LAPORAN TUGAS UDP SOCKET PROGRAMMING

MATA KULIAH II 2120

Kelas 02 / Kelompok 19

Dosen Pengampu

Ir. I Gusti Bagus Baskara Nugraha, S.T., M.T., Ph.D.

Disusun oleh

Nama	NIM
Mudzaki Kaarzaqiel Hakim	18223024
Henrycus Hugatama Risaldy	18223008

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

OKTOBER 2024

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Laporan Tugas Jaringan Komputer

A. Deskripsi Persoalan

Pada tugas ini, kami diminta untuk membuat suatu aplikasi *chat room* sederhana yang memungkinkan pengguna-pengguna untuk bertukar pesan secara daring dan instan. Aplikasi ini kami buat dengan menggunakan bahasa pemrograman Python 3.11.4 dan mengimplementasikan protokol *transport* UDP, yaitu salah satu protokol yang digunakan dalam lapisan transport pada model OSI. UDP mengirimkan data dalam bentuk datagram, yang merupakan unit data yang terpisah dan dikirim secara independen juga sehingga pengirimannya lebih cepat dan lebih efisien dibandingkan dengan protokol lainnya seperti TCP, walaupun pada UDP jaminan urutan pengiriman atau keberhasilan pengiriman lebih rendah bila dibandingkan dengan TCP.

B. Daftar Pembagian Tugas

Table 1. Tabel pembagian tugas:

No	Spesifikasi	NIM
1	Server mampu menerima pesan yang dikirim client dan mencetaknya ke layar.	18223008
2	Server mampu meneruskan pesan satu client ke client lain.	18223008
3	Client mampu mengirimkan pesan ke server dengan IP dan port yang ditentukan pengguna.	18223008
4	Client mampu menerima pesan dari client lain (yang diteruskan oleh server), dan mencetaknya ke layar.	18223024
5	Client harus memasukkan *password* untuk dapat bergabung ke chatroom.	18223024
6	Client memiliki username yang unik.	18223024
7	Seluruh pesan dienkripsi menggunakan algoritma kriptografi modern asimetris, misal cipher RSA, atau kombinasi algoritma kriptografi modern asimetris dan modern simetris.	18223008
8	Aplikasi mampu menyimpan pesan-pesan lampau meskipun telah ditutup; mekanisme dan tempat penyimpanan bebas, baik di *client* maupun di *server*.	18223024
9	Aplikasi mampu mengotentikasi pengguna.	18223024

C. Daftar Penyelesaian Spesifikasi

Table 2. Tabel spesifikasi dan juga status penyelesaiannya:

No	Spesifikasi	Selesai
1	Server mampu menerima pesan yang dikirim client dan mencetaknya ke layar.	✔

2	Server mampu meneruskan pesan satu client ke client lain.	✔
3	Client mampu mengirimkan pesan ke server dengan IP dan port yang ditentukan pengguna.	✔
4	Client mampu menerima pesan dari client lain (yang diteruskan oleh server), dan mencetaknya ke layar.	✔
5	Client harus memasukkan *password* untuk dapat bergabung ke chatroom.	✔
6	Client memiliki username yang unik.	✔
7	Seluruh pesan dienkripsi menggunakan algoritma kriptografi modern asimetris, misal cipher RSA, atau kombinasi algoritma kriptografi modern asimetris dan modern simetris.	✔
8	Aplikasi mampu menyimpan pesan-pesan lampau meskipun telah ditutup; mekanisme dan tempat penyimpanan bebas, baik di *client* maupun di *server*.	✔
9	Aplikasi mampu mengotentikasi pengguna.	✔

D. Cara Kerja Program

Program chat ini bekerja dengan memanfaatkan protokol UDP. UDP (User Datagram Protocol) adalah protokol komunikasi alternatif dari Transmission Control Protocol (TCP).Protokol UDP bertujuan untuk menciptakan komunikasi dengan paket berukuran header kecil dan cepat, UDP memiliki keunggulan berupa kecepatan dibandingkan dengan TC dikarenakan UDP menerapkan komunikasi connectionless oriented (Kurose dan Ross, 2017). . Pertama-tama server akan diikat menggunakan socket ke suatu IP address dan port tertentu. Implementasi dari pemanfaatan protokol UDP dapat dilihat pada tangkapan layar kode berikut:

```
def RunServer() -> None:
    host = '127.0.0.1'
    port = 9999

    global s
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.bind((host, port))
```

Figure 1. Tangkapan layar kode

Kode diatas menunjukan bagaimana cara mengikat server ke IP dan port yang ditentukan dalam contoh tersebut menggunakan localhost dan port 9999. Baris kode `s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)` menunjukan bahwa inisiasi socket menggunakan alamat IPv4 dan protokol UDP.

Kemudian setelah server terikat ke suatu IP dan port server akan men-generate sebuah public key dan private key untuk enkripsi pesan serta password untuk masuk ke room chat. Server memiliki mekanisme autentikasi dimana klien harus berhasil mengirimkan password room chat yang benar dan jika salah maka akan muncul pesan agar pengguna mengisi password kembali. Setelah password terautentikasi server akan mengirimkan public dan private key ke klien agar pesan dapat dienkripsi.

