* **内容**OpenSSLを使った暗号化通信を学ぶ。
* **目標**OpenSSLを使用し、暗号化通信を用いてサーバからファイルをダウンロードする。
* **重要キーワード**・OpenSSL・SSL  
  ・TLS

ロゴ

自動的に生成された説明

* TLSプロトコル、SSLプロトコルのオープンソースで開発・提供されているソフトウェア。  
  基本的な暗号化関数と多様なユーティリティ関数を実装しており、C言語で開発されています。  
  公開されているファイルからC/C++で使えるようにするためには別途作業が必要になるが、授業ではすでにスタティックライブラリ化したものをプロジェクトに設定しています。
* **SSL** **(Secure Sockets Layer)**インターネット上の通信を暗号化して保護するためのプロトコルです。  
  SSLは、ウェブブラウザとウェブサーバ間で送受信されるデータを保護するために使用され、通信の内容が第三者に盗聴されたり改ざんされたりするのを防ぎます。
* **TLS(Transport Layer Security)**SSL同様にインターネット上の通信を暗号化して保護するためのプロトコルです。  
  SSLの脆弱性を改善しており、SSLより強力なセキュリティを保つことが出来ます。  
  昔からの慣習で、暗号化通信をSSLと呼ぶことが多いですが内部ではTLSで構成されていることがほとんどです。  
    
  SSLとTLSの主な機能  
  ① データの暗号化：送受信されるデータを暗号化し、第三者がその内容を理解できないようにします。これにより、通信内容が盗聴されるリスクが軽減されます。

② データの整合性：データが送信中に改ざんされていないことを確認します。もしデータが変更されていれば、受信側で検出されます。

③ 認証：クライアントとサーバの両方が相手の正当性を確認できるようにします。通常は、サーバがクライアントに自分の証明書を提示し、クライアントがそれを検証することでサーバの信頼性を確認します。  
  
TLSは上記に追加して下記の機能もあります。  
④ 信頼性のある接続：データが正しい順序で届き、欠落していないことを確認するためのメカニズムが組み込まれています。  
  
SSLは主にウェブブラウジングだけでしたが、TLSは「電子メール」、「インスタントメッセージング(LINE、Discord、Slackなど)」、「VoIP」などセキュアな接続を確保するための標準技術となっています。

* OpenSSLを使用したSSL/TLS通信方法  
  SSL/TLS通信のプログラムの処理の流れは下記の通り。  
  **①OpenSSLの初期化**②WinsockAPI初期化  
  ③ソケット作成  
  ④サーバに接続  
  **⑤SSLコンテキスト作成  
  ⑥SSLオブジェクト作成  
  ⑦SSLオブジェクトとソケットを関連付け  
  ⑧サーバにSSL/TLS接続  
  ⑨リクエスト送信  
  ⑩レスポンス受信  
  ⑪SSLオブジェクト開放  
  ⑫SSLコンテキスト開放**⑬ソケット終了  
  ⑭WinsockAPI終了  
    
  ②③④⑬⑭の項目は通常の通信時と同じ設定で問題ありません。  
  残りの項目にて独自の設定、通信関数を用いていきます。  
  **①OpenSSLの初期化**  
  OPENSSL\_init\_ssl関数を使用して初期化を行います。  
  関数名：OPENSSL\_init\_ssl  
  第一引数…初期化オプションを指定するためのフラグ。特にオプションが無ければ0を指定。  
  第二引数…ライブラリ全体の設定に関する構造体へのポインタです。通常、この引数はNULLを指定。

戻り値：なし

// OpenSSLの初期化

OPENSSL\_init\_ssl(0, nullptr);

**⑤SSLコンテキスト作成**OpenSSLライブラリにおいて、複数のSSL/TLS接続に共通する設定やリソースを管理するための構造体です。そのため、SSL/TLS接続を行う前に作成する必要があります。  
作成するにはSSL\_CTX\_new関数を使います。  
関数名：SSL\_CTX\_new  
第一引数：通信プロトコルを指定。OpenSSLの場合、プロトコルごとに関数が用意されており、引数で関数を実行する。  
戻り値：SSL\_CTX\*

SSL\_CTX\* ctx = SSL\_CTX\_new(TLS\_client\_method());

if (!ctx) {

throw std::runtime\_error("SSL\_CTXの生成に失敗しました");

}

今回はTLS接続のため、TLS\_client\_method関数を引数に指定。

**⑥SSLオブジェクト作成**  
実際に暗号化接続を行うためのオブジェクトを作成します。作成にはSSL\_new関数を使用します。  
関数名：SSL\_new  
第一引数：SSLコンテキストのポインタ変数  
戻り値：SSL\*

SSL\* ssl = **SSL\_new**(ctx);

if (!ssl) {

SSL\_CTX\_free(ctx);

throw std::runtime\_error("SSLの生成に失敗しました");

}

引数には⑤で作成されたSSLコンテキストを指定する。  
  
**⑦SSLオブジェクトとソケットを関連付け**作成したSSLオブジェクトに実際の通信口であるソケットを関連付けさせ、SSLオブジェクトを使用した通信の際に関連付けされたソケットを使用できるようにする必要がある。  
関連付けするにはSSL\_set\_fd関数を使用する。  
関数名：SSL\_set\_fd  
第一引数：SSLオブジェクト

第二引数：int型にキャストされたソケット  
戻り値：1のとき正常、0のとき失敗。

if (**SSL\_set\_fd**(ssl, static\_cast<int>(sock)) == 0) {

SSL\_free(ssl);

SSL\_CTX\_free(ctx);

throw std::runtime\_error("ソケットとSSLの関連付けに失敗しました");

