

## MatrixSSL Public API documentation

One of the primary development goals in MatrixSSL was to create a simple and small public application programming interface for users to integrate with their client or server applications. The public interface and structures are contained in the *matrixSsl.h* header file. The following API documentation describes the entire set of functions an application would need to use in order to get the full benefits of secure socket communications using MatrixSSL.

### **Commercial Version**

Some functions or features described in this document are available only in the commercially licensed version of MatrixSSL. Sections of this document that refer to the commercial version will be noted and shaded.

MatrixSSL Public API documentation.....	1
Integer sizes.....	2
Structures.....	3
sslBuf_t.....	3
sslCertInfo_t.....	4
Functions.....	6
matrixSslOpen.....	6
matrixSslClose.....	6
matrixSslReadKeys.....	7
matrixRsaReadKeysEx.....	9
matrixSslReadKeysMem.....	10
matrixRsaParseKeysMem.....	10
matrixSslFreeKeys.....	11
matrixSslNewSession.....	12
matrixSslDeleteSession.....	13
matrixSslDecode.....	14
matrixSslEncode.....	16
matrixSslEncodeClosureAlert.....	17
matrixSslEncodeClientHello.....	18
matrixSslEncodeHelloRequest.....	19
matrixSslSetSessionOption.....	20
matrixSslHandshakeIsComplete.....	21
matrixSslGetSessionId.....	22
matrixSslFreeSessionId.....	23
matrixSslSetCertValidator.....	24
matrixSslGetAnonStatus.....	25
matrixSslAssignNewKeys.....	26

## ***Integer sizes***

MatrixSSL was designed with the assumption that integer sizes are 32-bit. This assumption has been clarified with the addition of *int32* and *uint32* type definitions. These have been defined in *matrixCommon.h* of the MatrixSSL distribution. This change was made to enable global redefinitions for platforms that do not support 32-bit integer types in the native *int* type. Although this document will continue to use the *int* type, the code will reflect the use of *int32*.

## Structures

There are five structure types used in the MatrixSSL public API set. Only the members of the *sslBuf\_t* and *sslCertInfo\_t* structures have been exposed to the user. The *ssl\_t*, *sslSessionId\_t* and *sslKeys\_t* structures have been defined in the header file to be opaque integer types because their members do not need to be accessed by the user.

### sslBuf\_t

#### Definition

```
typedef struct {
    unsigned char *buf;
    unsigned char *start;
    unsigned char *end;
    int             size;
} sslBuf_t;
```

#### Context

Client and Server

#### Description

This structure is used for input and output message buffers for the set of public APIs that decode and encode data. The start and end pointers in the buffer may be modified by the MatrixSSL APIs to indicate the data that was parsed or written to the buffer.

To get an idea of how to work with these buffers, here are some examples of buffer arithmetic:

$b.end - b.start$	Number of bytes of valid data in the buffer
$(b.buf + b.size) - b.end$	Number of bytes available in the buffer.
if ( $b.start > b.buf$ )	If there are unused bytes at the start of the buffer...

#### Members

buf	Pointer to the start of the buffer
start	Pointer to the first valid byte of data
end	Pointer one byte beyond the last valid byte of data.
size	Size of buffer in bytes

**sslCertInfo\_t****Definition**

```
typedef struct sslCertInfo {
    int                verified;
    unsigned char     *serialNumber;
    int                serialNumberLen;
    char               *notBefore;
    char               *notAfter;
    char               *sigHash;
    int                sigHashLen;
    subjectAltName_t  subjectAltName;
    distinguishedName_t subject;
    distinguishedName_t issuer;
    struct sslCertInfo *next
} sslCertInfo_t;
```

```
typedef struct {
    char *country;
    char *state;
    char *locality;
    char *organization;
    char *orgUnit;
    char *commonName;
} distinguishedName_t;
```

```
typedef struct {
    char *dns;
    char *uri;
    char *email;
} subjectAltName_t;
```

**Context**

Client.

