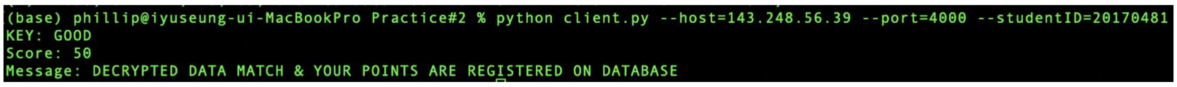
CS341 Practice #2 Report (20170481 Yuseung Lee)

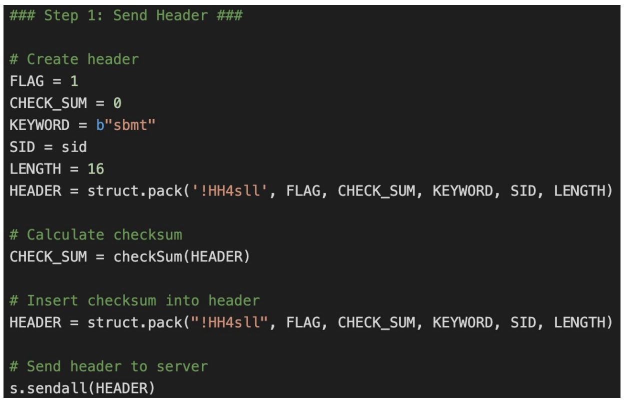
1) “client.py” Code Description (Task 1)



```
(base) phillip@iyuseung-ui-MacBookPro Practice#2 % python client.py --host=143.248.56.39 --port=4000 --studentID=20170481
KEY: GOOD
Score: 50
Message: DECRYPTED DATA MATCH & YOUR POINTS ARE REGISTERED ON DATABASE
```

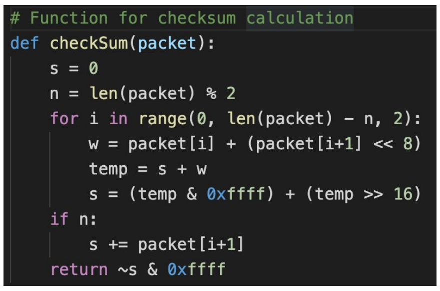
[“client.py” result]

Step 1: Send header to server



```python
### Step 1: Send Header ###

# Create header
FLAG = 1
CHECK_SUM = 0
KEYWORD = b"sbmt"
SID = sid
LENGTH = 16
HEADER = struct.pack('!HH4sll', FLAG, CHECK_SUM, KEYWORD, SID, LENGTH)

# Calculate checksum
CHECK_SUM = checkSum(HEADER)

# Insert checksum into header
HEADER = struct.pack("!HH4sll", FLAG, CHECK_SUM, KEYWORD, SID, LENGTH)

# Send header to server
s.sendall(HEADER)
```

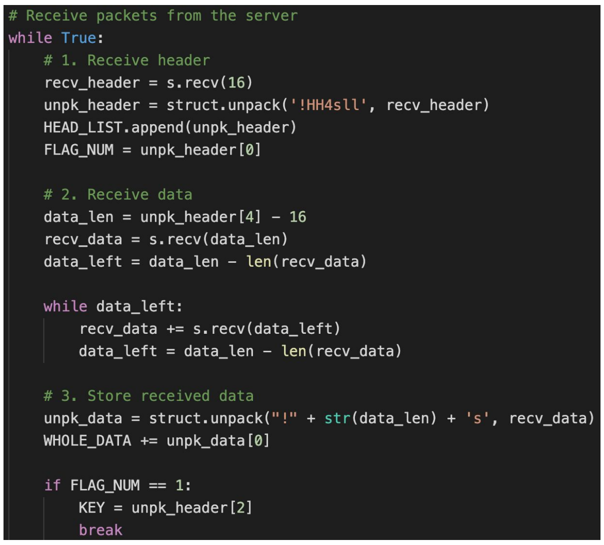
Make the initial header using the “struct.pack()” function, and calculate checksum by calling the “checkSum” function. Then, insert the checksum into the header and send the header to the server using “sendall()”.



```python
# Function for checksum calculation
def checkSum(packet):
    s = 0
    n = len(packet) % 2
    for i in range(0, len(packet) - n, 2):
        w = packet[i] + (packet[i+1] << 8)
        temp = s + w
        s = (temp & 0xffff) + (temp >> 16)
    if n:
        s += packet[i+1]
    return ~s & 0xffff
```

Define “checkSum” function which takes a bytes object as input and returns the checksum of that object. It divides the bytes into 16 bit segments and adds them, using wrap-around method when carry bits exist. Then it takes the 1’s complement of the sum and returns it as the checksum.

