

Keiran Berry

Networking

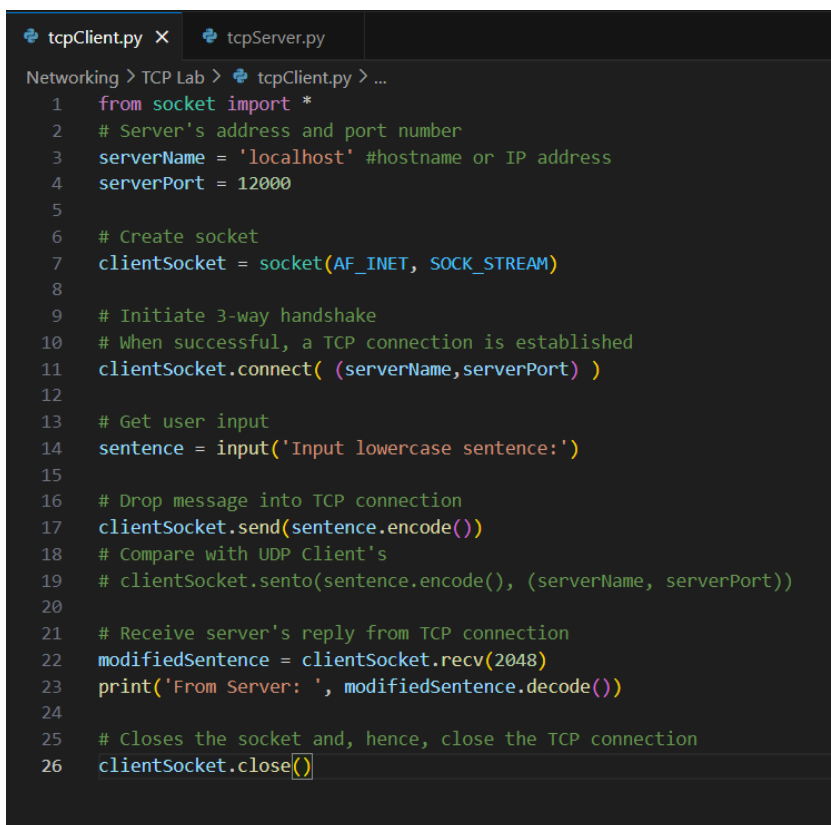
Lab 4: TCP/TLS Socket Programming

23 September 2024

## Task 1

Writing client and server code from the teaching slides

Client Code:



