

# CS3031: Project II

Conor McCauley - 17323203

April 13, 2020

## 1 Implementation

### 1.1 Overview

The project consists of two primary components: a client application and a server. The server is implemented using a Flask API in Python. It stores information on active users, groups and encrypted messages and responds to HTTP requests from instances of the client application.

The client application is also implemented in Python. It allows a user to login with a unique username and password and create or join messaging groups. From inside these groups the creator can add and remove participants while all members of the group can send messages.

All API requests require authentication in the form of a password which is provided by the client and compared with the server's hashed version.

### 1.2 Encryption

Each pair of users in a group share a unique RSA key pair. Whenever a user sends a message into the group it is encrypted separately for each other user in the group using the appropriate public key. Other users can sort through the different encrypted versions of the message and select the version intended for them and then decrypt it using the the appropriate private key.

In order to exchange keys each client listens for requests on a unique port. Whenever a new user joins a group they receive a list of user ports from the API and, for each user, generates new RSA keys and sends the public key to that user via a socket. The other user then responds in the same manner. No further direct communication between the two users is required.

The key exchange does not occur via the server meaning that only the two relevant users are aware of their key pairs. One improvement that could be made to the implementation would be to wrap all socket communications in SSL in order to prevent malevolent actors from eavesdropping on the client-to-client communication.

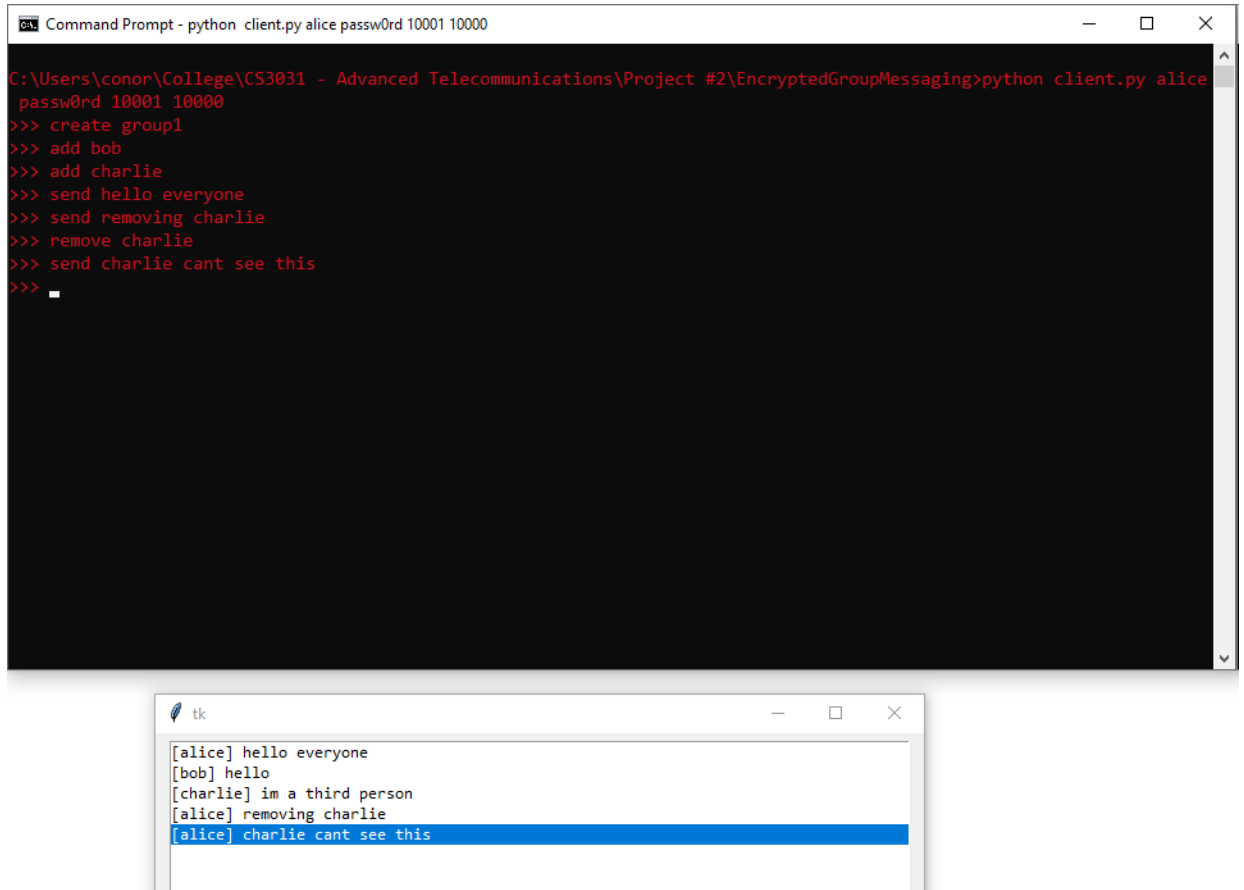
In summary the communication between two clients follows the following structure:

