CS3357A COMPUTER NETWORKING

Assignment #4: Building a multi-player server-client snake game.

Assignment Purpose

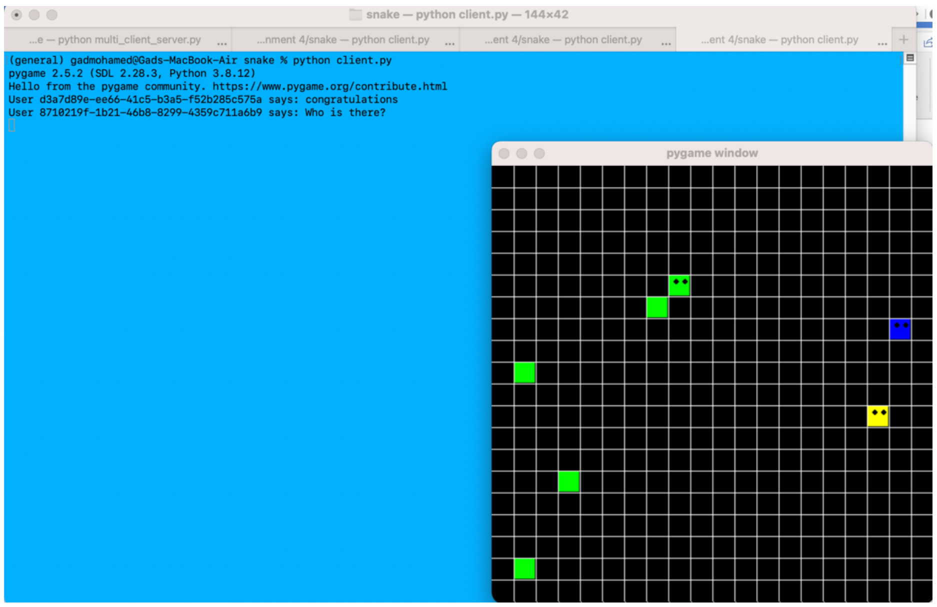
The goal of this assignment is to extend the functionality of the previously developed Snake game to support three additional features:

1- The server should handle multiple clients/snakes simultaneously. Each client will have its own snake, enabling a multiplayer environment where multiple snakes share the field. Each client will receive a game state that includes the position of all snakes to display them. You can decide colors. Check the "Communication Protocol and Game State" section for instruction on implementing the multi-player feature.

2- While playing, Clients can send a public message to the server to be broadcasted to all clients. Check the chatting section for instruction on implementing the public messaging part.

3- Both the server and the clients encrypt their messages using RSA encryption algorithm. In the RSA algorithm, each party generates a key pair (public key, private key), where the private key is kept as a secret at the sender and is used to encrypt the sender messages, and the public key is publicly shared with the receiver and is used to decrypt the messages at the receiver side. Check the Encryption section for instruction on implementing messages and control inputs encryption.

The image below shows the game window that should be displayed on the client side in this assignment.


snake — python client.py — 144×42
...e — python multi_client_server.py
...nment 4/snake — python client.py
...ent 4/snake — python client.py
...ent 4/snake — python client.py
pygame window

```
(general) gadmohamed@Gads-MacBook-Air snake % python client.py
pygame 2.5.2 (SDL 2.28.3, Python 3.8.12)
Hello from the pygame community. https://www.pygame.org/contribute.html
User d3a7d89e-ee66-41c5-b3a5-f52b285c575a says: congratulations
User 8710219f-1b21-46b8-8299-4359c711a6b9 says: Who is there?
```

Figure 1 A screenshot showing the graphical interface of the snake game on the client side, containing three snakes and 4 snacks. The screenshot also shows the terminal of one of the three connected clients with messages exchanged between clients.

