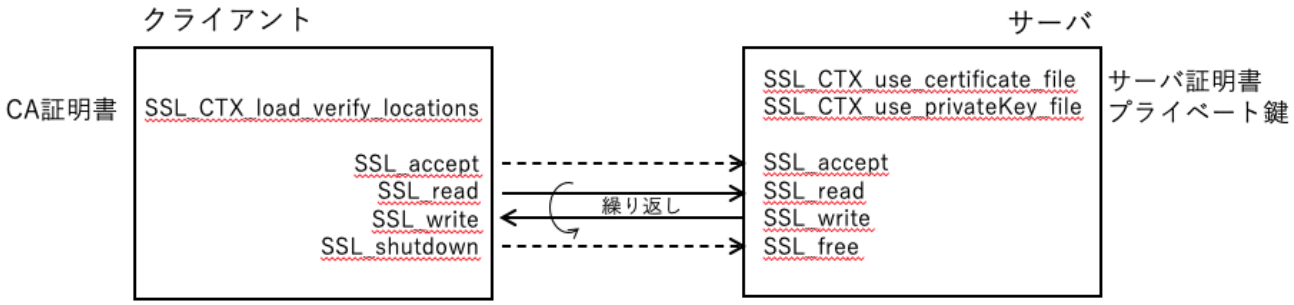
6.1 クライアント・サーバ通信

6.1.1 機能概要：

このサンプルはクライアントとサーバの間でTLS接続による簡単なアプリケーションメッセージ通信を行います。クライアントのコマンドアーギュメントで通信相手のIPアドレスを指定します。アーギュメントが無い場合はローカルホスト(127.0.0.1)に対して通信します。

クライアントはサーバとのTLS接続を確立した後、標準入力からのメッセージをサーバに送信します。サーバは受信したメッセージを標準出力に表示するとともに、所定のメッセージをクライアントに返送します。クライアントはサーバから返却されたメッセージを標準出力に表示します。クライアントは入力メッセージがある限りこれを繰り返します。サーバもクライアントからのメッセージがある限り、返信します。クライアントの入力メッセージが"shutodwn"の場合、クライアントはこれをサーバに送信した後、TLS接続を解除し処理を終了します。サーバ側も"shutdown"を受信した場合、処理を終了します。

TLS接続の際にピア認証を行います。サンプルプログラムではクライアント側がサーバ認証を行い、サーバ側はその認証要求に応えます。そのために、クライアント側にはCA証明書、サーバ側にはサーバ証明書とプライベート鍵をあらかじめ登録しておきます。



6.1.2 プログラム

1. クライアント

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

#define 定数定義

int main(int argc, char **argv)
{
    ソケット用変数, メッセージ用変数の定義

    SSL_CTX* ctx = NULL;    /* SSLコンテクスト */
    SSL*     ssl = NULL;    /* SSLオブジェクト */

    ライブラリの初期化

    /*  SSLコンテクストの確保し、CA証明書をロード  */
    if ((ctx = SSL_CTX_new(SSLv23_client_method())) == NULL)
        { エラーメッセージ出力; goto cleanup; }
    if ((ret = SSL_CTX_load_verify_locations
                        (ctx, CA_CERT_FILE, NULL)) != SSL_SUCCESS)
        { エラーメッセージ出力; goto cleanup; }

    TCPソケットの確保、サーバにTCP接続

    /* SSLオブジェクトの生成、ソケットをアタッチ、サーバにSSL接続 */
    if ((ssl = SSL_new(ctx)) == NULL)
        { エラーメッセージ出力; goto cleanup; }
    if ((ret = SSL_set_fd(ssl, sockfd)) != SSL_SUCCESS)
        { エラーメッセージ出力; goto cleanup; }
    if ((ret = SSL_connect(ssl)) != SSL_SUCCESS)
        { エラーメッセージ出力; goto cleanup; }

    /* アプリケーション層のメッセージング  */
    while (1) {
        送信メッセージを入力
        if ((ret = SSL_write(ssl, msg, sendSz)) != sendSz)
            { SSL詳細エラーメッセージ出力; break; }
        "shutdown" ならばbreak
        if ((ret = SSL_read(ssl, msg, sizeof(msg) - 1)) < 0)
            { SSL詳細エラーメッセージ出力; break;}
        受信メッセージを出力
    }
cleanup:
    リソースの解放
}
```

2. サーバ

