**Exercise 1 SSL/TLS Client and Server example**

In this exercise we will build a Java Socket program where a socket client establishes an SSL connection with a socket server and then sends a word to the server. The server reverses the word and sends it back to the client. The program needs a set of keys which are generated in Java using the keytool and also needs a keyStore (on the server side) and a trustStore (on the client side)

The main difference between trustStore vs keyStore is that trustStore (as name suggest) is used to store certificates from trusted Certificate authorities (CA) which are used to verify certificate presented by Server in SSL Connection while keyStore is used to store private key and own identity certificate which program should present to other parties (Server or client) to verify its identity.

Anytime a client will connect to the server, server will present its certificate stored in KeyStore and client will verify that certificate by comparing with certificates stored on its trustStore.  
  
Read more: <http://www.java67.com/2012/12/difference-between-truststore-vs.html#ixzz5dhkEUAQf>

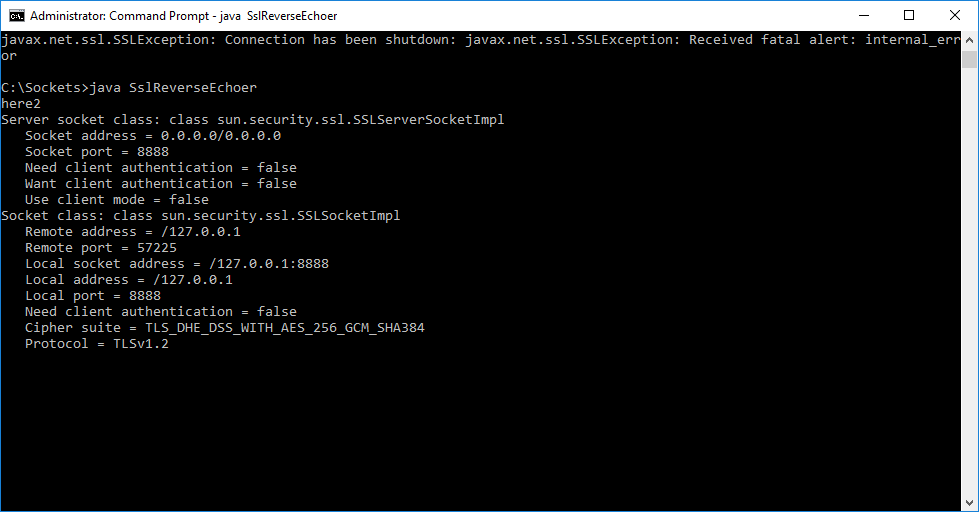
1. Create a new folder called ssl
2. Create a public/private key pair and store them in a keystore as follows:

keytool -genkeypair -v -keystore herong.jks -alias JKS

Note the keystore file is called herong.jks

Note you must give details and choose a password for the keystore

1. Copy the SslReverseEchoer.java file to your folder from the X drive
2. Add your keystore password to the file where it says password
3. Compile the file and run it. Review the code



1. Create a folder in the ssl folder called client
2. Copy and paste SslSocketClient.java to the client folder and compile it. Review the SslSocketClient code.
3. You must make a trust store on the client to hold the keys. To create the trust store we will takes the keys from the server and put them in a certificate. We will then use this certificate to create a trust store on the client.
4. To get the certificate, in the server folder run

keytool -exportcert -alias JKS -file my\_home.crt -keystore herong.jks -storepass password

Certificate stored in file <my\_home.crt>

A certificate contains the public part of the key.

The certificate, in addition to containing the public part of the key, contains additional information such as issuer, what the certificate is supposed to be used for, and other types of metadata.

