CyaSSL SSL チュートリアル

（CyaSSL SSL Tutorial, Rel 2.0.0）

目次

組込み向け SSL ライブラリーCyaSSL は SSL や TLS で通信セキュリティを強化するために既存のアプリケーションやデバイスに簡単に組み込むことができます。CyaSSL は超小型で高い性能を提供するなど、組込みや RTOS 環境向けに特化してきました。CyaSSL の最小ビルドサイズはビルドオプションやプラットフォームによって 30-100kB の範囲です。

CyaSSL は組込み用 SSL ライブラリーですが、その全機能はデスクトップ環境でも使用することができます。通常、新たなプラットフォームにも非常に容易にコンパイルすることができます。また、オペレーティングシステム層、カスタム I/O、C 標準ライブラリーなど、いくつかの抽象化レイヤーを含んでいます。全機能のリストとサポート・プラットフォームについては、
http://yassl.com/yaSSL/Products-yassl.html を参照ください。

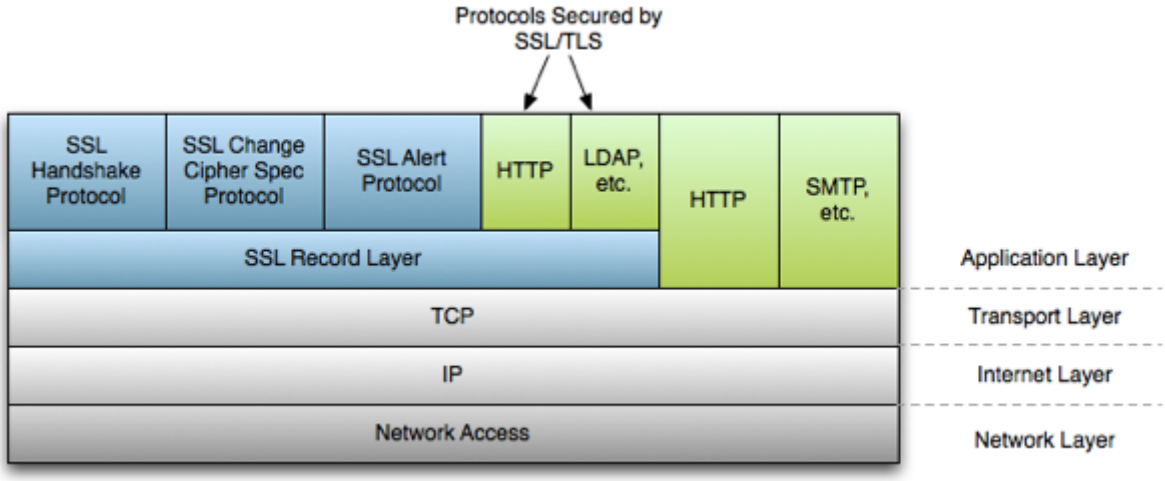
このチュートリアルでは、SSL や TLS を簡単なアプリケーションへの組込み手順を体験します。また、このチュートリアルを一通り体験することで SSL 一般への理解を深めていただくことも期待しています。ここでは、話を出来る限り簡単にするために CyaSSL と簡単な echoserver と echoclient の例を使いますが、アプリケーションに SSL サポートを追加する一般的手順をご覧いただけるものでもあります。echoserver と echoclient の例は広く知られた Richard Stevens, Bill Fenner,および Andrew Rudoff 編集の「Unix Network Programming（Vol1.3rd）」から参照させていただきました。元の例を参照したい方は以下のページに掲載されています。
echoclient – Figure 5.4, Page 124
echoserver – Figure 5.12, Page 139

このチュートリアルは、GNU GCC コンパイラーによる C コードの編集、コンパイルに慣れておられる方々、公開鍵暗号化の概念を理解しておられる方々を想定しています。なお、このチュートリアルでは「Unix Network Programming（邦題：UNIX ネットワークプログラミング）」へのアクセスは必要ではありません。