The server is responsible for:

1. Accepting multiple client connections.
2. Managing the game's logic for all connected clients.
3. Broadcasting the game state to all clients.
4. Receiving and parsing each client's input commands.
5. Applying the received game controls to update the game state.
6. Receive public messages from clients and broadcast them to all clients.
7. Encrypting outgoing public messages.
8. Decrypting incoming controls inputs and public messages.

The snake client is responsible for:

1. Connecting to the server.
2. Sending control inputs to the server.
3. Receive and parse the game state.
4. Displaying the game interface with all the snakes based on the received game state.
5. Send public messages to the server.
6. Receive the public messages broadcasted by the server and display them on the terminal.
7. Encrypts outgoing control inputs and public messages before sending them to the server.
8. Decrypts incoming public messages received from the server.

Assignment Description

Modify the snake_server.py and snake_client.py files of assignment 3 to support handling multiple players, RSA-based secure communication, and public messaging between clients. Each client will be assigned a snake where he will be able to control it by sending encrypted control inputs to the server. On the other hand, the server manages the game logic including recording snakes and snacks positions; applying control inputs; and returning the game state.

In the previous assignment (assignment #3), you were given the server side code which is separated into two files: snake.py and snake_server.py, and you had to implement the client side code. These files will be used as the starting code for this assignment.
In this assignment, no new files are given, but you are required to use and modify the server/client code files of the previous assignments to support the new functionalities.

As this assignment builds on previous ones, the following diagrams illustrate the connection between assignment 2, assignment 3, and assignment 4:

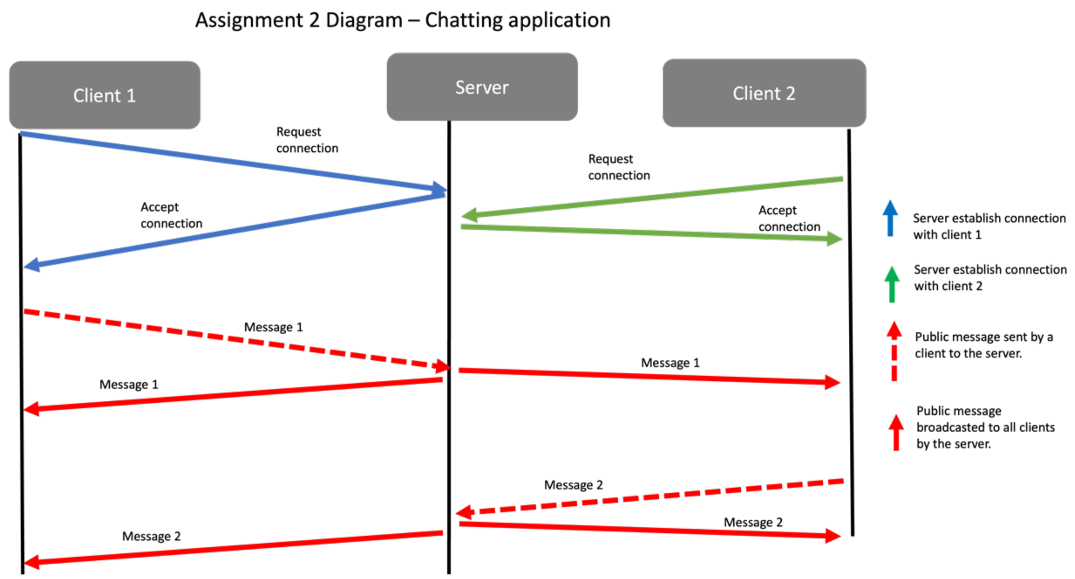

Assignment 2 Diagram – Chatting application
Client 1
Server
Client 2
Request connection
Accept connection
Request connection
Accept connection
Message 1
Message 1
Message 1
Message 2
Message 2
Message 2
Server establish connection with client 1
Server establish connection with client 2
Public message sent by a client to the server.
Public message broadcasted to all clients by the server.

Figure 2 In the chatting application of assignment 2, the server can handle concurrent client connections. A connected client can send a public message to the server to be broadcasted to all clients. No encryption is used.

