

# CS165 Program Assignment

Nov 16, 2005

**ATTN: the due date of this program assignment is at 11:59pm, Dec 10, 2005**

## 1 Introduction

In this project you will implement small client and server applications based on the SSL protocol.

Your applications should allow clients to authorize themselves and download documents from the server. For simplicity, consider that the document will be **FILE1** located at the root directory of the server (you can create a text file as FILE1). The client and server applications will establish an SSL connection, the client will then authenticate itself by using a digital signature and message digest. Once the client's identity is verified, the server will generate the hash value for each block of text and send them to the client through the SSL connection.

To implement the project you will use the C/C++ programming language, the OpenSSL security library, SHA implementation and RSA implementation in the OpenSSL package. OpenSSL is a software library that provides a full-featured cryptographic toolkit as well as an implementation of SSL. The Unix version, which we will be using, provides two libraries, one for SSL/TLS (ssl) and one for the cryptographic toolkit (crypto) as well as command-line tool (openssl).

The OpenSSL Project's Web page, <http://www.openssl.org>, contains HTML versions of much of the OpenSSL documentation, as well as links to FAQs and other documents available on the Web about using OpenSSL. You can also download your own copy of OpenSSL. The RSA implementation is a part of OpenSSL package.

Since OpenSSL's documentation of parts of the Crypto library are lacking, Ariel Glenn's documentation of SSLeay (the precursor to OpenSSL) at (<http://www.columbia.edu/~ariel/ssleay/>) may be of particular help.

## 2 Project details

You can running both the client and server on one machine by using localhost. Before the Client sends a request to the server, you need to make sure ths server is already up and listening for the incoming connection. For example, you can running server in terminal 1 (5000 is the port number that server listens on):

```
terminal 1> ./server 5000
```

The client should take command-line arguments as the following:

1. the filename that you request from the server
2. the address of the secure server to connect with (e.g. servername)

3. a flag to indicate the desired output of the webpage ( "c" for clear text or "e" for enciphered mode), the following command line should request for an encrypted version of **FILE1**

```
> your_application_name --server servername --mode e FILE1
```

For example, on terminal 2, you can run the client as follows. Here (*http://localhost*) is the name of the server, and 5000 is the port number that the client needs to connect to.

```
terminal 2> ./client --server http://localhost:5000 --mode e FILE1
```

## 2.1 Client side

The client will contact server as follows:

- initiates a SSL handshake
- receives the random number from the server
- generates a summary (message digest) of this random number by using SHA-1, and then appends it to the number,
- signs both the number and hash value with its private key (stored with the client) using the RSA implementation
- sends the signed message to the server
- receives and displays the message from the server
- if the handshake is done correctly, sends a request for FILE1 to the server(either in plaintext or encrypted mode)
- waits for the server's response
- if FILE1 is encrypted, decrypts FILE1 with client's private key
- closes the connection

## 2.2 Server side

- waits for a client to initiate a SSL handshake
- sends a random number as a challenge to the client using the SSL connection
- receives a message from the client through the SSL connection (the message contains two parts: random number and its hash value)
- decodes the message using the client public key and the RSA implementation (you can assume that the server has client's public key, which is stored in a .pem file)
- verifies the integrity of the message from client:
  1. compute the summary of the first part of the message by using SHA-1

2. compare with the one client sent (which should be the second part that appended to the random number)
  - verifies that the decoded message is the same as the random number
    1. if the message from client is correct, sends a confirmation of successful handshake to the client
    2. otherwise sends a "Wrong key" message to the client
  - waits for the client's request for file transfer
  - on receiving client's request for file transfer
    1. if the client requests the file in cleartext mode, sends the file in plaintext
    2. if the client requests the file in encrypted mode
      - (a) generates a hash value for the file and then appends the hash value to the file
      - (b) encrypts the file and hash value with client's public key
      - (c) sends the encrypted message to the client
  - closes the connection

### 3 Requirements

1. You should use Subversion when working this program. Please check URL (<https://svn.cs.ucr.edu/svn/cs165/loginname>) for your own repository. Tutorials on Subversion can be found in (<http://svnbook.red-bean.com/>).
2. Both applications should:
  - display console messages after each step
  - check errors during the communication of the two parties and display appropriate message indications for the specific error identified prior, during and after the connection
3. You should use C/C++ language to implement your application, and your code should be clearly written and very well documented. Please write a README file with your code. You should turn in your code using department's turn-in system.
4. This assignment is an *\*INDIVIDUAL\** work. You might discuss the concepts and OpenSSL library with other students but sharing the code will result in you failing the assignment. We will use automated tools to check for cooperation. Instructor and TA can also have direct access to student's individual repository.