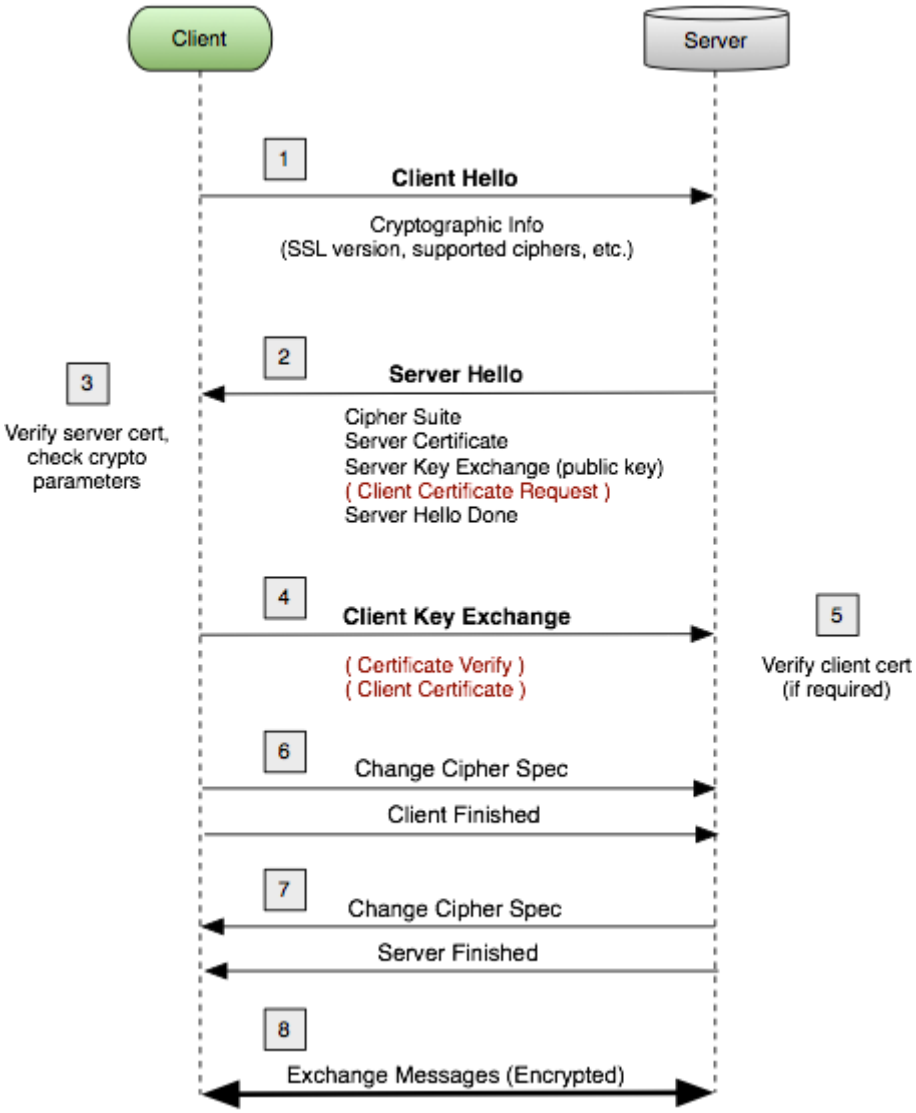
1. SSL/TLS の概要

TLS (Transport Layer Security)と SSL (Secure Sockets Layer)は主に TCP/IP における多種のトランスポート層プロトコルに対してセキュアな通信を実現する暗号化プロトコルです。SSL/TLS の最新バージョンは 1.2 です。CyaSSL は SSL3.0、TLS1.0.1.1 および 1.2 をサポートします。

SSL および TLS は OSI モデルのトランスポートとアプリケーション層の間に位置し、(TCP/IP、Bluetooth など) 多くのプロトコルがトランスポート媒体としての役割を果たします。アプリケーション・プロトコルは SSL の上位に位置し、HTTP、FTP また SMTP などを乗せることができます。SSL が OSI モデルにどのように納まるか次の図で示します。



コネクションの間、SSL または TLS はコネクション期間に使用するための暗号化と証明書のサブセットをネゴシエートします。SSL のハンドシェイクはいくつか段階を含んでおり、そのうちのいくつかは SSL クライアントとサーバ構成時に指定したオプションにより、オプショナルとなります。SSL のハンドシェーク・プロトコル・プロセスの簡単な流れを下図に示します。