Relevant to Server in commercial version as part of client authentication.

**Description**

This structure is passed to a user callback routine set by the application to perform any custom validation checks on a certificate. The default MatrixSSL validation check will previously have tested whether or not the certificate authority certificate has signed the certificate. The application code should call *matrixSslSetCertValidator* with the function that will receive the *sslCertInfo\_t* information of the certificate that was passed to the other side.

**Members**

verified	Status of the default validation check. The value will be -1 if the validation failed or 1 if it succeeded.
serialNumber	Serial number assigned by the issuer
serialNumberLen	Length of valid bytes in <i>serialNumber</i> member
notBefore	Start date of certificate validity
sigHash	The MD5 or SHA1 hash of the certificate signature
sigHashLen	The length of the sigHash member. Either 16 for MD5 or 20 for SHA1.
notAfter	End date of certificate validity
subjectAltName	The X509v3 subjectAltName extension often used in Web client applications for validating the FQDN
subject	The distinguished name info for the certificate being validated
issuer	The distinguished name info of the issuer of the certificate being validated
next	Pointer to the next sslCertInfo_t structure that represents the parent of the current certificate. NULL if there is no parent.

## Functions

The public API specifications follow. For sample usage, see the example code provided in the source code distribution.

### matrixSslOpen

**Prototype**

```
int matrixSslOpen();
```

**Context**

Client and Server

**Description**

This function performs the one-time initialization for MatrixSSL. Applications should call this function once as part of their own initialization to load the cipher suite and perform any operating system specific set up.

**Parameters**

None

**Return Value**

0	Success
< 0	Failure

### matrixSslClose

**Prototype**

```
void matrixSslClose();
```

**Context**

Client and Server

**Description**

This function performs the one-time final cleanup for MatrixSSL. Applications should call this function as part of their own final cleanup.

**Parameters**

None

**Return Value**

None

## matrixSslReadKeys

### Prototype

```
int matrixSslReadKeys(sslKeys_t **keys, char *certFile, char *privFile,  
                      char *privPass, char *trustedCAcertFiles);
```

### Context

Client and Server

Commercial users implementing memory pools should use *matrixRsaReadKeysEx*

### Description

This function is called to load the certificates and private key files from disk that are needed for client-server authentication. The key material is loaded into the *keys* output parameter.

The GNU MatrixSSL supports one-way authentication (server) so the parameters to this function are specific to the client/server role of the application. The *certFile*, *privFile*, and *privPass* parameters are server specific and should identify the certificate and private key file for that server. The *trustedCAcertFiles* is client specific and should identify the trusted root certificates that will be used to validate the certificates received from a server. Multiple trusted root certificates can be passed to this parameter as a semicolon delimited list of file names. Any key file or password parameter that does not apply to the application context should be passed in as NULL.

The commercial version of MatrixSSL supports two-way authentication. The *certFile* and *privFile* parameters are used to specify the certificate of the local entity. It is possible to identify a certificate chain (server side only) by using a semi-colon delimited list of certificate names. The list must be given in child/parent order with the first certificate being the server certificate. The private key file must correspond with this first certificate in the list. The *privPass* parameter is used if the *privFile* private key file is stored encrypted and password protected on disk. The *trustedCAcertFiles* parameter identifies the trusted root certificates that will be used to validate the certificates received from the other side of the communication channel. Multiple trusted root certificates can be passed to this parameter as a semicolon delimited list of file names. Any key file or password parameter that does not apply to the application context should be passed in as NULL.

In the commercial version the MatrixSSL library must be compiled with `USE_CLIENT_AUTH` defined for client authentication support.

The *sslKeys\_t* output parameter from this function is used as the input parameter when starting a new SSL session via *matrixSslNewSession*. The *sslKeys\_t* type has been defined in the public *matrixSsl.h* file to simply be an opaque integer type since applications do not need access to any of the structure members.

