COMPUTER ENGINEERING: 4DN4

Lab 2 Report

Zhaobo Wang – 400188525

Lifeng Mei – 400256678

Zhaohan Wang – 400188640

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by **[Zhaobo Wang, 400188525]**

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by **[LiFeng Mei, 400256678]**

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by **[Zhaohan Wang, 400188640]**

1.1 Server

```
class Server:

    HOSTNAME = "127.0.0.1"
    PORT = 50000
    RECV_BUFFER_SIZE = 1024
    MAX_CONNECTION_BACKLOG = 10
    MSG_ENCODING = "utf-8"
    SOCKET_ADDRESS = (HOSTNAME, PORT)


    def __init__(self):
        self.create_listen_socket()
        self.process_connections_forever()

    def create_listen_socket(self):
        try:
            self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            self.socket.bind(Server.SOCKET_ADDRESS)
            self.socket.listen(Server.MAX_CONNECTION_BACKLOG)
            print("Listening on port {} ...".format(Server.PORT))
            print("\n")
        except KeyboardInterrupt:
            sys.exit(1)
        except Exception as msg:
            print(msg)
            sys.exit(1)
```

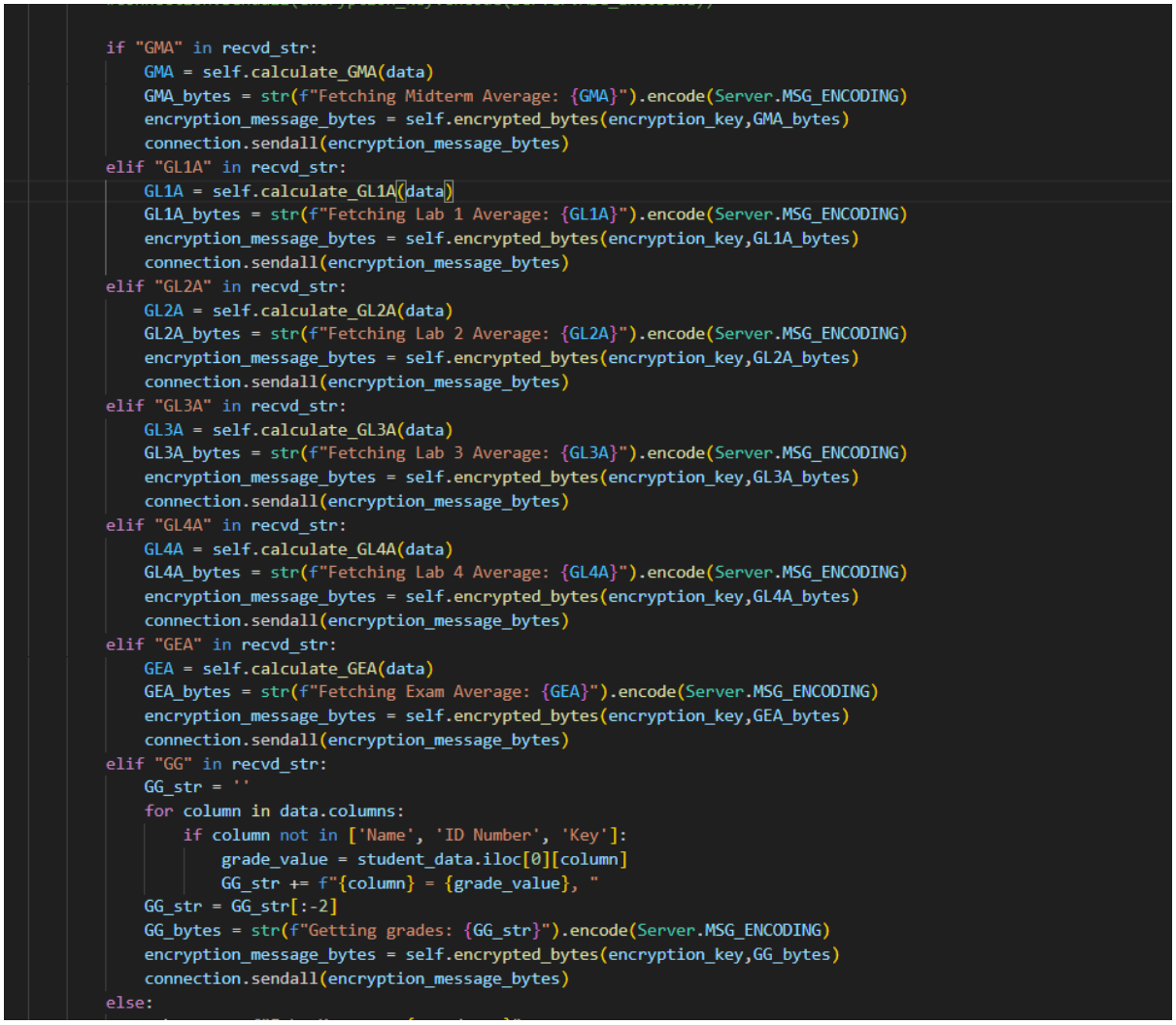
A server class is created, create_listen_socket help server continuedly listen to the socket port, with socket.socket()/bind()/listen() these functions, a listening socket is created.

```
def process_connections_forever(self):
    try:
        while True:
            try:
                self.connection_handler(self.socket.accept())
            except KeyboardInterrupt:
                self.socket.close()
                sys.exit(1)
    except Exception as msg:
        print(msg)


    finally:
        self.socket.close()
        sys.exit(1)

def connection_handler(self, client):

    connection, address_port = client
    Grade_File = 'course_grades_2024.csv'
    data = pd.read_csv(Grade_File)
    self.print_whole_csv(data)
    self.print_client_info(address_port[0],address_port[1])
```

