# Encrypted file server with OpenSSL

Martin Ting

University of California, Riverside
mting005@ucr.edu

## I. INTRODUCTION

Building a file server is a place where the consideration of security is very important. The time between transmission from a one place to another is the time where data transmissions are most vulnerable to third party attacks.

I have built two programs, a server and a client programs that can be run on two different hosts that are connected to the Internet. The client is able to connect to the host running the server program and either send a file or input a file to download, if it exists in the server's active directory. OpenSSL is used to secure the channel to transfer files.

## II. FUNCTIONS

This is an overview of the functions in both client and server programs.

**Server**

- acceptConnections(char* port)
- preparePort(char* port)
- performAuthentication(BIO* clientbio);
- getTransactionType(char* transType)
- recieveFile(BIO* clientbio)
- sendFile(BIO* clientbio)

**Client**

- connectToServer(char* serveraddress, port, transType, infile)
- establishSSLConnection(char* serveraddress, port, BIO* bio)
- performAuthentication(BIO* clientbio);
- verifyTransaction(char* serveraddress, port, transType, BIO* bio)
- getTransactionType(char* transType)
- recieveFile(char* serveraddress, port, transType, infile, BIO* bio)
- sendFile(char* serveraddress, port, transType, infile, BIO* bio)

## III.BUILDING & RUNNING

**Building**

A makefile is provided to build and clean the project directory for each program. To build, running the `make` command will create an executable named *client* or *server* in the same directory.

**Key Generation**

To generate keys run these commands:

```
$ openssl req -nodes -new -x509 -keyout
rsaprivatekey.pem -out cert
$ openssl rsa -in rsaprivatekey.pem
-outform PEM -pubout -out
rsapublickey.pem
```

**Running**

This is the command to run the client program:

```
./client --serverAddress=<ip>
         --port=<port-#>
         --send/recieve
     <path-to-file>
```

This is the command to run the Server program:

```
./server --port=<port-#>
```

## IV. Authentication

The server properly authenticates the client program with a Diffie Hellman exchange over a secure connection created with OpenSSL.

**Diffie Hellman**

This key exchange protocol allows the sender and receiver of a transmission to use some computation to exchange a secret. However, the Diffie Hellman key exchange is not completely secure because it does not authenticate both parties. So, it is vulnerable to a man in the middle attack. Someone who is listening in can intercept the transmission and do key exchanges with both the sender and receiver of the original channel. This would allow that person to have a key for both sender and receiver.

## V. IMPLEMENTATION

The server and client programs must follow a the Diffie Hellman key exchange protocol in order for the server to authenticate the client.

**Client Connection**

The first thing to do load the openSSL libraries and establish an openSSL connection with the Server. We do this by calling establishSSLConnection(serveraddress, port). We get a pointer to a BIO object which is an IO abstraction that openSSL uses as an identifier for the connection.

establishSSLConnection(...) opens a SSL connection by doing the following:

- ➔ creates a SSL_CTX object as a framework to establish the SSL connection.
  - ◆ We use the SSL_v23_client_method() to use the SSLv3 protocol.
- ➔ create the BIO chain by calling BIO_new_connect
- ➔ get the SSL pointer and configure SSL
  - ◆ set the callback function to null
  - ◆ set the mode to SSL_VERIFY_NONE so that we don't expect a certificate
- ➔ attempt to connect with BIO_do_connect then attempt an SSL handshake.

**Server Connection**

The server must do two things to accept a client connection with openSSL.

- ➔ Prepare the port.
- ➔ Create an accept BIO with the port passed in.
- ➔ Set up the socket and bind an address to it with BIO_do_accept()
- ➔ Call BIO_do_accept() again to wait for incoming connections

**Authentication**

We then authenticate by calling performAuthentication(...) on both client and server. This will perform the Diffie Hellman key exchange with the server. These are the steps taken by the client and server.

| Client | Server |
|---|---|
| - Generate a random number key.<br>- Load the public key into a BIO file and process it into a RSA object.<br>- Create a buffer message to be the same size as the RSA public key.<br>- Encrypt the key into the message buffer.<br>- Write the encrypted key to the server. | - Load the public /public key into a BIO file and process them into a RSA objects.<br>- Read the encrypted key from the client.<br>- Decrypt the encrypted key with the private key into a buffer.<br>- Perform a hash on the decrypted key.<br>- Encrypt the hashed key with the private key.<br>- Write the encrypted hash key to the client. |
| - Hash the unencrypted key<br>- Read the encrypted hash key from the server<br>- Decrypt the server's message.<br>- Compare the hashed key.<br>- Authenticated if hashed keys are the same. | |

**File Transfer**

After authenticating, we will verify the transaction by sending the transaction type via a back and forth communication of the word “send” or “receive.” This transaction identification and file transfer is done with the conventional BIO_write and BIO_read calls. The client sends “send”/”receive” and if the server accepts the transaction, it will write “send”/”receive” back.

Receive

If the client wants to receive a file, it will send “receive” then send the filename for the file it wants to request. Then, it will read the contents of the file into a buffer.

To handle a “receive” request, the server will accept a filename and load up the file into a buffer. It will then write that file to the client with BIO_write(...).

