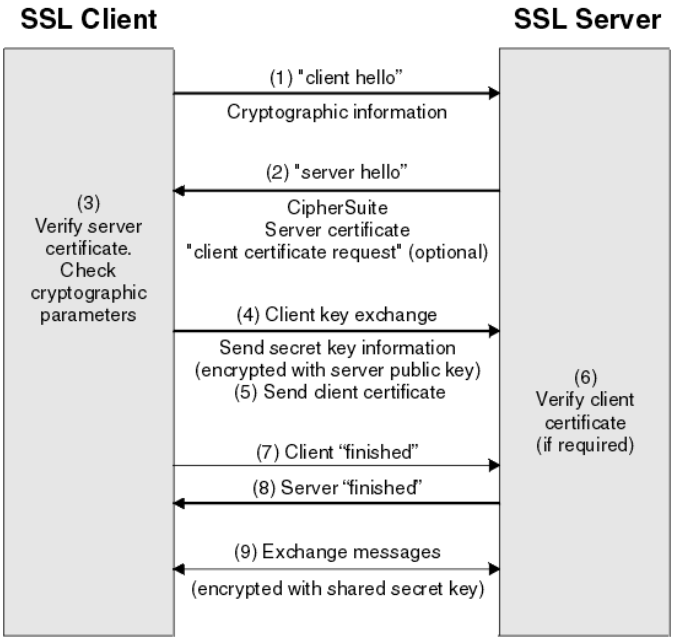
“I <Krishna Ganapathi, 860924522>, hereby certify that the present submission represents my own and original work.”

everything in this report that has a ‘-’ in front of it is an explanation from manpages or openssl intended to demonstrate the functionalities of the functions in question.

Overview:

This project was really challenging. The project required us to implement secure connections using a server and client and openssl. My code performs the basic OpenSSL when connecting according to the specifications: initializing, creating a context, creating an SSL object, creating a BIO object from a socket, and connecting the BIO and SSL objects. The connection I believe was the hardest part because there wasn’t enough well documented examples given. And the examples that were given were not explained well enough. I eventually understood how the client and the server is supposed to work after viewing a youtube video link: http://www.youtube.com/watch?v=Q8Eqby-xKKw. after watching this video I understood how the client and the server are supposed to work and that the server waits and listens for the client’s requests. I then used the OpenSSL resource to look up various functions that would be helpful such as SSL_CTX_new() to set up a new context pointer and BIO_new_ssl_connect() to make a connection.

The specification had certain steps that both the client and the server had to do. And in my naivete my first implementations first made me think that i had to complete all the client side steps first and then all the server side steps. But the video illustrated that it happens simultaneously and that the client and server are constantly interacting with each other. Another thing that helped me understand was this diagram:



Implementation details:
seeing this picture really helped a lot because I am a visual learner and this picture made my brain happy. So since then I started looking up ways on how to connect the client with the server and i tried implementing it once using the code in the video but that code was deemed by Rachid, the TA as not secure and then told me to look up functions on the OpenSSL database and cross referencing them with the references that were put on ilearn. I used these pdfs and the examples on ilearn to figure out how to setup the connection but i wasn't quite sure what the functions did so I looked them up on the man pages for OpenSSL and this is what I got.

client side:
-ERR_load_crypto_strings() registers the error strings for all **libcrypto** functions.
-SSL_load_error_strings() does the same, but also registers the **libssl** error strings.
-SSL_library_init() registers the available SSL ciphers and digests.
-OpenSSL_add_all_algorithms() adds all algorithms to the table (digests and ciphers).
-SSL_CTX_new() creates a new **SSL_CTX** object as framework to establish TLS/SSL enabled connections.
-SSL_CTX_set_verify() sets the verification flags for **ctx** to be **mode** and specifies the **verify_callback** function to be used. If no callback function shall be specified, the NULL pointer can be used for **verify_callback**. But according to the specifications we set flags to SSL_VERIFY_NONE and NULL to allow ADH to occur.
-BIO_get_ssl() retrieves the SSL pointer of BIO **b**, it can then be manipulated using the standard SSL library functions.
-BIO_new_connect() combines BIO_new() and BIO_set_conn_hostname() into a single call: that is it creates a new connect BIO with **name**.
-*BIO_do_connect()* attempts to connect the supplied BIO . It returns 1 if the connection was established successfully.
-BIO_do_handshake() attempts to complete an SSL handshake on the supplied BIO and establish the SSL connection. It returns 1 if the connection was established successfully. A zero or negative value is returned if the connection could not be established

server side:
--ERR_load_crypto_strings() registers the error strings for all **libcrypto** functions.
-SSL_load_error_strings() does the same, but also registers the **libssl** error strings.
-SSL_library_init() registers the available SSL ciphers and digests.
-OpenSSL_add_all_algorithms() adds all algorithms to the table (digests and ciphers).
-SSL_CTX_new() creates a new **SSL_CTX** object as framework to establish TLS/SSL enabled connections.
-SSL_CTX_set_verify() sets the verification flags for **ctx** to be **mode** and specifies the **verify_callback** function to be used. If no callback function shall be specified, the NULL pointer can be used for **verify_callback**. But according to the specifications we set flags to SSL_VERIFY_NONE and NULL to allow ADH to occur.
-BIO_get_ssl() retrieves the SSL pointer of BIO **b**, it can then be manipulated using the standard SSL library functions.