1. Client 1: joins group.
2. Client 1: generates RSA keys for communication with client 2 and stores the private key locally.
3. Client 1: sends the public key to client 2 via a socket.
4. Client 2: receives the public key and stores it locally.
5. Client 2: generates RSA keys for communication with client 1 and stores the private key locally.
6. Client 2: sends the public key to client 1 via the same socket.
7. Client 1: receives the public key and stores it locally.
8. Both clients now have local public and private keys for decrypting and encrypting messages.

### **1.3 Examples**

This section just contains a couple of screenshots of the messaging application being used.

### 1.3.1 Console and GUI for Alice



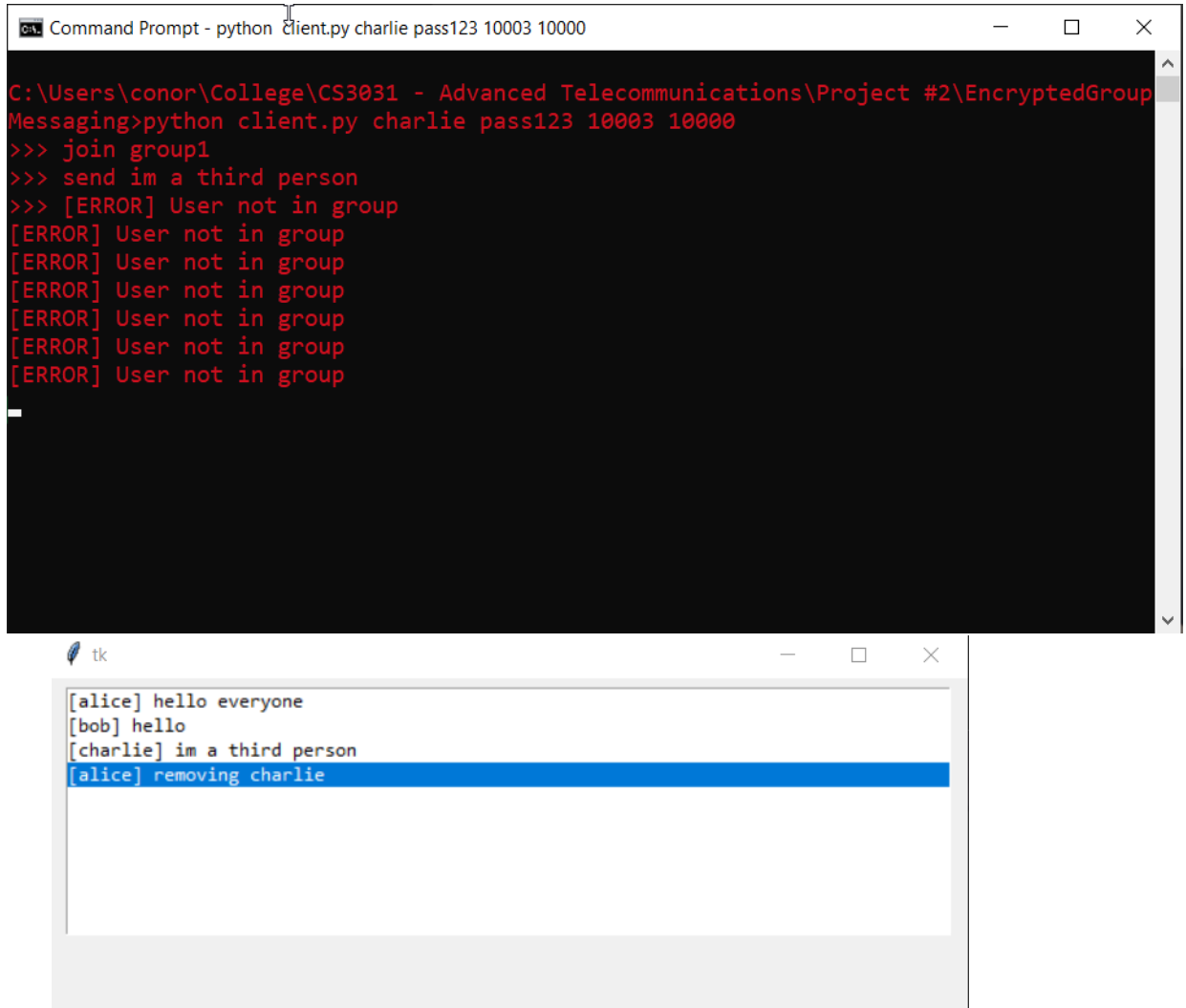
The image shows two windows. The top window is a Command Prompt titled "python client.py alice passwd 10001 10000". It displays the following Python code and its output:

```
C:\Users\conor\College\CS3031 - Advanced Telecommunications\Project #2\EncryptedGroupMessaging>python client.py alice
passwd 10001 10000
>>> create group1
>>> add bob
>>> add charlie
>>> send hello everyone
>>> send removing charlie
>>> remove charlie
>>> send charlie cant see this
>>> _
```