Setelah autentikasi password room chat maka client akan diberikan dua opsi yaitu ‘register’ dan ‘login’. Jika klien sudah memiliki akun sebelumnya, klien dapat memilih login dan memasukkan nama serta ID untuk diverifikasi oleh server. Namun jika klien belum memiliki akun klien dapat memilih opsi register yang mana akan memasukan nama username dan menerima 4 digit ID yang di-generate oleh server.

Ketika proses autentikasi nama dan ID telah selesai maka klien dapat bergabung ke dalam room chat. Pada room chat pesan yang dikirimkan oleh klien ke server akan di enkripsi menggunakan algoritma RSA cipher. RSA Cipher adalah metode enkripsi asimetris yang mengamankan data menggunakan dua kunci: kunci publik untuk enkripsi dan kunci privat untuk dekripsi, dengan keamanan berdasarkan sulitnya memfaktorkan bilangan besar hasil perkalian dua buah bilangan prima (Zhou & Tang, 2011). Kemudian dengan algoritma yang sama server mendekripsi pesan tersebut lalu kemudian menyebarkannya ke client yang lainnya. Pada sisi server juga dilakukan pencatatan pesan yang terenkripsi pada ‘log_encrypted.txt’ dan pesan yang sudah terdekripsi pada ‘log_server.txt’.

Pada sisi klien, program akan meminta input alamat IP dan port server serta melakukan pengecekan error ketika proses penyambungan ke server dan pesan akan terus berulang hingga akhirnya koneksi ke server sudah berhasil. Setelah klien menerima kunci publik dari server maka klien akan dapat menggunakan kunci tersebut untuk melakukan enkripsi dengan algoritma RSA cipher. Program klien juga memiliki thread yang memungkinkan klien untuk terus menerima pesan dari server tanpa harus menunggu pengguna selesai memberi input. Pengguna dapat keluar dari room chat dengan menulis pesan ‘qqq’ dan akan otomatis keluar dan akan terhapus dari daftar pengguna aktif.

E. Pengujian

1. Tata Cara dan Lingkungan Pengujian

Program diuji dengan cara menghubungkan dua buah perangkat, dalam kasus ini satu perangkat dengan sistem operasi Mac OS Sonoma 14.5 dan satu perangkat dengan sistem operasi Windows 11. Kedua perangkat dihubungkan dengan internet yang memiliki kecepatan download 12.68 Mbps dan kecepatan upload 8.09 Mbps seperti yang tertera pada gambar berikut.

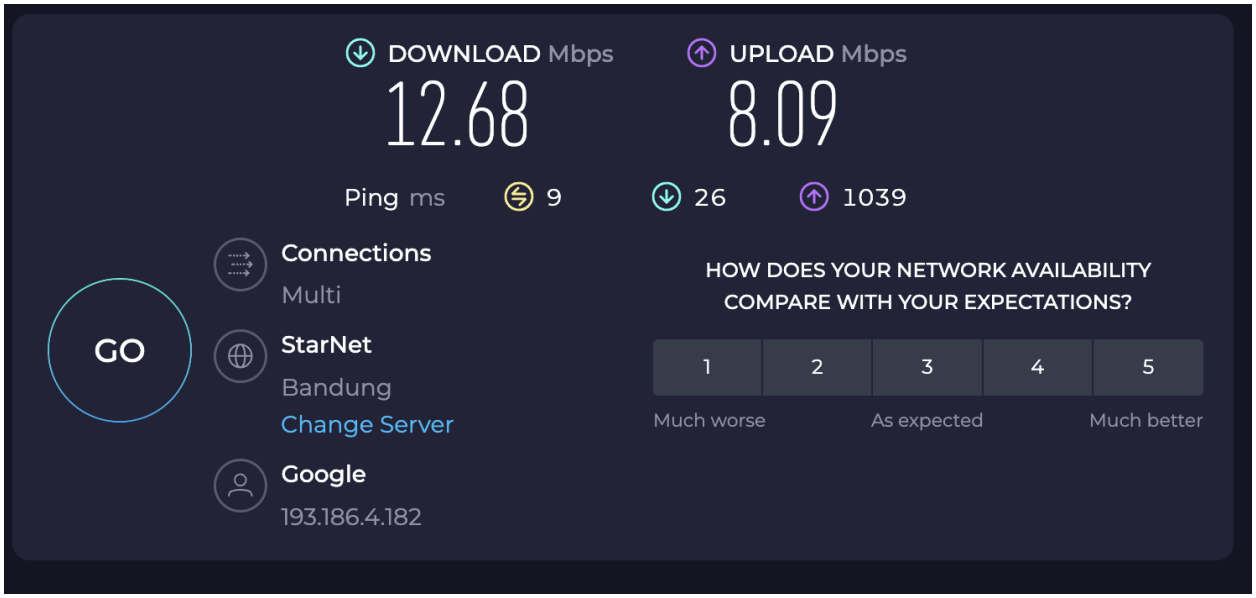


Figure 2. Kecepatan internet yang digunakan

Alasan dari penggunaan internet ini, adalah karena ketika kami mencoba menggunakan *personal hotspot* dari *handphone*, pesan terkirim dari *client* ke *server* dengan waktu yang cukup lama dan tidak konsisten, hal yang sama juga terjadi ketika pengiriman pesan dari *server* ke *client*. Bahkan, tidak sedikit terjadi kegagalan *client* untuk terhubung ke server. Namun, ketika kami mengganti internet yang digunakan menggunakan router, pesan terkirim jauh lebih cepat dan jauh lebih jarang terjadi kegagalan *client* terhubung ke *server*.