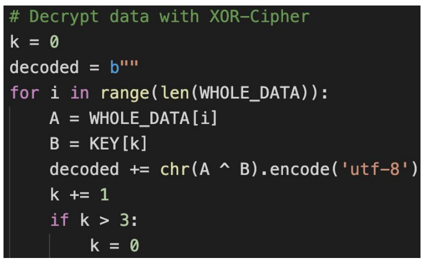
Step 2: Receive packets from server



```python
# Receive packets from the server
while True:
    # 1. Receive header
    recv_header = s.recv(16)
    unpk_header = struct.unpack('!HH4sll', recv_header)
    HEAD_LIST.append(unpk_header)
    FLAG_NUM = unpk_header[0]

    # 2. Receive data
    data_len = unpk_header[4] - 16
    recv_data = s.recv(data_len)
    data_left = data_len - len(recv_data)

    while data_left:
        recv_data += s.recv(data_left)
        data_left = data_len - len(recv_data)

    # 3. Store received data
    unpk_data = struct.unpack("!" + str(data_len) + 's', recv_data)
    WHOLE_DATA += unpk_data[0]

    if FLAG_NUM == 1:
        KEY = unpk_header[2]
        break
```

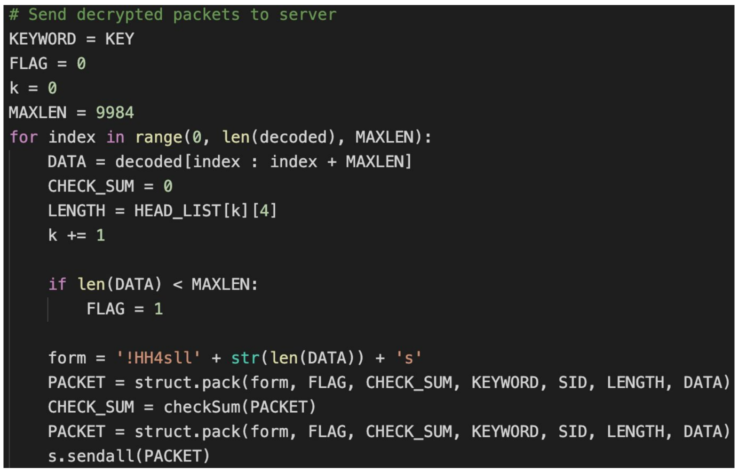
This is the while-loop for receiving packets from the server. Receive the header (first 16 bytes). From the header, get the length of the packet and receive the following data based on that given length. If the flag is 1, stop receiving packets.

Step 3: Send packets



```
# Decrypt data with XOR-Cipher
k = 0
decoded = b""
for i in range(len(WHOLE_DATA)):
    A = WHOLE_DATA[i]
    B = KEY[k]
    decoded += chr(A ^ B).encode('utf-8')
    k += 1
    if k > 3:
        k = 0
```

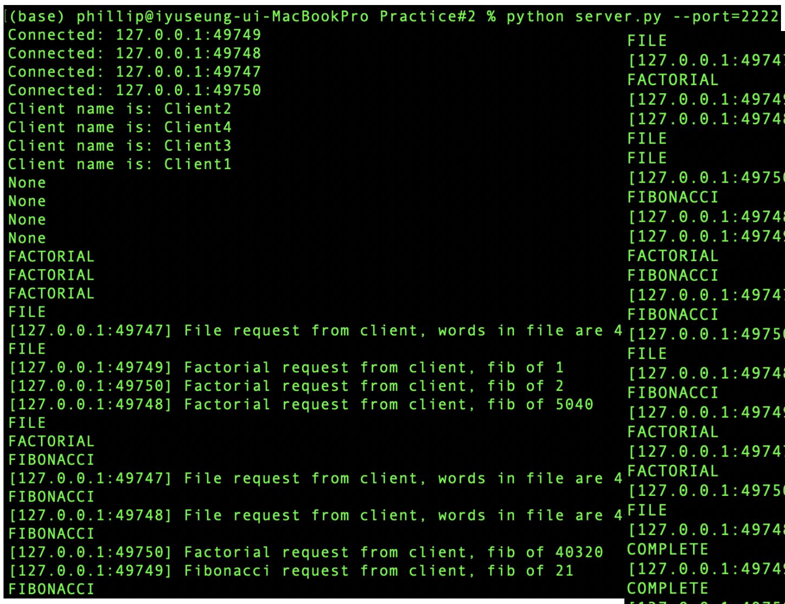
Decrypt the received data using XOR-cipher.