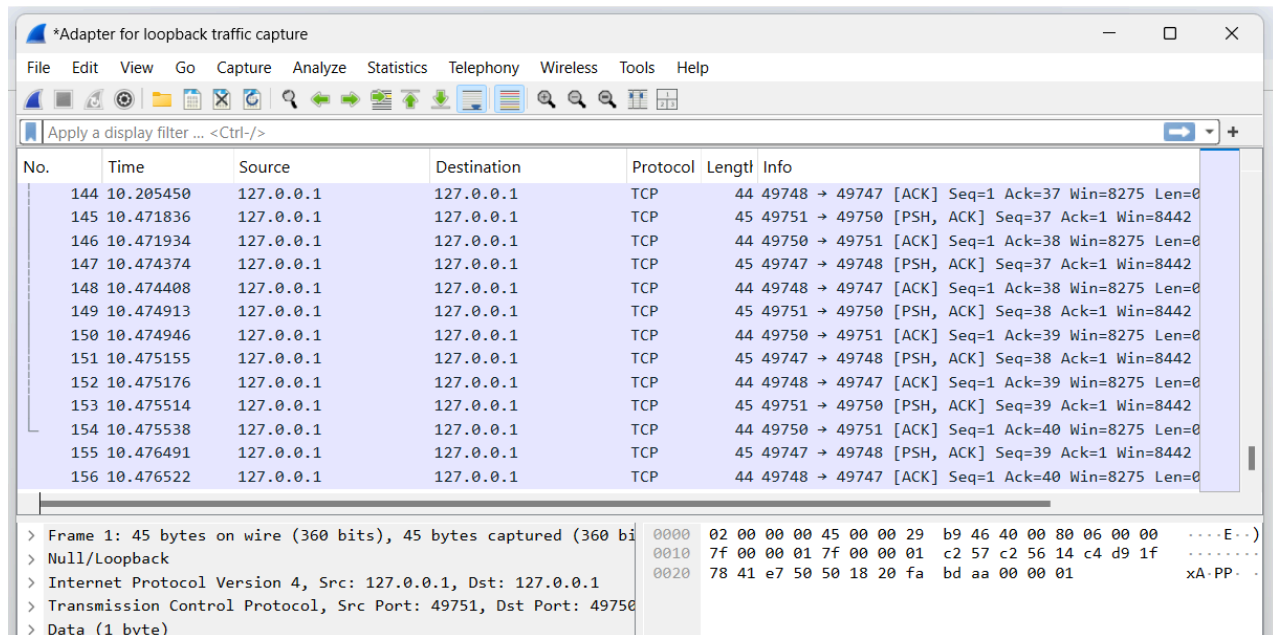
```
tcpClient.py X tcpServer.py
Networking > TCP Lab > tcpClient.py > ...
1  from socket import *
2  # Server's address and port number
3  serverName = 'localhost' #hostname or IP address
4  serverPort = 12000
5
6  # Create socket
7  clientSocket = socket(AF_INET, SOCK_STREAM)
8
9  # Initiate 3-way handshake
10 # When successful, a TCP connection is established
11 clientSocket.connect( (serverName,serverPort) )
12
13 # Get user input
14 sentence = input('Input lowercase sentence:')
15
16 # Drop message into TCP connection
17 clientSocket.send(sentence.encode())
18 # Compare with UDP Client's
19 # clientSocket.sendto(sentence.encode(), (serverName, serverPort))
20
21 # Receive server's reply from TCP connection
22 modifiedSentence = clientSocket.recv(2048)
23 print('From Server: ', modifiedSentence.decode())
24
25 # Closes the socket and, hence, close the TCP connection
26 clientSocket.close()
```

## Server Code:

```
tcpClient.py  tcpServer.py X
Networking > TCP Lab > tcpServer.py > ...
1  from socket import *
2
3  # Set port number
4  serverPort = 12000
5
6  # Create "welcoming" socket
7  serverSocket = socket(AF_INET, SOCK_STREAM)
8
9  # Explicitly assign port number to socket
10 serverSocket.bind(('',serverPort))
11
12 # Wait and listen for some client to knock
13 serverSocket.listen(1)
14
15 print('The server is ready to receive...')
16
17 while True:
18     # Welcome and create new dedicated socket
19     connectionSocket, addr = serverSocket.accept()
20
21     # Receive client's message from TCP connection
22     sentence = connectionSocket.recv(2048)
23
24     capitalizedSentence = sentence.decode().upper()
25
26     # Send reply
27     connectionSocket.send(capitalizedSentence.encode())
28
29     connectionSocket.close()
```

## Task 2

### 1. Getting to the capturing packets portion of Wireshark:



### 2. Running TCP server and client code (server first):

```
PS C:\Users\101080740\Documents> cd '.\Networking\TCP Lab\'
PS C:\Users\101080740\Documents\Networking\TCP Lab> python tcpClient.py
Input lowercase sentence:hello server!
From Server: HELLO SERVER!
PS C:\Users\101080740\Documents\Networking\TCP Lab> python tcpServer.py
The server is ready to receive...
PS C:\Users\101080740\Documents\Networking\TCP Lab>
PS C:\Users\101080740\Documents\Networking\TCP Lab> cd '.\Networking\TCP Lab\'
PS C:\Users\101080740\Documents\Networking\TCP Lab> python tcpClient.py
Input lowercase sentence:hello server!
From Server: HELLO SERVER!
PS C:\Users\101080740\Documents\Networking\TCP Lab>
```

Here, I make sure to not only run the server and client but complete a run of the program from the client side so that we can see the full “TCP handshake”.

