NAME

SSL_CTX_set_max_send_fragment, SSL_set_max_send_fragment, SSL_CTX_set_split_send_fragment, SSL_set_split_send_fragment, SSL_CTX_set_max_pipelines, SSL_set_max_pipelines, SSL_CTX_set_default_read_buffer_len, SSL_set_default_read_buffer_len, SSL_CTX_set_tlsext_max_fragment_length, SSL_set_tlsext_max_fragment_length, SSL_SESSION_get_max_fragment_length - Control fragment size settings and pipelining operations

SYNOPSIS

```
#include <openssl/ssl.h>

long SSL_CTX_set_max_send_fragment(SSL_CTX *ctx, long);
long SSL_set_max_send_fragment(SSL *ssl, long m);

long SSL_CTX_set_max_pipelines(SSL_CTX *ctx, long m);
long SSL_set_max_pipelines(SSL_CTX *ssl, long m);

long SSL_CTX_set_split_send_fragment(SSL_CTX *ctx, long m);
long SSL_set_split_send_fragment(SSL *ssl, long m);

void SSL_CTX_set_default_read_buffer_len(SSL_CTX *ctx, size_t len);
void SSL_set_default_read_buffer_len(SSL *s, size_t len);

int SSL_CTX_set_tlsext_max_fragment_length(SSL_CTX *ctx, uint8_t mode);
int SSL_set_tlsext_max_fragment_length(SSL *ssl, uint8_t mode);
uint8_t SSL_SESSION_get_max_fragment_length(const SSL_SESSION *session);
```

DESCRIPTION

Some engines are able to process multiple simultaneous crypto operations. This capability could be utilised to parallelise the processing of a single connection. For example a single write can be split into multiple records and each one encrypted independently and in parallel. Note: this will only work in TLS1.1+. There is no support in SSLv3, TLSv1.0 or DTLS (any version). This capability is known as "pipelining" within OpenSSL.

In order to benefit from the pipelining capability. You need to have an engine that provides ciphers that support this. The OpenSSL "dasync" engine provides AES128-SHA based ciphers that have this capability. However, these are for development and test purposes only.

SSL_CTX_set_max_send_fragment() and **SSL_set_max_send_fragment()** set the **max_send_fragment** parameter for SSL_CTX and SSL objects respectively. This value restricts the amount of plaintext bytes that will be sent in any one SSL/TLS record. By default its value is

SSL3_RT_MAX_PLAIN_LENGTH (16384). These functions will only accept a value in the range 512 - SSL3_RT_MAX_PLAIN_LENGTH.

SSL_CTX_set_max_pipelines() and **SSL_set_max_pipelines()** set the maximum number of pipelines that will be used at any one time. This value applies to both "read" pipelining and "write" pipelining. By default only one pipeline will be used (i.e. normal non-parallel operation). The number of pipelines set must be in the range 1 - SSL_MAX_PIPELINES (32). Setting this to a value > 1 will also automatically turn on "read_ahead" (see **SSL_CTX_set_read_ahead**(3)). This is explained further below. OpenSSL will only ever use more than one pipeline if a cipher suite is negotiated that uses a pipeline capable cipher provided by an engine.

Pipelining operates slightly differently for reading encrypted data compared to writing encrypted data. **SSL_CTX_set_split_send_fragment()** and **SSL_set_split_send_fragment()** define how data is split up into pipelines when writing encrypted data. The number of pipelines used will be determined by the amount of data provided to the **SSL_write_ex()** or **SSL_write()** call divided by **split_send_fragment**.

For example if **split_send_fragment** is set to 2000 and **max_pipelines** is 4 then:

SSL_write/SSL_write_ex called with 0-2000 bytes == 1 pipeline used

SSL_write/SSL_write_ex called with 2001-4000 bytes == 2 pipelines used

SSL_write/SSL_write_ex called with 4001-6000 bytes == 3 pipelines used

SSL_write/SSL_write_ex called with 6001+ bytes == 4 pipelines used

split_send_fragment must always be less than or equal to **max_send_fragment**. By default it is set to be equal to **max_send_fragment**. This will mean that the same number of records will always be created as would have been created in the non-parallel case, although the data will be apportioned differently. In the parallel case data will be spread equally between the pipelines.