The bottom window is a Tkinter window titled "tk". It displays the following text:

```
[alice] hello everyone
[bob] hello
[charlie] im a third person
[alice] removing charlie
[alice] charlie cant see this
```

### 1.3.2 Console and GUI for Charlie



The image shows two windows. The top window is a Command Prompt titled "Command Prompt - python Client.py charlie pass123 10003 10000". It displays the following text:

```
C:\Users\conor\College\CS3031 - Advanced Telecommunications\Project #2\EncryptedGroup Messaging>python client.py charlie pass123 10003 10000
>>> join group1
>>> send im a third person
>>> [ERROR] User not in group
[ERROR] User not in group
[ERROR] User not in group
[ERROR] User not in group
[ERROR] User not in group
[ERROR] User not in group
[ERROR] User not in group
```

The bottom window is a Tkinter GUI titled "tk". It displays a list of chat messages:

```
[alice] hello everyone
[bob] hello
[charlie] im a third person
[alice] removing charlie
```

The message "[alice] removing charlie" is highlighted in blue.

### 1.3.3 Server log for this example

```
Command Prompt - python server.py 10000
[USER] User 'alice' logged in
[USER] User 'bob' logged in
[USER] User 'charlie' logged in
[GROUP] Group 'group1' created by 'alice'
[GROUP] User 'bob' added to group 'group1'
[GROUP] User 'bob' joined group 'group1'
[GROUP] User 'charlie' added to group 'group1'
[GROUP] User 'charlie' joined group 'group1'
[GROUP] User 'alice' sent a message to group 'group1'
[MESSAGE] Encrypted message to user 'bob': 'Mw1/29zQMcbCzBqxmNNQRX0bxBo44pZ72UmNKA+G
juGBLw1fTskZIDuGVhV0HL/VwUsqa8UNQPrkQBwrtF37Du5f2u+PW1WMH4bKyBUctKf6UBC4sQ3etpumhvBOR
nraPLua+ha04so4zQuzo9gnP++mmThgRBucK41unCR+QU='
[MESSAGE] Encrypted message to user 'charlie': 'A9QfV6cBkSFc1euAmOLt2IFLDG3c/XQU5bwh0
GObRf/wokdCuryRZDK/pn9v8ijAQmaQ3vW58Bo5Neem9JYVCryDDZsGcsNf19YsleEhL/rqHHf7oyMYcGaVTN
wKMrmmKYRemunuCM4p04k/fIzm++QUcalwJuwmlHsNcMwSol4='