### 3. Applying filter to wireshark to only show TCP packets:

*Adapter for loopback traffic capture						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
tcp						
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	45	49751 → 49750 [PSH, ACK] Seq=1 Ack=1 Win=8442 Len=0
2	0.000066	127.0.0.1	127.0.0.1	TCP	44	49750 → 49751 [ACK] Seq=1 Ack=2 Win=8275 Len=0
3	0.001463	127.0.0.1	127.0.0.1	TCP	45	49747 → 49748 [PSH, ACK] Seq=1 Ack=1 Win=8442 Len=0
4	0.001495	127.0.0.1	127.0.0.1	TCP	44	49748 → 49747 [ACK] Seq=1 Ack=2 Win=8275 Len=0
5	0.093351	127.0.0.1	127.0.0.1	TCP	45	49747 → 49748 [PSH, ACK] Seq=2 Ack=1 Win=8442 Len=0
6	0.093400	127.0.0.1	127.0.0.1	TCP	44	49748 → 49747 [ACK] Seq=1 Ack=3 Win=8275 Len=0
7	0.093801	127.0.0.1	127.0.0.1	TCP	45	49751 → 49750 [PSH, ACK] Seq=2 Ack=1 Win=8442 Len=0
8	0.093832	127.0.0.1	127.0.0.1	TCP	44	49750 → 49751 [ACK] Seq=1 Ack=3 Win=8275 Len=0
9	0.267139	127.0.0.1	127.0.0.1	TCP	45	49751 → 49750 [PSH, ACK] Seq=3 Ack=1 Win=8442 Len=0
10	0.267201	127.0.0.1	127.0.0.1	TCP	44	49750 → 49751 [ACK] Seq=1 Ack=4 Win=8275 Len=0
11	0.268005	127.0.0.1	127.0.0.1	TCP	45	49747 → 49748 [PSH, ACK] Seq=3 Ack=1 Win=8442 Len=0
12	0.268035	127.0.0.1	127.0.0.1	TCP	44	49748 → 49747 [ACK] Seq=1 Ack=4 Win=8275 Len=0
13	0.268400	127.0.0.1	127.0.0.1	TCP	45	49751 → 49750 [PSH, ACK] Seq=4 Ack=1 Win=8442 Len=0

> Frame 1: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface 0  
 > Null/Loopback  
 > Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Here, I can observe the “TCP handshake” mentioned in class. Since there are lots of packets being picked up by Wireshark, they are not all clumped together. However, they are all here and can be seen a bit spread out.

112	4.119015	127.0.0.1	127.0.0.1	TCP	44	49748 → 49747 [ACK] Seq=1 Ack=29 Win=8260 Len=0
113	4.605080	127.0.0.1	127.0.0.1	TCP	56	50965 → 12000 [SYN] Seq=0 Win=65535 Len=0 MSS=65535
114	4.605119	127.0.0.1	127.0.0.1	TCP	56	12000 → 50965 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
115	4.605155	127.0.0.1	127.0.0.1	TCP	44	50965 → 12000 [ACK] Seq=1 Ack=1 Win=327424 Len=0
191	8.106803	127.0.0.1	127.0.0.1	TCP	44	50965 → 12000 [ACK] Seq=14 Ack=14 Win=327424 Len=0
192	8.106860	127.0.0.1	127.0.0.1	TCP	44	12000 → 50965 [FIN, ACK] Seq=14 Ack=14 Win=2161152 Len=0
193	8.106882	127.0.0.1	127.0.0.1	TCP	44	50965 → 12000 [ACK] Seq=14 Ack=15 Win=327424 Len=0
194	8.107062	127.0.0.1	127.0.0.1	TCP	44	50965 → 12000 [FIN, ACK] Seq=14 Ack=15 Win=327424 Len=0
195	8.107143	127.0.0.1	127.0.0.1	TCP	44	12000 → 50965 [ACK] Seq=15 Ack=15 Win=2161152 Len=0

#### 4. Following TCP stream from first packet of exchange:

Wireshark · Follow TCP Stream (tcp.stream eq 2) · Adapter for loopback traffic capture

hello server!  
HELLO SERVER!