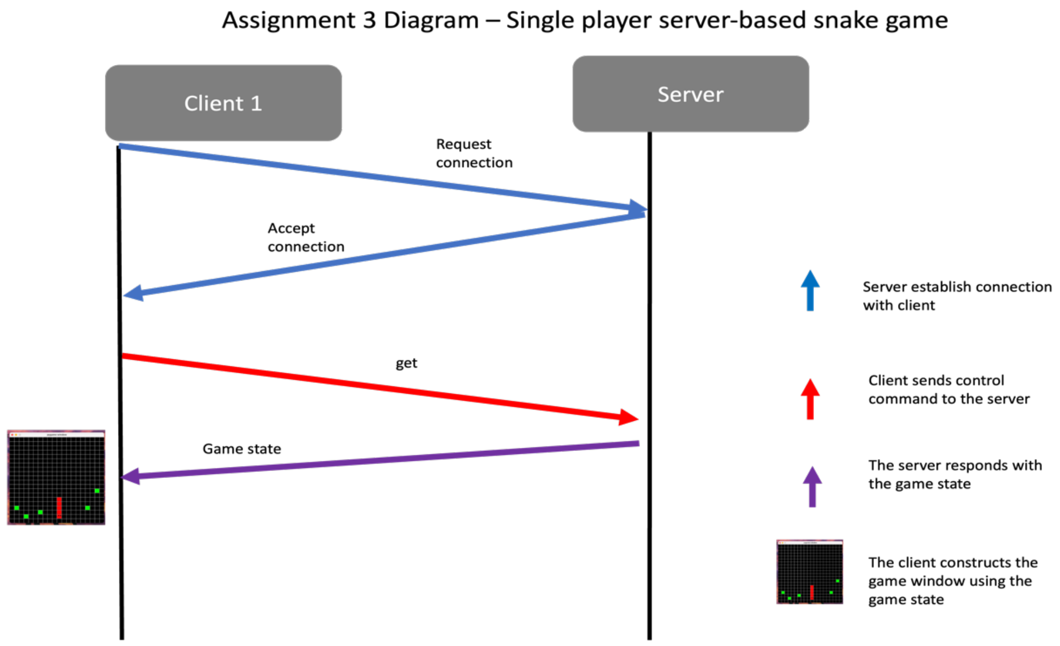

Assignment 3 Diagram – Single player server-based snake game
Client 1
Server
Request connection
Accept connection
get
Game state
Server establish connection with client
Client sends control command to the server
The server responds with the game state
The client constructs the game window using the game state

Figure 3 In the single-player snake game of assignment 3, the server can accept a connection from a single client. Each cycle, the client will send a control input, for example "get", and the server will reply with the game state. Finally, the client uses the game state to draw the game display. No encryption is applied; the server and client send data as plain text.