[GROUP] User 'bob' sent a message to group 'group1'
[MESSAGE] Encrypted message to user 'alice': 'cXfo39RU5jERRR435mRtmGZteeq+4Q0YNpvIw31
W5qe0/rBozqkEl+jHkqftXaxIszaVwHOpks2SdqIfj2LbNMPv02MLIytxv0Adw/yKCEzKI/Ddqb0MIKaJQVZY
HM7eyySM9AHezrBERBZh38/WVoNjVnyifhVF2wv0uSUzV+w='
[MESSAGE] Encrypted message to user 'charlie': 'DI5dyvHUXu5gwrmeaVS0eRbGQrtFJiE/9X1h
JRAMb7DzCdDCLnrd8z/I/G6CpuUx7hH0WxwQmrsIMjBTp4mc2q0nWqnbFi4ykXCmkVx8gXBnJcVRQmyDKCNGu
LEhyEfmT2Ybuul/OaicLveEARjnR/F10GRQPD4eu0LQfFhsYA='
[GROUP] User 'charlie' sent a message to group 'group1'
[MESSAGE] Encrypted message to user 'alice': 'f4DnSqRRbaeDnrz/5rTpau3acdRag0FpJc+67So
kdzCuGHPAz6ltsvNyyFZRQ7v3qr1fS10HJR1F2JRUdnBftrr0idkgzbivttzmtaPZ9Whc05niU0CIbkMxnQi
pG7K5sQVr0IEye3FFqoDPZ19eGdgA0asYZhN06DlglwUwA9g='
[MESSAGE] Encrypted message to user 'bob': 'jSltAhQTciT/QYiFJnZnfYJaS1+97edzATo1+iIf6
L3wCQUhKhNqQp+hXSieYrHv/002N5lQmauLgiXPV8BwgZzr4doTwCVcB+id9WYJRz9jC/j+WN+BpDbSANxgd6
dxXygvu34wmg09qrUn2Gqu4Z43Qnl+XRmc2H8Z4WxiUo4='
[GROUP] User 'alice' sent a message to group 'group1'
[MESSAGE] Encrypted message to user 'bob': 'OUhg9dWssErQ95OM2DzFW4Rg2n8tiCz+jtgOuWjbs
mBP0j0094Siidk+2hb+BBnljqSBko0UwdjSmUdPx+UH9HYH1P0lsc7aGyC55tnM7Kt3Rni0+JlBtVzTxJIV2
Pdr1xy6Yc1r5fpfx2VLNUdKIXiqZLax7qbz94mXT0J/c8='
[MESSAGE] Encrypted message to user 'charlie': 'v+xLMkM7jQD3fHwriTyfFgFFdwbo0z6E6bMHU
Ev6bomqHiWRThAAX59esmlcBaUwLy7Hc2xoYCRH8nQwVWp1woGY23oKBTG+MwsKYJg92booLvBVWos+zrZglU
p09q/jb6xxB01JJ+y1MAMc4hy68Y+p3ueYNdG0omqbpD+2NYE='
[GROUP] User 'charlie' removed from group 'group1'
[GROUP] User 'alice' sent a message to group 'group1'
[MESSAGE] Encrypted message to user 'bob': 'DqAIX7qKA9JQqKzjhZXFJ59zEm5LTrv6SNjCOaXeC
6MjJZTrj3+wtHYhVf/0uYJTfgVWDr/IBK3i5AbFuanWHK/ZJBK36sSLunSu0Yg83Me8ba+0ojMGC0452FP77t
M3zh0jZk0HXY8u7BuyqT/d/d3Wv2baJbcZWJU0sqjDzw='
[MESSAGE] Encrypted message to user 'charlie': 'DVBoMd6h3QjhN8f51sPapJo8mIMpeIN9KXOF
```

