when using the EAP - TLS protocol.
- Check if a session was resumed.
- Frees the read and write buffers created by the connection context.

## ssl\_GetNegotiatedProtocolVersion( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetNegotiatedProtocolVersion(  
    const ssl_ConnectionContext* connCtx,  
    ssl_ProtocolVersion* version  
);
```

### Description

Gets the version of the SSL protocol that was agreed upon during the handshake. Call this function after the handshake has finished. This function is optional.

### Parameters

**connCtx**

[ Input ] Pointer to the connection context.

**version**

[ Output ] Version of the SSL protocol which has been agreed upon. See the enumerated types on page 275.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SSL\_HANDSHAKE\_REQUIRED**

Handshake must be completed first.

## ssl\_GetContextProtocolVersion( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetContextProtocolVersion(  
    const ssl_ConnectionContext* connCtx,  
    ssl_ProtocolVersion* version  
);
```

### Description

Gets the version of the SSL protocol that is set in the connection context. This function is optional.

Notes:

( 1 ) Unlike **ssl\_GetNegotiatedProtocolVersion( )** , this function may be called **during** the handshake. However, this may not represent the version of the protocol that was agreed upon. This is only determined at the end of the handshake. This function is useful in the certificate chain callback, where your application may need to know which which protocol is being negotiated in order to carry out the appropriate validation.

( 2 ) If this function is called before the handshake is started the version returned is **SSL\_ProtocolVersion\_Undetermined** .

### Parameters

**connCtx**

[ Input ] Pointer to the connection context.

**version**

[ Output ] The version of the SSL protocol being negotiated for the connection. See the enumerated type on page....

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

## ssl\_GetNegotiatedCipher( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetNegotiatedCipher(  
    const ssl_ConnectionContext* connCtx,  
    cic_Uint16* ciphersuite  
);
```

### Description

Gets the identifier of the cipher suite that was agreed upon during the handshake. This function is optional.

### Parameters

**connCtx**

[ Input ] Pointer to the SSL connection context.

**ciphersuite**

[ Output ] Identifier of the cipher suite. The identifiers are listed on page 275.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SSL\_HANDSHAKE\_REQUIRED**

Handshake must be completed first.



## ssl\_GetMasterSecret( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetMasterSecret(
    const ssl_ConnectionContext* connCtx,
    cic_Uint16 secretBufLen,
    cic_Uint8* secret,
    cic_Uint16* secretLen
);
```

### Description

Gets the negotiated master secret for the connection context. This function is optional.

### Parameters

**connCtx**  
[ Input ] Pointer to the SSL connection context.

**secretBufLen**  
[ Input ] Size of the master secret buffer.

**secret**  
[ Output ] Buffer which contains the master secret.

**secretLen**  
[ Output ] Pointer to the number of bytes copied into the **secret** buffer.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_NO\_PTR**  
NULL pointer was passed.

**CIC\_ERR\_BAD\_LENGTH**  
**secretBufLen** is too small to hold the master secret. In this case **secretLen** returns the required size of the buffer.

**CIC\_ERR\_SSL\_HANDSHAKE\_REQUIRED**  
Handshake must be completed first.

## ssl\_GetClientRandom()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetClientRandom(  
    const ssl_ConnectionContext* connCtx,  
    cic_Uint16 randBufLen,  
    cic_Uint8* randBuf,  
    cic_Uint16* randLen  
);
```

### Description

Gets the random field which was encoded in the client\_hello handshake message. This function is intended for applications wishing to derive key material used in EAP-TLS or similar protocols.

Notes:

( 1 ) This function is only valid if the negotiated version is TLS 1.0 or greater.

### Parameters

**connCtx**

[ Input ] Pointer to the SSL connection context.

**randBufLen**

[ Input ] Size of the random buffer.

**randBuf**

[ Output ] Buffer which contains the random.

**randLen**

[ Output ] Pointer to the number of bytes copied into the **randomBuf** buffer.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

`CIC_ERR_BAD_LENGTH`

`randomBufLen` is too small to hold the random. In this case `randomLen` returns the required size of the buffer.

`CIC_ERR_SSL_HANDSHAKE_REQUIRED`

Handshake must be completed first.

`CIC_ERR_SSL_BAD_PROTOCOL_VERSION`

Protocol version is not TLS 1.0 or greater

## ssl\_GetServerRandom( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetServerRandom(  
    const ssl_ConnectionContext* connCtx,  
    cic_Uint16 randBufLen,  
    cic_Uint8* randBuf,  
    cic_Uint16* randLen  
);
```

### Description

Gets the random field which was encoded in the server\_hello handshake message. This function is intended for applications wishing to derive key material used in EAP-TLS or similar protocols.

Notes:

( 1 ) This function is only valid if the negotiated version is TLS 1.0 or greater.

### Parameters

**connCtx**

[ Input ] Pointer to the SSL connection context.

**randBufLen**

[ Input ] Size of the random buffer.

**randBuf**

[ Output ] Buffer which contains the random.

**randLen**

[ Output ] Pointer to the number of bytes copied into the **randomBuf** buffer.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

`CIC_ERR_BAD_LENGTH`

`randomBufLen` is too small to hold the random. In this case `randomLen` returns the required size of the buffer.

`CIC_ERR_SSL_HANDSHAKE_REQUIRED`

Handshake must be completed first.

`CIC_ERR_SSL_BAD_PROTOCOL_VERSION`

Protocol version is not TLS 1.0 or greater

## ssl\_WasSessionResumed( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_WasSessionResumed(  
    const ssl_ConnectionContext* connCtx,  
    cic_Bool* resumed  
);
```

### Description

Indicates if a session was resumed. This function is optional.

### Parameters

**connCtx**

[ Input ] Pointer to the SSL connection context.

**resumed**

[ Output ] Pointer to a flag which indicates if the SSL session was resumed.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SSL\_HANDSHAKE\_REQUIRED**

Handshake must be completed first.

## ssl\_GetSessionID()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetSessionID(
    const ssl_ConnectionContext* connCtx,
    cic_Uint16 sessionIDBufLen,
    cic_Uint8* sessionIDBuf,
    cic_Uint16* sessionIDLen
);
```

### Description

Gets the session ID from the connection context.

Notes:

### Parameters

**connCtx**

[ Input ] Pointer to the SSL connection context.

**sessionIDBufLen**

[ Input ] Size of the sessionID buffer.

**sessionIDBuf**

[ Output ] Buffer which contains the sessionID.

**sessionIDLen**

[ Output ] Pointer to the number of bytes copied into the **sessionIDBuf** buffer.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_BAD\_LENGTH**

**sessionIDBufLen** is too small to hold the sessionID. In this case **sessionIDLen** returns the required size of the buffer.

**CIC\_ERR\_SSL\_HANDSHAKE\_REQUIRED**

Handshake must be completed first.

## ssl\_FreeRecordBuffers ( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_FreeRecordBuffers(  
    ssl_ConnectionContext* connCtx  
);
```

### Description

Frees the read and write record buffers being used by the connection context.

Notes:

( 1 ) This function reduces the memory overhead of a connection which is still active but currently not in use. Normally the read and write record buffers will be allocated once during context creation and only freed when the context is destroyed. This default behavior helps prevent fragmentation of the dynamic memory heap by reducing the number of allocations.

( 2 ) If the read record buffer contains some data the buffer will not be released. You must read the data to empty the buffer.

( 3 ) If the write record buffer contains some data the buffer will not be released. You must service the write queue to empty the buffer.

( 4 ) If the buffers are freed, they will be automatically re-allocated the next time data is read or written.

### Parameters

**connCtx**

[ Input/Output ] Pointer to the connection context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SSL\_READ\_BUFFER\_NOT\_EMPTY**

The read buffer is not empty. The write buffer was freed.



`CIC_ERR_SSL_WRITE_BUFFER_NOT_EMPTY`

The write buffer is not empty. The read buffer was freed.

`CIC_ERR_SSL_BUFFERS_NOT_EMPTY`

Both read and write buffers are not empty. Neither buffer was freed.

# SSL Plus Callback Set Functions

The functions in this section perform the following tasks:

- Set the callback functions for reading and writing data on the network connection.
- Set the callback functions that the library uses to send alert messages to your application.
- Set the callback functions which manage sessions in the session database.
- Set the callback function for checking a certificate chain.
- Set the callback function that the library uses to generate a random seed.
- Set the callback function that the library calls during lengthy cryptographic operations.
- Set the callback function that logs status messages.

## ssl\_SetIOFuncs( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetIOFuncs(
    ssl_GlobalContext* globalCtx,
    const ssl_IOFunc readCb,
    const ssl_IOFunc writeCb
);
```

### Description

Sets the callback functions that the library uses to read and write data from the network connection.

This function is mandatory; you must set the **read ( )** and **write ( )** callback functions in your application.

See **ssl\_IOFunc ( )** for a description of the I/O callback functions.

Notes:

( 1 ) The **readCb** and **writeCb** arguments must be non-NULL.

( 2 ) The I/O functions must not be changed while any connection context created from this global context exists. Otherwise, the results will be unpredictable.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**readCb**

[ Input ] Read callback function.

**writeCb**

[ Input ] Write callback function.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SSL\_NULL\_CB**

NULL callback was passed.

## ssl\_SetAlertFunc( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetAlertFunc(  
    ssl_GlobalContext* globalCtx,  
    const ssl_AlertFunc alertCb  
);
```

### Description

Sets the callback function that the library uses to send alert messages to your application. This callback is optional.

Notes:

- ( 1 ) The **alertCb** argument may be NULL.
- ( 2 ) See **ssl\_AlertFunc ( )** for a description of the alert callback function.
- ( 3 ) The alert function must not be changed while any connection context created from this global context exists. Otherwise the results will be unpredictable.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**alertCb**

[ Input ] Callback function.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL context pointer was passed.

## ssl\_SetSessionFuncs( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetSessionFuncs(
    ssl_GlobalContext* globalCtx,
    const ssl_AddSessionFunc addCb,
    const ssl_GetSessionFunc getCb,
    const ssl_DeleteSessionFunc delCb
);
```

### Description

Sets the callback functions that the library uses to update the session database. The library stores enough information in the database to be able to resume a session.

These callbacks are optional ( see note ( 1 ) ).

Notes:

( 1 ) The **addCb** , **getCb** , and **delCb** arguments must be either all non-NULL or all NULL values. If any one of the callbacks is not set, the library will not attempt to resume sessions.

( 2 ) The session functions must not be changed while any connection context created from this global context exists. Otherwise the results will be unpredictable and application crashes possible.

( 3 ) See **ssl\_AddSessionFunc ( )** , **ssl\_GetSessionFunc ( )** , and **ssl\_DeleteSessionFunc ( )** for a description of the callback functions.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

**addCb**  
[ Input ] " Add session " callback function.

**getCb**  
[ Input ] " Get session " callback function.

**delCb**  
[ Input ] " Del session " callback function.

### Return Values

**CIC\_ERR\_NONE**  
Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_SSL_NULL_CB`

NULL callback was passed. All or none of the callbacks must be set.

## ssl\_SetCheckCertificateChainFunc ( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetCheckCertificateChainFunc(  
    ssl_GlobalContext* globalCtx,  
    const ssl_CheckCertificateChainFunc certCb  
);
```

### Description

Sets the callback function that the library calls when it receives a certificate from the client or server during the handshake. You can use the callback function to override any validation errors that the library found when it examined the certificate. This callback is optional.

See **ssl\_CheckCertificateChainFunc ( )** for a description of the callback function.

Notes:

( 1 ) If **certCb** is NULL, this function disables any previous callback function that may have been set.

( 2 ) The callback function must not be changed while any connection context created from this global context exists. Otherwise the results will be unpredictable.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**certCb**

[ Input ] Callback function.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL context pointer was passed.

## ssl\_SetLogFunc( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SetLogFunc(  
    ssl_GlobalContext* globalCtx,  
    const ssl_LogFunc logCb  
);
```

### Description

Sets the log callback for the SSL context. This callback is optional.

Notes:

- ( 1 ) If you set the **logCb** parameter to NULL, it disables any previous logging callback that was set.
- ( 2 ) See **ssl\_LogFunc** for a description of the callback function.
- ( 3 ) The log function must not be changed while any connection context created from this global context exists. Otherwise the results will be unpredictable.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**logCb**

[ Input ] Logging callback function.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL context pointer was passed.



# SSL Plus Random Seed Function

The SSL Plus library uses your callback function to generate random seed data. It uses this seed in an internal PRNG to generate random data for your application. You can use the function provided in this section to generate this seed, or, if you have the appropriate hardware, you can generate the random data directly.

## ssl\_GenerateRandomSeed( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GenerateRandomSeed(
    const cic_Buffer* additionalSeed,
    cic_Buffer* randout
);
```

### Description

Generates random seed data using the library's entropy collection code. The seed is generated from run-time data on the OS.

The library uses the seed generated by the random callback to seed its own PRNG. You can use this function ( **ssl\_GenerateRandomSeed( )** ) in the callback, however if your device has hardware capable of producing good quality random data you should use that instead and ignore this function.

Notes:

( 1 ) For added security, you can use the **seed** parameter to combine your own seed with that generated by the function. This parameter is optional ( set to NULL to return only the seed generated by the function ).

### Parameters

**additionalSeed**

[ Input ] Pointer to additional seed to be combined with that generated by this function.

**randout**

[ Input/Output ] Pointer to a buffer which will contain the seed. On input, set the **length** field of this buffer to the number of bytes of random seed that you need. On output, the **length** field contains the number of seed bytes generated.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_ILLEGAL\_PARAM**

The value of the **length** field in the **randout** parameter is invalid.

`CIC_ERR_MEMORY`

Error allocating memory.

`CIC_ERR_SSL_ENTROPY_COLLECTION`

Unable to generate sufficient data for seed. The **length** field may be too large.

# SSL Plus Certificate Functions

The functions in this section perform the following tasks:

- Extract an RDN from a certificate object.
- Extract the raw certificate data from a certificate object.
- Decode an SSL record into printable data for the logging function.

## ssl\_ExtractCertificateNameItem()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_ExtractCertificateNameItem(
    const ssl_ConnectionContext* connCtx,
    const ssl_Cert* cert,
    const ssl_CertNameItem whichNameItem,
    const cic_Uint16 instanceIndex,
    cic_Buffer* result
);
```

**Description** Extracts a subject RDN from a certificate object. This function is optional.

### Parameters

**connCtx**

[ Input ] Pointer to the SSL connection context.

**cert**

[ Input ] Certificate object.

**whichNameItem**

[ Input ] Identifies which RDN to extract.

**instanceIndex**

[ Input ] Identifies which instance of the RDN identified by **whichNameItem** to extract. There may be multiple instances of this RDN, but usually there is only one. Zero indicates the 1st instance. one indicates the 2nd instance, etc.

**result**

[ Input/Output ] Pointer to a buffer to return the RDN. On input, the **length** field must be set to the size of the buffer in the **data** field.