Calling this function is a relatively expensive operation because of the file access and parsing required to extract the key material. For this reason, it is typical that this function is only called once per set of key files for a given application. All new sessions associated with that certificate can reuse the returned key pointer. This function is separate from *matrixSslOpen* because some Web servers support virtual servers that each have different key pairs. The user must free the key structure using *matrixSslFreeKeys*.

### Parameters

keys	Output parameter for storing the key material
certFile	The filename (including path) of the certificate. Server only in GNU version.
privKeyFile	The filename (including path) of the private key file. Server only in GNU version.
privKeyPass	The password used to encrypt the private key file if used. Only 3DES CBC encryption is supported. Server only in GNU version.
trustedCAcertFile	The filename (including path) of a trusted root certificate. Multiple files may be passed in a semicolon delimited list. Client only in GNU version.

### Return Value

0	Success. A valid key pointer will be returned in the <i>keys</i> parameter for use in a subsequent call to <i>matrixSslNewSession</i>
<0	Failure

## **matrixRsaReadKeysEx**

### **Prototype**

```
int matrixRsaReadKeysEx(psPool_t *pool, sslKeys_t **keys, char *certFile,  
                        char *privFile, char *privPass, char *trustedCAcertFiles);
```

### **Context**

Client and Server commercial version

### **Header File**

Must include 'src/pki/matrixPki.h'

### **Description**

This extended version adds a *psPool\_t* parameter so a user implementing memory pools may specify the pool. Otherwise, it is identical in every way to the parameter inputs, return codes, and usage described above. The *matrixSslReadKeys* function will allocate the key structure from the PEERSEC\_BASE\_POOL. If an implementation requires an indefinite number of key reads and the extended version is not used, the base pool will become exhausted.

## matrixSslReadKeysMem

### Prototype

```
int matrixSslReadKeysMem(sslKeys_t **keys, char *certBuf,  
    int certLen, char *privBuf, int32 privLen, char *trustedCABuf,  
    int32 trustedCALen);
```

### Context

Client and Server

Commercial users implementing memory pools should use *matrixRsaParseKeysMem*

### Description

An in-memory version of the *matrixSslReadKeys* function. This version can be used in environments where the certificate material is not stored on disk. The usage is identical to the file version except that this memory version does not support certificate chaining. Reads an entire set of certificate, private key, and CA certificate buffers for an SSL session and returns the *sslKeys\_t* structure to be passed to *matrixSslNewSession*. The keys parameter must be freed with a call to *matrixSslFreeKeys*.

The buffers for the certificates and private key must be in ASN.1 standard format. For certificates, this is the X.509 standard. For private keys, this is the PKCS #8 specification.

## matrixRsaParseKeysMem

### Prototype

```
int matrixRsaParseKeysMem(psPool_t *pool, sslKeys_t **keys, char *certBuf,  
    int certLen, char *privBuf, int32 privLen, char *trustedCABuf,  
    int32 trustedCALen);
```

### Context

Client and Server

### Header File

Must include 'src/pki/matrixPki.h'

### Description

This extended version of *matrixSslReadKeysMem* adds a *psPool\_t* parameter so a user implementing memory pools may specify the pool. Otherwise, it is identical in every way to the parameter inputs, return codes, and usage as described above. The *matrixSslReadKeysMem* function will allocate the key structure from the PEERSEC\_BASE\_POOL. If an implementation requires an indefinite number of key reads and the extended version is not used, the base pool will become exhausted.

## matrixSslFreeKeys

### Prototype

```
void matrixSslFreeKeys(sslKeys_t *keys);
```

### Context

Client and Server

### Description

This function is called to free the key structure and elements allocated from a previous call to *matrixSslReadKeys*.

