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.

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 {
        int                   verified;
        unsigned char         *serialNumber;
        int                   serialNumberLen;
        char                  *notBefore;
        char                  *notAfter;
        char                  *sigHash;
        int                   sigHashLen;
        subjectAltName_t      subjectAltName;
        distinguishedName_t   subject;
        distinguishedName_t   issuer;
} 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

Description

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

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 *serialNumber* 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

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

Description

This function is called to load the certificates and private key files from disk that are needed for 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 *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*.

A buffered memory version of this function is included in the library for environments where the certificate material is not stored on disk. That version can be found by searching for *matrixSslReadKeysMem* in the source code.

Parameters

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

Return Value

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

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 *sslKeys_t* value returned from a previous call to *matrixSslReadKeys*

Return Value
None

matrixSslNewSession

Prototype
int matrixSslNewSession(ssl_t **ssl, sslKeys_t *keys, sslSessionId_t *sesssionId,
    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 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 *matrixSslReadKeys*
sessionId	Prior session id obtained from *matrixSslGetSessionId* if client is resuming a session. NULL otherwise.
flags	SSL_FLAGS_SERVER for server and 0 for client.

Return Value

0	Success. A newly allocated session structure will be returned in the *ssl* 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 *ssl_t* session pointer returned from the call to *matrixSslNewSession*

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 *ssl_t* session structure associated with this instance. Created by the call to *matrixSslNewSession*
in	The *sslBuf_t* 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. *matrixSslDecode* 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 *matrixSslHandshakeIsComplete* 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 *matrixSslDecode* 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 *matrixSslDecode* 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 *matrixSslDecode* 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 *ssl_t* 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 *ssl_t* 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*.

Parameters

session	The *ssl_t* 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

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 *ssl_t* 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 *ssl_t* 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 *sessionId*
<0	Failure (*sessionId* 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 *sslSession_t* identifier

Return Value

None

matrixSslSetCertValidator

Prototype

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

Context

Client

Description

This function is used by client 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.

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. 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.

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 >= 0 if the custom validation check is successful and the certificate is determined to be acceptable. The callback function must return a negative value if the validation checks fails for any reason. The negative return code will be passed back to the MatrixSSL library and the handshake process will terminate.

Parameters

session	The *ssl_t* 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