```
#include <openssl/ssl.h>
#define 定数定義

int main(int argc, char **argv)
{
    ソケット用変数, メッセージ用変数の定義
    SSL_CTX* ctx = NULL;    /* SSLコンテクスト */
    SSL*     ssl = NULL;    /* SSLオブジェクト */

    ライブラリの初期化

    /*  SSLコンテクストの確保し、サーバ証明書、プライベート鍵をロード  */
    if ((ctx = SSL_CTX_new(SSLv23_server_method())) == NULL)
        { エラーメッセージ出力; goto cleanup; }
    if ((ret = SSL_CTX_use_certificate_file
        (ctx, SERVER_CERT_FILE, SSL_FILETYPE_PEM)) != SSL_SUCCESS) {
        { エラーメッセージ出力; goto cleanup; }
    if ((ret = SSL_CTX_use_PrivateKey_file(ctx, SERVER_KEY_FILE,
        SSL_FILETYPE_PEM)) != SSL_SUCCESS)
        { エラーメッセージ出力; goto cleanup; }

    TCPソケットの確保、bind, listen

    while(1) {
        connd = accept() /` TCP アクセプト */

        /* SSLオブジェクトの生成、ソケットをアタッチ、アクセプト */
        if ((ssl = SSL_new(ctx)) == NULL)
            { エラーメッセージ出力; goto cleanup; }
        if ((ret = SSL_set_fd(ssl, connd)) != SSL_SUCCESS)
            { エラーメッセージ出力; goto cleanup; }
        if ((ret = SSL_accept(ssl)) != SSL_SUCCESS)
            { エラーメッセージ出力; goto cleanup; }

        /* アプリケーション層のメッセージング  */
        while (1) {
            if ((ret = SSL_read(ssl, msg, sizeof(msg) - 1)) < 0)
                { SSL詳細エラーメッセージ出力; break;}