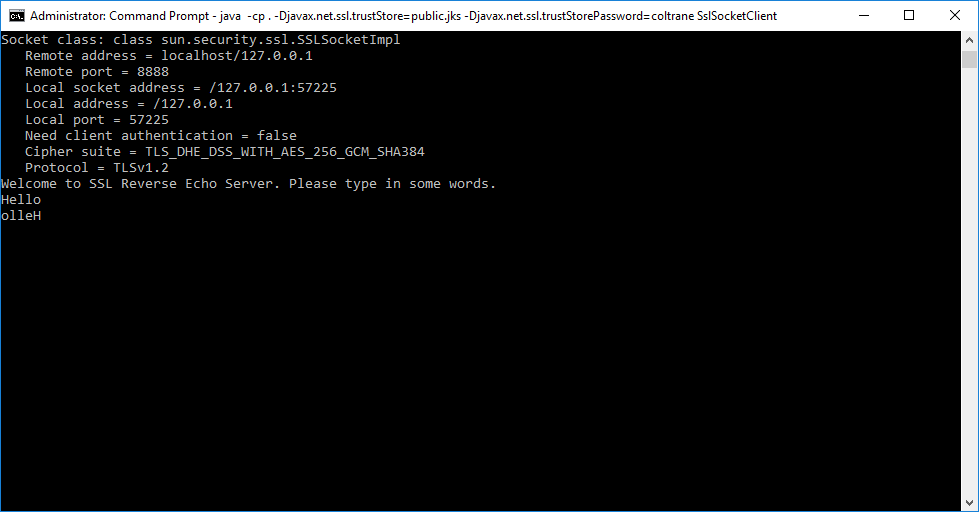
Typically, a certificate is itself signed with a private key that verifies its authenticity.

1. Now copy the certificate (my\_home.crt) to the client folder and in the client folder create a store and put the certificate keys in the store

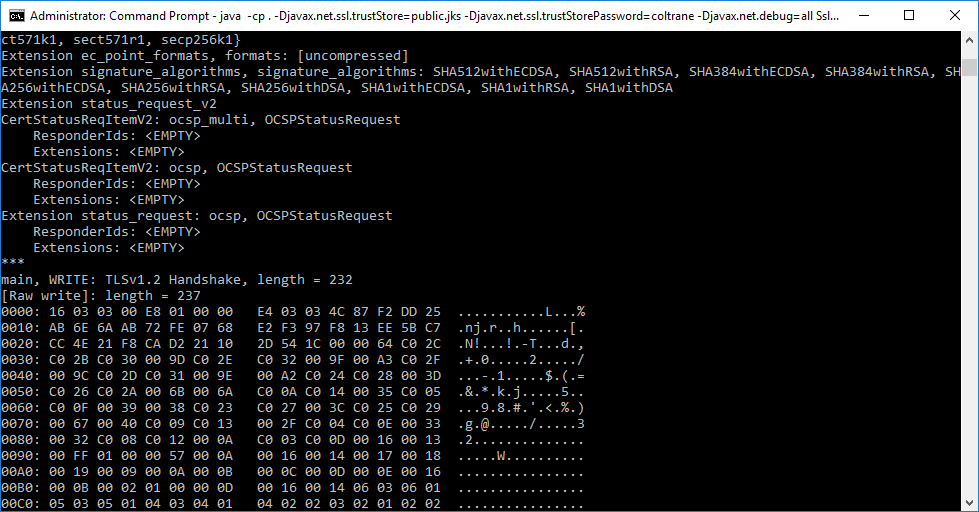
>keytool.exe -importcert -alias herong\_home -file my\_home.crt -keystore public.jks -storepass password

10. Now run SslReverseEchoer.java again (if it stopped). Then run SslSocketClient with public.jks as a trustStore.

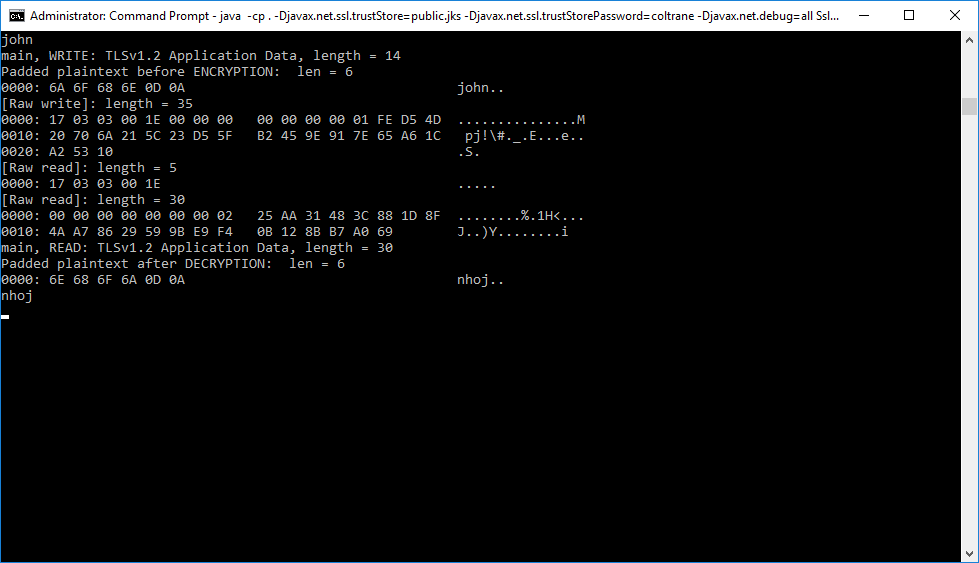
>java -cp . -Djavax.net.ssl.trustStore=public.jks -Djavax.net.ssl.trustStorePassword=password SslSocketClient