### Parameters

keys	A pointer to an <i>sslKeys_t</i> value returned from a previous call to <i>matrixSslReadKeys</i>
------	--

### Return Value

None

## matrixSslNewSession

### Prototype

```
int matrixSslNewSession(ssl_t **ssl, sslKeys_t *keys, sslSessionId_t *sessionId,
    int flags);
```

### Context

Client and Server

### Description

This function is called to start a new SSL session, or resume a previous one, with a client or server. The session is returned in the output parameter *ssl*. This function requires a pointer to an *sslKeys\_t* value returned from a previous call to *matrixSslReadKeys* and the *flags* parameter to specify whether this is a server side usage. MatrixSSL supports client initiated SSL sessions and the *sessionId* parameter is specific to client implementations only. If the client is resuming a prior session, this parameter will be the value returned from a call to *matrixSslGetSessionId*. Otherwise, this parameter must be NULL. The client must pass 0 as the flags parameter. A client will make a call to this function prior to calling *matrixSslEncodeClientHello*.

When a server application has received notice that a client is requesting a secure socket connection (a socket accept on a secure port), this function should be called to initialize the new session structure. The *sessionId* parameter must be set to NULL for server side implementations.

The server must pass SSL\_FLAGS\_SERVER as one of the flags, otherwise the resulting SSL session will be initialized to parse the client side protocol.

Commercial users may optionally include the SSL\_FLAGS\_CLIENT\_AUTH parameter if client authentication is desired. The MatrixSSL library must be compiled with USE\_CLIENT\_AUTH defined for client authentication support.

The output parameter is an *ssl\_t* structure that will be used as input parameters to the *matrixSslDecode* and *matrixSslEncode* family of APIs for decrypting and encrypting messages. The *ssl\_t* type has been defined in the public *matrixSsl.h* file to simply be an opaque integer type since users do not need access to any of the structure members. The user must free the *ssl\_t* structure using *matrixSslDeleteSession*.

### Parameters

ssl	Output. The new SSL session created by this call
keys	The opaque key material pointer returned from a call to <i>matrixSslReadKeys</i>
sessionId	Prior session id obtained from <i>matrixSslGetSessionId</i> if

	client is resuming a session. NULL otherwise.
flags	SSL_FLAGS_SERVER for server and 0 for client. Servers may optionally include SSL_FLAGS_CLIENT_AUTH in the commercial version.

**Return Value**

0	Success. A newly allocated session structure will be returned in the <i>ssl</i> parameter for use as the input parameter on session related decoding and encoding APIs
<0	Failure

**matrixSslDeleteSession****Prototype**

```
void matrixSslDeleteSession(ssl_t *session);
```

**Context**

Client and Server

**Description**

This function is called at the conclusion of an SSL session that was created using *matrixSslNewSession*. This function will free the allocated memory associated with the session. It should be called after the corresponding socket has been closed.

A client wishing to reconnect later to the same server may choose to call *matrixSslGetSessionId* prior to calling this delete session function to save aside the session id for later use with *matrixSslNewSession*.

**Parameters**

session	The <i>ssl_t</i> session pointer returned from the call to <i>matrixSslNewSession</i>
---------	---

**Return Value**

None

## matrixSslDecode

### Prototype

```
int matrixSslDecode(ssl_t *session, sslBuf_t *in, sslBuf_t *out,
    unsigned char *error, unsigned char *alertLevel,
    unsigned char *alertDescription);
```

### Context

Client and Server

### Description

This is a powerful function used to decode all messages received from a peer, including handshake and alert messages. The input parameters include the *ssl\_t* session from the previous call to *matrixSslNewSession* and an *sslBuf\_t* input buffer containing the message received from the client or server. This function is typically called in a loop during the handshake process. The return value indicates the type of message received and the *out* buffer parameter may contain an encoded message to send to the other side or a decoded message for the application to process. The *in* buffer may have its start pointer moved forward to indicate the bytes that were successfully decoded. The *out* buffer end pointer may be modified to reflect the output data written to the buffer.