            受信メッセージを出力
            "shutdown" ならばbreak

            if ((ret = SSL_write(ssl, msg, sendSz)) != sendSz)
                { SSL詳細エラーメッセージ出力; break; }
        }
    }
cleanup:
    リソースの解放
}
```

6.1.3 プログラムの説明：

1) ヘッダーファイル

#include "openssl/ssl.h": TLSプログラムで使用するAPI、データタイプなどの定義が含まれています

2) 管理構造体とポインタ

- SSL_CTX *ctx;
  一連のTLS接続処理(コンテクスト)を管理するための構造体です。同じサーバへのTLS接続のような類似の条件での複数のTLS接続を一つのコンテクストとして管理します。

- SSL *ssl;
  １つのTLS接続を管理するための構造体です。

- 構造体の確保と解放
  - 確保：SSL_CTX_new(), SSL_new()
  - 解放：SS*_CTX_free(), SSL_free()

- 関連情報
  SSL_CTXコンテクストに紐づけられる主な情報としては以下のようなものがあります。
  - TLSバージョン:
    コンテクストの確保時、SSL_CTX_newのアーギュメントでTLS接続時のプロトコルバージョンを指定します。 (表6.1.1 SSL_CTX_new メソッド, 表6.1.2 TLSバージョン指定関連の主なAPI参照)
  - ピア認証：
    認証のためのCA証明書、自ノードの証明書、プライベート鍵などを接続前にロードしておきます(表6.1.3 ピア認証関連のAPI参照)。
  - TLS接続に使用するソケット
    SSL_set_fd関数でTLS接続に使用するソケットをSSLに紐付けます。

3) 主なAPI

- SSL_CTX_load_verify_locations
  この例では、サーバ認証のためにクライアント側でCA証明書をTLSコンテクストにロードします。クライアント認証のためにサーバ側でも使用します。(関連APIは表6.1.3 ピア認証関連のAPIを参照)

- SSL_CTX_use_certificate_file
  この例では、サーバ認証のためにサーバ側でサーバ証明書をTLSコンテクストにロードします。クライアント認証のためにクライアント側でも使用します。(関連APIは表6.1.3 ピア認証関連のAPIを参照)

- SSL_CTX_use_privateKey_file
  この例では、サーバ認証のためにサーバ側でプライベート鍵をTLSコンテクストにロードします。クライアント認証のためにクライアント側でも使用します。(関連APIは表6.1.3 ピア認証関連のAPIを参照)

- SSL_connect
  クライアントからサーバにTLS接続を要求するAPIです。サーバとのTCP接続が完了している状態で、

SSL_newで確保したSSLを指定してこのAPIで接続を要求します。TLSバージョンや暗号スイートの合意、サーバ認証などのハンドシェークを行います。すべての処理が正常に完了するとこのAPIは正常終了を返却します。

- SSL_accept
  クライアントからのTLS接続要求を受け付けるAPIです。クライアントからのTCP接続要求で接続が完了している状態で、SSL_newで確保したSSLを指定してこのAPIで接続要求を受付ます。TLSバージョンや暗号スイートの合意、必要ならばクライアント認証などのハンドシェークを行います。すべての処理が正常に完了するとこのAPIは正常終了を返却します。

- SSL_write
  接続の相手方に対して指定された長さのアプリケーションメッセージを暗号化し送信します。正常に送信が完了した場合、指定したメッセージ長と同じ値を返却します。

- SSL_read, SSL_pending
  接続の相手方から指定された最大長以下のアプリケーションメッセージを受信しバッファーに復号化し、格納します。正常に受信が完了した場合、受信したメッセージのバイト数を返却します。SSL_pendingは現在ペンディンングとなっている受信メッセージのバイト数を返却します。SSL_readではこのバイト数分のメッセージをノンブロッキングで読み出すことができます。

4) 処理の流れ

クライアント

- ライブラリ初期化
  プログラムの冒頭でSL_library_init()を呼び出しライブラリを初期化します。

- TLSコンテクストの確保
  SSL_CTX_newでコンテクストを一つ確保します。この時、接続に使用するTLSバージョンを指定します(表6.1.1 SSL_CTX_new メソッド参照)。また、サーバ認証のためのCA証明書をロードします。

- ソケット確保とTCP接続
  socket、connectによってソケットの確保とサーバとのTCP接続を要求します。

- SSLの確保とTLS接続要求
  SSL_newでSSL接続管理の構造体を確保します。SSL_set_fdでソケットをSSLに紐付けます。SSL_connectでTLS接続を要求します。

- アプリケーションメッセージ
  SSL_write、SSL_readで、アプリケーションメッセージの送信、受信を行ます。

- 切断とリソースの解放
  TLSとTCPの切断、リソースを解放します。確保したときの逆の順序で、TLS切断とSSLの解放、ソケットの解放、コンテクストの解放の順序で実行します。

サーバ

サーバ側もクライアント側とほぼ同様の処理の流れとなります。以下、クライアント側と異なる部分を説明します。

- TLSコンテクストの確保
  サーバ認証要求を受ける側となるので、サーバ証明書、プライベート鍵をロードします。

- TCP, TLS接続
  接続要求を受け付ける側となるので、acceptおよびSSL_acceptを呼び出します。

- アプリケーションメッセージ
  クライアント側と同様に、SSL_read, SSL_writeを呼び出しますが、送受信が逆順となります。

5) その他の注意点

TLSのセキュリティを確保するために、SSL_write、SSL_readで通信するメッセージは以下のような対応関係が維持されます。

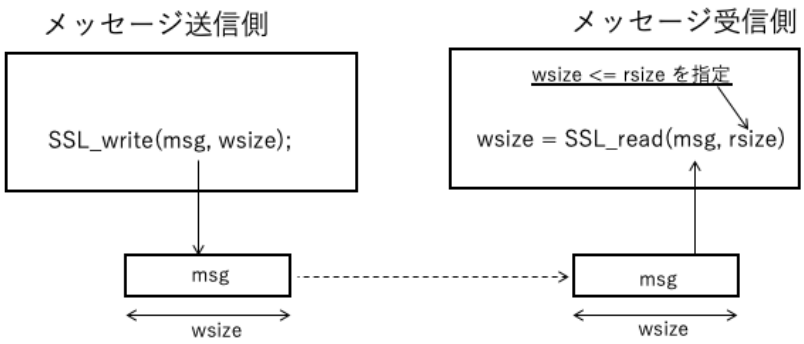
デフォルトでは、1回のSSL_write呼び出しによるメッセージは一つのTLSレコードとして送信されます。一つのTLSレコードのメッセージは１回ないし複数回のSSL_readします。SSL_readで指定するメッセージ長は送信側と同じか長い場合は１回で受信されます。送られてきたTLSレコードのサイズのほうがSSL_readで指定したメッセージサイズより長い場合は次のSSL_readで受信されます。

一方、SSL_readの指定サイズが長い場合でも、複数回のSSL_write呼び出しで送信された複数のレコードをまとめて一つのSSL_readで受信することはありません。

TLSレコードは最大16kバイトです。SSL_wirteで16kバイトを超えるメッセージを指定した場合は、メッセージを16kバイト X n のレコードと残り分のメッセージのレコードに分割して複数のレコードを送信します。これに対して、SSL_readは１回のAPI呼び出しに対して１レコードを読み込みます。したがって、メッセージのサイズとして最大レコードサイズ16kバイトを指定し、複数回APIを呼び出す必要があります。