On output, the **length** field will be set to the number of bytes copied into the **data** field.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_NOT_FOUND`

The subject RDN does not exist.

`CIC_ERR_BAD_INDEX`

The instance of the RDN identified by the `instanceIndex` parameter does not exist.

`CIC_ERR_ILLEGAL_PARAM`

The value of the `whichNameItem` or the `instanceIndex` parameter is invalid.

`CIC_ERR_SMALL_BUFFER`

The value of the `length` field in `result` is too small to hold the RDN. On output, the `length` field will be set to the size of buffer required for the RDN.

## ssl\_ExtractRawCertData( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_ExtractRawCertData(
    const ssl_ConnectionContext* connCtx,
    const ssl_Cert* cert,
    cic_Buffer* result
);
```

### Description

Extracts the raw certificate data from a certificate object. This function is optional.

### Parameters

**connCtx**

[ Input ] Pointer to the SSL connection context.

**cert**

[ Input ] Certificate object.

**result**

[ Input/Output ] Pointer to a buffer to return the raw certificate data. On input, the **length** field must be set to the size of the buffer in the **data** field. On output, the **length** field will be set to the number of bytes copied into the **data** field.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SMALL\_BUFFER**

The **result** buffer is too small to hold the certificate data. On output, the **length** will be set to the size of buffer required for the certificate data.

# SSL Plus Miscellaneous Functions

The functions in this section perform the following tasks:

- Return the version number of the SSL Plus library.
- Decode an SSL record into printable data for the logging function.
- Generates random data to be used in generating key material. This is useful if your application uses the EAP - TLS protocol.
- Sends an SSL3 or TLS1 alert message.



## ssl\_GetVersion( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_GetVersion(  
    cic_Uint16* length,  
    char* version  
);
```

### Description

Returns the version number of the SSL Plus library. The **version** is returned as a NULL-terminated string in the following format:

**major.minor.patch**

where

**major** = major release version ( numeric )

**minor** = minor release version ( numeric )

**patch** = patch release version ( alphanumeric )

### Parameters

**length**

[ Input/Output ] On input, set this to the size of the version buffer. On output, this is set to the number of bytes copied into the version buffer, including the NULL terminator byte. If the length is too small an error is returned and **length** is set to the size of buffer required.

**version**

[ Output ] Buffer to return the library version string.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SMALL\_BUFFER**

Version buffer is too small.

## ssl\_DecodeRecord( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_DecodeRecord(  
    const ssl_Callbacks* sslCallbacks,  
    const ssl_CB_LogSubType subType,  
    const cic_Uint32 dataLen,  
    const cic_Uint8* data,  
    cic_Uint32* outbufLen,  
    cic_Uint8* outbuf  
);
```

### Description

Utility function to be used in conjunction with **ssl\_LogFunc** to decode SSL records into printable data.

Notes:

- ( 1 ) **outbuf** will be terminated with a trailing NULL character.
- ( 2 ) **outbufLen** includes the trailing NULL character.

### Parameters

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**subType**

[ Input ] See **ssl\_CB\_LogSubType** for details.

**dataLen**

[ Input ] Length of data.

**data**

[ Input ] Data of interest.

**outbufLen**

[ Input/Output ] On input, the size of the **outbuf** buffer. On output, the size of the data in the **outbuf** buffer.

**outbuf**

[ Output ] Contents of data decoded into printable data.

## ssl\_TLSPRF( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_TLSPRF(
    const ssl_ConnectionContext* connCtx,
    const cic_Uint16 secretLength,
    const cic_Uint8* secretData,
    const cic_Uint16 labelLength,
    const cic_Uint8* labelData,
    const cic_Uint16 randomLength,
    const cic_Uint8* randomData,
    const cic_Uint16 outputLength,
    cic_Uint8* outputData
);
```

### Description

Exposes the TLS PRF function for use in deriving key material. This function is intended for applications wishing to derive key material used in EAP-TLS or similar protocols.

Notes:

( 1 ) The output will be generated by computing: PRF ( secret, label, random )

### Parameters

**connCtx**

[ Input ] Pointer to the connection context.

**secretLength**

[ Input ] Length of secret data.

**secretData**

[ Input ] Secret data. Can be NULL.

**labelLength**

[ Input ] Length of label data.

**labelData**

[ Input ] Label data. Must be non-NULL.

**randomLength**

[ Input ] Length of random data.

**randomData**

[ Input ] Random data. Must be non-NULL.

**outputLength**

[ Input ] Indicates the length of output to generate. **outputData** must be large enough to hold the output.

`outputData`

[ Output ] Buffer for output.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_SSL_BAD_PROTOCOL_VERSION`

Protocol version is not TLS 1.0 or greater

`CIC_ERR_SSL_HANDSHAKE_REQUIRED`

TLS Handshake must be completed first.

## ssl\_SendAlert( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV ssl_SendAlert(
    ssl_ConnectionContext* connCtx,
    const ssl_AlertLevel level,
    const ssl_AlertCode code
);
```

### Description

Sends an SSL3 or TLS1 alert.

Notes:

### Parameters

**connCtx**  
[ Input/Output ] Pointer to the connection context.

**level**  
[ Input ] Level of alert ( Warning or Fatal ).

**code**  
[ Input ] Alert code.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_NO\_PTR**  
NULL pointer was passed.

**CIC\_ERR\_SSL\_BAD\_PROTOCOL\_VERSION**  
Protocol version is not SSL 3.0, TLS 1.0, or greater

**CIC\_ERR\_ILLEGAL\_PARAM**  
Value of the **level** or **code** parameter is invalid.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

# SSL Plus API Protocol Objects

These functions are used with the SSL Plus API to provide support for specific versions of the SSL protocol. You can set the protocol version you require for your application with `ssl_SetProtocol()`. You must call `ssl_SetProtocol()` with at least one of these objects to create a usable configuration.

## SSL\_PROTOCOL\_SSLV2\_SERVERSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_PROTOCOL_SSLV2_SERVERSIDE(
    ssl_GlobalContext* globalCtx,
    const ssl_Callbacks* sslCallbacks,
    struct STM_StateTableRec** stateMachine,
    cic_Bool* isServer,
    ssl_ProtocolVersion* minVersion,
    ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the SSL version 2 Server engine. Negotiates SSL version 2, server-side only.

Notes:

( 1 ) This object installs the **SSL\_PKI\_POLICY\_SSLV2** by default. The default policy can be overridden by calling **ssl\_SetPolicy()**. See **ssl\_SetPolicy()** for more information.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_PROTOCOL\_SSLV2\_CLIENTSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_PROTOCOL_SSLV2_CLIENTSIDE(
    ssl_GlobalContext* globalCtx,
    const ssl_Callbacks* sslCallbacks,
    struct STM_StateTableRec** stateMachine,
    cic_Bool* isServer,
    ssl_ProtocolVersion* minVersion,
    ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the SSL version 2 client engine. Negotiates SSL version 2, client-side only.

Notes:

( 1 ) This object installs the **SSL\_PKI\_POLICY\_SSLV2** by default. The default policy can be overridden by calling **ssl\_SetPolicy**. See **ssl\_SetPolicy()** for more information.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.



## SSL\_PROTOCOL\_SSLV3\_SERVERSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_PROTOCOL_SSLV3_SERVERSIDE(
    ssl_GlobalContext* globalCtx,
    const ssl_Callbacks* sslCallbacks,
    struct STM_StateTableRec** stateMachine,
    cic_Bool* isServer,
    ssl_ProtocolVersion* minVersion,
    ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the SSL version 3 server engine. Negotiates SSL version 3, server-side only.

Notes:

( 1 ) This object installs the **SSL\_PKI\_POLICY\_SSLV3** by default. The default policy can be overridden by calling **ssl\_SetPolicy**. See **ssl\_SetPolicy()** for more information.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_PROTOCOL\_SSLV3\_CLIENTSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_PROTOCOL_SSLV3_CLIENTSIDE(
    ssl_GlobalContext* globalCtx,
    const ssl_Callbacks* sslCallbacks,
    struct STM_StateTableRec** stateMachine,
    cic_Bool* isServer,
    ssl_ProtocolVersion* minVersion,
    ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the SSL version 3 client engine. Negotiates SSL version 3, client side only.

Notes:

( 1 ) This object installs the **SSL\_PKI\_POLICY\_SSLV3** by default. The default policy can be overridden by calling **ssl\_SetPolicy**. See **ssl\_SetPolicy()** for more information.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to return allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_PROTOCOL\_TLSV1\_SERVERSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_PROTOCOL_TLSV1_SERVERSIDE(
    ssl_GlobalContext* globalCtx,
    const ssl_Callbacks* sslCallbacks,
    struct STM_StateTableRec** stateMachine,
    cic_Bool* isServer,
    ssl_ProtocolVersion* minVersion,
    ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the TLS version 1 server engine. Negotiates TLS version 1, server-side only.

Notes:

( 1 ) This object installs the **SSL\_PKI\_POLICY\_TLSV1** by default. The default policy can be overridden by calling **ssl\_SetPolicy**. See **ssl\_SetPolicy()** for more information.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to return allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_PROTOCOL\_TLSV1\_CLIENTSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_PROTOCOL_TLSV1_CLIENTSIDE(
    ssl_GlobalContext* globalCtx,
    const ssl_Callbacks* sslCallbacks,
    struct STM_StateTableRec** stateMachine,
    cic_Bool* isServer,
    ssl_ProtocolVersion* minVersion,
    ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the TLS version 1 client engine. Negotiates TLS version 1, client-side only.

Notes:

( 1 ) This object installs the **SSL\_PKI\_POLICY\_TLSV1** by default. The default policy can be overridden by calling **ssl\_SetPolicy**. See **ssl\_SetPolicy()** for more information.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_PROTOCOL\_SSLV3\_SSLV2\_SERVERSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_PROTOCOL_SSLV3_SSLV2_SERVERSIDE(
        ssl_GlobalContext* globalCtx,
        const ssl_Callbacks* sslCallbacks,
        struct STM_StateTableRec** stateMachine,
        cic_Bool* isServer,
        ssl_ProtocolVersion* minVersion,
        ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the SSL version 3 & version 2 server engine. Negotiates SSL version 2 or SSL version 3, server-side only.

Notes:

( 1 ) This object installs **SSL\_PKI\_POLICY\_SSLV3** and **SSL\_PKI\_POLICY\_SSLV2** as the default PKI policies for SSL3 and SSL2 connections. To override these policies see **ssl\_SetPolicy** .

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

### Return Values

**CIC\_ERR\_NONE**

Success.

`CIC_ERR_MEMORY`

Error allocating memory.

## SSL\_PROTOCOL\_SSLV3\_SSLV2\_CLIENTSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_PROTOCOL_SSLV3_SSLV2_CLIENTSIDE(
        ssl_GlobalContext* globalCtx,
        const ssl_Callbacks* sslCallbacks,
        struct STM_StateTableRec** stateMachine,
        cic_Bool* isServer,
        ssl_ProtocolVersion* minVersion,
        ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the SSL version 3 & version 2 client engine. Negotiates SSL version 2 or SSL version 3, client-side only.

Notes:

( 1 ) This object installs **SSL\_PKI\_POLICY\_SSLV3** and **SSL\_PKI\_POLICY\_SSLV2** as the default PKI policies for SSL3 and SSL2 connections. To override these policies see **ssl\_SetPolicy** .

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

### Return Values

**CIC\_ERR\_NONE**

Success.

`CIC_ERR_MEMORY`

Error allocating memory.



## SSL\_PROTOCOL\_TLSV1\_SSLV3\_SSLV2\_SERVERSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_PROTOCOL_TLSV1_SSLV3_SSLV2_SERVERSIDE(
        ssl_GlobalContext* globalCtx,
        const ssl_Callbacks* sslCallbacks,
        struct STM_StateTableRec** stateMachine,
        cic_Bool* isServer,
        ssl_ProtocolVersion* minVersion,
        ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the TLS version 1 or SSL version 3 or SSL version 2 server engine. Negotiates SSL version 2 or SSL version 3 or TLS version 1, server-side only.

Notes:

( 1 ) This object installs **SSL\_PKI\_POLICY\_TLSV1** , **SSL\_PKI\_POLICY\_SSLV3** and **SSL\_PKI\_POLICY\_SSLV2** as the default PKI policies for TLS1, SSL3 and SSL2 connections. To override these policies see **ssl\_SetPolicy** .

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_MEMORY`

Error allocating memory.

## SSL\_PROTOCOL\_TLSV1\_SSLV3\_SSLV2\_CLIENTSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_PROTOCOL_TLSV1_SSLV3_SSLV2_CLIENTSIDE(
        ssl_GlobalContext* globalCtx,
        const ssl_Callbacks* sslCallbacks,
        struct STM_StateTableRec** stateMachine,
        cic_Bool* isServer,
        ssl_ProtocolVersion* minVersion,
        ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the TLS version 1 or SSL version 3 or SSL version 2 client engine. Negotiates SSL version 2 or SSL version 3 or TLS version 1, client-side only.

Notes:

( 1 ) This object installs **SSL\_PKI\_POLICY\_TLSV1** , **SSL\_PKI\_POLICY\_SSLV3** and **SSL\_PKI\_POLICY\_SSLV2** as the default PKI policies for TLS1, SSL3 and SSL2 connections. To override these policies see **ssl\_SetPolicy** .

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_MEMORY`

Error allocating memory.

## SSL\_PROTOCOL\_TLSV1\_SSLV3\_SERVERSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_PROTOCOL_TLSV1_SSLV3_SERVERSIDE(
        ssl_GlobalContext* globalCtx,
        const ssl_Callbacks* sslCallbacks,
        struct STM_StateTableRec** stateMachine,
        cic_Bool* isServer,
        ssl_ProtocolVersion* minVersion,
        ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the TLS version 1 or SSL version 3 server engine. Negotiates SSL version 3 or TLS version 1, server-side only.

Notes:

( 1 ) This object installs **SSL\_PKI\_POLICY\_TLSV1** and **SSL\_PKI\_POLICY\_SSLV3** as the default PKI policies for TLS1 and SSL3 connections. To override these policies see **ssl\_SetPolicy** .

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

### Return Values

**CIC\_ERR\_NONE**

Success.

`CIC_ERR_MEMORY`

Error allocating memory.

## SSL\_PROTOCOL\_TLSV1\_SSLV3\_CLIENTSIDE

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_PROTOCOL_TLSV1_SSLV3_CLIENTSIDE(
        ssl_GlobalContext* globalCtx,
        const ssl_Callbacks* sslCallbacks,
        struct STM_StateTableRec** stateMachine,
        cic_Bool* isServer,
        ssl_ProtocolVersion* minVersion,
        ssl_ProtocolVersion* maxVersion );
```

### Description

**ssl\_ProtocolSide** object which provides support for the TLS version 1 or SSL version 3 client engine. Negotiates SSL version 3 or TLS version 1, client-side only.