Untuk tahapan pengujiannya sendiri, dilakukan sedikit modifikasi pada bagian ip host, yang mana awalnya diset sebagai '127.0.0.1' atau localhost, diganti menjadi IPv4 Address yang di-*assign* kepada perangkat yang digunakan untuk menjalankan server oleh router. Setelah itu, *source code* untuk *server* dijalankan pada perangkat *server* dan program akan memberikan kata sandi untuk masuk ke *chat room* pada terminal. Setelah itu, *source code* untuk *client* dapat dijalankan pada perangkat *client*, dan pada terminalnya akan keluar *prompt* untuk memasukkan ip dari server (dalam kasus ini, ip yang sudah dimodifikasi di awal tadi) dan juga port yang digunakan. Kami menggunakan port 9999, pemilihan ini dikarenakan port tersebut tidak digunakan oleh *protokol* atau layanan lain, sehingga meminimalisir kemungkinan terjadinya konflik dengan layanan lainnya.

Jika ip address dan port sesuai, maka akan terbentuk suatu koneksi antara *server* dengan *client*. Hal ini dapat diketahui dari jenis pesan yang dikirimkan oleh *server* ke terminal *client* setelah meng-*input* server ip dan juga server port. Jika pesan yang ditampilkan berupa "Berhasil terhubung ke server." maka pengguna bisa lanjut meng-*input* password dan kemudian log-in menggunakan akun dan id yang sudah ada atau *register* dengan menggunakan username baru yang tidak boleh sama dengan username yang sudah tercatat pada database server. Setelah itu, maka *client* dapat mulai mengirimkan pesan. Namun, bila pesan yang ditampilkan pada terminal *client* setelah melakukan *input* ip address server dan juga portnya adalah "Error: Tidak dapat terhubung ke server. Pastikan alamat dan port benar." berarti terjadi *timeout* dan *client* harus mengulang *input* ip address dan juga port hingga benar.

Pada bagian selanjutnya, terdapat tangkapan ketika dilakukan pengujian. Tangkapan antarmuka untuk *error handling* terdapat pada lampiran.