1. You can view the SSL handshake as described in the webcast with **-Djavax.net.debug=all**



And note the data is encrypted (I send the word “john”)



**Exercise 2 Sockets and HTTPS (do with week 3-4 Client Server)**

See: <https://blog.anvileight.com/posts/simple-python-http-server/>

In this example we build a very simple web server using Python. It needs certificates which we create using openssl.

We then run a Java client and communicate with the server using HTTPS that is HTTP over SSL

Again we need a trusted store on the client where it can put the keys it gets from the server. The client does a SSL handshake with the server before downloading the web page (in this case there is no GET method supported on the web server so it sends an error code instead but you can view the error code returned as html. Yo could put in a GET method later – see the code at the end of the lab.

**A Python Web Server**

Create a folder for the web server.

On the X drive, copy and paste simple code for a Python web server with SSL – “sslexample.py”

We need a public/private key pair stored in a certificate.

Download openssl (<https://sourceforge.net/projects/openssl/>) and run it to build certificate and keys with the command

openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 365 -out certificate.pem

See: [**https://www.ibm.com/support/knowledgecenter/en/SSWHYP\_4.0.0/com.ibm.apimgmt.cmc.doc/task\_apionprem\_gernerate\_self\_signed\_openSSL.html**](https://www.ibm.com/support/knowledgecenter/en/SSWHYP_4.0.0/com.ibm.apimgmt.cmc.doc/task_apionprem_gernerate_self_signed_openSSL.html)

Place the certificate.pem and key.pem files in the same folder as the sslexample.py file

Run the Python Web Server:

python sslexample.py (note it is listening on port 4443)

**Sending HTTP messages via sockets**

Now the server is running we need a client

Create a folder for the client

Download SSLSocketClient.java from the X drive (not SslSocketClient!!). Review the code.

We need a trusted store on client side

Can build one by

1. Copying the file InstallCert.java and compile and run it as follows

Java InstallCert localhost:4443

This makes a trusted store called jssecacerts

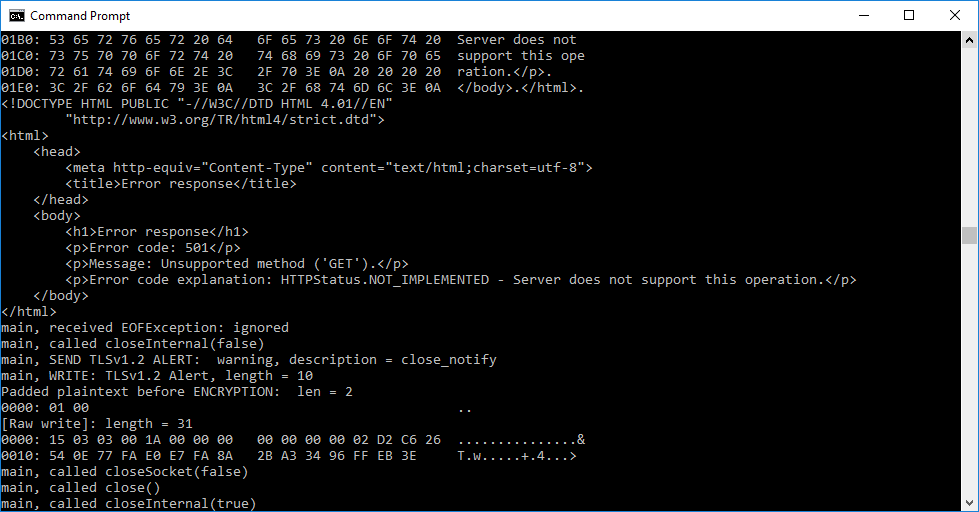
(Change the code in the server to 127.0.0.1:4443 if its not already changed)