MAX Fragmentを指定してしてTLSレコードの最大サイズに小さいサイズを指定した場合は、上記のレコードサイズもそのサイズとなります。

SSL_CTX_set_modeでSSL_MODE_ENABLE_PARTIAL_WRITEが指定されている場合は、SSL_writeは送信処理の状況によってメッセージ全体が送信できない場合、一部だけ送信しそのバイト数を返却します。



6.1.4 参照

分類	名前	説明
サーバー	SSLv23_server_method	両者のサポートする最も高いバージョンで接続
	TLSv1_3_server_method	TLS 1.3で接続
	TLSv1_2_server_method	TLS 1.2で接続
	TLSv1_1_server_method	TLS 1.1で接続
	TLSv1_server_method	TLS 1.0で接続
クライアント	SSLv23_client_method	両者のサポートする最も高いバージョンで接続
	TLSv1_3_client_method	TLS 1.3で接続
	TLSv1_2_client_method	TLS 1.2で接続
	TLSv1_1_client_method	TLS 1.1で接続
	TLSv1_client_method	TLS 1.0で接続
サーバー/クライアント	SSLv23_method	両者のサポートする最も高いバージョンで接続
	TLSv1_3_method	TLS 1.3で接続
	TLSv1_2_method	TLS 1.2で接続
	TLSv1_1_method	TLS 1.1で接続
	TLSv1_method	TLS 1.0で接続

表6.1.1 SSL_CTX_new メソッド

分類	名前	説明
設定	SSL_CTX_set_min_proto_version	使用する最も低いプロトコルバージョンを指定
	SSL_CTX_set_max_proto_version	使用する最も高いプロトコルバージョンを指定
参照	SSL_CTX_get_min_proto_version	設定済み最も低いプロトコルバージョンを参照
	SSL_CTX_get_man_proto_version	設定済み最も高いプロトコルバージョンを参照

表6.1.2 TLSバージョン指定関連の主なAPI

役割	機能	指定単位	ファイルシステムあり	ファイルシステムなし
証明する側	CA証明書のロード	コンテクスト	SSL_CTX_load_verify_locations	SSL_CTX_load_verify_buffer
証明される側	ノード証明書のロード	コンテクスト	SSL_CTX_use_certificate_file	SSL_CTX_use_certificate_buffer
		セッション	SSL_use_certificate_file	SSL_use_certificate_buffer
	プライベート鍵のロード	コンテクスト	SSL_CTX_use_privateKey_file	SSL_CTX_use_privateKey_buffer
		セッション	SSL_use_privateKey_file	SSL_use_privateKey_buffer

表6.1.3 ピア認証関連のAPI

6.2 事前共有鍵(PSK)

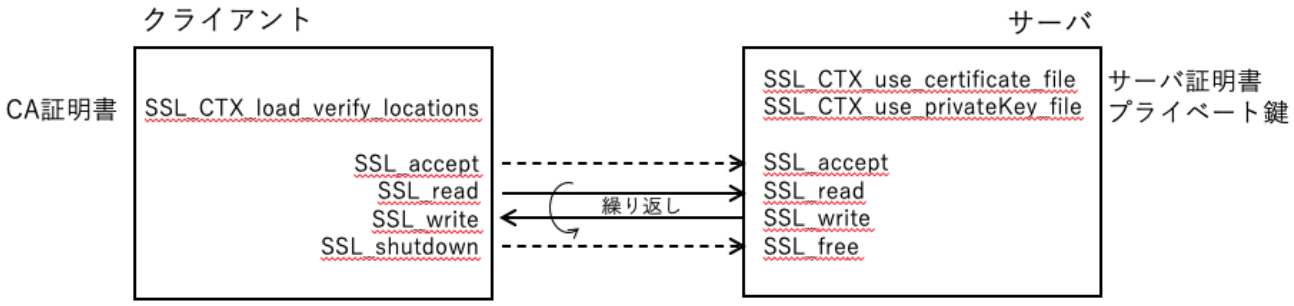
6.2.1 機能概要：

このサンプルはクライアントとサーバの間でTLS接続による簡単なアプリケーションメッセージ通信を行います。クライアントのコマンドアーギュメントで通信相手のIPアドレスを指定します。アーギュメントが無い場合はローカルホスト(127.0.0.1)に対して通信します。

クライアントはサーバとのTLS接続を確立した後、標準入力からのメッセージをサーバに送信します。サーバは受信したメッセージを標準出力に表示するとともに、所定のメッセージをクライアントに返送します。クライアントはサーバから返却されたメッセージを標準出力に表示します。クライアントは入力メッセージがある限りこれを繰り返します。サーバもクライアントからのメッセージがある限り、返信します。クライアントの入力メッセージ

が"shutodwn"の場合、クライアントはこれをサーバに送信した後、TLS接続を解除し処理を終了します。サーバ側も"shutdown"を受信した場合、処理を終了します。

TLS接続の際にピア認証を行います。サンプルプログラムではクライアント側がサーバ認証を行い、サーバ側はその認証要求に応えます。そのために、クライアント側にはCA証明書、サーバ側にはサーバ証明書とプライベート鍵をあらかじめ登録しておきます。



6.2.2 プログラム

1. クライアント

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

#define 定数定義

int main(int argc, char **argv)
{
    ソケット用変数, メッセージ用変数の定義

    SSL_CTX* ctx = NULL;    /* SSLコンテクスト */
    SSL*     ssl = NULL;    /* SSLオブジェクト */

    ライブラリの初期化

    /*  SSLコンテクストの確保し、CA証明書をロード  */
    if ((ctx = SSL_CTX_new(SSLv23_client_method())) == NULL)
        { エラーメッセージ出力; goto cleanup; }
    if ((ret = SSL_CTX_load_verify_locations
                        (ctx, CA_CERT_FILE, NULL)) != SSL_SUCCESS)
        { エラーメッセージ出力; goto cleanup; }

    TCPソケットの確保、サーバにTCP接続

    /* SSLオブジェクトの生成、ソケットをアタッチ、サーバにSSL接続 */
    if ((ssl = SSL_new(ctx)) == NULL)
        { エラーメッセージ出力; goto cleanup; }
    if ((ret = SSL_set_fd(ssl, sockfd)) != SSL_SUCCESS)
        { エラーメッセージ出力; goto cleanup; }
    if ((ret = SSL_connect(ssl)) != SSL_SUCCESS)
        { エラーメッセージ出力; goto cleanup; }

    /* アプリケーション層のメッセージング  */
    while (1) {
        送信メッセージを入力
        if ((ret = SSL_write(ssl, msg, sendSz)) != sendSz)
            { SSL詳細エラーメッセージ出力; break; }
        "shutdown" ならばbreak
        if ((ret = SSL_read(ssl, msg, sizeof(msg) - 1)) < 0)
            { SSL詳細エラーメッセージ出力; break;}
        受信メッセージを出力
    }
cleanup:
    リソースの解放
}
```