### Parameters

session	The <i>ssl_t</i> session structure associated with this instance. Created by the call to <i>matrixSslNewSession</i>
in	The <i>sslBuf_t</i> buffer containing the input message from the other side of the client/server communication channel
out	The output buffer after returned to the application
error	On SSL_ERROR conditions, this output parameter specifies the error description associated with the error
alertLevel	On SSL_ALERT conditions, this output parameter specifies the alert level associated with the client alert message
alertDescription	On SSL_ALERT conditions, this output parameter specifies the alert description associated with the client alert message

### Return Value

SSL_SUCCESS	A handshake message was successfully decoded and handled. No additional action is required for this message. <i>matrixSslDecode</i> can be called again immediately if more data is expected. This return code gives visibility into the handshake process and can be used in conjunction with <i>matrixSslHandshakeIsComplete</i> to determine when the handshake is complete and application data can be sent.
SSL_SEND_RESPONSE	This value indicates the input message was part of the

	SSLv3 internal protocol and a reply is expected. The application should send the data in the out buffer to the other side and then call <i>matrixSslDecode</i> again to see if any more message data needs to be decoded.
SSL_ERROR	This value indicates there has been an error while attempting to decode the data or that a bad message was sent. The application should attempt to send the contents of out buffer, if any (likely an error alert) to the other side as a reply and then close the communication layer (i.e. close the socket).
SSL_ALERT	This value indicates the message was an alert sent from the other side and the application should close the communication layer (i.e. close the socket).
SSL_PARTIAL	This value indicates that the input buffer was an incomplete message or record. The application must retrieve more data from the communications layer (socket) and call <i>matrixSslDecode</i> again when more data is available.
SSL_FULL	This value indicates the output buffer was too small to hold the output message. The application should grow the output buffer and call <i>matrixSslDecode</i> again with the same input buffer. The maximum size of the buffer output buffer will never exceed 16K per the SSLv3 standard.
SSL_PROCESS_DATA	This value indicates that the message is application specific data that does not require a response from the server. This message is an implicit indication that SSLv3 handshaking is complete. The decoded data has been written to the output buffer for application consumption.

**matrixSslEncode****Prototype**

```
int matrixSslEncode(ssl_t *session, unsigned char *in, int inLen, sslBuf_t *out);
```

**Context**

Client and Server

**Description**

This function is used by the application to generate encrypted messages to be sent to the other side of the client/server communication channel. Only application level messages should be generated with this API. Handshake messages are generated internally as part of *matrixSslDecode*. It is the responsibility of the application to actually transmit the generated output buffer to the other side.

**Parameters**

session	The <i>ssl_t</i> session identifier for this session.
in	The plain-text message buffer to encrypt
inLen	The length of valid data in the input buffer to encrypt
out	The encrypted message to be passed to the other side

**Return Value**

>= 0	Success. The value is the length of the encrypted data.
SSL_ERROR	Error. The connection should be closed, and session deleted.
SSL_FULL	The output buffer is not big enough to hold the encrypted data. Grow the buffer and retry.

**matrixSslEncodeClosureAlert****Prototype**

```
int matrixSslEncodeClosureAlert(ssl_t *session, sslBuf_t * out);
```

**Context**

Client and Server

**Description**

An optional function call made before closing the communication channel with a peer. This function alerts the peer that the connection is about to close. Some implementations simply close the connection without an alert, but per spec, this message should be sent first.

**Parameters**

session	The <i>ssl_t</i> session identifier for this session
out	The output alert closure message to be passed along to the client.

**Return Value**

0	Success
SSL_FULL	The output buffer is not big enough to hold the encrypted data. Grow the buffer and retry.
SSL_ERROR	Failure

## matrixSslEncodeClientHello

### Prototype

```
int matrixSslEncodeClientHello(ssl_t *session, sslBuf_t * out,
    unsigned short cipherSuite);
```

### Context

Client

### Description

This function builds the initial CLIENT\_HELLO message to be passed to a server to begin SSL communications. This function is called once by the client before entering into the *matrixSslDecode* handshake loop.