In this function, this server will process the client connection forever with the following connection handler, once socket.accept() is called, a connection and address_port will return from the client side. In the connection handler function, data is read and printed from the server launch panel and client information is also printed.

```
while True:
    try:

        recvd_bytes = connection.recv(Server.RECV_BUFFER_SIZE)

        if len(recvd_bytes) == 0:
            print("Closing client connection ... ")
            connection.close()
            break

        recvd_str = recvd_bytes.decode(Server.MSG_ENCODING)
        student_ID = self.extract_student_id(recvd_str)

        if len(student_ID) != 7 or not student_ID.isdigit():
            id_error_str = (f"invalid student number, correct student id should contain 7 digits \n"
                            f"Echo Message: {recvd_str}")
            connection.sendall(id_error_str.encode(Server.MSG_ENCODING))
            print("User not found \n")

        else:

            student_ID = int(student_ID)
            student_data = data[data['ID Number'] == student_ID]

            if student_data.empty:
                print("User not Found \n")
                print("Closing client connection ... ")
                connection.close()
```

In this part, the server deals with the returned msg, first decoded the msg and then extracted the student ID, it will see whether the student ID matches the csv database student ID.



```
        if "GMA" in recvd_str:
            GMA = self.calculate_GMA(data)
            GMA_bytes = str(f"Fetching Midterm Average: {GMA}").encode(Server.MSG_ENCODING)
            encryption_message_bytes = self.encrypted_bytes(encryption_key,GMA_bytes)
            connection.sendall(encryption_message_bytes)
        elif "GL1A" in recvd_str:
            GL1A = self.calculate_GL1A(data)
            GL1A_bytes = str(f"Fetching Lab 1 Average: {GL1A}").encode(Server.MSG_ENCODING)
            encryption_message_bytes = self.encrypted_bytes(encryption_key,GL1A_bytes)
            connection.sendall(encryption_message_bytes)
        elif "GL2A" in recvd_str:
            GL2A = self.calculate_GL2A(data)
            GL2A_bytes = str(f"Fetching Lab 2 Average: {GL2A}").encode(Server.MSG_ENCODING)
            encryption_message_bytes = self.encrypted_bytes(encryption_key,GL2A_bytes)
            connection.sendall(encryption_message_bytes)
        elif "GL3A" in recvd_str:
            GL3A = self.calculate_GL3A(data)
            GL3A_bytes = str(f"Fetching Lab 3 Average: {GL3A}").encode(Server.MSG_ENCODING)
            encryption_message_bytes = self.encrypted_bytes(encryption_key,GL3A_bytes)
            connection.sendall(encryption_message_bytes)
        elif "GL4A" in recvd_str:
            GL4A = self.calculate_GL4A(data)
            GL4A_bytes = str(f"Fetching Lab 4 Average: {GL4A}").encode(Server.MSG_ENCODING)
            encryption_message_bytes = self.encrypted_bytes(encryption_key,GL4A_bytes)
            connection.sendall(encryption_message_bytes)
        elif "GEA" in recvd_str:
            GEA = self.calculate_GEA(data)
            GEA_bytes = str(f"Fetching Exam Average: {GEA}").encode(Server.MSG_ENCODING)
            encryption_message_bytes = self.encrypted_bytes(encryption_key,GEA_bytes)
            connection.sendall(encryption_message_bytes)
        elif "GG" in recvd_str:
            GG_str = ''
            for column in data.columns:
                if column not in ['Name', 'ID Number', 'Key']:
                    grade_value = student_data.iloc[0][column]
                    GG_str += f"{column} = {grade_value}, "
            GG_str = GG_str[:-2]
            GG_bytes = str(f"Getting grades: {GG_str}").encode(Server.MSG_ENCODING)
            encryption_message_bytes = self.encrypted_bytes(encryption_key,GG_bytes)
            connection.sendall(encryption_message_bytes)
        else:
```

