The following tests have commands that you can use to verify functionality of your server. **Client coming soon.**

Legend Test id in orange Commands to Run in green Questions to verify after you run your command in yellow

Assignment 2 - Server Verification

[TEST # 1]

- To check if server supports SSLv2
  - $> openssl s_client -connect localhost:8765 -cert alice.pem -ssl2;
  - [type some stuff + press enter]
  - Check s_client
    - Is 42 returned by the server?
  - Check server
    - Does your message appear on the server?

[TEST # 2]

- To check if server supports SSLv3
  - $> openssl s_client -connect localhost:8765 -cert alice.pem -ssl3;
  - [type some stuff + press enter]
  - Check s_client
    - Is 42 returned by the server?
  - Check server
    - Does your message appear on the server?

[TEST # 3]

- To check if server supports TLSv1
  - $> openssl s_client -connect localhost:8765 -cert alice.pem -tls1;
  - [type some stuff + press enter]
  - Check s_client
    - Is 42 returned by the server?
  - Check server
    - Does your message appear on the server?

[TEST # 4]

- To check if the client certificate is not signed by the proper CA
  - We need to generate a cert for the CA and a fake x509 certificate for client Alice [Ref :

https://redhatlinuxtutorial.blogspot.com/2012/08/creating-certificate-authorities-and.html]

- Generate CA RSA Pair for CA and create root x509 certificate
  - $>openssl genrsa -des3 -out fakeECE568ca.key 4096
    - Enter "password" as passphrase
  - $>openssl req -new -x509 -days 365 -key fakeECE568ca.key -out fakeECE568ca.crt
    - Enter "password" as passphrase
    - C=CA
    - ST=Ontario
    - L=Toronto
    - O=University of Toronto
    - OU=ECE568
    - CN=**FAKE**_ECE568 Certificate Authority
    - emailAddress=ece568ca@ecf.utoronto.ca
- Generate RSA Pair for FakeAlice and Sign the csr with fakeECE568ca.crt
  - $> openssl genrsa -des3 -out FakeAlice.key 4096
    - Enter "password" as passphrase
  - $> openssl req -new -key FakeAlice.key -out FakeAlice.csr
    - Enter "password" as passphrase
    - C=CA
    - ST=Ontario
    - L=Toronto
    - O=University of Toronto
    - OU=ECE568
    - CN=Alice's Client
    - emailAddress=ece568alice@ecf.utoronto.ca
    - Challenge Password : password
- Sign FakeAlice with FakeRootCA
  - $> openssl x509 -req -days 365 -in FakeAlice.csr -CA fakeECE568ca.crt -CAkey fakeECE568ca.key -set_serial 01 -out FakeAlice.crt
    - Enter "password" as passphrase
- Generate FakeAlice.pem
  - $> openssl x509 -in FakeAlice.crt -text > FakeAlice.pem
  - $> cat FakeAlice.key >> FakeAlice.pem
    - Appends private key to the end of pem to generate a similar structure to alice.pem
- Use FakeAlice.pem to start SSL handshakes
  - $> openssl s_client -connect localhost:8765 -cert FakeAlice.pem -ssl2;
  - $> openssl s_client -connect localhost:8765 -cert FakeAlice.pem -ssl3;
  - $> openssl s_client -connect localhost:8765 -cert FakeAlice.pem -tls1;
    - Does it say **"SSL ACCEPT ERROR no certificate returned"** on the server side?

[TEST # 5]

- To check if the client does not present a certificate
  - $> openssl s_client -connect localhost:8765 -ssl2;
  - $> openssl s_client -connect localhost:8765 -ssl3;
  - $> openssl s_client -connect localhost:8765 -tls1;
  - Check server
    - Does the server have an **SSL accept error**? Does it either say "**peer error no certificate**" (for ssl2) or "**peer did not return a certificate**" (for ssl3 or tls1)?