SSL、TLS の経緯と詳細について詳しくはウィキペディアまたはそれぞれの RFC ドキュメントを参照してください。

ウィキペディア：http://en.wikipedia.org/wiki/Transport_Layer_Security）

SSL v3.0 http://tools.ietf.org/id/draft-ietf-tls-ssl-version3-00.txt

TLS v1.0 http://tools.ietf.org/rfc/rfc2246.txt

TLS v1.1 http://tools.ietf.org/rfc/rfc4346.txt

TLS v1.2 http://tools.ietf.org/rfc/rfc5246.txt

2. ソースコードを入手する

このチュートリアルで使用されているすべてのソースコードは yaSSL サイト（下記 URL)からダウンロードすることができます。ダウンロードには、このチュートリアルで使われている echoserver と echoclient のオリジナルと完成後のソースコードが含まれています。内容について下記に示します。

http://www.yassl.com/jp/documentation/ssl-tutorial-2.0.zip

ダウンロード ZIP ファイルには以下のものが含まれます:

CyaSSL_SSL_Tutorial.pdf
/finished_src
  /echoclient
    （完成後の echoclient コード）
  /echoserver
    （完成後の echoserver コード）
  /include
    （共通ヘッダーファイル[UNIX ネットワークプログラミングのもを修正しています]）
  /lib
    （共通ライブラリー関数）
/original_src
  /echoclient
    （オリジナルの echoclient コード）
  /echoserver
    （オリジナルの echoserver コード）
  /include
    （共通ヘッダーファイル[UNIX ネットワークプログラミングのもを修正しています]）
  /lib
    （共通ライブラリー関数）

3. もとの例題の修正

このチュートリアルとそれに関連したソースコードは、複数のプラットフォーム間でできる限りポータブルになるよう設計されています。そのためと、我々は SSL と TLS をアプリケーションに組込む方法にフォーカスしたいという理由で、もとの例題はできる限り単純なものにしてあります。無用な複雑さを省いたり、プラットフォームのサポート範囲を拡大したりするために、UNIX ネットワークプログラミングからの例にいくつかの修正が加えられています。もし、このチュートリアルのポータビリティをさらに改善できる点などありましたら support@yassl.com にご連絡ください。

echoserver と echoclient に加えた修正点は以下の通りです。

echoserver (tcpserv04.c) の修正点

- fork()関数は Windows でサポートされていなため、fork()への呼び出しを削除。これによって echoserver は同時に一つのクライアントしかアクセプトできないものになっています。これとともに、シグナル処理も削除されています。
- Str_echo()関数は str_echo.c ファイルから cpserv04.c ファイルに移動。
- コネクトされたクライアントアドレスとポートを見るために printf ステートメントの追加:

```
printf("Connection from %s, port %d¥n",
        inet_ntop(AF_INET, &cliaddr.sin_addr, buff, sizeof(buff)),
        ntohs(cliaddr.sin_port));
```

- bind の“Address already in use”エラーを回避するために、ソケットリストの生成後 setsockopt() 呼び出しを追加。

echoclient (tcpcli01.c) の修正点

- str_cli() 関数を str_cli.c から tcpcli01.c に移動。

unp.h ヘッダーの修正点

- このヘッダーは、この例で必要なものだけを含むよう簡略化しました。

これらのソースコード例内では、特定の関数の先頭文字が大文字になっている点に注意してください。例えば Fputs()、Writen()など。UNIX ネットワークプログラミングの著者は、エラーチェックを明確に取り扱うために、通常の関数に対して特別のラッパー関数を書いています。より詳細の説明については、UNIX ネットワークプログラミングのセクション 1.4 (原著:11 ページ) を参照してください。

4. CyaSSL のビルドとインストール

はじめる前に、”ソースコードを入手する” の通りに例題のコード（echoserver と echoclient）をダウンロードします。このセクションでは、組込み SSL ライブラリーCyaSSL のダウンロード、コンフィグレーションとインストール方法を説明します。

yaSSL ダウンロード・ページ から最新版の CyaSSL をダウンロード、インストールしてください。CyaSSL は、DTLS、証明書生成、OpenSSL 互換性、その他多数の機能を必要に応じて有効・無効にするビルドオプション指定して、必要なだけビルドすることができます。

ビルドオプションの詳細な説明は Building CyaSSL ガイドを参照してください。CyaSSL はポータビリティーを念頭において書かれていて、ほとんどのシステムにおいて通常簡単にビルドすることができます。もし、CyaSSL のビルドに関してお気づきの点がありましたら、遠慮なく product support forums を通じてサポートをご依頼ください。

Linux、各種 BSD、OS X、Solaris またはその他の各種*nix 系システムの上で CyaSSL をビルドする場合は、autoconf システムを利用できます。Windows 個別の説明は CyaSSL マニュアルの Building CyaSSL セクションを参照して、ターミナルから次の二つのコマンドを実行してください。必要なビルドオプションを ./configure （例えば、./configure -enable-opensslExtra のように）に追加することができます。

```
./configure
make
```