2. サーバ

```
#include <openssl/ssl.h>
#define 定数定義

int main(int argc, char **argv)
{
    ソケット用変数, メッセージ用変数の定義
    SSL_CTX* ctx = NULL;    /* SSLコンテクスト */
    SSL*     ssl = NULL;    /* SSLオブジェクト */

    ライブラリの初期化

    /*  SSLコンテクストの確保し、サーバ証明書、プライベート鍵をロード  */
    if ((ctx = SSL_CTX_new(SSLv23_server_method())) == NULL)
        { エラーメッセージ出力; goto cleanup; }
    if ((ret = SSL_CTX_use_certificate_file
        (ctx, SERVER_CERT_FILE, SSL_FILETYPE_PEM)) != SSL_SUCCESS) {
        { エラーメッセージ出力; goto cleanup; }
    if ((ret = SSL_CTX_use_PrivateKey_file(ctx, SERVER_KEY_FILE,
        SSL_FILETYPE_PEM)) != SSL_SUCCESS)
        { エラーメッセージ出力; goto cleanup; }

    TCPソケットの確保、bind, listen

    while(1) {
        connd = accept() /` TCP アクセプト */

        /* SSLオブジェクトの生成、ソケットをアタッチ、アクセプト */
        if ((ssl = SSL_new(ctx)) == NULL)
            { エラーメッセージ出力; goto cleanup; }
        if ((ret = SSL_set_fd(ssl, connd)) != SSL_SUCCESS)
            { エラーメッセージ出力; goto cleanup; }
        if ((ret = SSL_accept(ssl)) != SSL_SUCCESS)
            { エラーメッセージ出力; goto cleanup; }

        /* アプリケーション層のメッセージング  */
        while (1) {
            if ((ret = SSL_read(ssl, msg, sizeof(msg) - 1)) < 0)
                { SSL詳細エラーメッセージ出力; break;}

            受信メッセージを出力
            "shutdown" ならばbreak

            if ((ret = SSL_write(ssl, msg, sendSz)) != sendSz)
                { SSL詳細エラーメッセージ出力; break; }
        }
    }
cleanup:
    リソースの解放
}
```