Notes:

( 1 ) This object installs **SSL\_PKI\_POLICY\_TLSV1** and **SSL\_PKI\_POLICY\_SSLV3** as the default PKI policies for TLS1 and SSL3 connections. To override these policies see **ssl\_SetPolicy** .

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**sslCallbacks**

[ Input ] Pointer to the OS callbacks.

**stateMachine**

[ Output ] Pointer to return allocated and initialized state machine array.

**isServer**

[ Output ] Pointer to flag which indicates if the protocol side is a server.

**minVersion**

[ Output ] Pointer to return minimum supported protocol version.

**maxVersion**

[ Output ] Pointer to return maximum supported protocol version.

### Return Values

**CIC\_ERR\_NONE**

Success.

`CIC_ERR_MEMORY`

Error allocating memory.



# SSL Plus API Authentication Mode Objects

These functions are used with the SSL Plus API to provide support for specific client authentication modes. You can set the authentication modes for your application with **ssl\_SetClientAuthModes()**. All of these modes are optional and must be available on both the client and the server to enact client authentication.

Servers with client authentication always authenticate the client. Authentication failures are handled by the **ssl\_CheckCertificateChainFunc ( )** callback.

Clients with client authentication are only authenticated if the server requests it.

## SSL\_ALG\_CLIENT\_AUTH\_MODE\_RSA\_SIGN\_SERVERSIDE ( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CLIENT_AUTH_MODE_RSA_SIGN_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_ClientAuthMode** object which provides support for client authentication using RSA signatures on server side.

Notes:

( 1 ) This object is a server side object which must only be installed in applications which implement the server side of the SSL protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()** , or with **ssl\_SetCipherSuites()** .

# SSL\_ALG\_CLIENT\_AUTH\_MODE\_RSA\_SIGN\_CLIENTSIDE ( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CLIENT_AUTH_MODE_RSA_SIGN_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

## Description

**ssl\_ClientAuthMode** object which provides support for client authentication using RSA signatures on client side.

Notes:

( 1 ) This object is a client side object which must only be installed in applications which implement the client side of the SSL protocol.

## Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

## Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()** , or with **ssl\_SetCipherSuites()** .

## SSL\_ALG\_CLIENT\_AUTH\_MODE\_ECDSA\_SIGN\_SECT163K1\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CLIENT_AUTH_MODE_ECDSA_SIGN_SECT163K1_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_ClientAuthMode** object which provides support for client authentication using the ECDSA\_sign method, with the curve sect163k1, on the server side of the protocol.

Notes:

( 1 ) This object is a server side object which must only be installed in applications which implement the server side of the SSL protocol.

( 2 ) In the ECDSA\_sign method, the client supplies a certificate containing an ECDSA public key, and authenticates itself by signing the certificate verify message with its ECDSA key pair.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetCipherSuites()**.

## SSL\_ALG\_CLIENT\_AUTH\_MODE\_ECDSA\_SIGN\_SECT163K1\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CLIENT_AUTH_MODE_ECDSA_SIGN_SECT163K1_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_ClientAuthMode** object which provides support for client authentication using the ECDSA\_sign method, with the curve sect163k1, on the client side of the protocol.

Notes:

( 1 ) This object is a client side object which must only be installed in applications which implement the client side of the SSL protocol.

( 2 ) In the ECDSA\_sign method, the client supplies a certificate containing an ECDSA public key, and authenticates itself by signing the certificate verify message with its ECDSA key pair.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetCipherSuites( )**.

## SSL\_ALG\_CLIENT\_AUTH\_MODE\_ECDSA\_FIXED\_ECDH\_SECT163K1\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CLIENT_AUTH_MODE_ECDSA_FIXED_ECDH_SECT163K1_SERVER
    SIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_ClientAuthMode** object which provides support for client authentication using the ECDSA\_fixed\_ECDH, with the curve sect163k1, on the server side of the protocol.

Notes:

( 1 ) This object is a server side object which must only be installed in applications which implement the server side of the SSL protocol.

( 2 ) In the ECDSA\_fixed\_ECDH method, the client supplies an ECDSA-signed certificate containing an ECDH public key using the same parameters as the server's ECDH public key. The client authenticates itself by computing the master secret and the finished message. This achieves authentication because these computations can only be performed by a party possessing the private key corresponding to one of the ECDH public keys exchanged.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetCipherSuites()**.

## SSL\_ALG\_CLIENT\_AUTH\_MODE\_ECDSA\_FIXED\_ECDH\_SECT163K1\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CLIENT_AUTH_MODE_ECDSA_FIXED_ECDH_SECT163K1_CLIENT
    SIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_ClientAuthMode** object which provides support for client authentication using the ECDSA\_fixed\_ECDH method, with the curve sect163k1, on the client side of the protocol.

Notes:

( 1 ) This object is a client side object which must only be installed in applications which implement the server side of the SSL protocol.

( 2 ) In the ECDSA\_fixed\_ECDH method, the client supplies an ECDSA-signed certificate containing an ECDH public key using the same parameters as the server's ECDH public key. The client authenticates itself by computing the master secret and the finished message. This achieves authentication because these computations can only be performed by a party possessing the private key corresponding to one of the ECDH public keys exchanged.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetCipherSuites( )**.

## SSL\_ALG\_CLIENT\_AUTH\_MODE\_DSS\_SIGN\_CLIENTSIDE ( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CLIENT_AUTH_MODE_DSS_SIGN_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_ClientAuthMode** object which provides support for client authentication using DSS signatures on client side.

Notes: ( 1 ) This object is a client side object which must only be installed in applications which implement the client side of the SSL protocol.

### Parameters

**globalCtx**  
[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()** , or with **ssl\_SetCipherSuites()** .



# SSL\_ALG\_CLIENT\_AUTH\_MODE\_DSS\_SIGN\_SERVERSIDE ( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CLIENT_AUTH_MODE_DSS_SIGN_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

## Description

**ssl\_ClientAuthMode** object which provides support for client authentication using DSS signatures on server side.

Notes: ( 1 ) This object is a server side object which must only be installed in applications which implement the server side of the SSL protocol.

## Parameters

**globalCtx**  
[ Input/Output ] A pointer to the SSL Global Context.

## Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()** , or with **ssl\_SetCipherSuites()** .

## SSL\_ALG\_CLIENT\_AUTH\_MODE\_RSA\_SIGN\_CLIENTSIDE\_CS( )

```
CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CLIENT_AUTH_MODE_RSA_SIGN_CLIENTSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_ClientAuthMode** object which provides support for client authentication using RSA signatures on the client side. Use this object if you have the Cryptoswift RSA hardware accelerator from Rainbow Technologies.

Notes: ( 1 ) This object is a client side object which must only be installed in applications which implement the client side of the SSL protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
This protocol side of this mode is different from the side selected with a previous mode or with **ssl\_SetProtocol( )** , or with **ssl\_SetCipherSuites( )** .

## SSL\_ALG\_CLIENT\_AUTH\_MODE\_RSA\_SIGN\_SERVERSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CLIENT_AUTH_MODE_RSA_SIGN_SERVERSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_ClientAuthMode** object which provides support for client authentication using RSA signatures on the server side. Use this object if you have the Cryptoswift RSA hardware accelerator from Rainbow Technologies.

Notes: ( 1 ) This object is a server side object which must only be installed in applications which implement the server side of the SSL protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
This protocol side of this mode is different from the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetCipherSuites()**.

# SSL Plus API Ciphersuite Objects

These functions are used with the SSL Plus API to provide support for specific cipher suites. You can set the cipher suites for your application with **`ssl_SetCipherSuites()`**.

You must install at least one of these objects in your application. In a client/server environment, both applications must have at least one of these objects in common.

## SSL\_ALG\_CIPHER\_ECDH\_ECDSA\_SECT163K1\_WITH\_RC4\_128\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_ECDH_ECDSA_SECT163K1_WITH_RC4_128_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_ECDH\_ECDSA\_WITH\_RC4\_128\_SHA, using the curve sect163k1, on the server side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_ECDH\_ECDSA\_SECT163K1\_WITH\_RC4\_128\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_ECDH_ECDSA_SECT163K1_WITH_RC4_128_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_ECDH\_ECDSA\_WITH\_RC4\_128\_SHA, using the curve sect163k1, on the client side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_ECDH\_ECDSA\_SECT163K1\_NULL\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_ECDH_ECDSA_SECT163K1_NULL_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_ECDH\_ECDSA\_NULL\_SHA, using the curve sect163k1, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_ECDH\_ECDSA\_SECT163K1\_NULL\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_ECDH_ECDSA_SECT163K1_NULL_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_ECDH\_ECDSA\_NULL\_SHA, using the curve sect163k1, on the client side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.



## SSL\_ALG\_CIPHER\_RSA\_WITH\_RC4\_128\_MD5\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_RC4_128_MD5_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_RC4\_128\_MD5 on server side.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_RC4\_128\_MD5\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_RC4_128_MD5_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_RC4\_128\_MD5 on client side.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_RC4\_128\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_RC4_128_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_RC4\_128\_SHA on server side.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_RC4\_128\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_RC4_128_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_RC4\_128\_SHA on client side.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_AES\_128\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_AES_128_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA on server side.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_AES\_128\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_AES_128_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA on client side.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_AES\_256\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_AES_256_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA on server side.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_AES\_256\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_AES_256_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA on client side.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.



## SSL\_ALG\_CIPHER\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_3DES_EDE_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA on server side.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_3DES_EDE_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA on client side.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_EXPORT_WITH_RC4_40_MD5_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5 on server side.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_EXPORT_WITH_RC4_40_MD5_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5 on client side.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_EXPORT_WITH_DES40_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA on server side.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_EXPORT_WITH_DES40_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA on client side.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_DES\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_DES_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_DES\_CBC\_SHA on server side.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_DES\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_DES_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_DES\_CBC\_SHA on client side.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.



## SSL\_ALG\_CIPHER\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_EXPORT\_WITH\_DES40\_CBC\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DHE\_DSS\_WITH\_DES\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_WITH_DES_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_WITH\_DES\_CBC\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_DHE\_DSS\_WITH\_DES\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_WITH_DES_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_WITH\_DES\_CBC\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DHE\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_WITH_3DES_EDE_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_DHE\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_WITH_3DES_EDE_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_WITH\_3DES\_EDE\_CBC\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DHE\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA\_CLIENTSIDE()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DHE\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.



## SSL\_ALG\_CIPHER\_DHE\_RSA\_WITH\_DES\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_RSA_WITH_DES_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_RSA\_WITH\_DES\_CBC\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_DHE\_RSA\_WITH\_DES\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_RSA_WITH_DES_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_RSA\_WITH\_DES\_CBC\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_RSA_WITH_3DES_EDE_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_RSA_WITH_3DES_EDE_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DH\_ANON\_EXPORT\_WITH\_DES40\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DH_ANON_EXPORT_WITH_DES40_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DH\_anon\_EXPORT\_WITH\_DES40\_CBC\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_DH\_ANON\_EXPORT\_WITH\_DES40\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DH_ANON_EXPORT_WITH_DES40_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DH\_anon\_EXPORT\_WITH\_DES40\_CBC\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DH\_ANON\_EXPORT\_WITH\_RC4\_40\_MD5\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DH_ANON_EXPORT_WITH_RC4_40_MD5_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DH\_anon\_EXPORT\_WITH\_RC4\_40\_MD5, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DH\_ANON\_EXPORT\_WITH\_RC4\_40\_MD5\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DH_ANON_EXPORT_WITH_RC4_40_MD5_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DH\_anon\_EXPORT\_WITH\_RC4\_40\_MD5, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.



## SSL\_ALG\_CIPHER\_DH\_ANON\_WITH\_RC4\_128\_MD5\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DH_ANON_WITH_RC4_128_MD5_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DH\_anon\_WITH\_RC4\_128\_MD5, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_DH\_ANON\_WITH\_RC4\_128\_MD5\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DH_ANON_WITH_RC4_128_MD5_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DH\_anon\_WITH\_RC4\_128\_MD5, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DH\_ANON\_WITH\_DES\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DH_ANON_WITH_DES_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DH\_anon\_WITH\_DES\_CBC\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_DH\_ANON\_WITH\_DES\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DH_ANON_WITH_DES_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DH\_anon\_WITH\_DES\_CBC\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DH\_ANON\_WITH\_3DES\_EDE\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DH_ANON_WITH_3DES_EDE_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DH\_anon\_WITH\_3DES\_EDE\_CBC\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_DH\_ANON\_WITH\_3DES\_EDE\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DH_ANON_WITH_3DES_EDE_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DH\_anon\_WITH\_3DES\_EDE\_CBC\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DHE\_DSS\_EXPORT1024\_WITH\_DES\_CBC\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_EXPORT1024\_WITH\_DES\_CBC\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DHE\_DSS\_EXPORT1024\_WITH\_DES\_CBC\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_EXPORT1024\_WITH\_DES\_CBC\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.



## SSL\_ALG\_CIPHER\_DHE\_DSS\_EXPORT1024\_WITH\_RC4\_56\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_EXPORT1024\_WITH\_RC4\_56\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_DHE\_DSS\_EXPORT1024\_WITH\_RC4\_56\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_EXPORT1024\_WITH\_RC4\_56\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_DHE\_DSS\_WITH\_RC4\_128\_SHA\_CLIENTSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_WITH_RC4_128_SHA_CLIENTSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_WITH\_RC4\_128\_SHA, on the client side of the protocol.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_DHE\_DSS\_WITH\_RC4\_128\_SHA\_SERVERSIDE( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_DHE_DSS_WITH_RC4_128_SHA_SERVERSIDE(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_DHE\_DSS\_WITH\_RC4\_128\_SHA, on the server side of the protocol.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5\_CLIENTSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_EXPORT_WITH_RC4_40_MD5_CLIENTSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5 on client side for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**  
[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5\_SERVERSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_EXPORT_WITH_RC4_40_MD5_SERVERSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5 on server side, for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**

[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA\_CLIENTSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_3DES_EDE_CBC_SHA_CLIENTSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA on client side for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**

[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA\_SERVERSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_3DES_EDE_CBC_SHA_SERVERSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA on server side, for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**

[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.



## SSL\_ALG\_CIPHER\_RSA\_WITH\_AES\_128\_CBC\_SHA\_CLIENTSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_AES_128_CBC_SHA_CLIENTSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA on client side, for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**  
[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_AES\_128\_CBC\_SHA\_SERVERSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_AES_128_CBC_SHA_SERVERSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA on server side, for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**

[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_AES\_256\_CBC\_SHA\_CLIENTSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_AES_256_CBC_SHA_CLIENTSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA on client side, for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**  
[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_AES\_256\_CBC\_SHA\_SERVERSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_AES_256_CBC_SHA_SERVERSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA on server side, for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**

[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_RC4\_128\_MD5\_CLIENTSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_RC4_128_MD5_CLIENTSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_RC4\_128\_MD5 on client side, for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**  
[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_RC4\_128\_MD5\_SERVERSIDE\_CS()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_RC4_128_MD5_SERVERSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_RC4\_128\_MD5 on server side, for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**

[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_RC4\_128\_SHA\_CLIENTSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_RC4_128_SHA_CLIENTSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_RC4\_128\_SHA on client side, for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**  
[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_RC4\_128\_SHA\_SERVERSIDE\_CS()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_RC4_128_SHA_SERVERSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_RC4\_128\_SHA on server side, for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**

[ Input/Output ] A pointer to the SSL Global Context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

This protocol side of this mode is different than the side selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.



## SSL\_ALG\_CIPHER\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA\_SERVERSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_EXPORT_WITH_DES40_CBC_SHA_SERVERSIDE_CS
    (
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA on server side for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**  
The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA\_CLIENTSIDE\_CS()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_EXPORT_WITH_DES40_CBC_SHA_CLIENTSIDE_CS
(
    ssl_GlobalContext* globalCtx
);
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA on client side for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_DES\_CBC\_SHA\_SERVERSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_DES_CBC_SHA_SERVERSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_DES\_CBC\_SHA on server side for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**  
[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol( )**, or with **ssl\_SetClientAuthModes( )**.

## SSL\_ALG\_CIPHER\_RSA\_WITH\_DES\_CBC\_SHA\_CLIENTSIDE\_CS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_CIPHER_RSA_WITH_DES_CBC_SHA_CLIENTSIDE_CS(
        ssl_GlobalContext* globalCtx
    );
```

### Description

**ssl\_CipherSuite** object which provides support for the ciphersuite TLS\_RSA\_WITH\_DES\_CBC\_SHA on client side for Rainbow Technologies CryptoSwift.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_SSL\_BAD\_SIDE**

The protocol side of this mode is different from that selected with a previous mode or with **ssl\_SetProtocol()**, or with **ssl\_SetClientAuthModes()**.