2. Tangkapan Antarmuka Pengujian

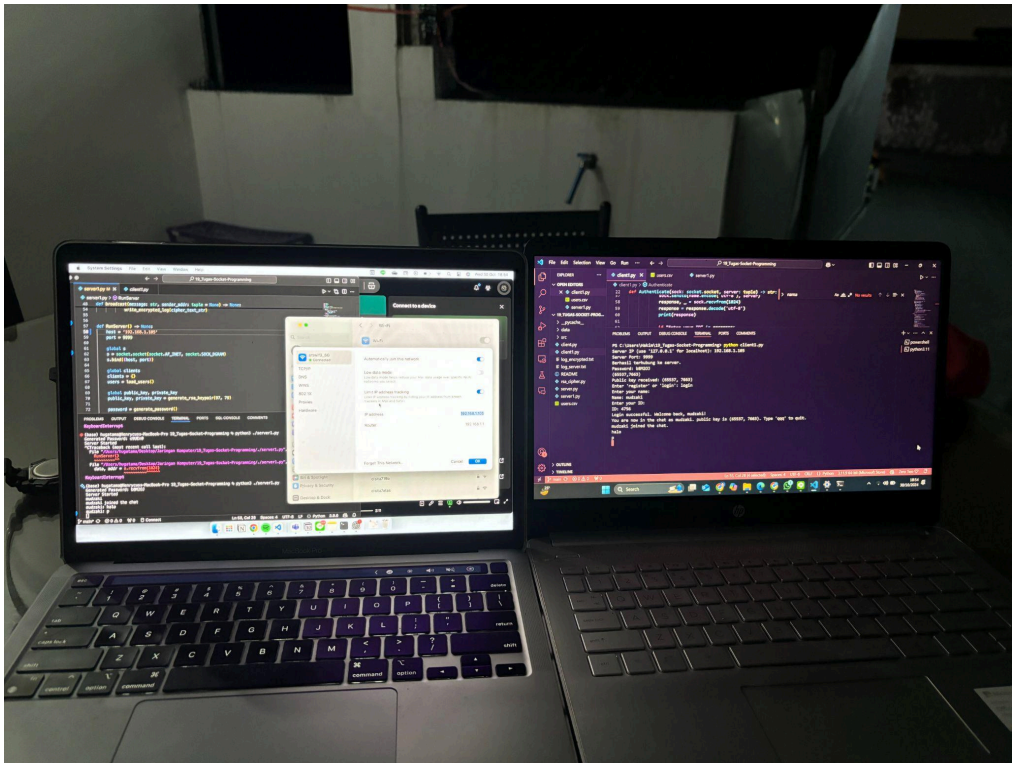


Figure 3. *Set-up* secara keseluruhan

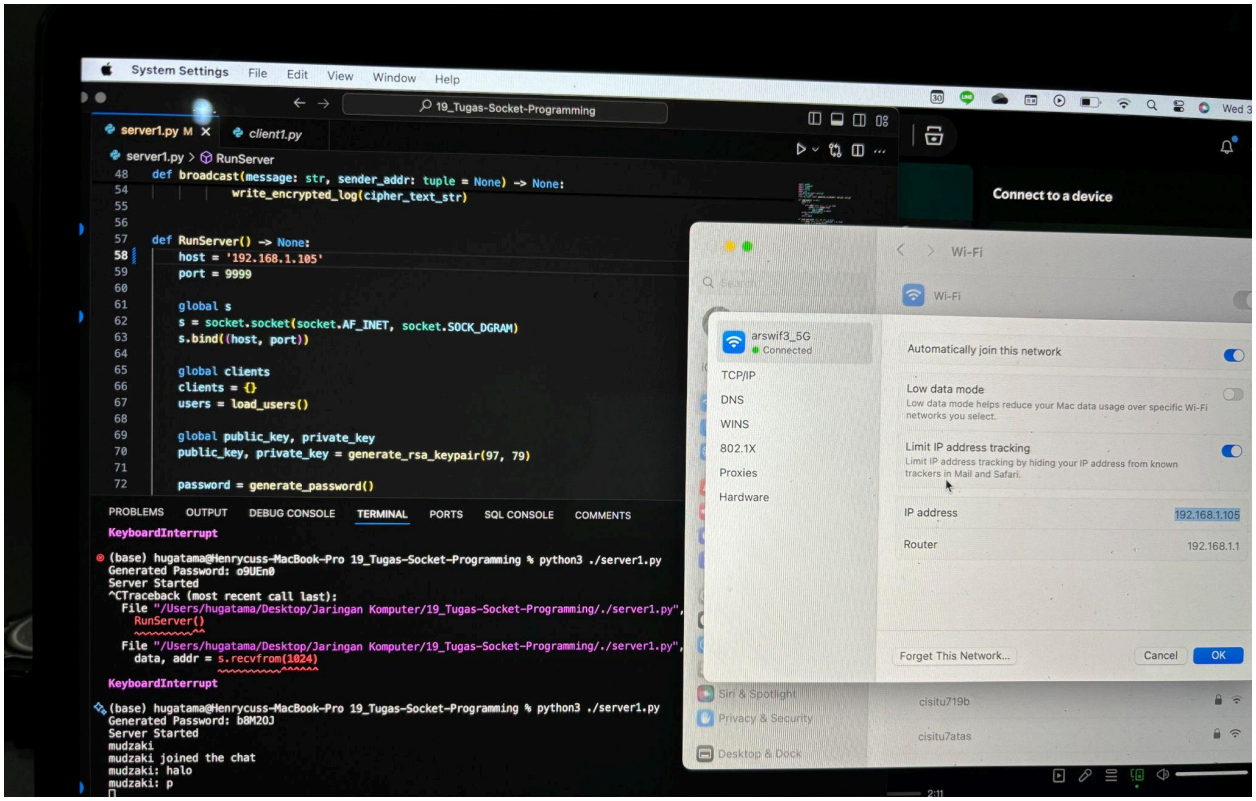


Figure 4. Tampilan antarmuka pada server

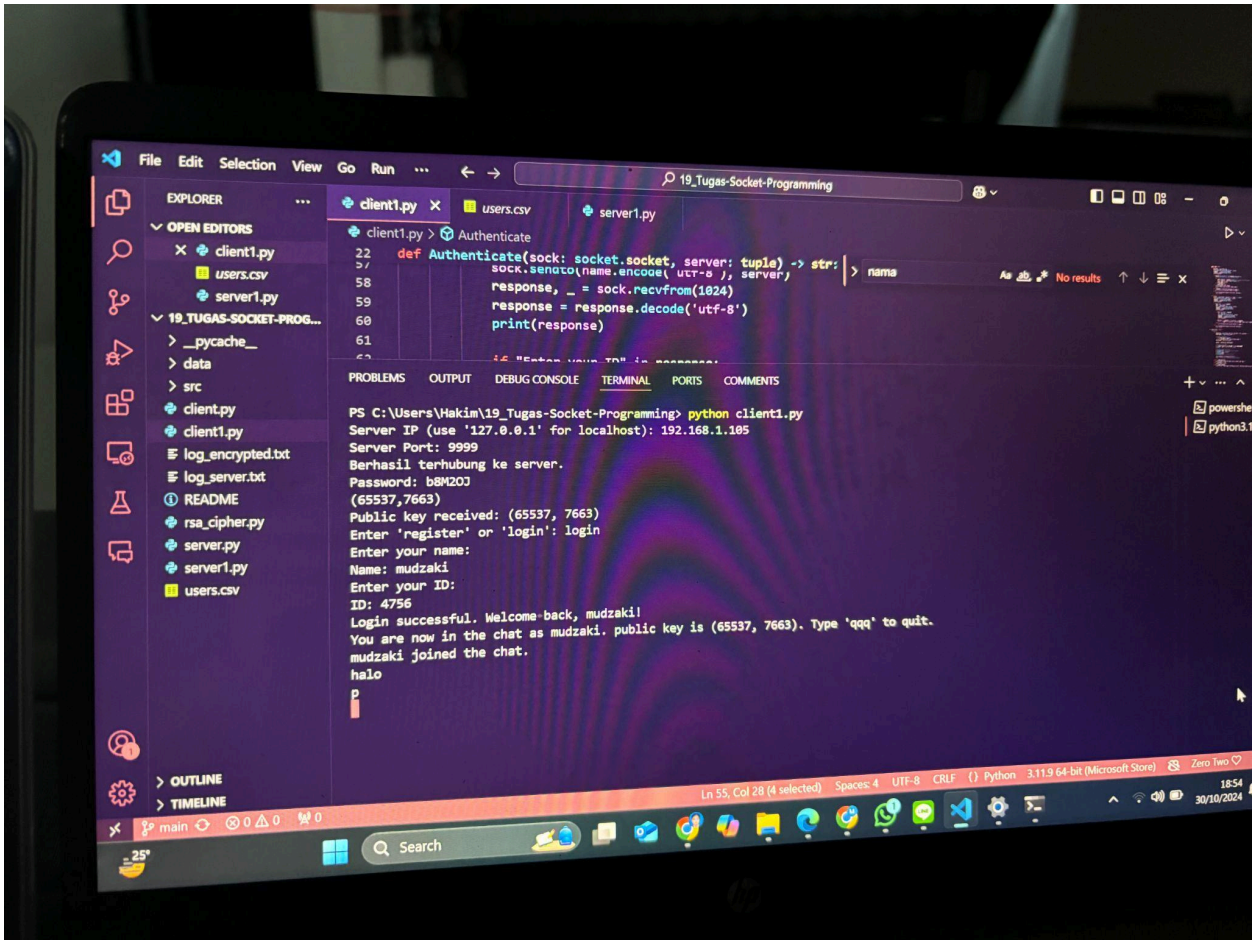


Figure 5. Tampilan antarmuka pada client

F. Source Code

1. Source Code server1.py

```
import socket
import threading
import random
import string
import csv
from datetime import datetime
import rsa_cipher
from rsa_cipher import generate_rsa_keypair, decrypt, encrypt

def load_users() -> dict:
    """
    Memuat data user dari file CSV sebagai dictionary
    """

    users = {}
```

```
    try:
        with open('users.csv', 'r') as file:
            reader = csv.reader(file)
            for row in reader:
                users[row[0]] = row[1]
    except FileNotFoundError:
        pass
    return users

def save_user(name: str, id: str) -> None:
    """
    Menyimpan data pengguna baru berupa nama dan ID ke dalam file CSV
    """

    with open('users.csv', 'a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([name, id])

def generate_id() -> str:
    """
    Menghasilkan ID unik untuk pengguna baru
    """

    return str(random.randint(1000, 9999))

def write_log(message: str) -> None:
    """
    Menulis log chat room dengan timestamp ke dalam file .txt
    """

    log_file = "log_server.txt"
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    log_message = f"[{timestamp}] {message}\n"

    with open(log_file, "a") as file:
        file.write(log_message)

def write_encrypted_log(encrypted_message: str) -> None:
    """
    Menulis log pesan terenkripsi dengan timestamp
    """
```

```
    log_file = "log_encrypted.txt"
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    log_message = f"[{timestamp}] {encrypted_message}\n"

    with open(log_file, "a") as file:
        file.write(log_message)

def generate_password(length: int = 6) -> str:
    """
    Menghasilkan password untuk bergabung dengan chat room
    """

    return ''.join(random.choices(string.ascii_letters + string.digits, k=length))

def broadcast(message: str, sender_addr: tuple = None) -> None:
    """
    Mengirimkan pesan ke semua client kecuali pengirim
    """

    for client in clients:
        if client != sender_addr:
            s.sendto(message.encode('utf-8'), client)
            encrypted_message = rsa_cipher.encrypt(public_key, message)
            cipher_text_str = " ".join(map(str, encrypted_message))
            write_encrypted_log(cipher_text_str)

def RunServer() -> None:
    """
    Menjalankan server UDP
    """

    host = '127.0.0.1'
    port = 9999

    global s
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.bind((host, port))

    global clients
    clients = {}
    users = load_users()
```

```
    global public_key, private_key
    public_key, private_key = generate_rsa_keypair(97, 79)

    password = generate_password()
    print(f"Generated Password: {password}")
    write_log(f"Server started. Password: {password}")
    print("Server Started")

    while True:
        try:
            data, addr = s.recvfrom(1024)
            data = data.decode('utf-8')

            if addr not in clients:
                if data == password:
                    s.sendto(f"({public_key[0]},{public_key[1]})".encode('utf-8'),
addr)
                    clients[addr] = None  # Menandai bahwa klien telah melewati tahap
