Secure Document Exchange System

Solution Code:

```
#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <openssl/ssl.h>

#include <openssl/bio.h>

#include <openssl/err.h>


#define SERVER_URL "https://seg-receiver.example.com:443/api/receive"

#define CERT_FILE "seg_sender.crt"

#define KEY_FILE "seg_sender.key"

#define CA_FILE "rootCA.pem"


void send_xml(const std::string& xml_file) {

  SSL_CTX* ctx = SSL_CTX_new(TLS_client_method());

  if (!ctx) {

    std::cerr << "SSL context creation failed!\n";

    exit(EXIT_FAILURE);

  }


  // Load certificates

  if (!SSL_CTX_load_verify_locations(ctx, CA_FILE, nullptr) ||
    !SSL_CTX_use_certificate_file(ctx, CERT_FILE, SSL_FILETYPE_PEM) ||
    !SSL_CTX_use_PrivateKey_file(ctx, KEY_FILE, SSL_FILETYPE_PEM)) {
    std::cerr << "SSL certificate configuration failed!\n";
    SSL_CTX_free(ctx);
    exit(EXIT_FAILURE);
}

BIO* bio = BIO_new_ssl_connect(ctx);
SSL* ssl = nullptr;

if (!bio || BIO_set_conn_hostname(bio, SERVER_URL) <= 0) {
    std::cerr << "Failed to connect to server!\n";
    BIO_free_all(bio);
    SSL_CTX_free(ctx);
    exit(EXIT_FAILURE);
}

BIO_do_connect(bio);
BIO_get_ssl(bio, &ssl);
SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY);

// Read XML file
std::ifstream file(xml_file);
if (!file.is_open()) {
    std::cerr << "Error opening XML file.\n";
    BIO_free_all(bio);
```

```
        SSL_CTX_free(ctx);
        exit(EXIT_FAILURE);
    }

    std::ostringstream buffer;
    buffer << file.rdbuf(); // Read entire file into buffer
    file.close();

    // Send XML data securely
    std::string xml_data = buffer.str();
    BIO_write(bio, xml_data.c_str(), xml_data.size());

    std::cout << "Document sent successfully!\n";

    BIO_free_all(bio);
    SSL_CTX_free(ctx);
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
        std::cerr << "Usage: " << argv[0] << " <XML-file>\n";
        return EXIT_FAILURE;
    }

    send_xml(argv[1]);
    return EXIT_SUCCESS;
```

```
}
```

Documentation part

Overview

This program securely transmits an XML document over **SSL/TLS** using **OpenSSL**. It establishes a **secure connection** with a remote server and ensures authentication via **certificates and encryption**.

Features

- **Secure Connection:** Uses TLS encryption for safe data exchange.
- **Certificate-Based Authentication:** Ensures integrity using a trusted **CA**.
- **XML File Handling:** Reads and transmits an XML document.

Dependencies

Ensure you have **OpenSSL** installed on your system to run this program.

Setup Instructions

1. Install **OpenSSL**:
    - Windows: Use **MinGW** or install OpenSSL manually.
    - Linux/macOS: Install via package manager (apt, brew, yum).
2. Configure **Certificates**:
    - seg_sender.crt – Client Certificate
    - seg_sender.key – Private Key
    - rootCA.pem – CA Certificate
3. Compile:

```
g++ secure_exchange.cpp -o secure_exchange -lssl -lcrypto
```

Code Explanation

1️⃣SSL Initialization

- Initializes an SSL context using **TLS_client_method()**.

- Loads **CA**, **certificate**, and **private key** for authentication.

2️⃣ Establishing a Secure Connection

- Uses BIO_new_ssl_connect(ctx) to create an SSL connection.
- Connects to SERVER_URL.
- Implements SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY) for reliable communication.

3️⃣ Reading and Sending XML File

- Reads the XML file into a buffer using std::ostringstream.
- Transmits data over the **SSL connection** using BIO_write.

Error Handling

The program checks:

- **SSL initialization failures** (SSL_CTX_new).
- **Certificate loading errors** (SSL_CTX_use_certificate_file).
- **Connection issues** (BIO_set_conn_hostname).
- **File errors** (std::ifstream).

Execution

Run the program as:

```
./secure_exchange document.xml
```

Where document.xml is the **XML file** to be transmitted.