CyaSSL をインストール、実行:

```
sudo make install
```

これにより、CyaSSL のヘッダーが /usr/local/include/cyassl に、CyaSSL ライブラリーが /usr/local/lib にインストールされます。ビルドをテストするには、CyaSSL の root ディレクトリーからテストスーツを実行します:

```
./testsuite/testsuite
```

正しくインストールされたことをチェックするために、CTaoCrypt とCyaSSL に対して一連のテストが実行されます。テストスーツ・アプリケーションの実行が成功すると、概ね下記のような出力があります:

```
MD5        test passed!
MD4        test passed!
SHA        test passed!
SHA-256    test passed!
HMAC       test passed!
ARC4       test passed!
Rabbit     test passed!
DES        test passed!
DES3       test passed!
AES        test passed!
RANDOM     test passed!
RSA        test passed!
DH         test passed!
DSA        test passed!
PWDBASED test passed!
OPENSSL    test passed!
peer's cert info:
issuer :
/C=US/ST=Oregon/L=Portland/O=yaSSL/OU=programming/CN=www.yassl.com/emailAddress=i
nfo@yassl.com
subject:
/C=US/ST=Oregon/L=Portland/O=yaSSL/OU=programming/CN=www.yassl.com/emailAddress=i
nfo@yassl.com
serial number:c5:d7:6c:11:36:f0:35:e1
SSL version is TLSv1.2
SSL cipher suite is TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
peer's cert info:
```

issuer : /C=US/ST=Montana/L=Bozeman/O=sawtooth/OU=consulting/CN=www.sawtooth-consulting.com/emailAddress=info@yassl.com

subject:

/C=US/ST=Montana/L=Bozeman/O=yaSSL/OU=support/CN=www.yassl.com/emailAddress=info@yassl.com

serial number:01

SSL version is TLSv1.2

SSL cipher suite is TLS_DHE_RSA_WITH_AES_256_CBC_SHA256

Client message: hello cyassl!

Server response: I hear you fa shizzle!

sending server shutdown command: quit!

client sent quit command: shutting down!

b88596cd2362310b2506f9d73693cefd input

b88596cd2362310b2506f9d73693cefd output

All tests passed!

これで CyaSSL はインストールされました。SSL 機能を追加するために例題コードを修正することができます。

最初は echoclient への SSL の追加から始め、echoserver に進みます。

5. 初期コンパイル

ssl_tutorial.zip からの echoclient と echoserver の例題コードをコンパイル、実行するためには、一緒に含まれている Makefile を使います。echoclient または echoserver にディレクトリを移動し(cd)、実行:

```
make
```

これで例題コードのコンパイルが完了し、ビルド対象によって echoclient または echoserver という名前の実行ファイルが生成されます。Makefile で使用される GCC コマンドは以下で参照できます。提供されている Makefile をしようしないでいずれかの例題をビルドしたい場合は、例題ディレクトリ

に移動し、以下のコマンドの tcpserv04.c を例題に対して正しいソースファイルと入れ替えてください：

```
gcc -o echoserver ../lib/*.c tcpserv04.c -I ../include
```

これによって、現在の例題が実行ファイルにコンパイルされ、"echoserver" または "echoclient" アプリケーションが生成されます。コンパイル完了後どれかの例題を実行するには、実行したい例題ディレクトリに移動し、アプリケーションを起動します。例えば、echoserver を起動するには：

```
./echoserver
```

echoclient を実行する場合は、アプリケーションの起動時に IP アドレスを指定する必要があります。我々の場合、127.0.0.1 となります。カレントディレクトリーを"echoclient" ディレクトリーに移動し、次のコマンドを実行します。echoserver がすでに実行中でないといけない点に注意：

```
./echoclient 127.0.0.1
```

echoserver と echoclient が実行中になったら、echoserver は echocliet から受けとった任意の入力をエコーバックするはずです。echoserver または echoclient の実行を終了するために、[Ctrl + C] でアプリケーションを終了させます。現在、二つの例題の間で行き来するエコーデータは明白な形で送信されます。これは、多少のスキルのある人なら誰でも簡単に自分でクライントとサーバ間に挿入でき、自分の通信を見ることができるようにするためです。