Send

If the client wants to send a file, it will send “Send” then send a filename for the file it want to send. Then, it will load up the file into a buffer and write it to the server.

To handle a “send” request, it will read in the filename for the file. Then, it will read in the contents of the file into a buffer from the client with BIO_read(...).

# VI. Sample execution receive

Server receive

```
mting005@well $ ./server --port=1234
================================
Welcome to CS165 Crypto Server
application
================================

Opening port 1234 for connections.
Ready for connections.
Waiting for client request.
Connection with client accepted.
Connection Established
Reading client encrypted key.Client
encrypted key read.Decrypting client's
encrypted key.
Decrypted client key: 733
Hashing decrypted client key
Hashed decrypted client key:
a��G#x�+S���'k6��1
Signing the hash with RSA key and sending
to client.
Signed hash has been sent.
Verifying transfer type from client.
Transfer type recieved: recieve
Sending back transfer type.
Transfer type verified.
getTransactionType got: recieve
returning recieve
Get filename request from client
Recieved file request for file:
./makefile
count number of characters in the file
Counted: 217
Reading file.
File read.
Sending file.
File sent.
Closing connection to client.
Server application is exiting.
GoodBye!
```

**Client receive Output**

```
mting005@well $ ./client --s=127.0.0.1
--port=1234 --recieve ./makefile
====================================
Welcome to CS165 client application
====================================
 Loading. . . . . .
server: 127.0.0.1
port: 1234
transType: recieve
file: ./makefile
Attempting to establish SSL Connection.
SSL Connection has been established!
Verifying client with RSA
============================
Message: 733
Encrypting message with random seed: 733
Message has been encrypted using RSA
public key
Sending encryptedSeed to server and
generating hash of message.
Hash of message: a��G#x�+S���'k6��
Recieving signed hash from server.
Got signed hash from server.
Recovering hash using public key.
Hash recovered from public key.
Comparing generated hash with recovered
hash.
Received correct SHA1!
Hash comparison was a success!
Sending transfer type to server to
notify: recieve
Transfer type sent.
Waiting for verify.
getTransactionType got: recieve
returning recieve
Transfer type verified: recieve | 2
getTransactionType got: recieve
returning recieve
Sending file request to server for file:
./makefile
File name sent.
Waiting for file to return
File has been recieved.
File recieved <file_contents_printed>
SSL Connection closed.
Client application is exiting.
Good Bye!
```

# VII. Sample execution send

**Server send output**

```
mting005@well $ ./server --port=1234
========================================
Welcome to CS165 Crypto Server
application
========================================

Opening port 1234 for connections.
Ready for connections.
Waiting for client request.
Connection with client accepted.
Connection Established
Reading client encrypted key.Client
encrypted key read.Decrypting client's
encrypted key.
Decrypted client key: 901
Hashing decrypted client key
��1hed decrypted client key:
������9U��NL�-�>�L
Signing the hash with RSA key and sending
to client.
Signed hash has been sent.
Verifying transfer type from client.
Transfer type recieved: send
Sending back transfer type.
Transfer type verified.
getTransactionType got: send
returning send
Getting filename from client.
Recieved filename from client.:
./makefile
Getting file from client.
File recieved <File-Contents-here>
Server application is exiting.
GoodBye!
```

**Client send output**

```
mting005@well $ ./client --s=127.0.0.1
--port=1234 --send ./makefile
========================================
Welcome to CS165 client application
========================================
 Loading. . . . . .
server: 127.0.0.1
port: 1234
transType: send
file: ./makefile
Attempting to establish SSL Connection.
SSL Connection has been established!
Verifying client with RSA
============================
Message: 901
Encrypting message with random seed: 901
Message has been encrypted using RSA public
key
Sending encryptedSeed to server and
generating hash of message.
�� h of message: ������9U��NL�-�>�L
Recieving signed hash from server.
Got signed hash from server.
Recovering hash using public key.
Hash recovered from public key.
Comparing generated hash with recovered
hash.
Received correct SHA1!
Hash comparison was a success!
Sending transfer type to server to notify:
send
Transfer type sent.
Waiting for verify.
getTransactionType got: send
returning send
Transfer type verified: send | 1
getTransactionType got: send
returning send
send filename for new file.
Sending file name to server: ./makefile
Filename sent.
count number of characters in the file
Counted: 218
Reading file.
File read.
Sending file.
File sent.
SSL Connection closed.
Client application is exiting.
Good Bye!
```

## VII. Security

This implementation for an encrypted file server accomplishes what it needs to do. However, there is a flaw in the Diffie Hellman key exchange protocol that we have discussed earlier which is that it is susceptible to a man in the middle attack.

Someone who is interested in intercepting this communication channel can mirror the server's actions to the client and the client's actions to the server. So, the client think's its connected to the server when it is really connected to the interceptor. Likewise, the server is connected to the interceptor when it thinks it is connected to the client.

One way we could remedy this vulnerability is by using certificates to verify the identity of the server and client. This would ensure that the connection has not been invaded by an interceptor.

## VII. Resources


- https://www.openssl.org/docs/
- http://h71000.www7.hp.com/doc/83final/ba554_90007/ch04s03.html