## 2 Code Listing

### 2.1 Client

```
import base64
import requests
import socket
import threading
import time
import tkinter
from Crypto import Random
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
from enum import Enum
from queue import Queue
from tkinter import *

# command enum
class Command(Enum):
    CREATE = 1
    ADD = 2
    REMOVE = 3
    JOIN = 4
    SEND = 5
    DESTROY = 6

# config
USERNAME = '' # username to use
PASSWORD = '' # password to use for auth
CLIENT_PORT = -1 # port to listen for key exchanges on
SERVER_URL = 'http://127.0.0.1' # api url
SERVER_PORT = -1 # api port
MOD_LEN = 1024 # modulus length for rsa

# global variables
currently_in_group = False # is the user currently in a group
group_name = '' # the name of the group the user is in
keys = {} # local key storage
our_messages = Queue() # messages to be printed when there's a
                        chance

# start the client
```

```

def start_client():
    login_result = login()
    if not login_result:
        quit()
    while True:
        command, param = get_command()
        if command == Command.CREATE:
            create_group(param)
        elif command == Command.JOIN:
            join_group(param)
        elif command == Command.ADD:
            add_user(param)
        elif command == Command.REMOVE:
            remove_user(param)
        elif command == Command.SEND:
            send_message(param)

# start a new thread to exchange keys with other group members
def start_key_exchange(users):
    for user in users:
        # generate keys
        private_key = RSA.generate(MOD_LEN, Random.new().read)
        public_key = private_key.publickey()
        keys[user] = {}
        # send our username and public key to the other user
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect(('127.0.0.1', users[user]))
        sock.send(f'{USERNAME}|{public_key.exportKey().decode()}'
                  '.encode())
        # receive the other users public key
        encrypt_key = sock.recv(8192).decode()
        keys[user]['encrypt'] = encrypt_key
        keys[user]['decrypt'] = private_key

# start a new thread to listen for future key exchanges
def start_key_listener():
    # start listening on the client port
    sock = socket.socket()
    sock.bind('', CLIENT_PORT)
    sock.listen(5)
    while True:
        conn, address = sock.accept()

```

```

        # extract the public key and username from the received data
        username, encrypt_key = conn.recv(8192).decode().split('
|')
        keys[username] = {'encrypt': encrypt_key}
        # generate keys
        private_key = RSA.generate(MODLEN, Random.new().read)
        public_key = private_key.publickey()
        keys[username]['decrypt'] = private_key
        # send the public key
        conn.send(public_key.exportKey())
        conn.close()

# create a Tkinter display to show group messages
def start_group_terminal():
    ui = tkinter.Tk()
    ui.geometry("600x400")
    message_box = Listbox(ui, font=('Consolas', 10))
    message_box.pack(fill=BOTH, padx=10, pady=5)
    num_cur_messages = 0
    while True:
        ui.update_idletasks()
        ui.update()
        # add our messages to the ui
        while not our_messages.empty():
            message_box.insert(END, f"[{USERNAME}] {our_messages.get()}")
        # add new messages to the ui (if any exist)
        new_messages = get_messages()
        num_new_messages = len(new_messages)
        if num_new_messages > num_cur_messages:
            new_messages = new_messages[num_cur_messages:]
            num_cur_messages = num_new_messages
            for new_message in new_messages:
                message_box.insert(END, f"[{new_message['sender']}] {new_message['message']}")
        time.sleep(1)

# encrypt a message
def encrypt_message(text):
    messages = {}
    for user in keys:
        key = keys[user]['encrypt']

```



```

        # encrypt and encode the message
        encryptor = PKCS1_OAEP.new(RSA.importKey(key))
        messages[user] = base64.b64encode(encryptor.encrypt(text
            .encode())).decode()
    return messages

# decrypt received messages
def decrypt_messages(messages):
    decrypted_messages = []
    for message in messages:
        if USERNAME in message['message']:
            key = keys[message['sender']]['decrypt']
            encrypted_message = message['message'][USERNAME]
            # decrypt and decode the message and format it in a
            dictionary
            decryptor = PKCS1_OAEP.new(key)
            decrypted_text = decryptor.decrypt(base64.b64decode(
                encrypted_message.encode())).decode()
            decrypted_message = {'sender': message['sender'], '
                message': decrypted_text}
            decrypted_messages.append(decrypted_message)
    return decrypted_messages

# login to the server
def login():
    status_code, message = make_request('login', {
        'username': USERNAME,
        'password': PASSWORD,
        'port': CLIENT_PORT
    })
    success = (status_code == 200)
    if not success:
        print(f'[ERROR] {message}')
    return success

# get a command from the user
def get_command():
    while True:
        text = input('>>>').strip()
        if text == 'help':
            print_help()
        elif not currently_in_group:

```

```

        if text.startswith('create_'):
            return Command.CREATE, text[7:]
        elif text.startswith('join_'):
            return Command.JOIN, text[5:]
        else:
            print(' [ERROR] _Unrecognised _command_-_type_\'
                  help\' _for _a _list _of _commands')
    else:
        if text.startswith('add_'):
            return Command.ADD, text[4:]
        elif text.startswith('remove_'):
            return Command.REMOVE, text[7:]
        elif text.startswith('send_'):
            return Command.SEND, text[5:]
        else:
            print(' [ERROR] _Unrecognised _command_-_type_\'
                  help\' _for _a _list _of _commands')

# print command help
def print_help():
    print('>>>_Commands:')
    print('>>>_...help')
    if not currently_in_group:
        print('>>>_...create_{GROUP}')
        print('>>>_...join_{GROUP}')
    else:
        print('>>>_...add_{USER}')
        print('>>>_...remove_{USER}')
        print('>>>_...send_{MESSAGE}')

# create a group
def create_group(_group_name):
    global currently_in_group, group_name
    status_code, message = make_request('group/create', {
        'username': USERNAME,
        'password': PASSWORD,
        'groupname': _group_name
    })
    if status_code != 200:
        print(f' [ERROR] _{message}')
        return
    currently_in_group = True
    group_name = _group_name

```

```

# start key listener and ui threads
threading.Thread(target=start_key_listener , daemon=True).
    start()
threading.Thread(target=start_group_terminal , daemon=True).
    start()

# join a group
def join_group(_group_name):
    global currently_in_group , group_name
    status_code , message , json = make_request('group/join' , {
        'username': USERNAME,
        'password': PASSWORD,
        'groupname': _group_name
    } , json_response=True)
    if status_code != 200:
        print(f'[ERROR] {message}')
        return
    currently_in_group = True
    group_name = _group_name
    # start key exchange , key listener and ui threads
    threading.Thread(target=start_key_exchange , args=(json , ),
        daemon=True).start()
    threading.Thread(target=start_key_listener , daemon=True).
        start()
    threading.Thread(target=start_group_terminal , daemon=True).
        start()

# add a user to a group
def add_user(add_username):
    status_code , message = make_request('group/add' , {
        'username': USERNAME,
        'password': PASSWORD,
        'groupname': group_name ,
        'add_username': add_username
    })
    if status_code != 200:
        print(f'[ERROR] {message}')

# remove a user from a group
def remove_user(rem_username):
    status_code , message = make_request('group/remove' , {
        'username': USERNAME,

```

```

        'password': PASSWORD,
        'groupname': group_name,
        'rem_username': rem_username
    })
    if status_code != 200:
        print(f'[ERROR]_{message}')

# send a message to a group
def send_message(message):
    our_messages.put(message)
    status_code, status_message = make_request('message/send', {
        'username': USERNAME,
        'password': PASSWORD,
        'groupname': group_name,
        'encrypted_messages': encrypt_message(message)
    })
    if status_code != 200:
        print(f'[ERROR]_{status_message}')

# get a groups messages
def get_messages():
    status_code, status_message, json = make_request('message/
    get', {
        'username': USERNAME,
        'password': PASSWORD,
        'groupname': group_name
    }, json_response=True)
    if status_code != 200:
        print(f'[ERROR]_{status_message}')
        return []
    return decrypt_messages(json) # return the decrypted
    messages

# make an api request
def make_request(route, params, json_response=False):
    try:
        req = requests.post(f'{SERVER_URL}:{SERVER_PORT}/api/{
            route}', json=params)
        if not json_response:
            return req.status_code, req.text
        else:

```

```

        return req.status_code, req.text, ({} if req.
            status_code != 200 else req.json())
except:
    return 500, 'Could_not_connect_to_server'

if __name__ == '__main__':
    import argparse
    argp = argparse.ArgumentParser()
    argp.add_argument('username', help='unique_username_to_use')
    argp.add_argument('password', help='password_to_use')
    argp.add_argument('client_port', type=int, help='port_to_
        listen_on')
    argp.add_argument('server_port', type=int, help='port_the_
        server_is_on')
    args = argp.parse_args()
    USERNAME = args.username
    PASSWORD = args.password
    CLIENT_PORT = args.client_port
    SERVER_PORT = args.server_port
    start_client()

```

## 2.2 Server

```

from datetime import datetime
from flask import Flask, jsonify, request
import hashlib
import logging

app = Flask(__name__)
log = logging.getLogger('werkzeug')
log.setLevel(logging.ERROR)

USERS, GROUPS = {}, {}

# login to the server
@app.route('/api/login', methods=['POST'])
def login():
    is_valid, message, code = validate_request(request, ('
        username', 'password', 'port'))
    if not is_valid:
        return message, code
    json = request.get_json()
    username, password, port = json['username'], json['password',

```

```

    ], json['port'])
    if username in USERS:
        return 'Username_in_use', 400
    else:
        USERS[username] = {'password': hash_password(password),
                           'port': port}
        log('USER', f"User_{username}_logged_in")
        return 'Login_successful', 200

# create a group
@app.route('/api/group/create', methods=['POST'])
def group_create():
    is_valid, message, code = validate_request(request, ('
        username', 'password', 'groupname'))
    if not is_valid:
        return message, code
    json = request.get_json()
    username, password, groupname = json['username'], json['
        password'], json['groupname']
    if username not in USERS:
        return 'User_does_not_exist', 400
    elif not validate_password(username, password):
        return 'Invalid_password', 401
    elif groupname in GROUPS:
        return 'Group_name_in_use', 400
    else:
        GROUPS[groupname] = {'admin': username, 'users': {
            username}, 'messages': []}
        log('GROUP', f"Group_{groupname}_created_by_{username
            }'")
        return 'Group_created_successfully', 200

# destroy a group
@app.route('/api/group/destroy', methods=['POST'])
def group_destroy():
    is_valid, message, code = validate_request(request, ('
        username', 'password', 'groupname'))
    if not is_valid:
        return message, code
    json = request.get_json()
    username, password, groupname = json['username'], json['
        password'], json['groupname']
    if username not in USERS:

```

```

        return 'User_does_not_exist', 400
    elif not validate_password(username, password):
        return 'Invalid_password', 401
    elif groupname not in GROUPS:
        return 'Group_does_not_exist', 400
    elif username != GROUPS[groupname]['admin']:
        return 'User_does_not_have_permission_to_modify_group',
        401
    else:
        del GROUPS[groupname]
        log('GROUP', f"Group_{groupname}'_destroyed_by_{username}")
        return 'Group_destroyed_successfully', 200

# add a user to a group
@app.route('/api/group/add', methods=['POST'])
def group_add():
    is_valid, message, code = validate_request(request, ('username', 'password', 'groupname', 'add_username'))
    if not is_valid:
        return message, code
    json = request.get_json()
    username, password, groupname, add_username = json['username'], json['password'], json['groupname'], json['add_username']
    if username not in USERS or add_username not in USERS:
        return 'User_does_not_exist', 400
    elif not validate_password(username, password):
        return 'Invalid_password', 401
    elif groupname not in GROUPS:
        return 'Group_does_not_exist', 400
    elif username != GROUPS[groupname]['admin']:
        return 'User_does_not_have_permission_to_modify_group',
        401
    else:
        GROUPS[groupname]['users'].add(add_username)
        log('GROUP', f"User_{add_username}'_added_to_group_{groupname}")
        return 'User_added_successfully', 200

# remove a user from a group
@app.route('/api/group/remove', methods=['POST'])
def group_remove():