# SSL Plus API Private Key Decryption Suite Objects

These functions are used with the SSL Plus API to provide support for specific private key decryption suites. You can set the decryption suite for your application with **`ssl_CreateCertList()`**.

When setting up a local identity at least one of these suites must be installed.

## SSL\_ALG\_PRV\_KEY\_DECRYPT\_NULL

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PRV_KEY_DECRYPT_NULL(  
    ssl_GlobalContext* globalCtx,  
    const ssl_Encoding prvKeyEncoding,  
    struct CTR_Buffer* encryptedPrvKeyInfo,  
    struct CTR_Buffer* password,  
    struct PKC_Object** prvKey );
```

### Description

A private key decryption suite which supports decrypting PKCS #8 ( version 1.2 ) encoded private keys which have been not been encrypted.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**prvKeyEncoding**

[ Input ] The encoding object to decode the private key info.

**encryptedPrvKeyInfo**

[ Input ] Pointer to the encrypted private key info.

**password**

[ Input ] The password.

**prvKey**

[ Output ] Pointer to return the private key object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_ALG\_PRV\_KEY\_DECRYPT\_PBE\_MD5\_DES

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    SSL_ALG_PRV_KEY_DECRYPT_PBE_MD5_DES(
        ssl_GlobalContext* globalCtx,
        const ssl_Encoding prvKeyEncoding,
        struct CTR_Buffer* encryptedPrvKeyInfo,
        struct CTR_Buffer* password,
        struct PKC_Object** prvKey );
```

### Description

A private key decryption suite which supports decrypting PKCS #8 ( version 1.2 ) encoded private keys which have been encrypted with the algorithm PBE\_MD5\_DES from PKCS #5 ( version 1.5 ).

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**prvKeyEncoding**

[ Input ] The encoding object to decode the private key info.

**encryptedPrvKeyInfo**

[ Input ] Pointer to the encrypted private key info.

**password**

[ Input ] The password.

**prvKey**

[ Output ] Pointer to return the private key object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

# SSL Plus API Certificate Format Objects

These functions are used with the SSL Plus API to provide support for specific certificate formats.



## SSL\_CERT\_FMT\_X509( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_CERT_FMT_X509(  
    ssl_GlobalContext* globalCtx,  
    ssl_CertFormatObj* format  
);
```

### Description

**ssl\_CertFormat** object which installs in the global context a feature to support X509 certificates and returns an internal format object which references this feature. This object supports both RSA and ECC certificates.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**format**

[ Output ] Pointer to return format.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_CERT\_FMT\_X509\_RSA( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_CERT_FMT_X509_RSA(  
    ssl_GlobalContext* globalCtx,  
    ssl_CertFormatObj* format  
);
```

### Description

**ssl\_CertFormat** object which installs in the global context a feature to support X509 certificates and returns an internal format object which references this feature. This object supports only RSA.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**format**

[ Output ] Pointer to return format.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_CERT\_FMT\_X509\_ECC( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_CERT_FMT_X509_ECC(  
    ssl_GlobalContext* globalCtx,  
    ssl_CertFormatObj* format  
);
```

### Description

**ssl\_CertFormat** object which installs in the global context a feature to support X509 certificates and returns an internal format object which references this feature. This object supports only ECC certificates.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**format**

[ Output ] Pointer to return format.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_CERT\_FMT\_X509\_DSS( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_CERT_FMT_X509_DSS(  
    ssl_GlobalContext* globalCtx,  
    ssl_CertFormatObj* format  
);
```

### Description

**ssl\_CertFormat** object which installs in the global context a feature to support X509 certificates and returns an internal format object which references this feature. This object supports only DSS certificates.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**format**

[ Output ] Pointer to return format.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

# SSL Plus API Key and Certificate Decoding Objects

These functions are used with the SSL Plus API to provide support for decoding certificates and keys. Both client and server applications must install at least one of these objects.

## SSL\_ENC\_PEM( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ENC_PEM(  
    ssl_GlobalContext* globalCtx,  
    ssl_EncodingObj* encoding  
);
```

### Description

**ssl\_Encoding** object which allocates and initializes an encoding structure to support PEM base64 encoding.

### Parameters

**globalCtx**

[ Input ] A pointer to the global context.

**encoding**

[ Output ] Pointer to return the encoding.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_ENC\_DER( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ENC_DER(  
    ssl_GlobalContext* globalCtx,  
    ssl_EncodingObj* encoding  
);
```

### Description

**ssl\_Encoding** object which allocates and initializes an encoding structure to support DER encoding.

### Parameters

**globalCtx**  
[ Input ] A pointer to the global context.

**encoding**  
[ Output ] Pointer to return the encoding.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_MEMORY**  
Error allocating memory.

# SSL Plus API PKCS #12 Certificate and Private Key Decryption Suite Objects

These functions are used with the SSL Plus API to provide support for specific algorithms necessary to parse and decrypt a PKCS #12 PFX. You can set the decryption suite ( s ) for your application with **ssl\_AddPkcs12Pfx()**.

When setting up a local identity with a PKCS #12 PFX, at least one of these suites must be installed. Multiple algorithms may be required depending on how the PFX was constructed.



**Decryption Suite Objects****SSL\_ALG\_PKCS12\_PBE\_SHA\_RC4\_128( )**

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_SHA_RC4_128(  
    ssl_GlobalContext* globalCtx,  
    ssl_P12AlgObj* algObj  
);
```

**Description**

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using 128-bit RC4 with SHA1. ie. pbeWithSHAAnd128BitRC4 from the PKCS #12 specification.

**Parameters**

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

**Return Values**

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_ALG\_PKCS12\_PBE\_SHA\_RC4\_40( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_SHA_RC4_40(
    ssl_GlobalContext* globalCtx,
    ssl_P12AlgObj* algObj
);
```

### Description

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using 40-bit RC4 with SHA1. ie. pbeWithSHAAnd40BitRC4 from the PKCS #12 specification.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**Decryption Suite Objects****SSL\_ALG\_PKCS12\_PBE\_SHA\_3DES ( )**

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_SHA_3DES(  
    ssl_GlobalContext* globalCtx,  
    ssl_P12AlgObj* algObj  
);
```

**Description**

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using 128-bit Triple DES with SHA1. ie. pbeWithSHAAnd3-KeyTripleDES-CBC from the PKCS #12 specification.

**Parameters**

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

**Return Values**

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_ALG\_PKCS12\_PBE\_SHA\_2KEY\_3DES( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_SHA_2KEY_3DES(  
    ssl_GlobalContext* globalCtx,  
    ssl_P12AlgObj* algObj  
);
```

### Description

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using 2 Key Triple DES with SHA1. ie. pbeWithSHAAnd2-KeyTripleDES-CBC from the PKCS #12 specification.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**Decryption Suite Objects****SSL\_ALG\_PKCS12\_PBE\_SHA\_RC2\_128( )**

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_SHA_RC2_128(  
    ssl_GlobalContext* globalCtx,  
    ssl_P12AlgObj* algObj  
);
```

**Description**

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using 128-bit RC2 with SHA1. ie. pbeWithSHAAnd128BitRC2-CBC from the PKCS #12 specification.

**Parameters**

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

**Return Values**

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_ALG\_PKCS12\_PBE\_SHA\_RC2\_40( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_SHA_RC2_40(  
    ssl_GlobalContext* globalCtx,  
    ssl_P12AlgObj* algObj  
);
```

### Description

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using 40-bit RC2 with SHA1. ie. pbewithSHAAnd40BitRC2-CBC from the PKCS #12 specification.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**Decryption Suite Objects****SSL\_ALG\_PKCS12\_PBE\_MD2\_DES( )**

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_MD2_DES(  
    ssl_GlobalContext* globalCtx,  
    ssl_P12AlgObj* algObj  
);
```

**Description**

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using DES with MD2. ie. pbeWithMD2AndDES-CBC from the PKCS #5 specification.

**Parameters**

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

**Return Values**

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_ALG\_PKCS12\_PBE\_MD2\_RC2( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_MD2_RC2(  
    ssl_GlobalContext* globalCtx,  
    ssl_P12AlgObj* algObj  
);
```

### Description

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using RC2 with MD2. ie. pbeWithMD2AndRC2-CBC from the PKCS #5 specification.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.



**Decryption Suite Objects****SSL\_ALG\_PKCS12\_PBE\_MD5\_DES( )**

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_MD5_DES(  
    ssl_GlobalContext* globalCtx,  
    ssl_P12AlgObj* algObj  
);
```

**Description**

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using DES with MD5. ie. pbeWithMD5AndDES-CBC from the PKCS #5 specification.

**Parameters**

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

**Return Values**

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_ALG\_PKCS12\_PBE\_MD5\_RC2( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_MD5_RC2(
    ssl_GlobalContext* globalCtx,
    ssl_P12AlgObj* algObj
);
```

### Description

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using RC2 with MD5. ie. pbeWithMD5AndRC2-CBC from the PKCS #5 specification.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**Decryption Suite Objects****SSL\_ALG\_PKCS12\_PBE\_SHA1\_DES ( )**

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_SHA1_DES(  
    ssl_GlobalContext* globalCtx,  
    ssl_P12AlgObj* algObj  
);
```

**Description**

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using DES with SHA1. ie. pbeWithSHA1AndDES-CBC from the PKCS #5 specification.

**Parameters**

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

**Return Values**

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

## SSL\_ALG\_PKCS12\_PBE\_SHA1\_RC2( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_PKCS12_PBE_SHA1_RC2(  
    ssl_GlobalContext* globalCtx,  
    ssl_P12AlgObj* algObj  
);
```

### Description

A PKCS #12 certificate and private key decryption suite which supports decrypting private keys or certificates encrypted using RC2 with SHA1. ie. pbeWithSHA1AndRC2-CBC from the PKCS #5 specification.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**algObj**

[ Input/Output ] Pointer to the algorithm object.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

# SSL Plus API RNG Objects

These objects provide support for generating pseudo - random data.

## SSL\_ALG\_ANSIPRNG( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_ALG_ANSIPRNG(  
    ssl_GlobalContext* globalCtx  
);
```

### Description

A pseudo-random number generator provider which uses the ANSI RNG feature from Security Builder.

Notes:

### Parameters

`globalCtx`

[ Input ] Pointer to the global context.

### Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_MEMORY`

Error allocating memory.

# SSL Plus API Policy Objects

These functions define the PKI policies that govern how the certificate format objects are used. At least one of the policy objects is installed by each protocol object, but this behaviour can be overridden. Use **ssl\_SetPolicy()** to override the default policy for a protocol.

## SSL\_PKI\_POLICY\_SSLV2( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_PKI_POLICY_SSLV2(  
    ssl_GlobalContext* globalCtx,  
    ssl_ProtocolVersion protocolVersion  
);
```

### Description

**ssl\_PKIPolicyObject** object which enforces the default PKI policy required by SSL2. This policy includes only the policy requirements for SSL 2.0

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**protocolVersion**

[ Input ] Protocol version which will use this policy.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_ILLEGAL\_PARAM**

An invalid protocol version, such as

**SSL\_ProtocolVersion\_Undetermined** was specified.



## SSL\_PKI\_POLICY\_SSLV3( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_PKI_POLICY_SSLV3(  
    ssl_GlobalContext* globalCtx,  
    ssl_ProtocolVersion protocolVersion  
);
```

### Description

**ssl\_PKIPolicyObject** object which enforces the default PKI policy required by SSL3. This policy includes only the policy requirements for SSL 3. 0.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**protocolVersion**

[ Input ] Protocol version which will use this policy.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_ILLEGAL\_PARAM**

An invalid protocol version, such as

**SSL\_ProtocolVersion\_Undetermined** was specified.

## SSL\_PKI\_POLICY\_TLSV1 ( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_PKI_POLICY_TLSV1(  
    ssl_GlobalContext* globalCtx,  
    ssl_ProtocolVersion protocolVersion  
);
```

### Description

**ssl\_PKIPolicyObject** object which enforces the default PKI policy required by RFC2246. This policy includes only the policy requirements for TLS 1.0.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**protocolVersion**

[ Input ] Protocol version which will use this policy.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_ILLEGAL\_PARAM**

An invalid protocol version, such as

**SSL\_ProtocolVersion\_Undetermined** was specified.

## SSL\_PKI\_POLICY\_WAPV2( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_PKI_POLICY_WAPV2(  
    ssl_GlobalContext* globalCtx,  
    ssl_ProtocolVersion protocolVersion  
);
```

### Description

**ssl\_PKIPolicyObject** object which enforces the default PKI required by WAP 2. 0. This policy includes the policy requirements for TLS 1. 0 and WAP 2. 0.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**protocolVersion**

[ Input ] Protocol version which will use this policy.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_MEMORY**

Error allocating memory.

**CIC\_ERR\_ILLEGAL\_PARAM**

An invalid protocol version, such as

**SSL\_ProtocolVersion\_Undetermined** was specified.

## SSL\_PKI\_POLICY\_MIDPV2( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV SSL_PKI_POLICY_MIDPV2(
    ssl_GlobalContext* globalCtx,
    ssl_ProtocolVersion protocolVersion
);
```

### Description

**ssl\_PKIPolicyObject** object which enforces the default PKI required by MIDP V2.0. This policy includes the policy requirements for MIDP V2.0.