The *cipherSuite* parameter can be used to force the client to send a single cipher to the server rather than the entire set of supported ciphers. Set this value to 0 to send the entire cipher suite list. Otherwise the value is the two byte value of the cipher suite specified in the standards. The supported values can be found in *matrixInternal.h*.

This function may also be called by a client at the conclusion of the initial handshake at any time to initiate a re-handshake. A re-handshake is a complete SSL handshake protocol performed on an existing connection to derive new symmetric key material and/or to change the cipher spec of the communications. All re-handshake messages will be encrypted using the previously negotiated cipher suite. If the caller wants to assure that a new session id is used for the re-handshake, the function *matrixSslDeleteCurrentSessionId* should be called prior to calling *matrixSslEncodeClientHello*. It is always at the discretion of the server whether or not to resume on a session id passed in by the client in the CLIENT\_HELLO message. However, the client can force a new session if the session id is not passed in originally.

### Parameters

session	The <i>ssl_t</i> session identifier for this session
out	The output alert closure message to be passed along to the client.
cipherSuite	The two byte cipher suite identifier

### Return Value

0	Success
SSL_FULL	The output buffer is not big enough to hold the encrypted data. Grow the buffer and retry.
SSL_ERROR	Failure

## matrixSslEncodeHelloRequest

### Prototype

```
int matrixSslEncodeHelloRequest(ssl_t *session, sslBuf_t * out);
```

### Context

Server

### Description

This function builds a HELLO\_REQUEST message to be passed to a client to initiate a re-handshake. This is the only mechanism in the SSL protocol that allows the server to initiate a handshake. A re-handshake can be done on an existing session to derive new symmetric cryptographic keys, perform client authentication, or to change the cipher spec. All messages exchanged during a re-handshake are encrypted under the currently negotiated cipher suite.

If the server wishes to change any session options for the re-handshake it should call *matrixSslSetSessionOption* to modify the handshake behavior.

Note: The SSL specification allows clients to ignore a HELLO\_REQUEST message. The MatrixSSL client does not ignore this message and will send a CLIENT\_HELLO message with the current session id.

### Parameters

session	The <i>ssl_t</i> session identifier for this session
out	The output alert closure message to be passed along to the client.

### Return Value

0	Success
SSL_FULL	The output buffer is not big enough to hold the data. Grow the buffer and retry.
SSL_ERROR	Failure

## matrixSslSetSessionOption

### Prototype

```
void matrixSslSetSessionOption(ssl_t *session, int option, void *arg);
```

### Context

Client and Server

### Description

The *matrixSslSetSessionOption* function is used to modify the behavior of the SSL handshake protocol for a re-handshake. This function is only meaningful to call on an existing SSL session before a re-handshake to give the client or server control over which handshake type to perform (full, resumed, or client authentication).

A server initiated re-handshake is done by sending the HELLO\_REQUEST message which can be constructed by calling *matrixSslEncodeHelloRequest*. Prior to sending this message, the server may wish to disallow a resumed re-handshake by passing the option of SSL\_OPTION\_DELETE\_SESSION as the *option* parameter to this function. This will delete the current session information from the local cache so it will not be found if the client passes a session id in the subsequent CLIENT\_HELLO message.

The commercial version the server also has the ability to enable or disable a client authentication re-handshake by passing the option SSL\_OPTION\_ENABLE\_CLIENT\_AUTH or SSL\_OPTION\_DISABLE\_CLIENT\_AUTH as the *option* parameter to this function.

A client initiated re-handshake is done by simply sending a new CLIENT\_HELLO message over an existing connection. If the client application wishes a full re-handshake to be performed, it should call this function with SSL\_OPTION\_DELETE\_SESSION.

In both the client and server cases, a resumed re-handshake may be performed by excluding any calls to this function before sending the HELLO\_REQUEST or CLIENT\_HELLO messages.

For more information about re-handshaking, see the Re-handshake section of the MatrixSSL Developers Guide.