```

```

is_valid, message, code = validate_request(request, ('
    username', 'password', 'groupname', 'rem_username'))
if not is_valid:
    return message, code
json = request.get_json()
username, password, groupname, rem_username = json['username',
    json['password'], json['groupname'], json['
    rem_username']
if username not in USERS or rem_username not in USERS:
    return 'User_does_not_exist', 400
elif not validate_password(username, password):
    return 'Invalid_password', 401
elif groupname not in GROUPS:
    return 'Group_does_not_exist', 400
elif username != GROUPS[groupname]['admin']:
    return 'User_does_not_have_permission_to_modify_group',
        401
elif username == rem_username:
    return 'Cannot_remove_admin_from_group', 401
elif rem_username not in GROUPS[groupname]['users']:
    return 'User_not_in_group', 400
else:
    GROUPS[groupname]['users'].remove(rem_username)
    log('GROUP', f"User '{rem_username}'_removed_from_group_
        '{groupname}'")
    return 'User_removed_successfully', 200

# join a group you're a member of
@app.route('/api/group/join', methods=['POST'])
def group_join():
    is_valid, message, code = validate_request(request, ('
        username', 'password', 'groupname'))
    if not is_valid:
        return message, code
    json = request.get_json()
    username, password, groupname = json['username'], json['
        password'], json['groupname']
    if username not in USERS:
        return 'User_does_not_exist', 400
    elif not validate_password(username, password):
        return 'Invalid_password', 401
    elif groupname not in GROUPS:
        return 'Group_does_not_exist', 400
    elif username not in GROUPS[groupname]['users']:

```



```

        return 'User_not_in_group', 401
    else:
        response = {}
        for member in GROUPS[groupname]['users']:
            if member != username:
                response[member] = USERS[member]['port']
        log('GROUP', f"User_{username}'_joined_group_{groupname}")
        return jsonify(response)

# send a message to a group
@app.route('/api/message/send', methods=['POST'])
def message_send():
    is_valid, message, code = validate_request(request, ('username', 'password', 'groupname', 'encrypted_messages'))
    if not is_valid:
        return message, code
    json = request.get_json()
    username, password, groupname, encrypted_messages = json['username'], json['password'], json['groupname'], json['encrypted_messages']
    if username not in USERS:
        return 'User_does_not_exist', 400
    elif not validate_password(username, password):
        return 'Invalid_password', 401
    elif groupname not in GROUPS:
        return 'Group_does_not_exist', 400
    elif username not in GROUPS[groupname]['users']:
        return 'User_not_in_group', 401
    else:
        GROUPS[groupname]['messages'].append({
            'sender': username,
            'timestamp': datetime.now(),
            'message': encrypted_messages
        })
        log('GROUP', f"User_{username}'_sent_a_message_to_group_{groupname}")
        for recipient in encrypted_messages:
            log('MESSAGE', f"Encrypted_message_to_user_{recipient}':_{encrypted_messages[recipient]}")
        return 'Message_sent_successfully', 200

```

```

# get the messages for a group
@app.route('/api/message/get', methods=['POST'])
def message_get():
    is_valid, message, code = validate_request(request, ('
        username', 'password', 'groupname'))
    if not is_valid:
        return message, code
    json = request.get_json()
    username, password, groupname = json['username'], json['
        password'], json['groupname']
    if username not in USERS:
        return 'User_does_not_exist', 400
    elif not validate_password(username, password):
        return 'Invalid_password', 401
    elif groupname not in GROUPS:
        return 'Group_does_not_exist', 400
    elif username not in GROUPS[groupname]['users']:
        return 'User_not_in_group', 401
    else:
        return jsonify(GROUPS[groupname]['messages'])

# check for valid password
def validate_password(username, password):
    return hash_password(password) == USERS[username]['password'
    ]

# get the hash of a password
def hash_password(password):
    return hashlib.sha256(password.encode('utf-8')).digest()

# ensure request is correctly formatted
def validate_request(req, params):
    if not req.is_json:
        return False, 'Invalid_request:_not_JSON', 400
    json = req.get_json()
    for param in params:
        if param not in json:
            return False, 'Invalid_request:_missing_parameters',
            400
    return True, None, None

```

```

# log to console
def log(log_type, text):
    print(f'[{log_type}] {text}')

if __name__ == '__main__':
    import argparse
    argp = argparse.ArgumentParser()
    argp.add_argument('port', type=int, help='port to host the
        server on')
    args = argp.parse_args()
    app.run(port=args.port)

```