Note:

The key usage is prepared according to the MIDP v2.0(JSR-118) profile.

The profile states:

The MIDP V2.0 Profile (Draft 7) follows the WAPCert policy with the inclusion of the following points:

- Excluding Section 6.2, (WAP Cert) User Certificates for Authentication.
- Excluding Section 6.3, (WAP Cert) User Certificates for Digital Signature.
- Exclude (rfc2459) the requirements from Paragraphs 4 of Section 4.2 - Standard Certificate Extensions. A conforming implementation of this specification does not need to recognize extensions that must or may be critical including certificate policies, name constraints and policy constraints.
- Exclude rfc2459 Section 6.2 Extending Path Validation Support for Policy Certificate Authority or a policy attributes is not required.

Certificate processing for OTA also states that the key usage extension MUST be recognized if present. This conflicts with the third point above. We therefore recognize the key usage extension, but we do not recognize the other extensions listed in Section 4.2 paragraph 2 of rfc2459 such as the basic constraints extension.

### Parameters

**globalCtx**

[ Input/Output ] Pointer to the SSL global context.

**protocolVersion**

[ Input ] Protocol version which will use this policy.

### Return Values

**CIC\_ERR\_NONE**

Success.

`CIC_ERR_MEMORY`

Error allocating memory.

`CIC_ERR_ILLEGAL_PARAM`

An invalid protocol version, such as

`SSL_ProtocolVersion_Undetermined` was specified.



# 2

## SSL Plus Callbacks

---

### Mandatory Callbacks

This section details the memory, system and I/O callbacks that are mandatory for using SSL Plus. These functions are called by the SSL Plus library.

Some callbacks are provided in source form for various platforms. You may need to adjust these default implementations to suit your particular needs.

## cic\_MallocFunc

```
typedef void* (CIC_CALLCONV *cic_MallocFunc )(
    cic_Uint32 size,
    void * memRef
);
```

**Description**                      Template for the callback function which the library uses to allocate memory.

**Parameters**

<b>size</b>	[Input]   Number of bytes to allocate.
<b>memRef</b>	[Input]   User reference parameter.

**Return Values**                      Pointer to allocated memory



## cic\_FreeFunc

```
typedef void (CIC_CALLCONV *cic_FreeFunc )(
    void *block,
    void * memRef
);
```

**Description**                      Template for the callback function which the library uses to free memory.

**Parameters**

<b>block</b>	[Input/Output] Pointer to the block to be freed.
<b>memRef</b>	[Input] User reference parameter.

**Return Values**                      None

## cic\_MemsetFunc

```
typedef void (CIC_CALLCONV *cic_MemsetFunc )(
    void *block,
    cic_Uint8 value,
    cic_Uint32 length
);
```

**Description**                      Template for the callback function which the library uses to initialize memory.

**Parameters**

<b>block</b>	[Input/Output] Block to initialize.
<b>value</b>	[Input] Value with which to initialize the block.
<b>length</b>	[Input] Number of bytes to initialize.

**Return Values**                      None

## cic\_MemcpyFunc

```
typedef void (CIC_CALLCONV *cic_MemcpyFunc )(
    void *dest,
    const void *src,
    cic_Uint32 length
);
```

**Description**      Template for the callback function which the library uses to copy memory.

**Parameters**

<code>dest</code>	[Output] Destination block.
<code>src</code>	[Input] Source block.
<code>length</code>	[Input] Number of bytes to copy.

**Return Values**      None

## cic\_MemcmpFunc

```
typedef cic_Int32(  
    CIC_CALLCONV* cic_MemcmpFunc )(  
    const void* a,  
    const void* b,  
    cic_Uint32 length );
```

**Description**      Template for the callback function which the library uses to compare memory.

**Parameters**

<b>a</b>	[ Output ] First memory address.
<b>b</b>	[ Input ] Second memory address.
<b>length</b>	[ Input ] Number of bytes to compare.

**Return Values**

0	if <b>a</b> == <b>b</b>
less than 0	if <b>a</b> < <b>b</b>
greater than 0	if <b>a</b> > <b>b</b>

## cic\_TimeFunc

```
typedef cic_Uint32(  
    CIC_CALLCONV* cic_TimeFunc )(  
    void );
```

**Description**      Template for the callback function which the library uses to get the time.

**Return Values**      The number of seconds elapsed since 1 / 1 / 1970.

## ssl\_Callbacks

```
typedef struct ssl_Callbacks {  
    cic_MallocFunc pmalloc;  
    cic_FreeFunc pfree;  
    cic_MemsetFunc pmemset;  
    cic_MemcpyFunc pmemcpy;  
    cic_MemcmpFunc pmemcmp;  
    cic_TimeFunc ptime;  
    void* memRef;  
    ssl_RandomFunc prandom;  
    void* randomRef;  
    ssl_SurrenderFunc psurrender;  
    void* surrenderRef;  
} ssl_Callbacks;
```

### Description

This structure is passed to **ssl\_CreateGlobalContext()** to set the mandatory callback functions in the global context.

## ssl\_RandomFunc

```
typedef cic_Err(  
    CIC_CALLCONV* ssl_RandomFunc )(  
    cic_Buffer* data,  
    void* const randomRef );
```

### Description

Template for the callback function that the library uses to generate seeding material for an SSL connection.

Notes:

This callback generates a seed which is used by SSL Plus to help seed a PRNG.

### Parameters

**data**

[ Input / Output ] On input **data.length** indicates the number of seed bytes to generate. On output **data.data** contains the generated random seed bytes and **data.length** contains the number of seed bytes actually generated.

**randomRef**

[ Input ] Optional user reference parameter.

### Return Values

Any error codes that your callback function generates will cause the SSL API function that called your function to fail.

## ssl\_IOFunc

```
typedef cic_Err(
    CIC_CALLCONV* ssl_IOFunc )(
        cic_Buffer data,
        cic_Uint32* processed,
        ssl_IOState ioState,
        void* const connRef );
```

### Description

Template for the I/O callback functions that the library uses to transfer data on an SSL connection.

Design of the read callback must be as follows:

Return value must be **CIC\_ERR\_NONE** if, and only if, the requested number of bytes has been received. Return value must be **CIC\_ERR\_WOULD\_BLOCK** if the requested number of bytes could not be read. If the network connection closes or there is a network error, you should return **CIC\_ERR\_SSL\_IO**. However, in the case of an SSL 3.0 connection, if the peer closes the connection without sending a "goodbye kiss", you should return **CIC\_ERR\_SSL\_CONNECTION\_CLOSED\_GRACEFUL**. This prevents SSL Plus from treating this condition as an error.

Design of the write callback must be as follows:

Return value must be **CIC\_ERR\_NONE** if, and only if, all the bytes in the buffer have been sent. Return value must be **CIC\_ERR\_WOULD\_BLOCK** if not all of the bytes could be sent. In this case you should set **processed** to the number of bytes that were actually sent.

### Parameters

**data**

[ Input / output ] Data that was read (read callback) or data to be written (write callback).

**processed**

[ Input / Output ] Buffer to return number of bytes read/written.

**ioState**

[Input] Set by SSL Plus to **ssl\_IOState\_ContainsFinished** if the outgoing message is a TLS "Finished" message. See the chapter on EAP in the User's Guide for more information.

**connRef**

[ Input ] Optional user reference parameter.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_WOULD\_BLOCK**

I/O is blocking.



`CIC_ERR_SSL_IO`

I/O error.

`CIC_ERR_SSL_CONNECTION_CLOSED_GRACEFUL`

I/O is closed, override the goodbye kiss.

# Optional Callbacks

These are the templates for the optional SSL callback functions.

## ssl\_AlertFunc

```
typedef cic_Err(  
    CIC_CALLCONV* ssl_AlertFunc )(  
    const ssl_ConnectionContext* connCtx,  
    const ssl_AlertLevel level,  
    const ssl_AlertCode code,  
    void* const alertRef );
```

### Description

Template for the callback functions that processes alerts for a SSL connection.

### Parameters

**connCtx**

[ Input ] A pointer to the SSL context structure.

**level**

[ Input ] Alert level.

**code**

[ Input ] Alert code.

**alertRef**

[ Input ] Optional user reference parameter.

### Return Values

Any error codes that your callback function generates will cause the SSL API function that called your function to fail.

## ssl\_AlertLevel

```
typedef cic_Uint16 ssl_AlertLevel
```

**Description** Defines the alert level used in **ssl\_AlertFunc**.

**Enumeration**

```
SSL_AlertLevel_Fatal
```

Fatal alert.

```
SSL_AlertLevel_Warning
```

Warning alert.

## ssl\_AlertCode

```
typedef cic_Uint16 ssl_AlertCode
```

### Description

The alert code type.

Note:

Features (including alerts) that are based on internet drafts are subject to change and should be used with caution.

### Enumeration

#### SSL\_AlertCode\_CloseNotify

This alert notifies the recipient that the sender will not send any more messages on this connection. The session becomes un-resumable if any connection is terminated without proper close\_notify messages with level equal to warning. This alert exists in SSL 3.0 and TLS 1.0.

#### SSL\_AlertCode\_UnexpectedMessage

An inappropriate message was received. This alert is always fatal and should never be observed in communication between proper implementations. This alert exists in SSL 3.0 and TLS 1.0.

#### SSL\_AlertCode\_BadRecordMac

This alert is returned if a record is received with an incorrect MAC. This message is always fatal. This alert exists in SSL 3.0 and TLS 1.0.

#### SSL\_AlertCode\_DecryptionFailed

Ciphertext decrypted in an invalid way: either it wasn't an even multiple of the block length or its padding values, when checked, weren't correct. This message is always fatal. This alert exists in TLS 1.0.

#### SSL\_AlertCode\_RecordOverflow

A Ciphertext record was received which had a length more than  $2^{14}+2048$  bytes, or a record decrypted to a Compressed record with more than  $2^{14}+1024$  bytes. This message is always fatal. This alert exists in TLS 1.0.

#### SSL\_AlertCode-DecompressionFailure

The decompression function received improper input (e.g. data that would expand to excessive length). This message is always fatal. This alert exists in SSL 3.0 and TLS 1.0.

#### SSL\_AlertCode\_HandshakeFailure

Reception of a handshake\_failure alert message indicates that the sender was unable to negotiate an acceptable set of security parameters given the options available. This is a fatal error. This alert exists in SSL 3.0 and TLS 1.0.

#### SSL\_AlertCode\_NoCertificate

This alert may be sent in response to a certification request if no appropriate certificate is available. This alert exists in SSL 3.0. This alert shares the same value as the alert **certificate\_unobtainable**.

**SSL\_AlertCode\_CertificateUnobtainable**

This alert is sent by servers who are unable to retrieve a certificate chain from the URL supplied by the client. This message may be fatal. This alert exists in the draft TLS Extensions <draft-ietf-tls-extensions-01.txt> This alert is shares the same value as the TLS1 alert **no\_certificate**.

**SSL\_AlertCode\_BadCertificate**

A certificate was corrupt, contained signatures that did not verify correctly, etc. This alert exists in SSL 3.0 and TLS 1.0.

**SSL\_AlertCode\_UnsupportedCertificate**

A certificate was of an unsupported type. This alert exists in SSL 3.0 and TLS 1.0.

**SSL\_AlertCode\_CertificateRevoked**

A certificate was revoked by its signer. This alert exists in SSL 3.0 and TLS 1.0.

**SSL\_AlertCode\_CertificateExpired**

A certificate has expired or is not currently valid. This alert exists in SSL 3.0 and TLS 1.0.

**SSL\_AlertCode\_CertificateUnknown**

Some other (unspecified) issue arose in processing the certificate, rendering it unacceptable. This alert exists in SSL 3.0 and TLS 1.0.

**SSL\_AlertCode\_IllegalParameter**

A field in the handshake was out of range or inconsistent with other fields. This is always fatal. This alert exists in SSL 3.0 and TLS 1.0.

**SSL\_AlertCode\_UnknownCa**

A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or couldn't be matched with a known, trusted CA. This message is always fatal. This alert exists in TLS 1.0.

**SSL\_AlertCode\_AccessDenied**

A valid certificate was received, but when access control was applied, the sender decided not to proceed with negotiation. This message is always fatal. This alert exists in TLS 1.0.

**SSL\_AlertCode\_DecodeError**

A message could not be decoded because some field was out of the specified range or the length of the message was incorrect. This message is always fatal. This alert exists in TLS 1.0.

**SSL\_AlertCode\_DecryptError**

A handshake cryptographic operation failed, including being unable to correctly verify a signature, decrypt a key exchange or validate a finished message. This alert exists in TLS 1.0.

**SSL\_AlertCode\_ExportRestriction**

A negotiation not in compliance with export restrictions was detected; for example, attempting to transfer a 1024 bit ephemeral RSA key for the RSA\_EXPORT handshake method. This message is always fatal. This alert exists in TLS 1.0.

**SSL\_AlertCode\_ProtocolVersion**

The protocol version the client has attempted to negotiate is recognized, but not supported. (For example, old protocol versions might be avoided for security reasons). This message is always fatal. This alert exists in TLS 1.0.

**SSL\_AlertCode\_InsufficientSecurity**

Returned instead of handshake\_failure when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client. This message is always fatal. This alert exists in TLS 1.0.

**SSL\_AlertCode\_InternalError**

An internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue (such as a memory allocation failure). This message is always fatal. This alert exists in TLS 1.0.

**SSL\_AlertCode\_UserCanceled**

This handshake is being canceled for some reason unrelated to a protocol failure. If the user cancels an operation after the handshake is complete, just closing the connection by sending a close\_notify is more appropriate. This alert should be followed by a close\_notify. This message is generally a warning. This alert exists in TLS 1.0.

**SSL\_AlertCode\_NoRenegotiation**

Sent by the client in response to a hello request or by the server in response to a client hello after initial handshaking. Either of these would normally lead to renegotiation; when that is not appropriate, the recipient should respond with this alert; at that point, the original requester can decide whether to proceed with the connection. One case where this would be appropriate would be where a server has spawned a process to satisfy a request; the process might receive security parameters (key length, authentication, etc.) at startup and it might be difficult to communicate changes to these parameters after that point. This message is always a warning. This alert exists in TLS 1.0.