password
                else:
                    s.sendto("Incorrect password. Try again.".encode('utf-8'), addr)
            elif addr in clients and clients[addr] is None:
                if data.lower() == 'register':
                    s.sendto("Enter your name:".encode('utf-8'), addr)
                    clients[addr] = 'registering'
                elif data.lower() == 'login':
                    s.sendto("Enter your name:".encode('utf-8'), addr)
                    clients[addr] = 'logging_in'
                else:
                    s.sendto("Invalid command. Use 'register' or
'login'.".encode('utf-8'), addr)
            elif addr in clients and clients[addr] == 'registering':
                name = data
                if name in users:
                    s.sendto("Name already exists. Try logging in.".encode('utf-8'),
addr)
                    clients[addr] = None
                else:
                    id = generate_id()
                    users[name] = id
                    save_user(name, id)
                    clients[addr] = name
```

```
                    s.sendto(f"Registered. Your ID is {id}. Welcome
{name}".encode('utf-8'), addr)
                    write_log(f"{name} joined the chat")
                    print(f"{name} joined the chat")
                    broadcast(f"{name} joined the chat.")
            elif addr in clients and clients[addr] == 'logging_in':
                name = data
                if name not in users:
                    s.sendto("Name not found. Try registering.".encode('utf-8'), addr)
                    clients[addr] = None
                else:
                    s.sendto("Enter your ID:".encode('utf-8'), addr)
                    clients[addr] = ('verifying', name)
            elif addr in clients and isinstance(clients[addr], tuple) and
clients[addr][0] == 'verifying':
                _, name = clients[addr]
                id = data
                if users[name] == id:
                    clients[addr] = name
                    s.sendto(f"Login successful. Welcome back,
{name}!".encode('utf-8'), addr)
                    write_log(f"{name} joined the chat")
                    print(f"{name} joined the chat")
                    broadcast(f"{name} joined the chat.")
                else:
                    s.sendto("Incorrect ID. Try again.".encode('utf-8'), addr)
                    clients[addr] = None
            else:
                name = clients[addr]
                encrypted_message_str = data.replace('[', '').replace(']',
'').replace(',', '')
                write_encrypted_log(encrypted_message_str)
                encrypted_message = list(map(int, encrypted_message_str.split()))
                decrypted_message = decrypt(private_key, encrypted_message)
                message = f"{name}: {decrypted_message}"
                broadcast(message, addr)
                write_log(message)
                print(message)

                if decrypted_message.lower() == 'qqq':
                    write_log(f"{name} left the chat")
                    print(f"{name} left the chat")
```

```
                broadcast(f"{name} left the chat")
                del clients[addr]

        except Exception as e:
            print(f"An error occurred: {e}")
            write_log(f"Error: {e}")
            print(e)
            print("Server stopped")
```