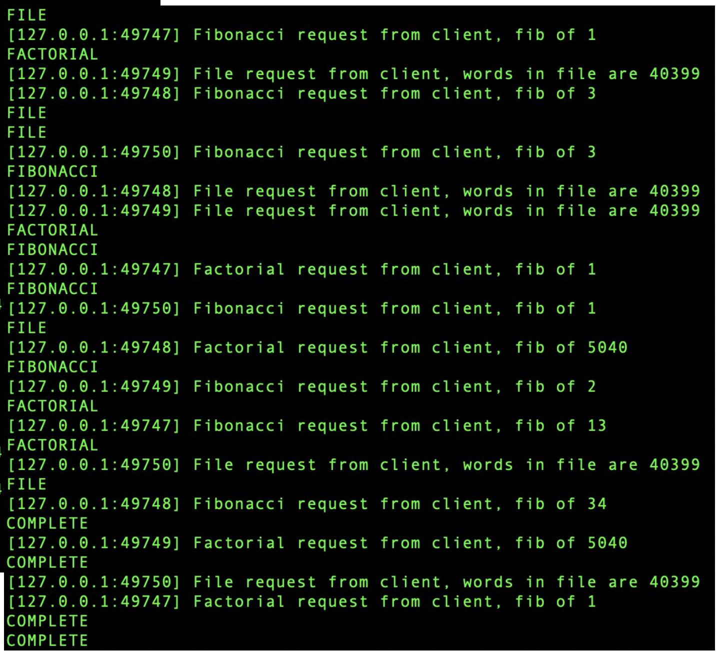
```
# Send decrypted packets to server
KEYWORD = KEY
FLAG = 0
k = 0
MAXLEN = 9984
for index in range(0, len(decoded), MAXLEN):
    DATA = decoded[index : index + MAXLEN]
    CHECK_SUM = 0
    LENGTH = HEAD_LIST[k][4]
    k += 1

    if len(DATA) < MAXLEN:
        FLAG = 1

    form = '!HH4sll' + str(len(DATA)) + 's'
    PACKET = struct.pack(form, FLAG, CHECK_SUM, KEYWORD, SID, LENGTH, DATA)
    CHECK_SUM = checkSum(PACKET)
    PACKET = struct.pack(form, FLAG, CHECK_SUM, KEYWORD, SID, LENGTH, DATA)
    s.sendall(PACKET)
```

Send the packets containing decrypted data. Since the maximum size of each packet is 10,000 bytes, we need to divide the data and send them through numerous packets. Make the initial packet using "struct.pack()" and calculate its checksum. Then insert the checksum into the packet and send it to the server.

2) "server.py" Code Description (Task 2)