6. ライブラリー

CyaSSL ライブラリーはコンパイルされると libcyassl と命名されます。また、CyaSLL がほかの指定でコンフィグレーションされない限り、CyaSSL のビルドとインストールプロセスでは、次のディレクトリー下の共有ライブラリーだけを生成する。共有、静的双方のライブラリーは適当なビルドオプションによって、イネーブルまたはディセーブルとなる：

```
/usr/local/lib
```

第一ステップとして、我々は CyaSSL ライブラリーを例題アプリケーションに組み込まなければなりません。GCC コマンドを修正し（例の echoserver を例題として使って）、次のような新しいコマンドを得ます。CyaSSL はヘッダーファイルとライブラリーを標準のロケーションにインストールするので、GCC は明白な指示（-l や -L を使った）無しにそれらを見つけることが出来なければいけない。GCC は -lcyssl を使用してコンパイラーは自動的に正しい種類のライブラリー(静的か共有)をみつけることができるはずです。

```
gcc -o echoserver ../lib/*.c tcpserv04.c -I ../include -lm -lcyassl
```

7. ヘッダー

echoclient と echoserver アプリケーションは CyaSSL ライブラリーに対してコンパイルとリンクができ、我々はアプリケーションのソースコードを修正できるようになりました。まず最初 echoclient を見てみましょう。そして、echoserver に移ります。最初にすべきことは、CyaSSL にネイティブ API ヘッダーをインクルードすることです。Tcpcli01.c ファイルを開き、先頭近くに次の行を挿入します：

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

8. 起動/終了

CyaSSL をコードの中で使えるようになる前に、ライブラリと CYASSL__CTX を初期化しなければいけません。CyaSSL は CyaSSL_init()の呼び出しで初期化されます。これを、ライブラリーに対して他の何かが行われる以前に実行しなければなりません。

CYASSL_CTX 構造体（CyaSSL コンテキスト）各 SSL コネクションへのグローバル変数を含んでおり、証明書情報を含んでいます。単一の CYASSL_CTX は生成された任意数の CYASSL で使用することができます。これを使って、例えば一回だけの信頼できる CA 証明書のリストのような、特定の情報をロードすることができます。

新しい CYASSL_CTX を生成するには、CyaSSL_CTX_new()を使用します。この関数は、クライアントが使用する SSL または TLS プロトコルを定義する引数を必要とします。CyaSSL は現在、SSLv3, TLSv1, TLSv1.1, TLSv1.2, および DTLS をサポートしています。これらのプロトコル

は CyaSSL_CTX_new().にアーギュメントとして使用できる対応する関数を持っています。使用できるプロトコルオプションを以下に示します。SSL2.0 は、近年セキュアーとはいえないので、CyaSSL ではサポートされません：

```
CyaSSLv3_client_method();       // SSL 3
CyaTLSv1_client_method();       // TLS 1
CyaTLSv1_1_client_method();     // TLS 1.1
CyaTLSv1_2_client_method();     // TLS 1.2
CyaSSLv23_client_method();      // Use highest version possible from
                                SSLv3 - TLS 1.2
CyaDTLSv1_client_method();      // DTLS
```

echoclient が echoserver にコネクトするときにサーバーのアイデンティティを証明することができるように、CYASSL_CXT に我々の CA（証明書認証局）証明書をロードする必要があります。
CYASSL_CTX に CA 証明書をロードするためには CyaSSL_CTX__load_verify_locations() を使用します。この関数は3つのアーギュメントを必要とします： CYASSL_CTX ポインター、証明書ファイル、そしてパスバリューです。パスバリューは PEM フォーマットの CA 証明書を格納しているはずのディレクトリーへのポインターです。証明書を探すとき、CyaSSL はそのパスロケーションを見る前に、証明書ファイルの値を見ます。今回の場合、パスアーギュメントとして値0を使用するというように、一つの CA ファイルを指定するので、証明書パスを指定する必要はありません。
CyaSSL_CTX_load_verify_locations 関数は SSL_SUCCESS または SSL_FAILURE を返却します：

```
CyaSSL_CTX_load_verify_locations(CYASSL_CTX* ctx, const char* file,
                                 const char* path)
```

ライブラリ初期化、プロトコル選択、および CA 証明書、これらを一緒にしたもは、以下の通りです。ここでは TLS1.0 を選択します。

```
CyaSSL_Init();// Initialize CyaSSL
CYASSL_CTX* ctx;

/* CYASSL_CTX を生成 */
```

```
if ( (ctx = CyaSSL_CTX_new(CyaTLSv1_client_method())) == NULL){
    fprintf(stderr, "CyaSSL_CTX_new error.¥n");
    exit(EXIT_FAILURE);
}

/* CA 証明書を CYASSL_CTX にロード */
if (CyaSSL_CTX_load_verify_locations(ctx,"./ca-cert.pem",0) !=
    SSL_SUCCESS) {
    fprintf(stderr, "Error loading ./ca-cert.pem, please check
            the file.¥n");
    exit(EXIT_FAILURE);
}
```