-BIO_new_accept() combines BIO_new() and BIO_set_accept_port() into a single call: that is it creates a new accept BIO with port **host_port**.
-*BIO_do_connect()* attempts to connect the supplied BIO . It returns 1 if the connection was established successfully.
-BIO_do_handshake() attempts to complete an SSL handshake on the supplied BIO and establish the SSL connection. It returns 1 if the connection was established successfully. A zero or negative value is returned if the connection could not be established

Once I understood this I rearranged and replaced the steps from the diagram above with the ones given in the specifications.

Client	Server
1. Establish an SSL connection with the server. ---------->	1. Wait for client connection request, and establish an SSL connection with the client. <----------
2. Seed a cryptographically secure PRNG and use it to generate a random number (challenge). 3. Encrypt the challenge using the server's RSA public key, and send the encrypted challenge to the server. ---------->	2. Receive an encrypted challenge from the client and decrypt it using the RSA private key. 3. Hash the challenge using SHA1. 4. Sign the hash. 5. Send the signed hash to the client. <----------
4. Hash the un-encrypted challenge using SHA1. 5. Receive the signed hash of the random challenge from the server, and recover the hash usingthe RSA public key. 6. Compare the generated and recovered hashes above, to verify that the server received and decrypted the challenge properly. 7. Send the server a filename (file request). ---------->	6.)Receive a filename request from the client. 7. Send the (entire) requested file back to the client. <----------
8. Receive and display the contents of the file	

requested. 9. Close the connection.	9. Close the connection.

note: read left to right and if an arrow is encountered follow the arrow all the way to the corresponding number and start reading again

I use BIO_read and BIO_write to read and or write to and or from the client or the server. According to OpenSSL:

-BIO_read() attempts to read **len** bytes from BIO **b** and places the data in **buf**.
-BIO_write() attempts to write **len** bytes from **buf** to BIO **b**.

and before I do a read or a write i use either encrypt or decrypt when necessary according to the specs. I use the encrypt and decrypt functions to encrypt and decrypt

-RSA_public_encrypt() encrypts the **flen** bytes at **from** (usually a session key) using the public key **rsa** and stores the ciphertext in **to**. **to** must point to RSA_size(**rsa**) bytes of memory. RSA_public_encrypt() returns the size of the encrypted data (i.e., RSA_size(**rsa**)). RSA_private_decrypt() returns the size of the recovered plaintext.

-*RSA_private_decrypt()* decrypts the **flen** bytes at **from** using the private key **rsa** and stores the plaintext in **to**. **to** must point to a memory section large enough to hold the decrypted data (which is smaller than RSA_size(**rsa**)). **padding** is the padding mode that was used to encrypt the data.

for the public and private keys for the public encrypt and private decrypt functions I used the same thing that was shown in lab8. I created a BIO pointer and read in the file using BIO_new_file() and then used the PEM_read_bio_RSAPrivateKey() or PEM_read_bio_RSA_PUBKEY to capture the private key or public key.

-*BIO_new_file()* creates a new file BIO with mode **mode** the meaning of **mode** is the same as the stdio function *fopen()*.
-The PEM functions read or write structures in PEM format. In this sense PEM format is simply base64 encoded data surrounded by header lines.The **RSAPrivateKey** functions process an RSA private key using an RSA structure. It handles the same formats as the **PrivateKey** functions but an error occurs if the private key is not RSA. The **RSA_PUBKEY** functions also process an RSA public key using an RSA structure. However the public key is encoded using a SubjectPublicKeyInfo structure and an error occurs if the public key is not RSA.

when I hashed the contents during the hash portion i used SHA1 as requested in the

specifications we were also exposed to this in lab8.

-SHA-1 (Secure Hash Algorithm) is a cryptographic hash function with a 160 bit output. SHA1() computes the SHA-1 message digest of the **n** bytes at **d** and places it in **md** (which must have space for SHA_DIGEST_LENGTH == 20 bytes of output). If **md** is NULL, the digest is placed in a static array.

I hash on both the server side and the client side this is to compare if the hashes are the same. I hash the unencrypted random number on the client side and receive the hash from the client and then decrypt it. when I compare them i use the strncmp function to compare the 2 strings if they are equal I send the server a filename the server then looks for that file exits with a message if the file is not found on the server. If it is found i send it back to the client on a BIO_write and do a BIO_read on the client end. It does a String compare and only sends the request if the hashes are the same. I then use an output filestream to output it to a text file and use a system call to display the text file.

after this i just use BIO_flush and BIO_free_all to close the connection on both the client and the server side.

sources:
https://www.openssl.org/docs(various links)

http://www.youtube.com/watch?v=Q8Eqby-xKKw

https://ilearn.ucr.edu/bbcswebdav/pid-2011575-dt-content-rid-10185969_1/courses/CS_165_001_13F/ssl-intro-part1.pdf

https://ilearn.ucr.edu/bbcswebdav/pid-2011575-dt-content-rid-10185970_1/courses/CS_165_001_13F/ssl-intro-part2.pdf

I talked with Alex barke and WIlly yong to clear up questions that I had on high level ideas.

I also used office hours and talked to Rachid Ounit(T.A) to clarify doubts about logic.