[TEST # 6][TEST IN PROGRESS]

- To check if server responds to incomplete client shutdown
  - Temporarily(Or use ifdef) put an "exit(0)" right after SSL_Connect, SSL_Write on client.c
  - $> make
  - $> ./server
  - $> ./client

Assignment 2 - Client Verification

Important Note : If you try to use s_server, it may not work with -accept <port>, if that's the case, run it without the accept keyword and run your client as follows:

./client localhost 4433

[TEST # 7] [TEST IN PROGRESS]

- Check if client only uses SSLv3 or TLSv1 (always test with SHA which corresponds to SHA1)
  - Start a server that listens on port 8765 using bob's pem
    - SSL3>openssl s_server -accept 8765 -cert bob.pem -ssl3 -cipher SHA1;
    - TLS1> openssl s_server -accept 8765 -cert bob.pem -tls1 -cipher SHA1;
  - Go run the client
    - $> ./client
  - Go on s_server
    - Type "42" and press enter
  - Go back to the client
    - Does the client print "**What's the question? 42**"?
  - Also make sure to check that SSLv2 fails (follow the above steps except start the server with this command instead)
    - openssl s_server -accept 8765 -cert bob.pem -ssl2;
      - Go back to the client :
        - Does the client print "**ECE568-CLIENT: SSL connect error"**?

[TEST # 8]

- C:heck if the client only uses the SHA1 hash function.
    - Start a server that uses SHA256 hash digest (other digests : md2, md4, md5, rmd160)
        - openssl s_server -accept 8765 -cert bob.pem -ssl3 -cipher SHA256;
        - openssl s_server -accept 8765 -cert bob.pem -tls1 -cipher SHA256;
        - $> ./client
            - Does the client print "**ECE568-CLIENT:** sslv3 alert handshake failure"?

[TEST # 9]

- Check if the client can verify that the server's certificate has a valid signature from the ECE568 CA. [Need to generate a fake bob.pem]
- Generate RSA Pair for FakeBob. Sign the csr with fakeECE568ca.crt from [TEST # 4]
    - $> openssl genrsa -des3 -out FakeBob.key 4096
        - Enter "password" as passphrase
    - $> openssl req -new -key FakeBob.key -out FakeBob.csr
        - Enter "password" as passphrase
        - C=CA
        - ST=Ontario
        - LN=Toronto
        - O=University of Toronto
        - OU=ECE568
        - CN=Fake Bob's Server
        - emailAddress=ece568fakebob@ecf.utoronto.ca
        - challenge password [] : password
- Sign FakeBob with FakeRootCA
    - $> openssl x509 -req -days 365 -in FakeBob.csr -CA fakeECE568ca.crt -CAkey fakeECE568ca.key -set_serial 01 -out FakeBob.crt
        - Enter "password" as passphrase
- Generate FakeBob.pem
    - $> openssl x509 -in FakeBob.crt -text > FakeBob.pem
    - $> cat FakeBob.key >> FakeBob.pem
        - Appends private key to the end of pem to generate a similar structure to alice.pem
- Use FakeBob.pem to start SSL handshakes
    - $> openssl s_server -accept 8765 -cert FakeBob.pem -ssl3 -cipher SHA1;
    - $> openssl s_server -accept 8765 -cert FakeBob.pem -tls1 -cipher SHA1;
        - Does the client print "**ECE568-CLIENT: Certificate does not verify**"?
            - This certificate does not verify because it's using another root CA's private key to sign the signature rather than 568ca's pem

[TEST # 10]

- Check if the client only communicates with Bob's Server by checking that the Common Name (CN) and/or the EMAIL [Need to generate a fake bob.pem. Use the same one from [TEST # 9]].
    - **Temporarily Comment out/Disable the line in your client which verifies the x509 certificate**
        - **SSL_get_verify_result (Make sure to add your code back before you submit !)**
    - Then run the s_server with FakeBob.pem
        - openssl s_server -accept 8765 -cert FakeBob.pem -ssl3 -cipher SHA1;
        - openssl s_server -accept 8765 -cert FakeBob.pem -tls1 -cipher SHA1;
            - Does the client print "**ECE568-CLIENT: Server Common Name doesn't match**" or "**ECE568-CLIENT: Server Email doesn't match**"?
- Alternatively, just use the correct .pem (the ones provided), but change the CN and/or email that you checked in the code (e.g. In your client.c, check with "Fake Bob's Server" instead of "Bob's Server", run the client etc. and see if it prints "**ECE568-CLIENT: Server Common Name** ")

[TEST # 11]

- Check if the client can report errors after detecting incorrect server shutdown of the SSL connection
    - Run openssl s_server -accept 8765 -cert bob.pem -ssl3 -cipher SHA1;
    - Run ./client
    - Press Ctrl+C on the server terminal
    - Client side should say 'ECE568-CLIENT: Premature close'