MatrixSSL Sockets API documentation

The MatrixSSL library is not written for a specific transfer protocol. The generic buffer based encryption/decryption routines make MatrixSSL ideal for securing data to be transported over virtually any communication channel. However, the majority of MatrixSSL integrations will be to secure data between computer systems over TCP sockets. The MatrixSSL package includes a sample socket layer wrapper around the public APIs as a reference implementation for customers wishing to use MatrixSSL to secure their sockets-based applications. This document details the socket layer APIs.

The example socket layer source code can be found in the *examples/sslSocket.c* file in the installed package. For working examples see the *httpsReflector* and *httpsClient* applications which take advantage of these socket level APIs.

The sockets interface is currently designed for blocking sockets operation. Support for non-blocking sockets is easily implemented using this code as a base. Or you may contact PeerSec for a non-blocking socket example.

Raw socket functions

For clarity, the initialization, shutdown, and raw socket control functions have been implemented separately from the security layer functions that wrap the MatrixSSL public APIs. This functionality will typically already be part of a networked application. The details of these routines are beyond the scope of this document but brief descriptions are given in this section.

```
SOCKET socketListen(short port, int *err);
```
Creates a new socket and binds to the given port

```
SOCKET socketAccept(SOCKET listenfd, int *err);
```
Creates new socket from an incoming request to a listen socket

```
SOCKET socketConnect(char *ip, short port, int *err);
```
Creates a new socket and connects to a listen socket on the given IP and port

```
Void socketShutdown(SOCKET sock);
```
Closes a socket

```
void setSocketBlock(SOCKET sock);
```
Sets a socket to blocking mode

```
void setSocketNonblock(SOCKET sock);
```
Sets a socket to non-blocking mode

```
void setSocketNodelay(SOCKET sock);
```
Sets the NO_DELAY option on the socket

Secure socket functions

Details for the functions that wrap the MatrixSSL public APIs are described in this section.

sslConnect

Prototype

```
int sslConnect(sslConn_t **cp, SOCKET fd, sslKeys_t *keys, sslSessionId_t *id,
               short cipherSuite, int (*certValidator)(sslCertInfo_t *t, void *arg));
```

Description

This client side function is used to connect to a server that has set up a listen socket. This function connects the socket and performs the SSL handshake with the server. The returned *cp* parameter is the new SSL connection context used as input for the *sslRead* and *sslWrite* routines. The remainder of the parameters are inputs that control the initialization of the new context.

At the successful completion of this function, the application may call *sslRead* and *sslWrite* as necessary to communicate with the server.

The *cp* parameter should be freed when no longer needed by calling *sslFreeConnection*.

Parameters

cp	Output. Initialize to NULL. Newly allocated connection context on success.
fd	Socket descriptor returned from a previous call to *socketConnect.*
keys	The MatrixSSL key structure returned from a previous call to *matrixSslReadKeys.*
id	An optional session id from a previous connection to initiate a resumed SSL session. The id can be retrieved by a call to *matrixSslGetSessionId* after a session has been successfully negotiated.
cipherSuite	An optional parameter that will restrict the cipher suite to the single specified value for the connection. Supported cipher suite values can be found in *matrixInternal.h.* Set to 0 for no preference.
certValidator	An optional function callback that will be invoked as part of the server

	certificate validation process. See the public API documentation for *matrixSslSetCertValidator*. Pass as NULL if not used.

Return Value

0	Success. Handshake complete.
< 0	Failure. Can't continue with this connection.

sslAccept

Prototype

```
int sslAccept(sslConn_t **cp, SOCKET fd, sslKeys_t *keys,
        int (*certValidator)(sslCertInfo_t *t, void *arg), int flags);
```

Description

This server side function is used to accept a client connection on an existing socket. This function connects with the client and performs the SSL handshake. The returned *cp* parameter is the new SSL connection context used as input for the *sslRead* and *sslWrite* routines. The remainder of the parameters are inputs that control the initialization of the new context.

At the successful completion of this function, the application may call *sslRead* and *sslWrite* as necessary to communicate with the client.

The *cp* parameter should be freed when no longer needed by calling *sslFreeConnection*.

Parameters

cp	Output. Initialize to NULL. Newly allocated connection context on success.
fd	Socket descriptor returned from a previous call to *socketAccept.*
keys	The MatrixSSL key structure returned from a previous call to *matrixSslReadKeys.*
certValidator	An optional function callback that will be invoked as part of the client certificate validation process. See the public API documentation for *matrixSslSetCertValidator*. Pass as NULL if not used. If set, flags must include SSL_FLAGS_CLIENT_AUTH.
flags	0 or SSL_FLAGS_CLIENT_AUTH

Return Value

0	Success. Handshake complete
< 0	Failure. Can't continue with this connection

sslFreeConnection

Prototype
void sslFreeConnection(sslConn_t **cp);

Description
Free a connection that was opened with *sslAccept* or *sslConnect*

Parameters

cp	The connection to close

Return Value
none

sslRead

Prototype
int sslRead(sslConn_t *cp, char *buf, int len, int *status);

Description
This function reads secure data from the given connection and returns the decoded data to the caller.

Parameters

cp	The connection to read from. Returned from a previous call to *sslAccept* or *sslConnect*
buf	A user allocated buffer to hold the returned decoded data that was read from the SSL socket.
len	The length in bytes of the allocated *buf* parameter
status	Status information. On a socket failure, the status will contain the error code. In a zero return code case, status may be set to SSLSOCKET_EOF if the connection was closed by the other side.

Return Value

Positive integer	The number of bytes successfully read into *buf*. The function may be called again with an updated buffer if there is more to read.
0	Success, but no data to be returned to the caller. This special case typically indicates a handshake message was successfully decoded and handled. No additional action is required for this message. This return code gives visibility into the handshake process and can be used in conjunction with *matrixSslHandshakeIsComplete* to determine when the handshake is complete. The other possible scenario for this return case is if the connection has been closed by the other side. In this case, the status parameter will be set to SSLSOCKET_EOF.
< 0	Failure. The status parameter contains specific information about the error if it is socket related. If using a non-blocking socket implementation the caller should check for non-fatal errors such as WOULD_BLOCK before determining whether to close the connection.

sslWrite

Prototype

int sslWrite(sslConn_t *cp, char *buf, int len, int *status);

Description

This function encodes and writes secure data to the given connection.

Parameters

cp	The connection to write to. Returned from a previous call to *sslAccept* or *sslConnect*
buf	The un-encoded data to be written to the

	connection
len	The length of the *buf* parameter in bytes
status	Status information. On a socket failure, the status will contain the error code. Set to 0 on internal function errors.

Return Value

Positive integer	Success. Return value is number of bytes written to the SSL connection. Should always match *len* parameter.
0	Indicates that *sslWrite* must be called again as all the data was not able to be written in one pass. Call again with same parameters.
< 0	Failure. If a socket level error, error code is contained in status. If using a non-blocking socket implementation the caller should check for non-fatal errors such as WOULD_BLOCK before closing the connection. A zero value in status indicates an error with this routine.

sslWriteClosureAlert

Prototype

void sslWriteClosureAlert(sslConn_t *cp);

Description

Writes an SSL closure alert to the connection.

Parameters

cp	The connection to write the alert to. Returned from a previous call to *sslAccept* or *sslConnect*

Return Value

none