6.2.3 プログラムの説明：

1) ヘッダーファイル

3) 主なAPI

4) 処理の流れ

クライアント

サーバ

サーバ側もクライアント側とほぼ同様の処理の流れとなります。以下、クライアント側と異なる部分を説明します。

- TLSコンテクストの確保
  サーバ認証要求を受ける側となるので、サーバ証明書、プライベート鍵をロードします。
- TCP, TLS接続
  接続要求を受け付ける側となるので、acceptおよびSSL_acceptを呼び出します。
- アプリケーションメッセージ
  クライアント側と同様に、SSL_read, SSL_writeを呼び出しますが、送受信が逆順となります。

5) その他の注意点

TLSのセキュリティを確保するために、SSL_write、SSL_readで通信するメッセージは以下のような対応関係が維持されます。

デフォルトでは、1回のSSL_write呼び出しによるメッセージは一つのTLSレコードとして送信されます。一つのTLSレコードのメッセージは１回ないし複数回のSSL_readします。SSL_readで指定するメッセージ長は送信側と同じか長い場合は１回で受信されます。送られてきたTLSレコードのサイズのほうがSSL_readで指定したメッセージサイズより長い場合は次のSSL_readで受信されます。

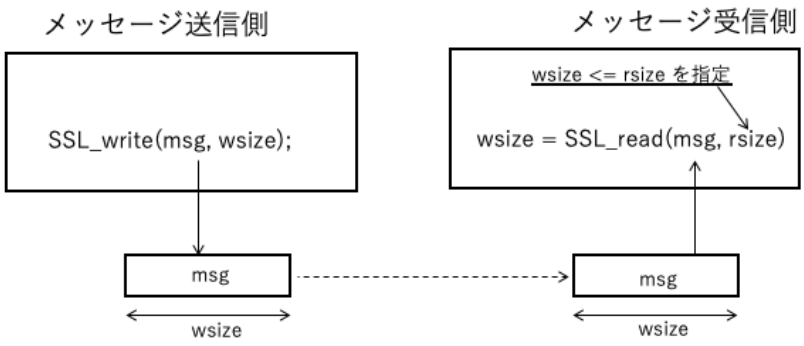
一方、SSL_readの指定サイズが長い場合でも、複数回のSSL_write呼び出しで送信された複数のレコードをまとめて一つのSSL_readで受信することはありません。

TLSレコードは最大16kバイトです。SSL_wirteで16kバイトを超えるメッセージを指定した場合は、メッセージを16kバイト X n のレコードと残り分のメッセージのレコードに分割して複数のレコードを送信します。これに対し

て、SSL_readは１回のAPI呼び出しに対して１レコードを読み込みます。したがって、メッセージのサイズとして最大レコードサイズ16kバイトを指定し、複数回APIを呼び出す必要があります。

MAX Fragmentを指定してしてTLSレコードの最大サイズに小さいサイズを指定した場合は、上記のレコードサイズもそのサイズとなります。

SSL_CTX_set_modeでSSL_MODE_ENABLE_PARTIAL_WRITEが指定されている場合は、SSL_writeは送信処理の状況によってメッセージ全体が送信できない場合、一部だけ送信しそのバイト数を返却します。



6.1.4 参照