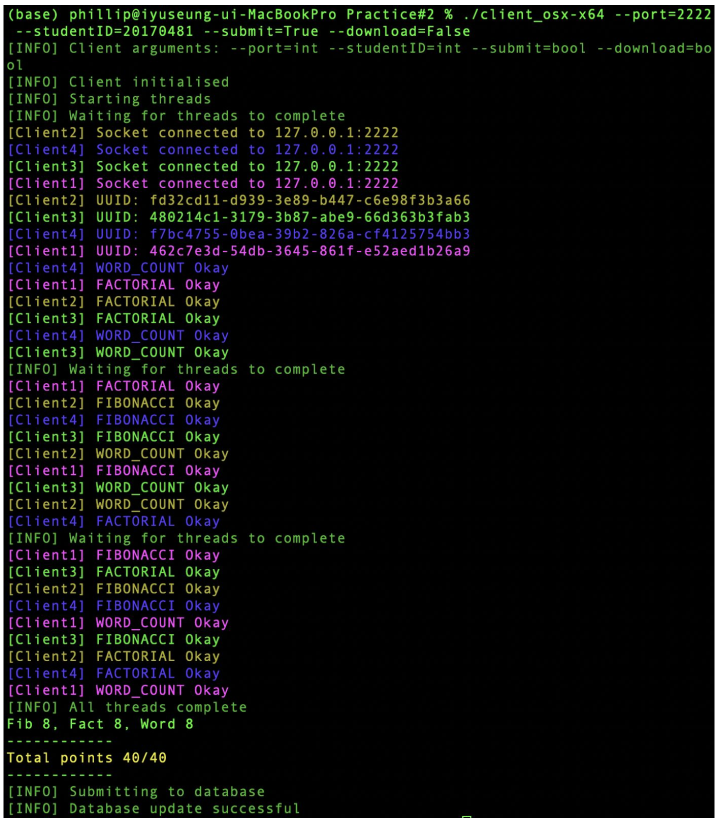


```
(base) phillip@iyuseung-ui-MacBookPro Practice#2 % python server.py --port=2222
Connected: 127.0.0.1:49749
Connected: 127.0.0.1:49748
Connected: 127.0.0.1:49747
Connected: 127.0.0.1:49750
Client name is: Client2
Client name is: Client4
Client name is: Client3
Client name is: Client1
None
None
None
None
FACTORIAL
FACTORIAL
FACTORIAL
FILE
[127.0.0.1:49747] File request from client, words in file are 4
FILE
[127.0.0.1:49749] Factorial request from client, fib of 1
[127.0.0.1:49750] Factorial request from client, fib of 2
[127.0.0.1:49748] Factorial request from client, fib of 5040
FILE
FACTORIAL
FIBONACCI
[127.0.0.1:49747] File request from client, words in file are 4
FIBONACCI
[127.0.0.1:49748] File request from client, words in file are 4
FIBONACCI
[127.0.0.1:49750] Factorial request from client, fib of 40320
[127.0.0.1:49749] Fibonacci request from client, fib of 21
FIBONACCI
```



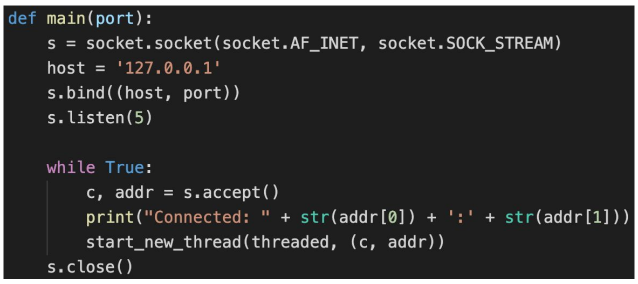
```
FILE
[127.0.0.1:49747] Fibonacci request from client, fib of 1
FACTORIAL
[127.0.0.1:49749] File request from client, words in file are 40399
[127.0.0.1:49748] Fibonacci request from client, fib of 3
FILE
FILE
[127.0.0.1:49750] Fibonacci request from client, fib of 3
FIBONACCI
[127.0.0.1:49748] File request from client, words in file are 40399
[127.0.0.1:49749] File request from client, words in file are 40399
FACTORIAL
FIBONACCI
[127.0.0.1:49747] Factorial request from client, fib of 1
FIBONACCI
[127.0.0.1:49750] Fibonacci request from client, fib of 1
FILE
[127.0.0.1:49748] Factorial request from client, fib of 5040
FIBONACCI
[127.0.0.1:49749] Fibonacci request from client, fib of 2
FACTORIAL
[127.0.0.1:49747] Fibonacci request from client, fib of 13
FACTORIAL
[127.0.0.1:49750] File request from client, words in file are 40399
FILE
[127.0.0.1:49748] Fibonacci request from client, fib of 34
COMPLETE
[127.0.0.1:49749] Factorial request from client, fib of 5040
COMPLETE
[127.0.0.1:49750] File request from client, words in file are 40399
[127.0.0.1:49747] Factorial request from client, fib of 1
COMPLETE
COMPLETE
```

["server.py" result]