2. Source Code client1.py

```
import socket
import threading
import rsa_cipher
from rsa_cipher import encrypt, decrypt

def ReceiveData(sock: socket.socket, private_key: tuple) -> None:
    """
    Menerima data dari socket dan melakukan dekripsi pesan dengan private key
    """

    while True:
        try:
            data, addr = sock.recvfrom(1024)
            data = data.decode()
            if data.startswith('[') and data.endswith(']'):
                encrypted_message_str = data.decode('utf-8')
                encrypted_message = list(map(int, encrypted_message_str.split()))  #
Convert back to list of integers
                decrypted_message = decrypt(private_key, encrypted_message)
                print(decrypted_message)
            else:
                print(data)
        except Exception as e:
            print(f"Error receiving data: {e}")
            break

def Authenticate(sock: socket.socket, server: tuple) -> str:
    """
    Melakukan otentikasi pengguna dan memberikan akses ke chatroom serta public key
jika berhasil
    """
```

```
public_key = None
while True:
    password = input("Password: ")
    sock.sendto(password.encode('utf-8'), server)
    response, _ = sock.recvfrom(1024)
    response = response.decode('utf-8')
    print(response)

    if response.startswith("(") and response.endswith(")"):
        # Parsing tuple dari string
        try:
            public_key_str = response.strip("()")
            e, n = map(int, public_key_str.split(","))
            public_key = (e, n)
        except ValueError:
            public_key = None
    if isinstance(public_key, tuple):
        break
    elif response == "Incorrect password. Try again.":
        print("Incorrect password. Please try again.")
    else:
        print("Unexpected response from server. Exiting.")
        return False

while True:
    while True:
        action = input("Enter 'register' or 'login': ").lower()
        if action in ['register', 'login']:
            sock.sendto(action.encode('utf-8'), server)
            break
        else:
            print("Invalid command. Please enter 'register' or 'login'.")
    response, _ = sock.recvfrom(1024)
    response = response.decode('utf-8')
    print(response)

    if "Enter your name" in response:
        name = input("Name: ")
        sock.sendto(name.encode('utf-8'), server)
        response, _ = sock.recvfrom(1024)
        response = response.decode('utf-8')
        print(response)
```

```
            if "Enter your ID" in response:
                id = input("ID: ")
                sock.sendto(id.encode('utf-8'), server)
                response, _ = sock.recvfrom(1024)
                response = response.decode('utf-8')
                print(response)

            if "Welcome" in response or "successful" in response:
                return f"Logged in successfull, welcome {name}. Public key is
{public_key}"
            else:
                print("Authentication failed. Please try again.")
        else:
            print("Unexpected response from server. Exiting.")
            return False

def RunClient() -> None:
    """
    Mencoba koneksi ke server, melakukan otentikasi dan bisa mulai mengirim/menerima
pesan
    """

    while True:
        try:
            server_ip = input("Server IP: ")
            server_port = input("Server Port: ")

            # Validasi input port
            if not server_port.isdigit():
                raise ValueError("Port must be a number.")
            server_port = int(server_port)

            server = (server_ip, server_port)

            # Membuat socket UDP
            s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            s.bind(('', 0))  # Bind ke port random

            # Mencoba koneksi ke server
            s.settimeout(5)  # Set timeout untuk koneksi
            s.sendto(b'test', server)
```

```
            s.recvfrom(1024)
            s.settimeout(None)  # Reset timeout

            print("Connected to server.")
            break  # Keluar dari loop jika koneksi berhasil

        except socket.gaierror:
            print("Error: Invalid server IP. please try again.")
        except ValueError as ve:
            print(f"Error: {ve}")
        except socket.timeout:
            print("Error: can't connect to server. Please try again.")
        except Exception as e:
            print(f"Error {e}")

        print("Please Try Again\n")

    auth_response = Authenticate(s, server)
    if not auth_response:
        s.close()
        return

    public_key_str = auth_response.split("Public key is ")[1]
    e, n = map(int, public_key_str.strip("()").split(", "))
    public_key = (e, n)

    name = auth_response.split("welcome ")[1]
    print(f"You are now in the chat as {name}. Type 'qqq' to quit.")
    threading.Thread(target=ReceiveData, args=(s, public_key), daemon=True).start()

    while True:
        data = input()
        if data.lower() == 'qqq':
            break
        elif data == '':
            continue
        message = encrypt(public_key, data)
        s.sendto(str(message).encode('utf-8'), server)

    s.sendto('qqq'.encode('utf-8'), server)
    s.close()
    print("Disconnected from server.")
```