上記コードは tcpcli01.c の先頭の変数定義とユーザが IP アドレスつきでクライアントを起動完了した後に付け加えます。完全なコードは参考のために ssl_tutorial.zip に格納してあります。

これで CyaSSL と CYASSL_CTX は初期化されたので、CYASSL_CTX オブジェクトと CyaSSL ライブラリーがアプリケーションが SSL/TLS の使用を完全に完了して CyaSSL ライブラリーを開放するのを確認しましょう。

以下の2行を echoclient の main()関数の最後に置きます ― exit(0)の直前です:

```
CyaSSL_CTX_free(ctx);
CyaSSL_Cleanup();
```

9. CYASSL オブジェクト

CYASSL オブジェクトは各 TCP コネクトの後に生成する必要があります。また、ソケット・ファイルディスクリプターはセッションに対応する必要があります。echoclient の例題では Connect()関数の呼び出し後に、下記のように行います:

```
/* Connect to socket file descriptor */
```

```
Connect(sockfd, (SA *) &servaddr, sizeof(servaddr));
```

CyaSSL_new()関数を使って新しい CYASSL を生成してください。この関数は、正常時には CYASSL オブジェクトへのポインターまたは異常時には NULL を返却します。これで、ソケットファイルディスクリプター（sockfd)を新しい CYASSL オブジェクト（ssl)に関連付けることができます：

```
/* Create CYASSL object */
CYASSL* ssl;

if( (ssl = CyaSSL_new(ctx)) == NULL) {
    fprintf(stderr, "CyaSSL_new error.¥n");
    exit(EXIT_FAILURE);
}

CyaSSL_set_fd(ssl, sockfd);
```

ここで一つ注意しなければいけない点は、この例題では CyaSSL_connect() 関数の呼び出しを行っていない点です。CyaSSL_connect() はサーバーとの SSL/TLS ハンドシェイクを開始させます。もし以前に呼び出されていない場合は CyaSSL_read() の中で呼び出されます。この例題では CyaSSL_connect()を明示的に呼び出すのではなく、 CyaSSL_read()の最初の呼び出しで必要に応じて呼び出させるようにしています。

10. データの設定

次のステップは、セキュアーなデータ送信を開始することです。echoclient の例題では echoserver との間のデータ送信と受信には Writen()と Readline()関数を使用しています。これらの関数は CyaSSL の CyaSSL_write() と CyaSSL_read()関数の呼び出しに置き換える必要があります。

echoclient の例題に注目すると、main()関数は str_cli()に送信、受信の作業を任せています。 関数の書き換えを行う場所は str_cli()関数です。まず、 str_cli() 関数内の CYASSL オブジェクトにアクセスする必要があるので、アーギュメントを一つ加えて str_cli() に ssl 変数を渡します。また、

CYASSL オブジェクトは str_cli()関数の内部で使用されるので、sockfd パラメータを削除します。この修正後の新しい str_cli()関数のシグナチャーは以下の通りです:

```
void
str_cli(FILE *fp, CYASSL* ssl)
```

main()関数内で、新しいアーギュメント(ssl)は str_cli()に引き渡されます:

```
str_cli(stdin, ssl);
```

str_cli()関数内では、Writen() と Readline() は CyaSSL 関数と置き換えられ、元のファイルディスクリプター(sockfd)ではなく CYASSL オブジェクト(ssl)が使用されます。新しい str_cli()関数は以下の通りです。
CyaSSL_write と CyaSSL_read が正しく呼び出されていることを確認する必要がある点に注意しましょう。

UNIX ネットワークプログラミングの著者は Writen()関数に対するエラーチェックを書いていますが、これを置き換えた後、適切に仕上げなければなりません。 CyaSSL_read の返却値を監視するために 新しい int 変数 ”n” を追加します。また、recvline バッファの内容をプリントアウトする前に リードデータの終わりに‘¥0’で目印をつけます:

```
void
str_cli(FILE *fp, CYASSL* ssl)
{
    charsendline[MAXLINE], recvline[MAXLINE];
    int    n = 0;

    while (Fgets(sendline, MAXLINE, fp) != NULL) {

        if(CyaSSL_write(ssl, sendline, strlen(sendline)) !=
                strlen(sendline)){
```

```
            err_sys("CyaSSL_write failed");
        }

        if ((n = CyaSSL_read(ssl, recvline, MAXLINE)) <= 0)
            err_quit("CyaSSL_read error");

        recvline[n] = '¥0';
        Fputs(recvline, stdout);
    }
}
```