1 client pkt, 1 server pkt, 1 turn.

Entire conversation (26 bytes) Show as ASCII No delta times Stream 2

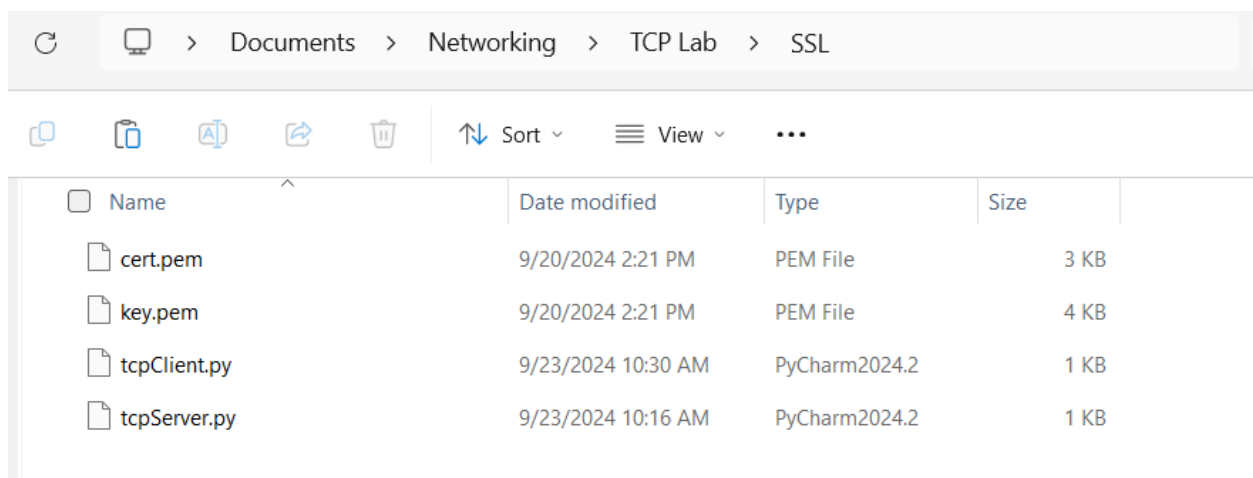
Find:  ☐ Case sensitive Find Next

Filter Out This Stream Print Save as... Back Close Help

This is really interesting as Wireshark can not only see the packets, but also decode and display the encoded messages being sent back and forth. As we can see in the running server and client step, these messages are both the one that I sent to the server and the one which the server returned to me. This is because there is no security on these messages, so if this were my password being put in then it could easily be viewed by malicious parties.

### Task 3: Adding Encryption to the TCP Server and Client

1. Downloading and extracting the key and certificate, and copying the client and server files into the folder:



2. Updating tcpServer.py using the code from the lab document:

```
tcpServer.py X
Networking > TCP Lab > SSL > tcpServer.py > ...
1  from socket import *
2  import ssl
3  serverPort = 12345
4
5  serverSocket = socket(AF_INET, SOCK_STREAM)
6  serverSocket.bind(('localhost', serverPort))
7  serverSocket.listen(1)
8
9  # Wrap the socket with SSL
10 context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
11 context.load_cert_chain(certfile="cert.pem", keyfile="key.pem")
12 serverSocket = context.wrap_socket(serverSocket, server_side=True)
13
14 print("Server listening on port 12345...")
15
16 while True:
17     connectionSocket, addr = serverSocket.accept()
18     print(f"Connection from {addr}")
19
20     # Receive data from the client
21     sentence = connectionSocket.recv(2048)
22     print(f"Received: { sentence.decode()}")
23
24     capitalizedSentence = sentence.decode().upper()
25
26     # Send a response
27     connectionSocket.send(capitalizedSentence.encode())
28
29     # Close the connection
30     connectionSocket.close()
```

- a. In the code snippet from lines 10-12:
- i. An SSLContext object is created, specifically for a server and using TLS protocol.
  - ii. The certificate and private key are loaded into the previously created SSLContext object, using the files from the same directory.
  - iii. The socket is wrapped using the SSLContext object which was created earlier. We also specify once again that this is the server side here.
3. Test run of server:

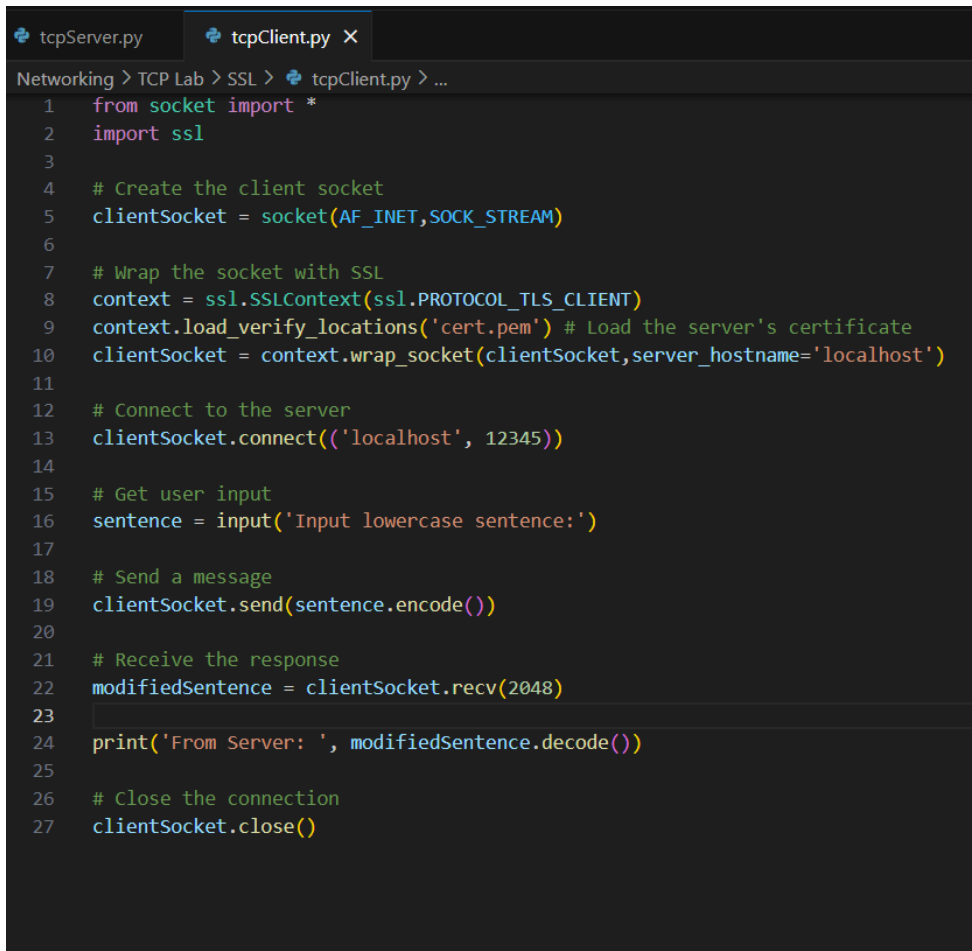
```
PS C:\Users\101080740\Documents\Networking\TCP Lab\SSL> python tcpServer.py
Server listening on port 12345...
```

As of now, we do not yet have the client edited to match what the server is listening for.

However, the server does run as intended and listens for the incoming encrypted connections.

## Part B: Modify the Client for Encryption

1. Modifying client code to use SSL with code from lab document:



```
1  from socket import *
2  import ssl
3
4  # Create the client socket
5  clientSocket = socket(AF_INET, SOCK_STREAM)
6
7  # Wrap the socket with SSL
8  context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
9  context.load_verify_locations('cert.pem') # Load the server's certificate
10 clientSocket = context.wrap_socket(clientSocket, server_hostname='localhost')
11
12 # Connect to the server
13 clientSocket.connect(('localhost', 12345))
14
15 # Get user input
16 sentence = input('Input lowercase sentence:')
17
18 # Send a message
19 clientSocket.send(sentence.encode())
20
21 # Receive the response
22 modifiedSentence = clientSocket.recv(2048)
23
24 print('From Server: ', modifiedSentence.decode())
25
26 # Close the connection
27 clientSocket.close()
```

- a. In the code from lines 8-10:

- i. An instance of SSLContext is created for the client side, using TLS protocol.
- ii. The server's certificate is loaded into the context, so that the client can confirm that it is connecting to the correct server and not a potentially harmful one.
- iii. Finally, the socket is wrapped to enable the encrypted communication with the server. An argument to specify the name of the server which will be connected to is also passed in.