DAFTAR PUSTAKA

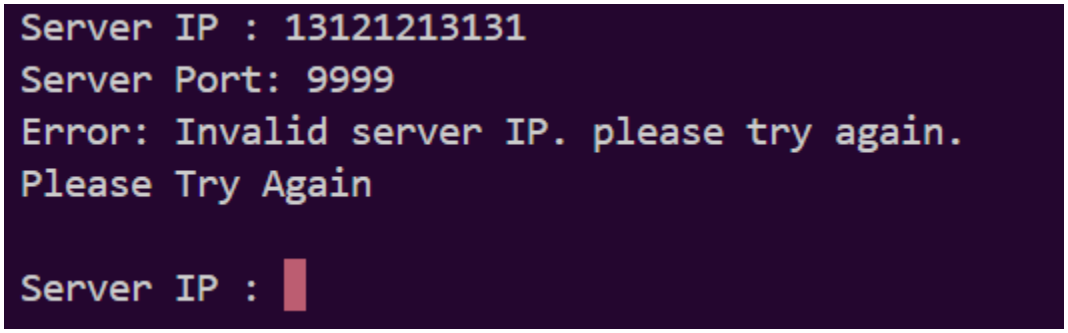
Kurose, J.F. & Ross, K.W., 2017. Computer Networking A Top-Down Approach. 7th ed. New Jersey: Pearson

Xin Zhou and Xiaofei Tang, "Research and implementation of RSA algorithm for encryption and decryption," Proceedings of 2011 6th International Forum on Strategic Technology, Harbin, Heilongjiang, 2011, pp. 1118-1121, doi: 10.1109/IFOST.2011.6021216. keywords: {Cryptography;Algorithm design and analysis;RSA algorithm;encryption;decryption},