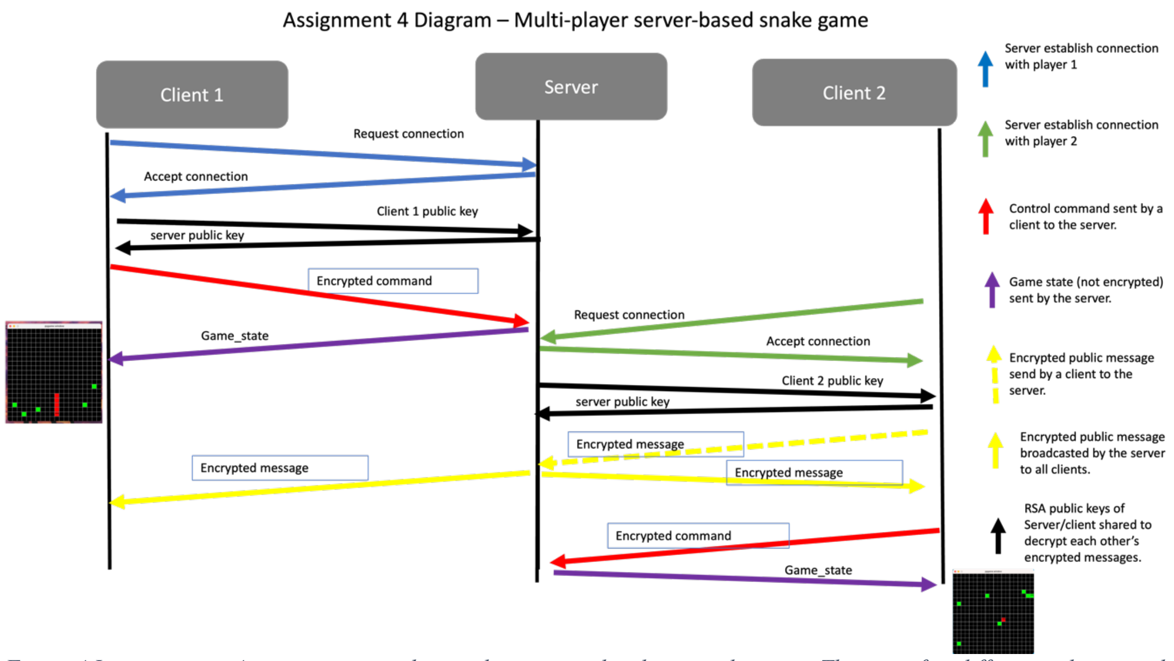

Assignment 4 Diagram – Multi-player server-based snake game
Client 1
Server
Client 2
Request connection
Accept connection
Client 1 public key
server public key
Encrypted command
Game_state
Request connection
Accept connection
Client 2 public key
server public key
Encrypted message
Encrypted message
Encrypted message
Encrypted command
Game_state
Server establish connection with player 1
Server establish connection with player 2
Control command sent by a client to the server.
Game state (not encrypted) sent by the server.
Encrypted public message send by a client to the server.
Encrypted public message broadcasted by the server to all clients.
RSA public keys of Server/client shared to decrypt each other's encrypted messages.

Figure 4 In assignment 4, you are required to implement a multi-player snake game. There are few differences between this version of the game and the version implemented in the previous assignment. First, the server can accept concurrent client connections. Once a client connects to the server, the server deploys a new snake and assigns it to that user. Second, in addition to control inputs, clients can send public messages. Third, control inputs and public messages (sent by both the server and clients) are encrypted using RSA encryption algorithm.

Snake game

The snake game that we will implement in this assignment is similar to the one implemented in the previous assignment with additional features.
The rules of the game are implemented in the snake.py file which is used by the server script (snake_server.py).

Game controls

In assignment 3, we implemented seven game controls sent by the client to enable him to control the snake. Since we are using the code scripts of the previous assignment as a starting code for this assignment, these seven game controls are still used in this assignment. However, feel free to change the format of the control messages. For example, instead of sending the get command as "get", you can send "control:get" instead. This way the server knows this is a control input and not a public message that should be broadcasted to all clients.

Chatting

Players will be able to send a message to all other players by sending a message to the server and the server will broadcast the message to all players. Therefore, in each cycle, the server can receive a control input or a message from each client. The two conditions apply:

1. A player can only send a public message to all players, he cannot send a private message to a particular player. For example, if three players are connected, A, B, and C. Player A can send a

public message to the server and the server will broadcast this message to A, B, and C. Player A can't send a private message to player B or C.

2. Instead of having the user type in the message it wants to send to the server, each player should have a set of predefined messages each associated with a hotkey. When the user presses a hotkey, the corresponding message should be sent to the server to be broadcasted to all other players. Each player chooses its unique set of messages. For example, player A can have the following set of messages: ["Congratulations!", "It works!", "Ready?"] assigned to the following keys: ['z', 'x', 'c']. When user A presses the key 'x', the message "It works!" should be sent to the server. The server will then broadcast the message to users B and C. Upon receiving player A's message from the server, users B and C will display the received message on their terminal. You are required to implement a client that has a set of three predefined messages each assigned a key.