## 2. Test run of client:

```
PS C:\Users\101080740\Documents\Networking\TCP Lab\SSL> python tcpClient
.py
Input lowercase sentence:hello encrypted server!
From Server:  HELLO ENCRYPTED SERVER!
PS C:\Users\101080740\Documents\Networking\TCP Lab\SSL> █
```

When running the client side of the updated code, we can see that we still get the same capitalized version of the text which we send to the server.

Now that the client can successfully connect to the server, here is the server's side of this interaction:

```
PS C:\Users\101080740\Documents\Networking\TCP Lab\SSL> python tcpServer
.py
Server listening on port 12345...
Connection from ('127.0.0.1', 51640)
Received: hello encrypted server!
█
```

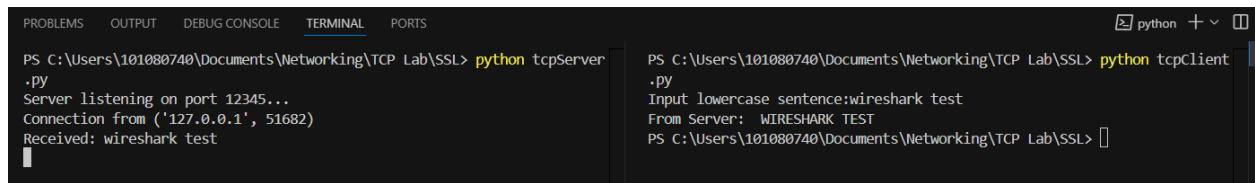


Here, we can verify that the client is able to securely connect with the server and exchange messages.

## Part C: Validation

Verifying encryption using Wireshark:

1. Sending a message to the server from the client



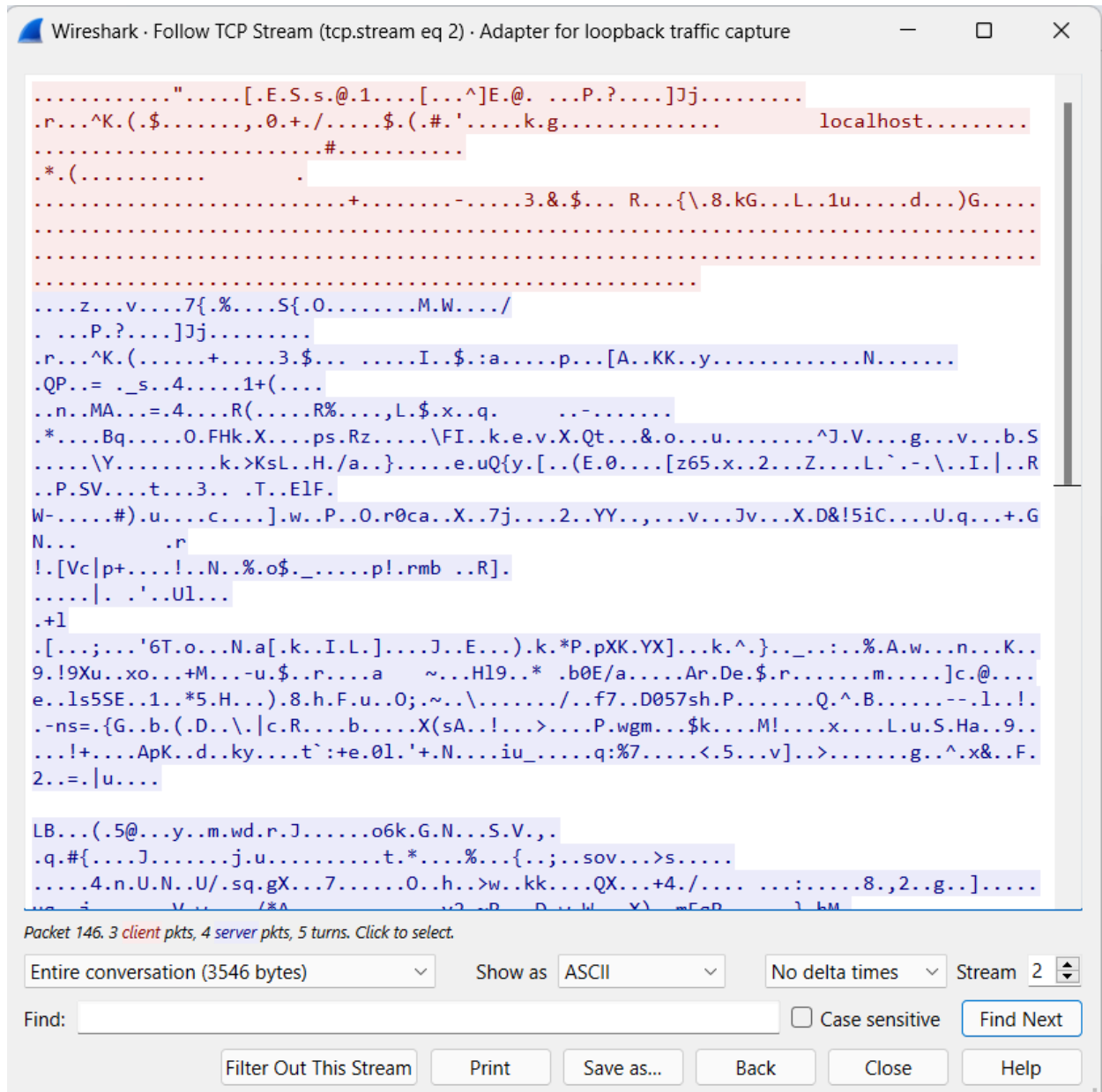
```
PS C:\Users\101080740\Documents\Networking\TCP Lab\SSL> python tcpServer.py
Server listening on port 12345...
Connection from ('127.0.0.1', 51682)
Received: wireshark test

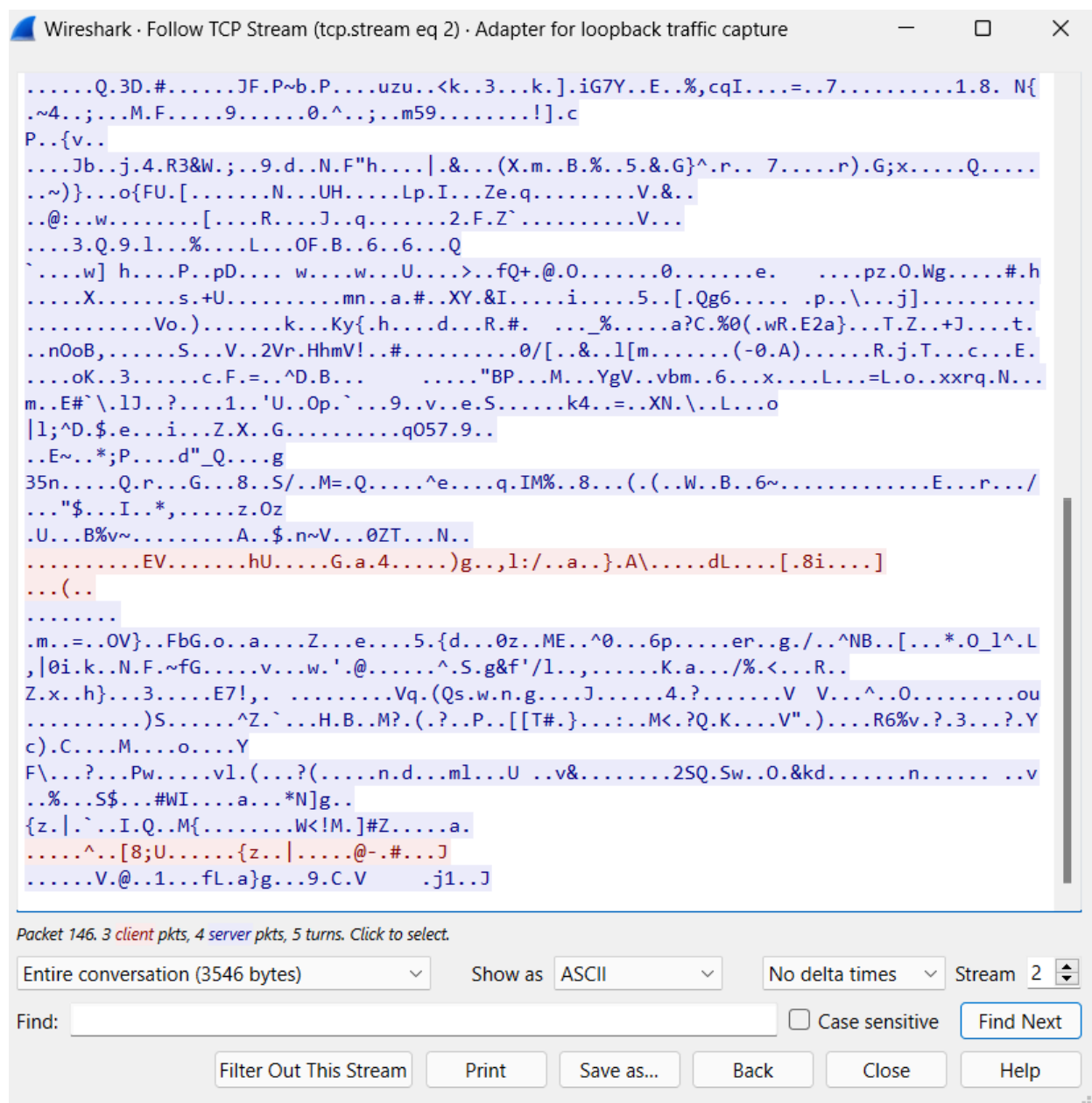
PS C:\Users\101080740\Documents\Networking\TCP Lab\SSL> python tcpClient.py
Input lowercase sentence:wireshark test
From Server: WIRESHARK TEST
PS C:\Users\101080740\Documents\Networking\TCP Lab\SSL>
```