Compile and run the client using

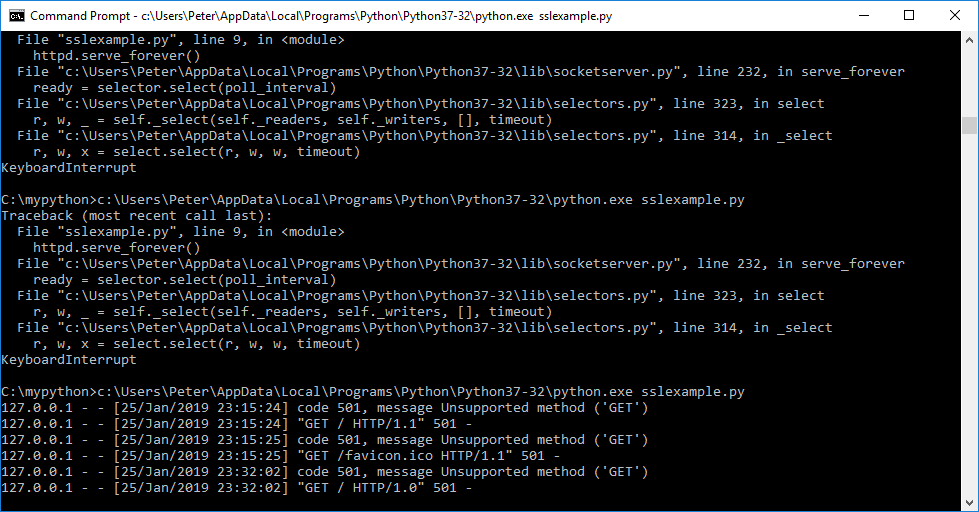
java -cp . –Djavax.net.ssl.trustStore=jssecacerts –Djavax.net.debug=all)