最後に、完了時に CYASSL オブジェクトを解放するようにします。 main()関数内の CYASSL_CTX 解放の直前で CyaSSL_free()を呼び出します:

```
str_cli(stdin, ssl);

CyaSSL_free(ssl);          // Free CYASSL object
CyaSSL_CTX_free(ctx);      // Free CYASSL_CTX object
CyaSSL_Cleanup();          // Free CyaSSL
```

11. シグナル処理

ユーザは “Ctrl+C”を使って echoclient を終了させる可能性があります。CyaSSL の資源が解放されプログラムがきちんと終了するために、このシグナルを捉える必要があります。そのために二つのことをします:

- シグナル処理関数の追加(ここでは str_cli() 関数の前に追加します)

```
    void sig_handler(const int sig)
    {
```

```
    printf("¥nSIGINT handled.¥n");
    CyaSSL_Cleanup();/* Free CyaSSL */
    exit(EXIT_SUCCESS);
}
```

- signal() を使ってこの関数をシグナルハンドラーとして登録。 echoclient の main() メソッド内の変数宣言の直後に追加します:

```
/* define a signal handler for when the user closes the
   program with Ctrl-C */

signal(SIGINT, sig_handler);
```

これで完了です。echoclient は TLSv1 化されました。CyaSSL ヘッダーをインクルードし、CyaSSL を初期化、使用したいプロトコルを選択した中で CYASSL_CTX 構造体を生成、送信、受信データに使用する CYASSL オブジェクトを生成、 Writen() と Readline() への呼び出しを CyaSSL_write() と CyaSSL_read() に置き換え、CYASSL、CYASSL_CTX、そして CyaSSL を解放、Ctrl+C シグナル処理を確認。

SSL コネクションの挙動を構成しコントロールする局面と方法はまだたくさあります。より詳細な情報については、CyaSSL の他のドキュメントやリソースを参照してください。次のセクションでは、echoserver の例題の TLSv1 化に取り組みます。

12. Echo Server

echoserver の例題での SSL/TLS 化は echoclient のステップと非常に似ています。プロトコル・バージョンの選択時(上記、起動・終了内の CYASSL_CTX 構造体生成時)にサーバメソッドを使用しなければならない点をのぞいて、これまでのステップに従ってください:

```
CyaSSLv3_server_methods();      // SSLv3
CyaTLSv1_server_method();       // TLSv1
```

```
CyaTLSv1_1_server_method();     // TLSv1.1
CyaTLSv1_2_server_method();     // TLSv1.2
CyaSSLv23_server_method();      // Allow clients to connect with
                                SSLv3 or TLSv1+
CyaDTLSv1_server_method();      // DTLS
```

CyaSSL_CTX_new() の呼び出し結果はこのような感じに:

```
/* Create and initialize SSL_CTX structure */
if ( (ctx = CyaSSL_CTX_new(CyaTLSv1_server_method())) == NULL){
    fprintf(stderr, "CyaSSL_CTX_new error.¥n");
    exit(EXIT_FAILURE);
}
```

CYASSL_CTX の証明書のロード時に、CA 証明書に加えサーバ証明書と鍵ファイルをロードしなければなりません。これによって、アイデンティフィケーション確認のためにサーバがその証明書をクライアントに送ることができるようになります:

```
if (CyaSSL_CTX_use_certificate_file(ctx,"./server-cert.pem",
        SSL_FILETYPE_PEM) != SSL_SUCCESS){
    fprintf(stderr, "Error loading ./server-cert.pem, please
        check the file.¥n");
    exit(EXIT_FAILURE);
}

if (CyaSSL_CTX_use_PrivateKey_file(ctx,"./server-key.pem",
        SSL_FILETYPE_PEM) != SSL_SUCCESS){
    fprintf(stderr, "Error loading ./server-key.pem, please check
        the file.¥n");
    exit(EXIT_FAILURE);
}
```