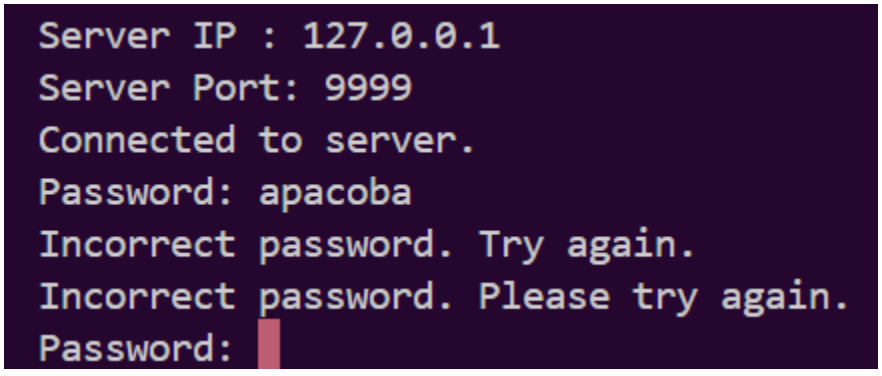
LAMPIRAN

Tampilan antarmuka ketika port salah

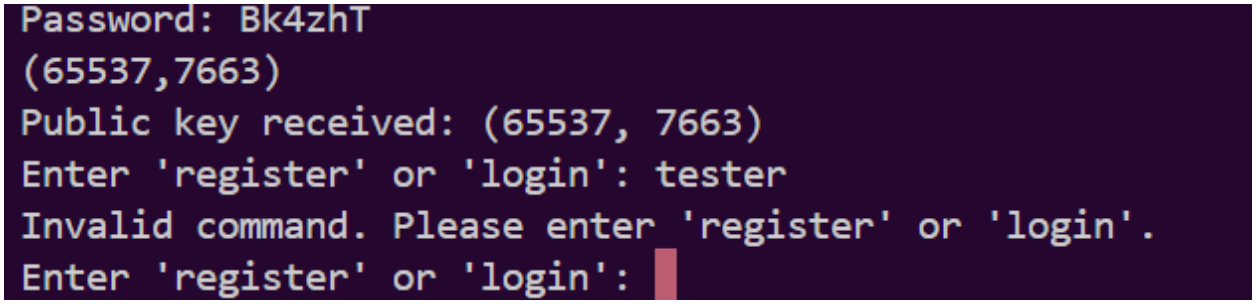
Tampilan antarmuka ketika gagal membuat koneksi ke server



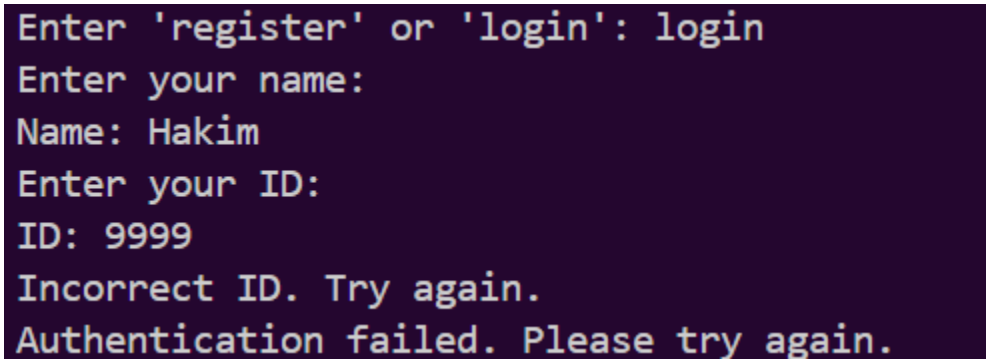
Tampilan antarmuka ketika ip tidak sesuai format



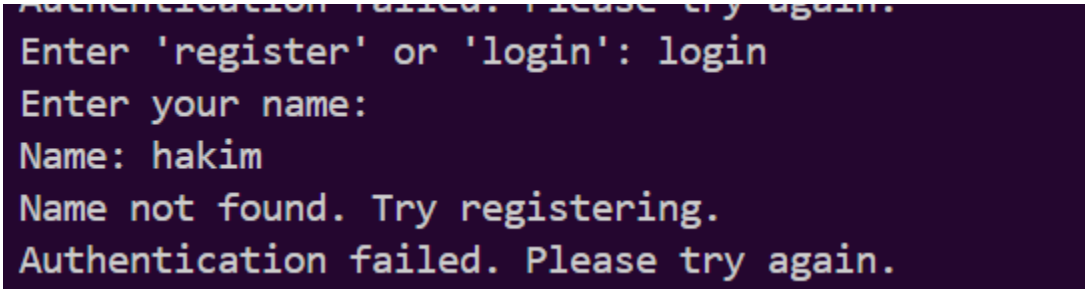
Tampilan antarmuka ketika password salah



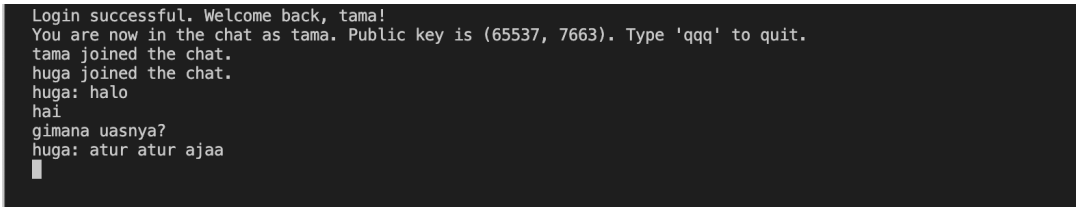
Tampilan antarmuka ketika user menginput string selain ‘register’ dan ‘login’



Tampilan antarmuka ketika user salah ID

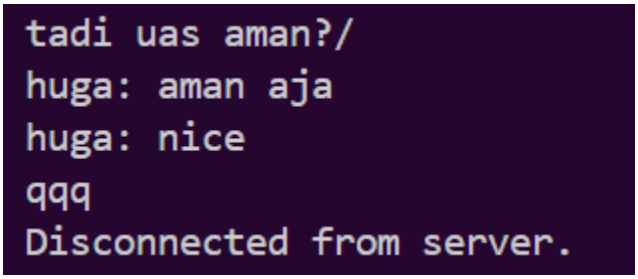


Tampilan antarmuka ketika user login namun belum pernah register



Tampilan antarmuka ketika 2 user

Tampilan antarmuka ketika 2 user dari sisi server



Tampilan antarmuka ketika user keluar dengan mengetik ‘qqq’