2. Finding the “TCP handshake” in Wireshark:

703	44.863797	127.0.0.1	127.0.0.1	TCP	56	51682 → 12345	[SYN] Seq=0 Win=65535 Len=0 MSS=6
704	44.863845	127.0.0.1	127.0.0.1	TCP	56	12345 → 51682	[SYN, ACK] Seq=0 Ack=1 Win=65535
705	44.863874	127.0.0.1	127.0.0.1	TCP	44	51682 → 12345	[ACK] Seq=1 Ack=1 Win=32768 Len=0
851	52.628933	127.0.0.1	127.0.0.1	TCP	44	12345 → 51682	[ACK] Seq=634 Ack=2915 Win=32768
852	52.628618	127.0.0.1	127.0.0.1	TCP	44	12345 → 51682	[FIN, ACK] Seq=2914 Ack=634 Win=2
853	52.628637	127.0.0.1	127.0.0.1	TCP	44	51682 → 12345	[ACK] Seq=634 Ack=2915 Win=324352
854	52.628857	127.0.0.1	127.0.0.1	TCP	44	51682 → 12345	[FIN, ACK] Seq=634 Ack=2915 Win=3
855	52.628925	127.0.0.1	127.0.0.1	TCP	44	12345 → 51682	[ACK] Seq=2915 Ack=635 Win=216064

3. Following TCP stream:





Here, we can see that the traffic is definitely encrypted and is not readable in plaintext. This is a very stark contrast from the plaintext exchange we were able to see before. We can conclude that our encryption has succeeded, and the traffic is not so easily read anymore.