}

**⑧サーバにSSL/TLS接続**実際にSSLオブジェクトでサーバに暗号化通信で接続を行っていきます。  
通常のソケットでconnect関数を使用して接続はしていますがこちらは暗号化通信を行っていません。そのため、改めて暗号化通信での接続を行う必要があります。  
暗号化通信で接続を行うにはSSL\_connect関数を使用していきます。  
関数名：SSL\_connect  
第一引数：SSLオブジェクト

戻り値：1のとき正常、0以下のとき失敗。

if (**SSL\_connect**(ssl) <= 0) {

SSL\_free(ssl);

SSL\_CTX\_free(ctx);

throw std::runtime\_error("SSL接続に失敗しました");

}

**⑨リクエスト送信**サーバへの接続が出来たので実際にサーバにデータのリクエストを送信していきます。  
暗号化を行わないときのリクエストは下記のようなメッセージでした。  
GET ファイルパス HTTP/1.0 \r\n\r\n  
暗号化通信を行う際にはまずHTTPのバージョンが1.1以上に変更する必要があります。  
また、1.0のときは一度ファイルダウンロードを完了するとサーバから自動的に切断命令を受信して再度サーバへの接続からやり直さないと続けてファイルダウンロードが出来ませんでした。  
バージョン1.1からはデフォルトで連続してファイルのダウンロードが出来るように接続が保持されるようになりました。  
今回の課題では連続で複数のファイルをダウンロードしませんので明示的に切断するようにリクエストに記述して送信します。  
**GET ファイルパス HTTP/1.1\r\nHost: ホスト名\r\nConnection: Close\r\n\r\n**  
最後の「\r\nConnection: Close」の部分を削除すると接続は保持されますが、別途終了するタイミングで切断リクエストを送信する必要があるので注意してください。また、このリクエストをサーバに送信する際にはsend関数を使わず、OpenSSLのSSL\_Write関数を使用します。  
関数名：SSL\_Write  
第一引数：SSLオブジェクト第二引数：リクエスト  
第三引数：リクエストの文字数

戻り値：1以上成功、0以下失敗。

// リクエスト送信

if (**SSL\_write**(ssl, request, static\_cast<int>(strlen(request))) <= 0) {

throw std::runtime\_error("送信に失敗しました");

}

**⑩レスポンス受信**サーバにリクエストを送信したらレスポンスを受信します。  
受信には送信時同様にrecv関数は使わず、OpenSSLの関数を使用します。  
関数名：SSL\_Read  
第一引数：SSLオブジェクト  
第二引数：受信用バッファ  
第三引数：受信バッファのサイズ  
戻り値：1以上の場合受信データがある。0のとき受信データなし。0未満の場合、受信エラー。

while ((size = **SSL\_read**(ssl, buf, sizeof(buf))) > 0) {

Logger::Print("%s",buf);

data.insert(data.end(), buf, buf + size);

}

if (size < 0) {

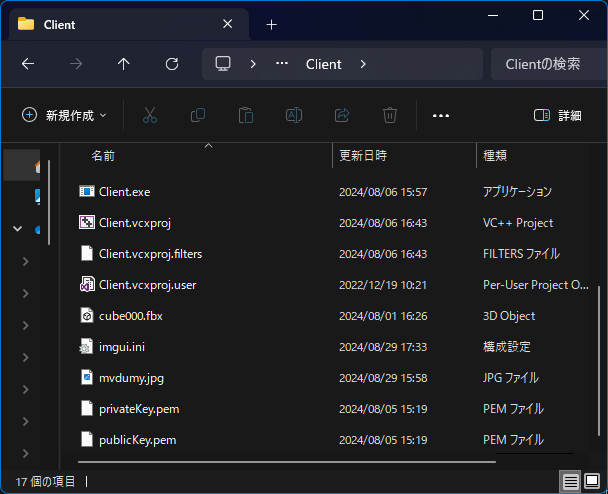
throw std::runtime\_error("受信エラー");

}

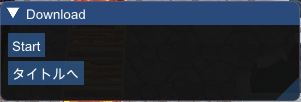
**⑪SSLオブジェクト開放**SSLオブジェクトの開放には2段階の手続きが必要となります。  
最初にSSL\_shutdown関数を用いてサーバ側にも通信処理の終了を通知。  
その後にSSL\_free関数でSSLオブジェクトが使用していたリソースを開放。  
関数名：**SSL\_shutdown**  
第一引数：SSLオブジェクト  
戻り値：1のとき、正常終了。0のときサーバからの応答待ち。-1のときエラーが発生。  
  
関数名：**SSL\_free**  
第一引数：開放したいSSLオブジェクト  
戻り値：なし  
 **⑫SSLコンテキスト開放**SSLコンテキストの開放にはSSL\_CTX\_free関数を使用します。  
  
関数名：**SSL\_CTX\_free**

第一引数：開放したいSSLコンテキスト。

戻り値：なし

* 課題  
  **Start**ボタンを押したとき、下記の接続先情報から暗号化通信を使用し、jpgファイルのダウンロードを行うようにしなさい。  
  ホスト名： comp.ecc.ac.jp  
  パス： /img/  
  ファイル名： mvdumy.jpg  
  **clientフォルダ下にダウンロードされます**。

ダウンロードされたファイル

FileDownload関数内でOpenSSLの設定を行い、ファイルダウンロードの処理を完成させること。  
※ ダウンロードしたバッファからレスポンスヘッダの削除などはすでに関数内に記載済み  
  
上記データのダウンロードが終われば接続情報を下記に変更し、fbxファイルのダウンロードも行えるか確認を行う。  
ホスト名： 10.14.10.40  
パス： /  
ファイル名： cube000.fbx

今回作成するのはClient側のみです。

図形 が含まれている画像

自動的に生成された説明

mvdumy.jpg

cube000.fbx