Encryption

RSA Encryption Workflow

Consider a client k that wants to securely communicate with a server using RSA algorithm. First, client k must generate its own RSA key pair: (client_k_public_key, client_k_private_key). When client k connects to the server, it will send its public key to the server. After that, whenever Client k wants to send a message to the server, it will encrypt the message using its own private key, and send the encrypted message to the server, the server will then use client_k_public_key to decrypt the received encrypted message. Similarly, the server will construct its own RSA key pair: (server_public_key, server_private_key). The server will share its public key with every client that connects to it. After that, if the server wants to send an encrypted message to client k, it will encrypt the message using its own private key. Client k will then be able to decrypt the received message using the server's public key.

What to encrypt

In this assignment, the server exchanges various types of data with clients. Each cycle, the client must send a control message and possibly a public message to be broadcasted to all clients. On the other hand, the server sends the game state and the public messages that it received from any client to broadcast to other clients. Therefore, the client can send two types of data: control inputs and public messages, and the server can send two types of data: the game state and public messages. **You are required to encrypt all the exchanged data between the server and client except the game state which can be sent without encryption.** We don't encrypt the game state because encryption becomes more expensive (takes a lot of time) as the length of the text to be encrypted grows.

Communication protocol and game state

In the previous assignment, the exact form of the game state was provided which contains the coordinates of the snakes and the coordinates of the snacks separated by a "|" character. As this assignment introduces more features that require adjusting both the server and client scripts, the communication protocol provided for the previous assignment will not work. For example, in the previous assignment, the server and the client exchange the control inputs and the game state each cycle. In this assignment, clients can send a public message to the server to be relayed to the other clients. Therefore, the communication protocol should be redesigned such that under the new communication protocol, the client can send both types of data (i.e. control inputs and public messages), and the server can differentiate between both types.

Redesign the game communication protocol. This includes re-formatting of the control inputs and the game state; and modifying the sequence and contents of exchanged data. **Feel free to make any changes in the client and server scripts of the previous assignment to add the new functionalities (multi-client connection, public chatting between clients, and message encryption).**

Deliverables

Submit four files:

1. snake_server.py: Snake game server-side code which can handle multiple client connections and public messaging.
2. Snake.py: Helper file for snake_server.py. It contains the functions to handle the game logic.
3. Screenshot.png: Screenshot of the game display showing at least three players (snakes) in the field. The screenshot should also show the terminal of one of the clients with exchanged messages printed on the terminal (similar to figure 1).
4. snake_client.py: Snake game client side

Use the snake.py, snake_server.py, and snake_client.py files from the previous assignment as starting code. Modify these files as you want to implement the features of this assignment.

Rubric

This assignment is out of 50. Marks are distributed as follows:

1. Server can handle multiple client connections concurrently. 5 marks
2. Server manages the game's logic for all clients (the game runs as intended). 10 marks
3. Server receives messages from clients and broadcasts them to all clients. 5 marks
4. Server broadcasts the game state to all clients. 5 marks
5. Server encrypts the public messages it broadcasts to clients. 5 marks
6. Client displays the game with all snakes based on the received game state. 5 marks
7. Client sends movement commands to the server. 5 marks
8. Client sends a message to the server when the user presses the assigned hotkey. 5 marks
9. Client encrypts the control inputs and public messages it sends to the server. 5 marks

Late submission

- Late assignments will be accepted for up to two days after the due date, with weekends counting as a single day; the late penalty is 20% of the available marks per day. Lateness is based on the time the assignment is submitted.
- Extensions will be granted only by your course instructor. If you have serious medical or compassionate grounds for an extension, you **must** take supporting documentation to the Academic Counselling unit of your faculty, who will contact the instructor.

Good luck!