### Parameters

session	The <i>ssl_t</i> session identifier for a currently connected session
option	If server, one of: SSL_OPTION_DELETE_SESSION, SSL_OPTION_DISABLE_CLIENT_AUTH, or

	SSL_OPTION_ENABLE_CLIENT_AUTH (commercial version only for CLIENT_AUTH options) If client: SSL_OPTION_DELETE_SESSION
arg	NULL. Reserved for future use.

**Return Value**

None

**matrixSslHandshakeIsComplete****Prototype**

```
int matrixSslHandshakeIsComplete(ssl_t *session);
```

**Context**

Client and Server

**Description**

This function returns whether or not the handshake portion of the *session* is complete. This API can be used to test when it is OK to send the first application data record on an SSL connection.

**Parameters**

session	The <i>ssl_t</i> session identifier for this session
---------	--

**Return Value**

1	Handshake is complete
0	Handshake is NOT complete

**matrixSslGetSessionId****Prototype**

```
int matrixSslGetSessionId(ssl_t *session, sslSessionId_t **sessionId);
```

**Context**

Client

**Description**

This function is used by a client application to extract the session id from an existing session for use in a subsequent call to *matrixSslNewSession* wishing to resume a session. A resumed session is much faster to negotiate because the public key encryption process does not need to be performed and two handshake messages are avoided. The *sessionId* return parameter of this function is valid even after *matrixSslDeleteSession* has been called on the current session. This function should only be called by a client SSL session after the handshake is complete (session id is established).

The *sslSessionId\_t* structure has been defined in the public header as an opaque integer type since the contents of the structure do not need to be accessed by the application. The session id must be freed with a call to *matrixSslFreeSessionId*.

**Parameters**

session	The <i>ssl_t</i> session identifier for this session
sessionId	Output. The returned session id for the given SSL session

**Return Value**

0	Success. An allocated session id is returned in <i>sessionId</i>
<0	Failure ( <i>sessionId</i> unavailable)

## matrixSslFreeSessionId

**Prototype**

```
void matrixSslFreeSessionId(sslSessionId_t *sessionId);
```

**Context**

Client

**Description**

This function is used by a client application to free a session id returned from a previous call to *matrixSslGetSessionId*.

**Parameters**

sessionId	The <i>sslSession_t</i> identifier
-----------	------------------------------------

**Return Value**

None

## matrixSslSetCertValidator

### Prototype

```
void matrixSslSetCertValidator(ssl_t *session,  
    int (*certValidator)(sslCertInfo_t*, void *arg), void *arg);
```

### Context

Client. Relevant to Server in the commercial version for client authentication.

### Description

This function is used by applications to register a callback routine that will be invoked during the certificate validation process. This optional registration will enable the application to perform custom validation checks or to pass certificate information on to end users wishing to manually validate certificates.

In the commercial version this functionality may be used on the server side if client authentication is being used (the MatrixSSL library must be compiled with `USE_CLIENT_AUTH` defined and the `SSL_FLAGS_CLIENT_AUTH` must be passed to *matrixSslNewSession*.)

The registered function must have the following prototype:

```
int appCertValidator(sslCertInfo_t *certInfo, void *arg);
```

The *certInfo* parameter is the incoming *sslCertInfo\_t* structure containing information about the certificate. This certificate information is read-only from the perspective of the validating callback function. The structure members are available in the *Structures* section in this document and in the *matrixSsl.h* public header file.

The *verified* member of *certInfo* will indicate whether or not the certificate passed the default MatrixSSL validation checks. If the *subjectCert* is a chain, the *parent* member will link to the next certificate in the chain. A typical callback implementation might be to check the value of the *verified* member and pass the certificate information along to the user if it had not passed the default validation checks. Additional tests a callback may want to perform on the certificate information might include date validation and hostname verification.

The *arg* parameter is a user specific argument that was specified in the *arg* parameter to the *matrixSslSetCertValidator* routine. This argument can be used to give session context to the callback if needed.