Watch as it negotiates a secure socket for HTTPS with the server

Client Output



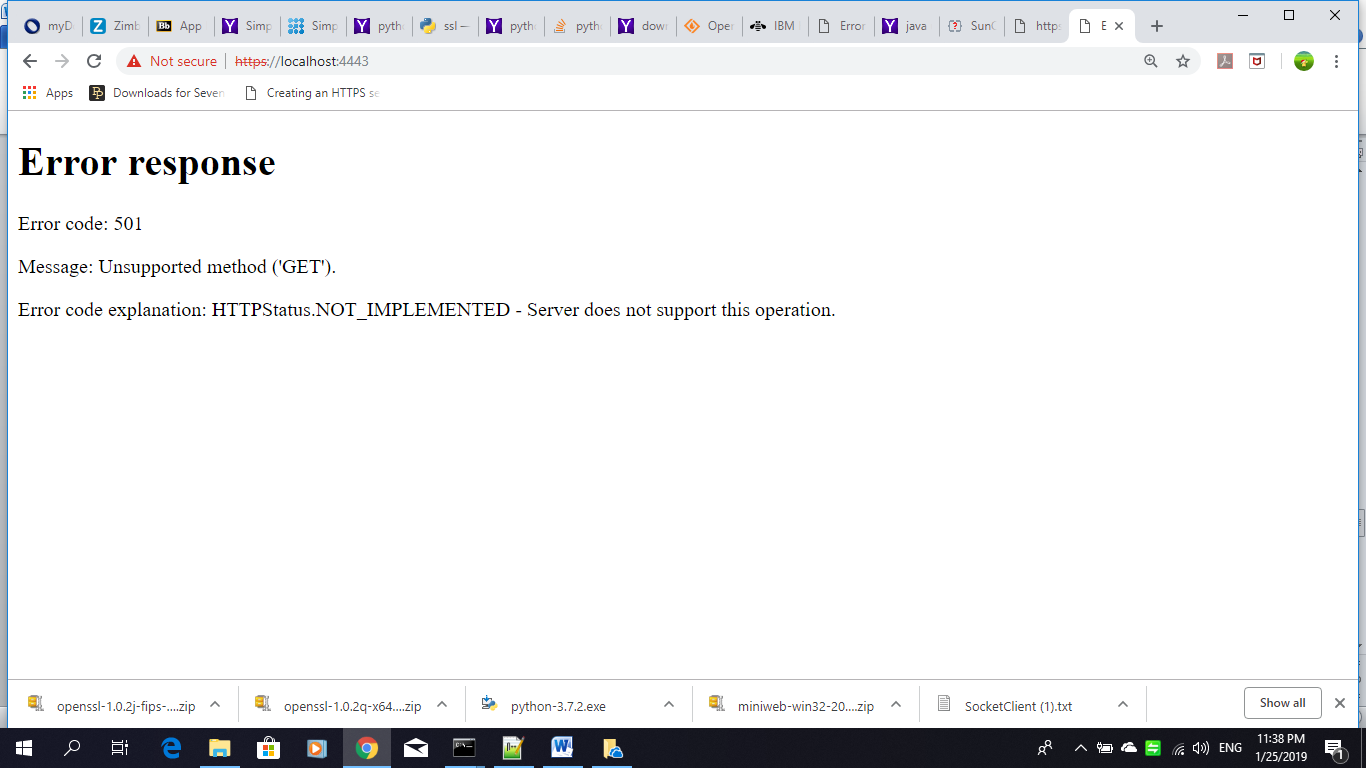
Server output



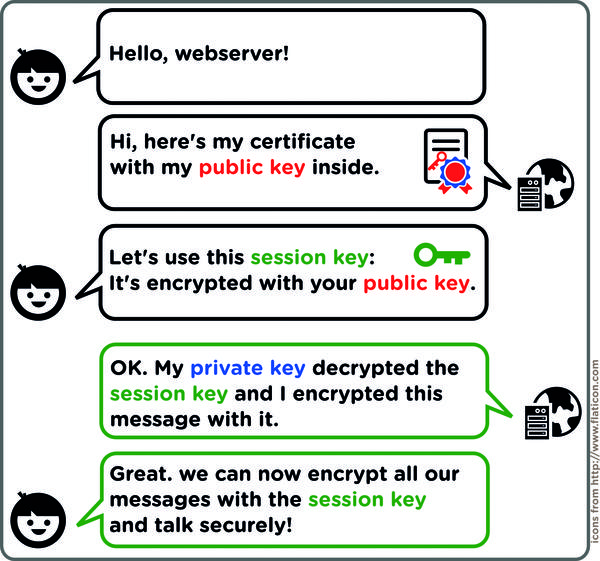
Open the same address in a browser

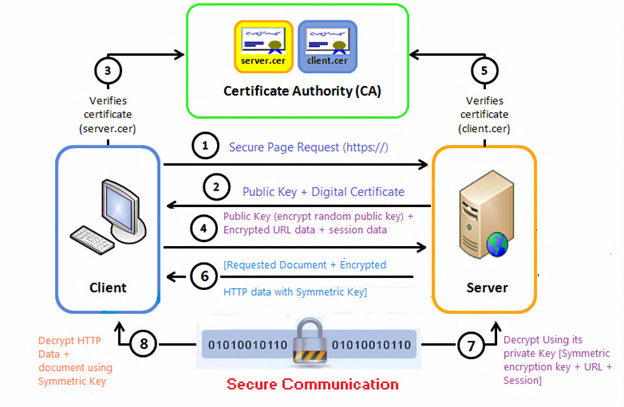
Try <http://localhost:4443>

Try <https://localhost:4443>



**SSL Handshake**





Python GET

from http.server import HTTPServer, BaseHTTPRequestHandler

class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):

def do\_GET(self):

self.send\_response(200)

self.end\_headers()

self.wfile.write(b'Hello, world!')

httpd = HTTPServer(('localhost', 8000), SimpleHTTPRequestHandler)

httpd.serve\_forever()