**SSL\_AlertCode\_UnsupportedExtension**

This alert is sent by clients that receive an extended server hello containing an extension that they did not put in the corresponding client hello. This alert is always fatal. This alert exists in the draft TLS Extensions <draft-ietf-tls-extensions-01.txt>

**SSL\_AlertCode\_UnrecognisedDomain**

This alert is sent by servers that receive a server\_name extension request, but do not recognize the server name as belonging to a domain they are responsible for. This alert MAY be fatal. This alert exists in the draft TLS Extensions <draft-ietf-tls-extensions-01.txt>

**SSL\_AlertCode\_BadCertificateStatusResponse**

This alert is sent by clients that receive an invalid certificate status response. This alert is always fatal. This alert exists in the draft TLS Extensions <draft-ietf-tls-extensions-01.txt>



## ssl\_CheckCertificateChainFunc

```
typedef cic_Err(
    CIC_CALLCONV* ssl_CheckCertificateChainFunc )(
        const ssl_ConnectionContext* connCtx,
        const ssl_Cert* const certChain [ ],
        const ssl_Cert* trustedCert,
        const ssl_CertWarnings warnings [ ],
        const cic_Err extError,
        void* const certChainRef );
```

### Description

Template for a callback function which the library calls after it has evaluated a certificate chain. You can indicate how you want to continue by returning an error code. Return **CIC\_ERR\_NONE** if you want to override the validation error and continue normally, or a non-zero value if you want the library to handle the error condition. This callback function is called whenever the library receives a TLS 1.0, SSL3 or SSL2 certificate. It is also called when a **SSL\_AlertCode\_NoCertificate** alert message is received during the handshake. In that case the **certChain** parameter is NULL, indicating that no certificates were received.

Note:

No matter how many things have failed, this function will ALWAYS be called, so that it is possible to proceed with the handshake even if the certificate is invalid. It is up to the user to block the access for suspicious servers by returning a nonzero value.

### Parameters

**connCtx**

[ Input ] Pointer to the SSL context structure.

**certChain**

[ Input ] Pointer to an array of certificate structures sent by the remote side and to be checked and validated by this callback. The last element is NULL.

**trustedCert**

[ Input ] Pointer to a certificate structure containing the trusted or CA certificate that was used to verify the certificate chain.

**warnings**

[ Input ] An array of certificate related warning conditions that were found to be true, one per certificate in **certChain**. Each element in this array is a bit-mask of warnings.

**extError**

Extended error information for internal errors. The library will pass a descriptive error code regarding certificate processing failures that occur during the handshake process. Often, this information may elaborate on information passed through the **warnings** parameter. Any extended error code passed to the callback will be returned by the last call to

**ssl\_Handshake( )**. If the certificate(s) are rejected by your callback, but no **extError** was inputted to the callback, the error code **CIC\_ERR\_SSL\_CERT\_CHECK\_CALLBACK** will be returned by the last call to **ssl\_Handshake( )**.

**certChainRef**

[ Input ] Optional user reference parameter.

## Return Values

This function should return zero if the verification is successful and return a non-zero value if the function failed or the certificate did not pass verification.

## ssl\_SurrenderFunc

```
typedef cic_Err(  
    CIC_CALLCONV* ssl_SurrenderFunc )(  
    void* const surrenderRef );
```

**Description** Template for the callback function that the library calls during lengthy cryptographic operations.

**Parameters** `surrenderRef`  
[ Input ] Optional user reference parameter.

**Return Values** Any error codes that your callback function generates will cause the SSL API function that called your function to fail.

## ssl\_GetSessionFunc

```
typedef cic_Err(  
    CIC_CALLCONV* ssl_GetSessionFunc )(  
    const cic_Buffer sessionKey,  
    cic_Buffer sessionData,  
    void* const sessionRef );
```

### Description

Template for the callback function that gets a session record from the session database.

### Parameters

**sessionKey**

[ Input ] (Array) Session key to search for.

**sessionData**

[ Output ] Buffer to return session data.

**sessionRef**

[ Input ] Optional user reference parameter.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_NOT\_FOUND**

Record not found.

## ssl\_AddSessionFunc

```
typedef cic_Err(  
    CIC_CALLCONV* ssl_AddSessionFunc )(  
    const cic_Buffer sessionKey,  
    const cic_Buffer sessionData,  
    void* const sessionRef );
```

### Description

Template for the callback function that adds a session record to the session database.

### Parameters

**sessionKey[]**  
[ Input ] Session record key.

**sessionData**  
[ Input ] Session record data.

**sessionRef**  
[ Input ] Optional user reference parameter.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_NO\_PTR**  
NULL pointer was passed.

## ssl\_DeleteSessionFunc

```
typedef cic_Err(  
    CIC_CALLCONV* ssl_DeleteSessionFunc )(  
    const cic_Buffer sessionKey,  
    void* const sessionRef );
```

**Description** Template for the callback function that deletes a session record from the session database.

**Parameters**

`sessionKey[ ]`  
[ Input ] Session record key.

`sessionRef`  
[ Input ] Optional user reference parameter.

**Return Values**

`CIC_ERR_NONE`  
Success.

`CIC_ERR_NO_PTR`  
NULL pointer was passed.

`CIC_ERR_SSL_SESSION_NOT_FOUND`  
Record not found.

## ssl\_LogFunc

```
typedef void (CIC_CALLCONV *ssl_LogFunc)(  
    const ssl_CB_LogType type,  
    const ssl_CB_LogSubType subType,  
    const cic_Uint32 dataLen,  
    const cic_Uint8 * data,  
    void * const logRef  
);
```

### Description

Template for the callback function called by the SSL library to allow for detailed debugging.

Note:

See the **ssl\_CB\_LogType** and **ssl\_CB\_LogSubType** definitions and comments for more details.

### Parameters

<b>type</b>	[Input] The log message type.
<b>subType</b>	[Input] The log message sub-type. Use for finer granularity.
<b>dataLen</b>	[Input] Length of data.
<b>data</b>	[Input] Data of interest
<b>logRef</b>	[Input] User-defined parameter.

### Return Values

none

## ssl\_CB\_LogType

```
typedef cic_Uint16 ssl_CB_LogType
```

### Description

Every log that is passed into the log callback function will have a type as one of the parameters; it is up to the application to decide what to print; for a production release this would probably be nothing unless a user debug flag is turned on by some means.

### Enumeration

```
SSL_RECORD_RX_CB_LOG_TYPE
```

Incoming SSL record (encrypted or un-encrypted)

```
SSL_RECORD_TX_CB_LOG_TYPE
```

Outgoing SSL record (encrypted or un-encrypted)

```
SSL_DEBUG_MSG_CB_LOG_TYPE
```

Debugging message



## ssl\_CB\_LogSubType

```
typedef cic_Uint16 ssl_CB_LogSubType
```

### Description

Logs that are passed into the log callback function usually have a sub type.

This sub type value is meant to be used in conjunction with **ssl\_CB\_LogType**. This allows for finer granularity; for example only printing the records that have problems.

Note:

Not every type has a sub-type associated with it. You have to match up the type and subtype (e.g. **SSL\_CB\_LOG\_TYPE\_RECORD\_XXX** and **SSL\_CB\_LOG\_SUB\_TYPE\_RECORD\_YYY**).

### Enumeration

```
SSL_NONE_CB_LOG_SUB_TYPE
```

Encrypted or un-encrypted SSL record

```
SSL_RECORD_ALERT_CB_LOG_SUB_TYPE
```

Un-encrypted alert message

```
SSL_RECORD_HANDSHAKE_CB_LOG_SUB_TYPE
```

Un-encrypted handshake message

```
SSL_RECORD_APPLICATION_CB_LOG_SUB_TYPE
```

Un-encrypted application message

```
SSL_DEBUG_HASH_CB_LOG_SUB_TYPE
```

Debug hash data

```
SSL_DEBUG_PREMASTER_CB_LOG_SUB_TYPE
```

Pre-master secret

```
SSL_DEBUG_MASTER_CB_LOG_SUB_TYPE
```

Master secret

```
SSL_DEBUG_KEY_BLOCK_CB_LOG_SUB_TYPE
```

Key block used by encryption keys

```
SSL_DEBUG_CLIENT_WRITE_MAC_SECRET_CB_LOG_SUB_TYPE
```

Client write MAC secret

```
SSL_DEBUG_SERVER_WRITE_MAC_SECRET_CB_LOG_SUB_TYPE
```

Server write MAC secret

```
SSL_DEBUG_CIPHER_CLIENT_WRITE_KEY_CB_LOG_SUB_TYPE
```

Client symmetric cipher write key

```
SSL_DEBUG_CIPHER_SERVER_WRITE_KEY_CB_LOG_SUB_TYPE
```

Server symmetric cipher write key

```
SSL_DEBUG_CIPHER_CLIENT_FINAL_WRITE_KEY_CB_LOG_SUB_TYPE
```

Client symmetric cipher final write key

`SSL_DEBUG_CIPHER_SERVER_FINAL_WRITE_KEY_CB_LOG_SUB_TYPE`

Server symmetric cipher final write key

`SSL_DEBUG_CLIENT_WRITE_IV_CB_LOG_SUB_TYPE`

Client write symmetric cipher iv

`SSL_DEBUG_SERVER_WRITE_IV_CB_LOG_SUB_TYPE`

Server write symmetric cipher iv

# Cipher Suite and Protocol Enumerated Types

This section contains the enumerated type definitions for all the supported cipher suites and protocol versions. These enumerated identifiers are returned by the

`ssl_GetNegotiatedCipher()`,  
`ssl_GetNegotiatedProtocolVersion()` and  
`ssl_GetContextProtocolVersion()` functions.

## ssl\_CipherSuite

```
enum {
    TLS_RSA_EXPORT_WITH_RC40_40_MD5 = 0x0003,
    TLS_RSA_WITH_RC4_128_MD5 = 0x0004,
    TLS_RSA_WITH_RC4_128_SHA = 0x0005,
    TLS_RSA_EXPORT_WITH_DES40_CBC_SHA = 0x0008,
    TLS_RSA_WITH_DES_CBC_SHA = 0x0009,
    TLS_RSA_WITH_3DES_EDE_CBC_SHA = 0x000A,
    TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA = 0x0011,
    TLS_DHE_DSS_WITH_DES_CBC_SHA = 0x0012,
    TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA = 0x0013,
    TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA = 0x0014,
    TLS_DHE_RSA_WITH_DES_CBC_SHA = 0x0015,
    TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA = 0x0016,
    TLS_DH_anon_EXPORT_WITH_RC4_40_MD5 = 0x0017,
    TLS_DH_anon_WITH_RC4_128_MD5 = 0x0018,
    TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA = 0x0019,
    TLS_DH_anon_WITH_DES_CBC_SHA = 0x001A,
    TLS_DH_anon_WITH_3DES_EDE_CBC_SHA = 0x001B,
    TLS_RSA_WITH_AES_128_CBC_SHA = 0x002F,
    TLS_RSA_WITH_AES_256_CBC_SHA = 0x0035,
    TLS_ECDH_ECDSA_NULL_SHA = 0x0047,
    TLS_ECDH_ECDSA_WITH_RC4_128_SHA = 0x0048,
    TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA = 0x0063,
    TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA = 0x0065,
    TLS_DHE_DSS_WITH_RC4_128_SHA = 0x0066
}
```

### Description

Returned by the `ssl_GetNegotiatedCipher()` function.

## ssl\_ProtocolVersion

```
enum {  
    SSL_ProtocolVersion_Undetermined = 0x0000,  
    SSL_ProtocolVersion_SSLV2 = 0x0002,  
    SSL_ProtocolVersion_SSLV3 = 0x0300,  
    SSL_ProtocolVersion_TLSV1 = 0x0301  
}
```

### Description

Returned by the `ssl_GetNegotiatedProtocolVersion()` and `ssl_GetContextProtocolVersion()` functions.



# 3

## The SSL RAD API

---

### Rapid Application Development

The SSL Plus RAD API allows you to quickly develop secure client or server network applications by providing all the system and I/O callbacks required by SSL Plus. SSL Plus RAD is a complete API that "wraps" the original SSL Plus API. SSL Plus RAD provides simple functions for all the basic SSL protocol operations, including initializing the global context, choosing ciphersuites, performing authentication and performing secure data transfer. The source code for the SSL Plus RAD API is also provided.

## sslrad\_MallocCallback( )

```
extern CIC_EXPORT void* CIC_CALLCONV sslrad_MallocCallback(  
    cic_Uint32 size,  
    void* memRef  
);
```

**Description**      Callback function to allocate memory.

**Parameters**

<b>size</b>	[ Input ] Size of memory, in bytes, to allocate.
<b>memRef</b>	[ Input ] Memory reference. Not used by the library.

**Return Values**      Pointer to allocated memory. NULL if failure.



## sslrad\_FreeCallback( )

```
extern CIC_EXPORT void CIC_CALLCONV sslrad_FreeCallback(  
    void* block,  
    void* memRef  
);
```

### Description

Callback function to free memory allocated by the **sslrad\_MallocCallback( )**.

### Parameters

**block**

[ Input ] Pointer to the memory to be released.

**memRef**

[ Input ] Memory reference. Not used by the library.

### Return Values

None.

## sslrad\_MemsetCallback( )

```
extern CIC_EXPORT void CIC_CALLCONV sslrad_MemsetCallback(  
    void* block,  
    cic_Uint8 value,  
    cic_Uint32 length  
);
```

### Description

Callback function for setting the specified memory block to a certain value.

### Parameters

**block**

[ Input/Output ] Points to the memory to be set.

**value**

[ Input ] Value to which the memory block is to be set.

**length**

[ Input ] Size of the memory block.

### Return Values

None.

## sslrad\_MemcpyCallback( )

```
extern CIC_EXPORT void CIC_CALLCONV sslrad_MemcpyCallback(  
    void* dest,  
    const void* src,  
    cic_Uint32 length  
);
```

### Description

Callback function to copy contents of memory from one location to another.

### Parameters

**dest**

[ Output ] Destination memory address.

**src**

[ Input ] Source memory address.

**length**

[ Input ] Length of data to copy.

### Return Values

None.

## sslrad\_MemcmpCallback( )

```
extern CIC_EXPORT cic_Int32 CIC_CALLCONV sslrad_MemcmpCallback(  
    const void* a,  
    const void* b,  
    cic_Uint32 length  
);
```

### Description

Callback function to compare the contents of two memory blocks.

### Parameters

**a**

[ Input ] First memory address.

**b**

[ Input ] Second memory address.

**length**

[ Input ] Length of data to compare.

### Return Values

Returns 0 if they are equal, a negative number if a is less than b, and a positive number if a is greater than b.

## sslrad\_TimeCallback()

```
extern CIC_EXPORT cic_Uint32 CIC_CALLCONV sslrad_TimeCallback(  
    void  
);
```

**Description**                      Callback function for acquiring the time.

**Return Values**                      Returns a 32-bit value indicating the elapsed time in seconds since midnight, January 1, 1970, UTC.