Read pipelining is controlled in a slightly different way than with write pipelining. While reading we are constrained by the number of records that the peer (and the network) can provide to us in one go. The more records we can get in one go the more opportunity we have to parallelise the processing. As noted above when setting **max_pipelines** to a value greater than one, **read_ahead** is automatically set. The **read_ahead** parameter causes OpenSSL to attempt to read as much data into the read buffer as the network can provide and will fit into the buffer. Without this set data is read into the read buffer one record at a time. The more data that can be read, the more opportunity there is for parallelising the processing at the cost of increased memory overhead per connection. Setting **read_ahead** can impact the behaviour of the **SSL_pending()** function (see **SSL_pending**(3)). In addition the default size of the

internal read buffer is multiplied by the number of pipelines available to ensure that we can read multiple records in one go. This can therefore have a significant impact on memory usage.

The **SSL_CTX_set_default_read_buffer_len()** and **SSL_set_default_read_buffer_len()** functions control the size of the read buffer that will be used. The **len** parameter sets the size of the buffer. The value will only be used if it is greater than the default that would have been used anyway. The normal default value depends on a number of factors but it will be at least SSL3_RT_MAX_PLAIN_LENGTH + SSL3_RT_MAX_ENCRYPTED_OVERHEAD (16704) bytes.

SSL_CTX_set_tlsext_max_fragment_length() sets the default maximum fragment length negotiation mode via value **mode** to **ctx**. This setting affects only SSL instances created after this function is called. It affects the client-side as only its side may initiate this extension use.

SSL_set_tlsext_max_fragment_length() sets the maximum fragment length negotiation mode via value **mode** to **ssl**. This setting will be used during a handshake when extensions are exchanged between client and server. So it only affects SSL sessions created after this function is called. It affects the client-side as only its side may initiate this extension use.

SSL_SESSION_get_max_fragment_length() gets the maximum fragment length negotiated in **session**.

RETURN VALUES

All non-void functions return 1 on success and 0 on failure.

NOTES

The Maximum Fragment Length extension support is optional on the server side. If the server does not support this extension then **SSL_SESSION_get_max_fragment_length()** will return: TLSEXT_max_fragment_length_DISABLED.

The following modes are available:

TLSEXT_max_fragment_length_DISABLED
: Disables Maximum Fragment Length Negotiation (default).

TLSEXT_max_fragment_length_512
: Sets Maximum Fragment Length to 512 bytes.

TLSEXT_max_fragment_length_1024
: Sets Maximum Fragment Length to 1024.

TLSEXT_max_fragment_length_2048

Sets Maximum Fragment Length to 2048.

TLSEXT_max_fragment_length_4096
Sets Maximum Fragment Length to 4096.

With the exception of **SSL_CTX_set_default_read_buffer_len() SSL_set_default_read_buffer_len()**, **SSL_CTX_set_tlsext_max_fragment_length()**, **SSL_set_tlsext_max_fragment_length()** and **SSL_SESSION_get_max_fragment_length()** all these functions are implemented using macros.

SEE ALSO

ssl(7), **SSL_CTX_set_read_ahead**(3), **SSL_pending**(3)

HISTORY

The **SSL_CTX_set_max_pipelines()**, **SSL_set_max_pipelines()**, **SSL_CTX_set_split_send_fragment()**, **SSL_set_split_send_fragment()**, **SSL_CTX_set_default_read_buffer_len()** and **SSL_set_default_read_buffer_len()** functions were added in OpenSSL 1.1.0.

The **SSL_CTX_set_tlsext_max_fragment_length()**, **SSL_set_tlsext_max_fragment_length()** and **SSL_SESSION_get_max_fragment_length()** functions were added in OpenSSL 1.1.1.

COPYRIGHT