For testing purposes it is convenient to use a self-signed certificate. Follow [these instructions](https://devcenter.heroku.com/articles/ssl-certificate-self). You will be prompted for a password a few times:

openssl genrsa -des3 -out server.orig.key 2048

openssl rsa -in server.orig.key -out server.key

openssl req -new -key server.key -out server.csr

openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt

Here is **client.py**, slightly modified from the [Python 2.7.3 docs](http://docs.python.org/2/library/ssl.html):

import socket, ssl, pprint

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Require a certificate from the server. We used a self-signed certificate

# so here ca\_certs must be the server certificate itself.

ssl\_sock = ssl.wrap\_socket(s,

ca\_certs="server.crt",

cert\_reqs=ssl.CERT\_REQUIRED)

ssl\_sock.connect(('localhost', 10023))

print repr(ssl\_sock.getpeername())

print ssl\_sock.cipher()

print pprint.pformat(ssl\_sock.getpeercert())

ssl\_sock.write("boo!")

if False: # from the Python 2.7.3 docs

# Set a simple HTTP request -- use httplib in actual code.

ssl\_sock.write("""GET / HTTP/1.0\r

Host: www.verisign.com\n\n""")

# Read a chunk of data. Will not necessarily

# read all the data returned by the server.

data = ssl\_sock.read()

# note that closing the SSLSocket will also close the underlying socket

ssl\_sock.close()

And here is **server.py**:

import socket, ssl

bindsocket = socket.socket()

bindsocket.bind(('', 10023))

bindsocket.listen(5)

def do\_something(connstream, data):

print "do\_something:", data

return False

def deal\_with\_client(connstream):

data = connstream.read()

while data:

if not do\_something(connstream, data):

break

data = connstream.read()

while True:

newsocket, fromaddr = bindsocket.accept()

connstream = ssl.wrap\_socket(newsocket,

server\_side=True,

certfile="server.crt",

keyfile="server.key")

try:

deal\_with\_client(connstream)

finally:

connstream.shutdown(socket.SHUT\_RDWR)

connstream.close()

Note: if you try to use the standard system ca certificates, e.g. on Debian:

ssl\_sock = ssl.wrap\_socket(s,

ca\_certs="/etc/ssl/certs/ca-certificates.crt",

cert\_reqs=ssl.CERT\_REQUIRED)

then server.py explodes with:

Traceback (most recent call last):

File "server.py", line 24, in <module>

ssl\_version=ssl.PROTOCOL\_TLSv1)

File "/usr/lib/python2.6/ssl.py", line 338, in wrap\_socket

suppress\_ragged\_eofs=suppress\_ragged\_eofs)

File "/usr/lib/python2.6/ssl.py", line 120, in \_\_init\_\_

self.do\_handshake()

File "/usr/lib/python2.6/ssl.py", line 279, in do\_handshake

self.\_sslobj.do\_handshake()

ssl.SSLError: [Errno 1] \_ssl.c:490: error:1408F10B:SSL routines:SSL3\_GET\_RECORD:wrong version number

If you specify the SSL version, e.g.

connstream = ssl.wrap\_socket(newsocket,

server\_side=True,

certfile="server.crt",

keyfile="server.key",

ssl\_version=ssl.PROTOCOL\_TLSv1)

then you can run into other problems, e.g.

Traceback (most recent call last):

File "server.py", line 27, in <module>

ssl\_version=ssl.PROTOCOL\_TLSv1)

File "/usr/lib64/python2.6/ssl.py", line 338, in wrap\_socket

suppress\_ragged\_eofs=suppress\_ragged\_eofs)

File "/usr/lib64/python2.6/ssl.py", line 120, in \_\_init\_\_

self.do\_handshake()

File "/usr/lib64/python2.6/ssl.py", line 279, in do\_handshake

self.\_sslobj.do\_handshake()

ssl.SSLError: [Errno 1] \_ssl.c:490: error:1408F10B:SSL routines:SSL3\_GET\_RECORD:wrong version number