## sslrad\_RandomCallback( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV sslrad_RandomCallback(  
    cic_Buffer* data,  
    void* const randomRef  
);
```

### Description

Callback function to generate seeding material.

### Parameters

**data**

[ Input/Output ] On input, the **length** field of the **data** structure indicates the number of seed bytes to generate. On output, the **data** field contains the generated seed.

**randomRef**

[ Input ] Optional user reference parameter. Not used by the library.

### Return Values

**CIC\_ERR\_NONE**

Success.

## sslrad\_CheckCertificateChainCallback()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_CheckCertificateChainCallback(
        const ssl_ConnectionContext* connCtx,
        const ssl_Cert* const certChain [ ],
        const ssl_Cert* trustedCert,
        const ssl_CertWarnings warnings [ ],
        const cic_Err extError,
        void* const certChainRef
    );
```

### Description

Check certificate chain callback.

### Parameters

**connCtx**

[ Input ] Pointer to the context structure defining the SSL connection.

**certChain**

[ Input ] Pointer to an array of certificate structures sent by the remote side and to be checked and validated by this callback. The last element in the array is NULL.

**trustedCert**

[ Input ] Pointer to a certificate structure containing the trusted or CA certificate that was used to verify the certificate chain.

**warnings**

[ Input ] Array of certificate related warning conditions that were found to be true, one per certificate in **certChain**. Each element in this array is a bit-mask of warnings.

**extError**

[ Input ] Extended error information for internal errors. The library passes a descriptive error code regarding certificate processing failures that occur during the handshake process. Often, this information may elaborate on information passed through the **warnings** parameter. Any extended error code passed to the callback will be returned by the last call to **ssl\_Handshake()**. If the certificate(s) are rejected by your callback,

but no `extError` was passed to the callback, the error code `CIC_ERR_SSL_CERT_CHECK_CALLBACK` will be returned by the last call to `ssl_Handshake()`.

`certChainRef`

[ Input ] Optional user reference parameter.

## Return Values

`CIC_ERR_NONE`

Always returned.



## sslrad\_CreateSessionDB( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV sslrad_CreateSessionDB(  
    sslrad_SessionDB** sessionDB  
);
```

**Description** Create and initialize session database.

**Parameters** `sessionDB`  
[ Output ] Sesion database.

**Return Values**

`CIC_ERR_NONE`  
Success.

`CIC_ERR_NO_PTR`  
NULL pointer was passed.

`CIC_ERR_MEMORY`  
Error allocating memory.

## sslrad\_DestroySessionDB( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV sslrad_DestroySessionDB(  
    sslrad_SessionDB** sessionDB  
);
```

**Description** Destroys the session database created with `sslrad_CreateSessionDB( )`.

**Parameters**

<code>sessionDB</code>	
[ Input/Output ]	Session database to destroy.

**Return Values**

<code>CIC_ERR_NONE</code>	Success.
<code>CIC_ERR_NO_PTR</code>	NULL pointer was passed.

## sslrad\_GetSessionCallback()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV sslrad_GetSessionCallback(  
    const cic_Buffer sessionKey,  
    cic_Buffer sessionData,  
    void* const sessionDB  
);
```

### Description

Callback function that gets a session record from the session database.

Notes:

( 1 ) This function is not thread safe.

### Parameters

**sessionKey**

[ Input ] Session key to search for.

**sessionData**

[ Input/Output ] Pre-allocated buffer to return session data.

**sessionDB**

[ Input ] Pointer to an initialized **sslrad\_SessionDB**.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SSL\_SESSION\_NOT\_FOUND**

Record not found.

## sslrad\_AddSessionCallback()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV sslrad_AddSessionCallback(  
    const cic_Buffer sessionKey,  
    const cic_Buffer sessionData,  
    void* const sessionDB  
);
```

### Description

Callback function that adds a session record to the session database.

Notes:

( 1 ) This function is not thread safe.

### Parameters

**sessionKey**

[ Input ] Session key.

**sessionData**

[ Input ] Session data.

**sessionDB**

[ Input ] Pointer to a initialized **sslrad\_SessionDB**.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

## sslrad\_DeleteSessionCallback( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV sslrad_DeleteSessionCallback(
    const cic_Buffer sessionKey,
    void* const sessionDB
);
```

### Description

Callback function that deletes a session record from the session database.

Notes:

( 1 ) This function is not thread safe.

### Parameters

**sessionKey**

[ Input ] Session key.

**sessionDB**

[ Input ] Pointer to a initialized **sslrad\_SessionDB**.

### Return Values

**CIC\_ERR\_NONE**

Success.

**CIC\_ERR\_NO\_PTR**

NULL pointer was passed.

**CIC\_ERR\_SSL\_SESSION\_NOT\_FOUND**

Record not found.

## sslrad\_SocketLayerInit()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV sslrad_SocketLayerInit(  
    void  
);
```

### Description

Performs platform specific initialization of the socket I/O library. This function must be called before any other socket operations.

### Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_SSL_IO`

Error initializing the I/O Layer

## sslrad\_SocketLayerClose( )

```
extern CIC_EXPORT void CIC_CALLCONV sslrad_SocketLayerClose(  
    void  
);
```

**Description**                Performs a clean-up of the socket I/O layer.

**Return Values**            None.

## sslrad\_SocketConnect()

```
extern CIC_EXPORT sslrad_Socket* CIC_CALLCONV sslrad_SocketConnect(  
    const char* server,  
    cic_Int32 port,  
    sslrad_SocketBlockingMode blockingMode,  
    cic_Buffer* peerID  
);
```

**Description** Connects to the specified server.

### Parameters

**server**

[ Input ] Server to connect to.

**port**

[ Input ] Port number to try and connect on.

**blockingMode**

[ Input ] Set to either **SSLRad\_NonBlocking\_Socket** or **SSLRad\_Blocking\_Socket**.

**peerID**

[ Input/Output ] Data which uniquely identifies the server on the connection. This parameter is passed to **ssl\_CreateConnectionContext** for this connection. You are responsible for freeing memory for this buffer.

**Return Values** NULL if an error occurs; otherwise a **sslrad\_Socket**.



## sslrad\_SocketCreateServer()

```
extern CIC_EXPORT sslrad_Socket* CIC_CALLCONV sslrad_SocketCreateServer(  
    cic_Int32 port  
);
```

**Description** Creates a server socket.

**Parameters** `port`  
[ Input ] Port number to listen on.

**Return Values** NULL if an error occurs; otherwise a `sslrad_Socket` .

## sslrad\_SocketAccept ( )

```
extern CIC_EXPORT sslrad_Socket* CIC_CALLCONV sslrad_SocketAccept(  
    sslrad_Socket* serverSocket,  
    sslrad_SocketBlockingMode blockingMode,  
    cic_Buffer* peerId  
);
```

**Description** Returns the next incoming connection.

### Parameters

**serverSocket**

[ Input ] Server socket.

**blockingMode**

[ Input ] Should be **SSLRad\_NonBlocking\_Socket** to configure the returned socket to non-blocking or **SSLRad\_Blocking\_Socket**.

**peerId**

[ Input/Output ] Peer id. The buffer will be allocated and must be freed by the caller.

**Return Values** NULL if an error occurs; otherwise returns a **sslrad\_Socket**.

## sslrad\_SocketClose( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV sslrad_SocketClose(  
    sslrad_Socket* theSocket  
);
```

**Description** Closes a socket.

**Parameters** `theSocket`  
[ Input ] Socket to close.

**Return Values**

`CIC_ERR_NONE`  
Success.

`CIC_ERR_NO_PTR`  
The `theSocket` pointer was NULL.

`CIC_ERR_SSL_IO`  
Error closing socket.

## sslrad\_SocketReadCallback()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV sslrad_SocketReadCallback(  
    cic_Buffer data,  
    cic_Uint32* processed,  
    ssl_IOState ioState,  
    void* const connRef  
);
```

**Description** I/O callback for reading data from a socket.

### Parameters

**data**  
[ Input/Output ] On input, specifies the number of bytes to read. On output, the buffer to store data that was read.

**processed**  
[ Output ] Number of bytes read.

**ioState**  
[ Input ] Indicates additional state information. Unused.

**connRef**  
[ Input ] Pointer to **sslrad\_Socket**.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_WOULD\_BLOCK**  
I/O is blocking.

**CIC\_ERR\_SSL\_IO**  
I/O error.

**CIC\_ERR\_SSL\_CONNECTION\_CLOSED\_GRACEFUL**  
I/O is closed, override the goodbye kiss.



## sslrad\_SocketWriteCallback()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV sslrad_SocketWriteCallback(  
    cic_Buffer data,  
    cic_Uint32* processed,  
    ssl_IOState ioState,  
    void* const connRef  
);
```

**Description** IO callback for writing data to a socket.

### Parameters

**data**  
[ Input ] Data to be written.

**processed**  
[ Output ] Number of bytes written.

**ioState**  
[ Input ] Indicates additional state information. Unused.

**connRef**  
[ Input ] Pointer to **sslrad\_Socket**.

### Return Values

**CIC\_ERR\_NONE**  
Success.

**CIC\_ERR\_WOULD\_BLOCK**  
I/O is blocking.

**CIC\_ERR\_SSL\_IO**  
I/O error.

**CIC\_ERR\_SSL\_CONNECTION\_CLOSED\_GRACEFUL**  
I/O is closed, override the goodbye kiss.

# sslrad\_InitializeCompleteServerGlobalContext ( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeCompleteServerGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

## Description

Creates and configures a SSL global context with default parameters for SSL servers.

Notes:

The global context will be configured with the following characteristics:

- ( 1 ) Supports server side of protocol.
- ( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.
- ( 3 ) Supports RSA ciphersuites.
- ( 4 ) Supports ECC ciphersuites with the curve sect163k1.
- ( 5 ) Installs the following ciphersuites ( in order of preference ):  
**TLS\_ECDH\_ECDSA\_WITH\_RC4\_128\_SHA**  
**TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA** **TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA**  
**TLS\_RSA\_WITH\_RC4\_128\_MD5** **TLS\_RSA\_WITH\_RC4\_128\_SHA**  
**TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA**  
**TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5** **TLS\_ECDH\_ECDSA\_NULL\_SHA**
- ( 6 ) Requests client authentication with the client authentication modes ( in order of preference ): **ECDSA\_fixed\_ECDH**, **ECDSA\_sign** and **RSA\_sign**.
- ( 7 ) Supports partial ( **SSL\_IOSemantics\_PartialIO** ) I/O semantics.
- ( 8 ) Installs the following callbacks: **sslrad\_MallocCallback**  
**sslrad\_FreeCallback** **sslrad\_MemsetCallback**  
**sslrad\_MemcpyCallback** **sslrad\_MemcmpCallback**  
**sslrad\_MemcpyCallback** **sslrad\_TimeCallback**  
**sslrad\_RandomCallback** **sslrad\_SocketReadCallback**  
**sslrad\_SocketWriteCallback**  
**sslrad\_CheckCertificateChainCallback**
- ( 9 ) If **sessionDB** is non-NULL, then session resumption will be supported and the following callbacks will be installed: **sslrad\_GetSessionCallback**  
**sslrad\_AddSessionCallback** **sslrad\_DeleteSessionCallback**

## Parameters

**globalCtx**

[ Output ] Pointer to return the global context.

`sessionDB`

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.



# sslrad\_InitializeECOnlyServerGlobalContext( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeECOnlyServerGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

## Description

Creates and configures a SSL global context with default parameters for SSL servers which support ECC ciphers only.

Notes:

The global context will be configured with the following characteristics:

- ( 1 ) Supports server side of protocol.
- ( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.
- ( 3 ) Supports ECC ciphersuites with the curve sect163k1.
- ( 4 ) Installs the following ciphersuites ( in order of preference ):  
**TLS\_ECDH\_ECDSA\_WITH\_RC4\_128\_SHA** and  
**TLS\_ECDH\_ECDSA\_NULL\_SHA**.
- ( 5 ) Requests client authentication with the client authentication modes ( in order of preference ): **ECDSA\_fixed\_ECDH** and **ECDSA\_sign**.
- ( 6 ) Supports partial ( **SSL\_IOSemantics\_PartialIO** ) I/O semantics.
- ( 7 ) Installs the following callbacks: **sslrad\_MallocCallback**  
**sslrad\_FreeCallback** **sslrad\_MemsetCallback**  
**sslrad\_MemcpyCallback** **sslrad\_MemcmpCallback**  
**sslrad\_MemcpyCallback** **sslrad\_TimeCallback**  
**sslrad\_RandomCallback** **sslrad\_SocketReadCallback**  
**sslrad\_SocketWriteCallback**  
**sslrad\_CheckCertificateChainCallback**
- ( 8 ) If **sessionDB** is non-NULL, then session resumption will be supported and the following callbacks will be installed: **sslrad\_GetSessionCallback**  
**sslrad\_AddSessionCallback** **sslrad\_DeleteSessionCallback**

## Parameters

**globalCtx**

[ Output ] Pointer to return the global context.

`sessionDB`

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

# sslrad\_InitializeRSAOnlyServerGlobalContext( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeRSAOnlyServerGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

## Description

Creates and configures a SSL global context with default parameters for SSL servers which support RSA ciphers only.

Notes:

The global context will be configured with the following characteristics:

- ( 1 ) Supports server side of protocol.
- ( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.
- ( 3 ) Supports RSA ciphersuites.
- ( 4 ) Installs the following ciphersuites ( in order of preference ):
 