The callback function should return a value 1 if the custom validation check is successful and the certificate is determined to be acceptable. The callback may also choose to return `SSL_ALLOW_ANON_CONNECTION` if an anonymous connection is desired. The handshake will continue in this anonymous case and

the application data will be encrypted as usual. It is not typically advised to allow anonymous connections in a standard use case, but may sometimes such a connection is desirable.

The callback function must return a negative value if the validation checks fails or if the user rejects the certificate for any reason. The negative return code will be processed by the MatrixSSL library and the handshake process will terminate as an authentication failure.

### Parameters

session	The <i>ssl_t</i> session identifier for this session
certValidator	The function callback that will be invoked to validate the certificate
arg	Implementation specific data that will be received by the callback. Use to give session context if needed, NULL otherwise.

### Return Value

None

## matrixSslGetAnonStatus

### Prototype

```
void matrixSslGetAnonStatus(ssl_t *session, int *anonArg);
```

### Context

Client. Relevant to Server in the commercial version for client authentication.

### Description

This function returns whether or not the provided session is anonymous in the *anonArg* output parameter. A value of 1 indicates the connection is anonymous and a value of 0 indicates the connection has been authenticated. An anonymous connection in this case means the calling entity (client or server) explicitly allowed the SSL handshake to continue despite not being able to authenticate the certificate supplied by the other side with an available Certificate Authority. The mechanism to allow an anonymous connection is for the certificate validation callback function (see *matrixSslSetCertValidator*) to return `SSL_ALLOW_ANON_CONNECTION`.

The *matrixSslGetAnonStatus* is only meaningful to call after the successful completion of the SSL handshake to determine if the existing connection is anonymous. Anonymous connections are not normally recommended but can be useful in a scenario in which encryption is the primary security concern. Other reasons the caller may choose to use anonymous connections might be to allow a subset of the normal functionality to anonymous connectors or to temporarily accept a connection while a certificate upgrade is being performed.

The anonymous status is only relevant to the entity that calls this routine. For example, calling this routine from the server side is meaningless for an implementation that has not performed client authentication because the server can not know if it is anonymous to the client or not. Therefore, it is not possible for one side of the connection to know if the other side believes the connection to be anonymous from their standpoint. This is an easy rule to remember if you recall the mechanism to allow anonymous connections is controlled through the certificate validation callback routine when the `SSL_ALLOW_ANON_CONNECTION` define is returned. Client authentication is only available in the commercial version of MatrixSSL.

### Parameters

session	The <i>ssl_t</i> session identifier for this session
anonArg	Return status of the connection. 1 if anonymous and 0 if authenticated.

### Return Value

None

## matrixSslAssignNewKeys

### Prototype

```
void matrixSslAssignNewKeys(ssl_t *session, sslKeys_t *keys);
```

### Context

Client and Server

### Description

This routine is used to change the underlying certificate or certificate authority information for an existing, open connection. This function is intended to help the process of upgrading certificate material between two SSL entities that are currently connected. The *keys* parameter is an *sslKeys\_t* type that was created by a call to *matrixSslReadKeys*. The new key material is associated with the passed in *session*.

The user should have freed any previously allocated keys with a call to *matrixSslFreeKeys* before assigning new keys to the session with this routine. Once the new keys are associated with the session, the application may initiate a re-handshake over the existing connection to authenticate with the new key material. In the client case, a new CLIENT\_HELLO message (*matrixSslEncodeClientHello*) should be sent to kick off the re-handshake. In the server scenario, a HELLO\_REQUEST (*matrixSslEncodeHelloRequest*) message would be sent. The benefit of this method is that the current connection does not have to be closed in order to upgrade certificate material.

**Parameters**

session	The <i>ssl_t</i> session identifier for this session
keys	New keys returned from a previous call to <i>matrixSslReadKeys</i>

**Return Value**

None