In this part, server will try to read the specific command from the string, then it will return the specific value back to the client using connection.sendall() functions.

```python
def calculate_GMA(self,grade):
    return grade['Midterm'].mean()

def calculate_GEA(self,grade):
    exam_columns = ['Exam 1', 'Exam 2', 'Exam 3', 'Exam 4']
    exams = grade[exam_columns]
    total_exam_scores = exams.sum(axis=1)
    return total_exam_scores.mean()

def calculate_GL1A(self,grade):
    return grade['Lab 1'].mean()

def calculate_GL2A(self,grade):
    return grade['Lab 2'].mean()

def calculate_GL3A(self,grade):
    return grade['Lab 3'].mean()

def calculate_GL4A(self,grade):
    return grade['Lab 4'].mean()

def extract_student_id(self,recvd_str):
    student_id = recvd_str[:7]
    return student_id

def find_encryption_key(self,grade,student_id):
    student_data = grade[grade['ID Number'] == student_id]

    if not student_data.empty:
        return student_data.iloc[0]['Key']
    else:
        return None
```

In this part, it calculated the mean avg for the csv, extracted the student ID, found the encryption key, and it implements all these specific functions.

```
def encrypted_bytes(self,encryption_key,GA_bytes):

    encryption_key_bytes = encryption_key.encode(Server.MSG_ENCODING)
    fernet = Fernet(encryption_key_bytes)
    encryption_message_bytes = fernet.encrypt(GA_bytes)
    return encryption_message_bytes

def print_whole_csv(self,data):

    print("-" * 72)
    print("Data read from CSV file:")
    data_str = ""
    for index, row in data.iterrows():
        for column in data.columns:
            grade_value = row[column]
            data_str += f"{column} = {grade_value}, "
    data_str = data_str[:-2] +"\n"
    print(data_str)

def print_client_info(self,addr,port):
    print("-" * 72)
    print(f"connections recieved from {addr} on port {port}")
    print("\n")
```

In this part, it printed the whole csv, print client information, made the encrypted bytes, and it implements all these specific functions.

1.2 Client

```
def __init__(self):
    self.get_socket()
    self.connect_to_server()
    self.send_console_input_forever()

def get_socket(self):
    try:
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    except Exception as msg:
        print(msg)
        sys.exit(1)

def connect_to_server(self):
    try:

        self.socket.connect((Client.SERVER_HOSTNAME, Server.PORT))
        print("Connected to \"{}\" on port {}".format(Client.SERVER_HOSTNAME, Server.PORT))
        print("Now, you can input your (studentID+Your request Grade) \n")
        print("For example, 1803933GMA")
        print("\n")
```

A socket is created and fire up a connection with the server using socket.connect function. All the connection info will then be printed out.

```
def send_console_input_forever(self):
    while True:
        try:
            self.get_console_input()
            self.connection_send()
            self.connection_receive()
        except (KeyboardInterrupt, EOFError):
            print()
            print("Closing server connection ...")
            self.socket.close()
            sys.exit(1)
```

The client will run forerver without needing to terminate and restart the program by defining the above function. The function will continuesly take and send the input to the server and also decrypt and decode the recived massage in a while loop.

```
def get_console_input(self):

    while True:
        self.input_text = input("Input: ")
        if self.input_text != "":
            student_id = self.input_text[:7]
            print("command entered: ", self.input_text)

            Grade_File = 'Client_key_info.csv'
            data = pd.read_csv(Grade_File)

            student_id = int(student_id)
            filtered_row = data[data['ID Number'] == student_id]

            if not filtered_row.empty:
                self.key = filtered_row['Key'].values[0]
```

In the get input function, we find the key of the client by searching the key depending on the input student id in a csv file that we created with only ‘student id’ and ‘key’ colum. This may not work in practice because each client should hold their own key, but in this lab we assume a “general” client which means we pretent the user is the client of the input student id.

```
def connection_send(self):
    try:
        self.socket.sendall(self.input_text.encode(Server.MSG_ENCODING))
    except Exception as msg:
        print(msg)
        sys.exit(1)
```

This function send the command to the server using sendall().

```
def connection_receive(self):
    try:
        if len(self.key) == 0:
            print("Closing server connection ... ")
            self.socket.close()
            sys.exit(1)
        # first key is encrpyted
        if "contain 7 digits" in self.key:
            print(self.key)
            self.socket.close()
            sys.exit(1)
        elif "No encryption" in self.key:
            print(self.key)
            self.socket.close()
            sys.exit(1)
        else:
            encryption_key = self.key
            self.fernet = Fernet(encryption_key)

        recvd_bytes = self.socket.recv(Client.RECV_BUFFER_SIZE)
        if len(recvd_bytes) == 0:
            print("Closing server connection ... ")
            self.socket.close()
            sys.exit(1)

        try:
            decrypted_message_bytes = self.fernet.decrypt(recvd_bytes)
            decrypted_message = decrypted_message_bytes.decode(Server.MSG_ENCODING)
            print(f"Decrypted_message: \n{decrypted_message}")
            print("\n")
```

The encrypted massage byte will be recieved by using recv function. When the client recieves the encrypted massage byte from the server, it needs to be decrypted to the decrypted bytes first, then we can decode it and get the correct massages.