```
TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_RC4_128_MD5 TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5
```
- ( 5 ) Requests client authentication with the client authentication modes ( in order of preference ): RSA\_sign
- ( 6 ) Supports partial ( `SSL_IOSemantics_PartialIO` ) I/O semantics.
- ( 7 ) Installs the following callbacks: `sslrad_MallocCallback`  
`sslrad_FreeCallback` `sslrad_MemsetCallback`  
`sslrad_MemcpyCallback` `sslrad_MemcmpCallback`  
`sslrad_MemcpyCallback` `sslrad_TimeCallback`  
`sslrad_RandomCallback` `sslrad_SocketReadCallback`  
`sslrad_SocketWriteCallback`  
`sslrad_CheckCertificateChainCallback`
- ( 8 ) If `sessionDB` is non-NULL, then session resumption will be supported and the following callbacks will be installed: `sslrad_GetSessionCallback`  
`sslrad_AddSessionCallback` `sslrad_DeleteSessionCallback`

## Parameters

`globalCtx`

[ Output ] Pointer to return the global context.

`sessionDB`

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

## sslrad\_InitializeCompleteNoClientAuthServerGlobalContext ( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeCompleteNoClientAuthServerGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

### Description

Creates and configures a SSL global context with default parameters for SSL servers.

Notes:

The global context will be configured with the following characteristics:

- ( 1 ) Supports server side of protocol.
- ( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.
- ( 3 ) Supports RSA ciphersuites.
- ( 4 ) Supports ECC ciphersuites with the curve sect163k1.
- ( 5 ) Installs the following ciphersuites ( in order of preference ):  
**TLS\_ECDH\_ECDSA\_WITH\_RC4\_128\_SHA**  
**TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA**  
**TLS\_RSA\_WITH\_RC4\_128\_MD5 TLS\_RSA\_WITH\_RC4\_128\_SHA**  
**TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA**  
**TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5 TLS\_ECDH\_ECDSA\_NULL\_SHA**
- ( 6 ) Does not request client authentication.
- ( 7 ) Supports partial ( **SSL\_IOSemantics\_PartialIO** ) I/O semantics.
- ( 8 ) Installs the following callbacks: **sslrad\_MallocCallback**  
**sslrad\_FreeCallback sslrad\_MemsetCallback**  
**sslrad\_MemcpyCallback sslrad\_MemcmpCallback**  
**sslrad\_MemcpyCallback sslrad\_TimeCallback**  
**sslrad\_RandomCallback sslrad\_SocketReadCallback**  
**sslrad\_SocketWriteCallback**  
**sslrad\_CheckCertificateChainCallback**
- ( 9 ) If **sessionDB** is non-NULL, then session resumption will be supported and the following callbacks will be installed: **sslrad\_GetSessionCallback**  
**sslrad\_AddSessionCallback sslrad\_DeleteSessionCallback**

### Parameters

**globalCtx**

[ Output ] Pointer to return the global context.

`sessionDB`

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

## sslrad\_InitializeECOnlyNoClientAuthServerGlobalContext()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeECOnlyNoClientAuthServerGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

### Description

Creates and configures a SSL global context with default parameters for SSL servers which support ECC ciphers only.

Notes:

The global context will be configured with the following characteristics:

- ( 1 ) Supports server side of protocol.
- ( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.
- ( 3 ) Supports ECC ciphersuites with the curve sect163k1.
- ( 4 ) Installs the following ciphersuites ( in order of preference ):  
**TLS\_ECDH\_ECDSA\_WITH\_RC4\_128\_SHA\_SERVERSIDE** and  
**TLS\_ECDH\_ECDSA\_NULL\_SHA**.
- ( 5 ) Does not request client authentication.
- ( 6 ) Supports partial ( **SSL\_IOSemantics\_PartialIO** ) I/O semantics.
- ( 7 ) Installs the following callbacks: **sslrad\_MallocCallback**  
**sslrad\_FreeCallback** **sslrad\_MemsetCallback**  
**sslrad\_MemcpyCallback** **sslrad\_MemcmpCallback**  
**sslrad\_MemcpyCallback** **sslrad\_TimeCallback**  
**sslrad\_RandomCallback** **sslrad\_SocketReadCallback**  
**sslrad\_SocketWriteCallback**  
**sslrad\_CheckCertificateChainCallback**
- ( 8 ) If **sessionDB** is non-NULL, then session resumption will be supported and the following callbacks will be installed: **sslrad\_GetSessionCallback**  
**sslrad\_AddSessionCallback** **sslrad\_DeleteSessionCallback**

### Parameters

**globalCtx**

[ Output ] Pointer to return the global context.

`sessionDB`

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.



## sslrad\_InitializeRSAOnlyNoClientAuthServerGlobalContext()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeRSAOnlyNoClientAuthServerGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

### Description

Creates and configures a SSL global context with default parameters for SSL servers which support RSA ciphers only.

Notes:

The global context will be configured with the following characteristics:

- ( 1 ) Supports server side of protocol.
- ( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.
- ( 3 ) Supports RSA ciphersuites.
- ( 4 ) Installs the following ciphersuites ( in order of preference ):  
**TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA**  
**TLS\_RSA\_WITH\_RC4\_128\_MD5 TLS\_RSA\_WITH\_RC4\_128\_SHA**  
**TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA**  
**TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5**
- ( 5 ) Does not request client authentication.
- ( 6 ) Supports partial ( **SSL\_IOSemantics\_PartialIO** ) I/O semantics.
- ( 7 ) Installs the following callbacks: **sslrad\_MallocCallback**  
**sslrad\_FreeCallback sslrad\_MemsetCallback**  
**sslrad\_MemcpyCallback sslrad\_MemcmpCallback**  
**sslrad\_MemcpyCallback sslrad\_TimeCallback**  
**sslrad\_RandomCallback sslrad\_SocketReadCallback**  
**sslrad\_SocketWriteCallback**  
**sslrad\_CheckCertificateChainCallback**
- ( 8 ) If **sessionDB** is non-NULL, then session resumption will be supported and the following callbacks will be installed: **sslrad\_GetSessionCallback**  
**sslrad\_AddSessionCallback sslrad\_DeleteSessionCallback**

### Parameters

**globalCtx**

[ Output ] Pointer to return the global context.

`sessionDB`

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

## sslrad\_InitializeCompleteClientGlobalContext ( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeCompleteClientGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

### Description

Creates and configures a SSL global context with default parameters for SSL clients.

Notes:

The global context will be configured with the following characteristics:

- ( 1 ) Supports Client side of protocol.
- ( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.
- ( 3 ) Supports RSA ciphersuites.
- ( 4 ) Supports ECC ciphersuites with the curve sect163k1.
- ( 5 ) Installs the following ciphersuites ( in order of preference ):
 

```
TLS_ECDH_ECDSA_WITH_RC4_128_SHA_SERVERSIDE
TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_RC4_128_MD5 TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5 TLS_ECDH_ECDSA_NULL_SHA
```
- ( 6 ) Supports client authentication with the client authentication modes ( in order of preference ): **ECDSA\_fixed\_ECDH**, **ECDSA\_sign** and **RSA\_sign**
- ( 7 ) Supports partial ( **SSL\_IOSemantics\_PartialIO** ) I/O semantics.
- ( 8 ) Installs the following callbacks: **sslrad\_MallocCallback**  
**sslrad\_FreeCallback** **sslrad\_MemsetCallback**  
**sslrad\_MemcpyCallback** **sslrad\_MemcmpCallback**  
**sslrad\_MemcpyCallback** **sslrad\_TimeCallback**  
**sslrad\_RandomCallback** **sslrad\_SocketReadCallback**  
**sslrad\_SocketWriteCallback**  
**sslrad\_CheckCertificateChainCallback**
- ( 9 ) If **sessionDB** is non-NULL, then session resumption will be supported and the following callbacks will be installed: **sslrad\_GetSessionCallback**  
**sslrad\_AddSessionCallback** **sslrad\_DeleteSessionCallback**

## Parameters

`globalCtx`

[ Output ] Pointer to return the global context.

`sessionDB`

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

# sslrad\_InitializeECOnlyClientGlobalContext( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeECOnlyClientGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

## Description

Creates and configures a SSL global context with default parameters for SSL clients which support ECC ciphers only.

Notes:

The global context will be configured with the following characteristics:

- ( 1 ) Supports Client side of protocol.
- ( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.
- ( 3 ) Supports ECC ciphersuites with the curve sect163k1.
- ( 4 ) Installs the following ciphersuites ( in order of preference ):  
**TLS\_ECDH\_ECDSA\_WITH\_RC4\_128\_SHA** , **TLS\_ECDH\_ECDSA\_NULL\_SHA** .
- ( 5 ) Supports client authentication with the client authentication modes ( in order of preference ): **ECDSA\_fixed\_ECDH** and **ECDSA\_sign** .
- ( 6 ) Supports partial ( **SSL\_IOSemantics\_PartialIO** ) I/O semantics.
- ( 7 ) Installs the following callbacks: **sslrad\_MallocCallback**  
**sslrad\_FreeCallback** **sslrad\_MemsetCallback**  
**sslrad\_MemcpyCallback** **sslrad\_MemcmpCallback**  
**sslrad\_MemcpyCallback** **sslrad\_TimeCallback**  
**sslrad\_RandomCallback** **sslrad\_SocketReadCallback**  
**sslrad\_SocketWriteCallback**  
**sslrad\_CheckCertificateChainCallback**
- ( 8 ) If **sessionDB** is non-NULL, then session resumption will be supported and the following callbacks will be installed: **sslrad\_GetSessionCallback**  
**sslrad\_AddSessionCallback** **sslrad\_DeleteSessionCallback**

## Parameters

**globalCtx**

[ Output ] Pointer to return the global context.

`sessionDB`

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

# sslrad\_InitializeRSAOnlyClientGlobalContext( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeRSAOnlyClientGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

## Description

Creates and configures a SSL global context with default parameters for SSL clients which support RSA ciphers only.

Notes:

The global context will be configured with the following characteristics:

- ( 1 ) Supports Client side of protocol.
- ( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.
- ( 3 ) Supports RSA ciphersuites.
- ( 4 ) Installs the following ciphersuites ( in order of preference ):  
**TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA**  
**TLS\_RSA\_WITH\_RC4\_128\_MD5 TLS\_RSA\_WITH\_RC4\_128\_SHA**  
**TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA**  
**TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5**
- ( 5 ) Supports client authentication with the client authentication modes ( in order of preference ): **RSA\_sign**.
- ( 6 ) Supports partial ( **SSL\_IOSemantics\_PartialIO** ) I/O semantics.
- ( 7 ) Installs the following callbacks: **sslrad\_MallocCallback**  
**sslrad\_FreeCallback sslrad\_MemsetCallback**  
**sslrad\_MemcpyCallback sslrad\_MemcmpCallback**  
**sslrad\_MemcpyCallback sslrad\_TimeCallback**  
**sslrad\_RandomCallback sslrad\_SocketReadCallback**  
**sslrad\_SocketWriteCallback**  
**sslrad\_CheckCertificateChainCallback**
- ( 8 ) If **sessionDB** is non-NULL, then session resumption will be supported and the following callbacks will be installed: **sslrad\_GetSessionCallback**  
**sslrad\_AddSessionCallback sslrad\_DeleteSessionCallback**

## Parameters

**globalCtx**

[ Output ] Pointer to return the global context.

`sessionDB`

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.



## sslrad\_InitializeCompleteNoClientAuthClientGlobalContext( )

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeCompleteNoClientAuthClientGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

### Description

Creates and configures a SSL global context with default parameters for SSL clients.

Notes:

The global context will be configured with the following characteristics:

- ( 1 ) Supports Client side of protocol.
- ( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.
- ( 3 ) Supports RSA ciphersuites.
- ( 4 ) Supports ECC ciphersuites with the curve sect163k1.
- ( 5 ) Installs the following ciphersuites ( in order of preference ):  
**TLS\_ECDH\_ECDSA\_WITH\_RC4\_128\_SHA**  
**TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA**  
**TLS\_RSA\_WITH\_RC4\_128\_MD5 TLS\_RSA\_WITH\_RC4\_128\_SHA**  
**TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA**  
**TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5 TLS\_ECDH\_ECDSA\_NULL\_SHA**
- ( 6 ) Supports partial ( **SSL\_IOSemantics\_PartialIO** ) I/O semantics.
- ( 7 ) Installs the following callbacks: **sslrad\_MallocCallback**  
**sslrad\_FreeCallback sslrad\_MemsetCallback**  
**sslrad\_MemcpyCallback sslrad\_MemcmpCallback**  
**sslrad\_MemcpyCallback sslrad\_TimeCallback**  
**sslrad\_RandomCallback sslrad\_SocketReadCallback**  
**sslrad\_SocketWriteCallback**  
**sslrad\_CheckCertificateChainCallback**
- ( 8 ) If **sessionDB** is non-NULL, then session resumption will be supported and the following callbacks will be installed: **sslrad\_GetSessionCallback**  
**sslrad\_AddSessionCallback sslrad\_DeleteSessionCallback**

### Parameters

**globalCtx**

[ Output ] Pointer to return the global context.

`sessionDB`

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

# sslrad\_InitializeECOnlyNoClientAuthClientGlobalContext()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeECOnlyNoClientAuthClientGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

## Description

Creates and configures a SSL global context with default parameters for SSL clients which support ECC ciphers only.

Notes:

The global context will be configured with the following characteristics:

- ( 1 ) Supports Client side of protocol.
- ( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.
- ( 3 ) Supports ECC ciphersuites with the curve sect163k1.
- ( 4 ) Installs the following ciphersuites ( in order of preference ):  
**TLS\_ECDH\_ECDSA\_WITH\_RC4\_128\_SHA** and  
**TLS\_ECDH\_ECDSA\_NULL\_SHA**.
- ( 5 ) Supports the IO semantics *SSL\_IOSemantics\_PartialIO*.
- ( 6 ) Installs the following callbacks: **sslrad\_MallocCallback**  
**sslrad\_FreeCallback** **sslrad\_MemsetCallback**  
**sslrad\_MemcpyCallback** **sslrad\_MemcmpCallback**  
**sslrad\_MemcpyCallback** **sslrad\_TimeCallback**  
**sslrad\_RandomCallback** **sslrad\_SocketReadCallback**  
**sslrad\_SocketWriteCallback**  
**sslrad\_CheckCertificateChainCallback**
- ( 7 ) If **sessionDB** is non-NULL, then session resumption will be supported and the following callbacks will be installed: **sslrad\_GetSessionCallback**  
**sslrad\_AddSessionCallback** **sslrad\_DeleteSessionCallback**

## Parameters

**globalCtx**

[ Output ] Pointer to return the global context.

**sessionDB**

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.

## sslrad\_InitializeRSAOnlyNoClientAuthClientGlobalContext()

```
extern CIC_EXPORT cic_Err CIC_CALLCONV
    sslrad_InitializeRSAOnlyNoClientAuthClientGlobalContext(
        ssl_GlobalContext** globalCtx,
        sslrad_SessionDB** sessionDB
    );
```

### Description

Creates and configures a SSL global context with default parameters for SSL clients which support RSA ciphers only.

Notes:

The global context will be configured with the following characteristics:

( 1 ) Supports Client side of protocol.

( 2 ) Supports the TLS V1, SSL V3, and SSL V2 protocols.

( 3 ) Supports RSA ciphersuites.

( 4 ) Installs the following ciphersuites ( in order of preference ):

**TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA**

TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA

TLS\_RSA\_WITH\_RC4\_128\_MD5

TLS\_RSA\_WITH\_RC4\_128\_SHA

TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA

TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5

( 5 ) Supports the IO semantics *SSL\_IOSemantics\_PartialIO* .

( 6 ) Installs the following callbacks: **sslrad\_MallocCallback**

**sslrad\_FreeCallback** **sslrad\_MemsetCallback**

**sslrad\_MemcpyCallback** **sslrad\_MemcmpCallback**

**sslrad\_MemcpyCallback** **sslrad\_TimeCallback**

**sslrad\_RandomCallback** **sslrad\_SocketReadCallback**

**sslrad\_SocketWriteCallback**

**sslrad\_CheckCertificateChainCallback**

( 7 ) If **sessionDB** is non-NULL, then session resumption will be supported and the following callbacks will be installed: **sslrad\_GetSessionCallback**

**sslrad\_AddSessionCallback** **sslrad\_DeleteSessionCallback**

## Parameters

`globalCtx`

[ Output ] Pointer to return the global context.

`sessionDB`

[ Output ] Pointer to return a session database. If the pointer is NULL, the global context will not support session resumption.

## Return Values

`CIC_ERR_NONE`

Success.

`CIC_ERR_NO_PTR`

NULL pointer was passed.

`CIC_ERR_MEMORY`

Error allocating memory.