エコーサーバーは 読み出しと書き込みのために str_echo()を呼び出します（対するクライアントが str_cli()を呼び出すのに対して）。クライアントに対応して、 関数シグナチャーの sockfd パラメータを CYASSL オブジェクト(ssl)パラメータに置き換える修正を str_echo()に対して行います：

```
void str_echo(CYASSL* ssl)
```

read() と Writen() の呼び出しを CyaSSL_read() と CyaSSL_write() 関数への呼び出しに置き換えましょう。返却値のエラーチェックを含めて、str_echo()関数を下記のように修正します。 read() から CyaSSL_read() への変更に対処するために、変数“n”は ssize_t から int に変えられている点に注意してください：

```
void
str_echo(CYASSL* ssl)
{
    intn;
    charbuf[MAXLINE];

again:
    while ( (n = CyaSSL_read(ssl, buf, MAXLINE)) > 0) {
        if(CyaSSL_write(ssl, buf, n) != n) {
            err_sys("CyaSSL_write failed");
        }
    }

    if( n < 0 )
        printf("CyaSSL_read error = %d¥n", CyaSSL_get_error(ssl,n));

    else if( n == 0 )
        printf("The peer has closed the connection.¥n");
}
```

echoclient と同じように、ユーザが“Ctrl+C”で echoserver を閉じた時のために、シグナル処理を追加する必要があります。エコーサーバはループ内で継続して走り続けます。そのために、ユーザが“Ctrl+C”を押した時にループを脱出する方法を用意する必要があります。これをするために、まず無限ループを、出口変数（cleanup） が真にセットされた時に処理が終了する while ループに変更する必要があります。

tcpserv04.c の先頭、 #include ステートメントの直後で新しい static int 変数 cleanup を定義します。

```
static int cleanup;// To handle shutdown
```

echoserver の無限ループを while に変更します:

```
while(cleanup != 1)
{
    // echo server code here
}
```

echoserver に対して、このハンドラーが終了した後シグナルが処理される前に実行されるリスタート呼び出しをオペレーティングシステムに対して禁止する必要があります。これらを禁止することで、オペレーテリングシステムはシグナルが処理された後 accept() をリスタートしなくなります。もしこれをしないと、echoserver がリソースを解放して処理終了する前にコネクトとディスコネクトするために、別のクライアントを待たねばならなくなってしまいます。

シグナルハンドラーを定義し、SA_RESTART をオフにするために、まず echoserver の main()関数に act と oact 構造体を定義します:

```
struct sigaction      act, oact;
```

main() 関数の CyaSSL_Init() 呼び出しの前、変数宣言の後に以下のコードを挿入します:

```
/* Define a signal handler for when the user closes the program
   with Ctrl-C. Also, turn off SA_RESTART so that the OS doesn't
   restart the call to accept()after the signal is handled. */

act.sa_handler = sig_handler;
sigemptyset(&act.sa_mask);
act.sa_flags = 0;
sigaction(SIGINT, &act, &oact);
```

echoserver の sig_handler 関数は以下の通りです:

```
void sig_handler(const int sig)
{
    printf("¥nSIGINT handled.¥n");
    cleanup = 1;
    return;
}
```

繰り返しになりますが、完全なソースコードはダウンロード zip ファイルの中にあります。

13. 証明書

テストのために、CyaSSL の提供する証明書を使うことができます。CyaSSL ダウンロード内、特にこのチュートリアルのためのものは finished_src フォルダー下にあります。
製品アプリケーションのためには、認証局からの正しい正式の証明書を入手してください。

14. まとめ

このチュートリアルは CyaSSL 組み込み SSL ライブラリーを簡単なクライアントとサーバーアプリケーションに組合わせるプロセスを手順を追って説明しました。この事例は簡単ですが、同じ原理をご自分の アプリケーションに SSL や TLS を加えるのに適用することができると思います。CyaSSL

組み込みライブラリーは必要となるすべての機能をサイズと処理スピードの両面で最適化されたコンパクトで効率的なパッケージとして提供します。

GPLv2 と標準的な商用ライセンスのデュアルライセンスで、我々のサイトから直接 CyaSSL ソースコードを自由にダウンロードすることができます。質問、コメントは遠慮な support forums 宛投稿してください。製品に関してさらに詳細情報をご希望の際は info@yassl.com にコンタクトをお願いします。

このチュートリアルに関するご意見、フィードバックをお待ちしています。より有用に、わかりやすく、あるいはポータブルにするための改善、強化の提案についても support@yassl.com によろしくお願いします。