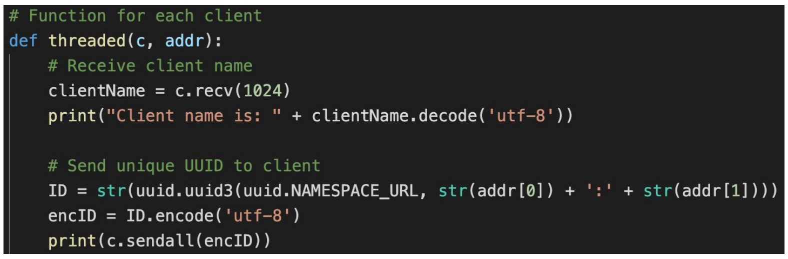
```
(base) phillip@iyuseung-ui-MacBookPro Practice#2 % ./client_osx-x64 --port=2222
 --studentID=20170481 --submit=True --download=False
[INFO] Client arguments: --port=int --studentID=int --submit=bool --download=bo
ol
[INFO] Client initialised
[INFO] Starting threads
[INFO] Waiting for threads to complete
[Client2] Socket connected to 127.0.0.1:2222
[Client4] Socket connected to 127.0.0.1:2222
[Client3] Socket connected to 127.0.0.1:2222
[Client1] Socket connected to 127.0.0.1:2222
[Client2] UUID: fd32cd11-d939-3e89-b447-c6e98f3b3a66
[Client3] UUID: 480214c1-3179-3b87-abe9-66d363b3fab3
[Client4] UUID: f7bc4755-0bea-39b2-826a-cf4125754bb3
[Client1] UUID: 462c7e3d-54db-3645-861f-e52aed1b26a9
[Client4] WORD_COUNT Okay
[Client1] FACTORIAL Okay
[Client2] FACTORIAL Okay
[Client3] FACTORIAL Okay
[Client4] WORD_COUNT Okay
[Client3] WORD_COUNT Okay
[INFO] Waiting for threads to complete
[Client1] FACTORIAL Okay
[Client2] FIBONACCI Okay
[Client4] FIBONACCI Okay
[Client3] FIBONACCI Okay
[Client2] WORD_COUNT Okay
[Client1] FIBONACCI Okay
[Client3] WORD_COUNT Okay
[Client2] WORD_COUNT Okay
[Client4] FACTORIAL Okay
[INFO] Waiting for threads to complete
[Client1] FIBONACCI Okay
[Client3] FACTORIAL Okay
[Client2] FIBONACCI Okay
[Client4] FIBONACCI Okay
[Client1] WORD_COUNT Okay
[Client3] FIBONACCI Okay
[Client2] FACTORIAL Okay
[Client4] FACTORIAL Okay
[Client1] WORD_COUNT Okay
[INFO] All threads complete
Fib 8, Fact 8, Word 8
------------
Total points 40/40
------------
[INFO] Submitting to database
[INFO] Database update successful
```

[client result]



```python
def main(port):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host = '127.0.0.1'
    s.bind((host, port))
    s.listen(5)

    while True:
        c, addr = s.accept()
        print("Connected: " + str(addr[0]) + ':' + str(addr[1]))
        start_new_thread(threaded, (c, addr))
    s.close()
```

Define “main” function of the server. Creates a socket and binds with the address. Then in the while-loop accepts multiple clients and call the “start_new_thread” function to run each client concurrently. After all clients are finished, close the socket.



```python
# Function for each client
def threaded(c, addr):
    # Receive client name
    clientName = c.recv(1024)
    print("Client name is: " + clientName.decode('utf-8'))

    # Send unique UUID to client
    ID = str(uuid.uuid3(uuid.NAMESPACE_URL, str(addr[0]) + ':' + str(addr[1])))
    encID = ID.encode('utf-8')
    print(c.sendall(encID))
```

Define “threaded” which defines what happens in each client concurrently. It sends an unique UUID to each client.

The “threaded” function receives a request message from the server and calls the corresponding function for each request (“FIBONACCI”, “FACTORIAL”, “FILE”). It sends the result back to the server. If the request is “COMPLETE”, it closes the connection with the client.

* I used “./client_osx-x64 --port=2222 --studentID=20170481 --submit=True --download=False” because I ran it on a macOS environment.