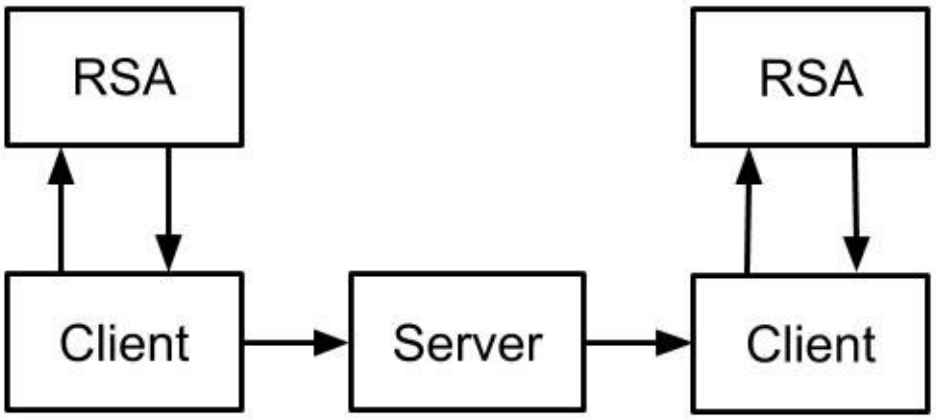
Design Document

Description:

The system we're planning to create involves peer to peer communication over a network, while having a security system which employs RSA. It includes logging in, entering and leaving chats, notifications of when someone enters the chat room, and built-in commands. The most important functionality of our system will be to allow easy ways of communicating using terminal. We're hoping to avoid complicated GUI based on past experiences and focus more on features we can add to a messaging system, as described below. The messaging platform is end to end encrypted and the server gets encrypted messages.

Additional features that we hope to add involve:

- Logging in
- Notifications when users join the chat room or request to message you
- Blocking users
- Message encryption
- Checking who is available to chat
- Leaving and re-entering chats



Our system will be broken down into modules according to the diagram above. First the Client module will parse the text input through the terminal. It will pass the processed message on to the RSA module where they will be encoded. From there the encoded message will be passed

back through the Client module and then to the Server module which will send the message to another device. Finally the message will go back through the RSA module to be decrypted and then displayed on the receiver's device using the Client module once more.

Client:

The purpose of the client module is to interact with the client and allow the client to send and receive messages and requests. To send a message, it takes the message from the client, performs the necessary encryption using the RSA module, and then sends the message along to the server. Similarly, for the case of receiving messages, it would take in the message from the server, decrypt it and then format the output for the user. The Client is able to parse the built-in commands so they are not encrypted before being passed to the server.

RSA:

The goal of the RSA module is to make sure that all messages are sent over the network in a safe manner. Hence, messages need to be encrypted when given to the Server module. The RSA module provides the interface for encrypting and decrypting messages based on a private key. The private key will be calculated based on the user id's of the people involved in the conversation. We are using the user id's so that the private key can be calculated on both sides of the conversation to decrypt any message sent between them. This module is called by the Client whenever a message is sent or received.

Server:

The server module acts as the interface between multiple users. It will connect users in a chat so that the messages sent by one user are displayed for the other user in the chat. It also handles built-in commands such as "//online" or "//leave" and executes them for the client. It also ensures that users only receive messages from people they chat with and supports blocking users.

Dependencies:

- Async
    - Contains the Deferred, Reader, Writer, and Tcp modules
- Core

    - Jane Street's version of the standard library. Needs to be used for Async to work properly
- Deferred
    - This will be used for concurrent programming
- Reader
    - This will be used to read from the network
- Writer
    - This will be used to write to the network
- Tcp
    - Used to set up a connection between the Server and Client
- Zarith
    - Used for avoiding overflow during RSA calculations

Testing:

We found it very difficult to write unit tests for the Server and Client modules. Even our helper methods all use a reader/writer given by the server/client connection. This external dependency made it hard to test. Thus, we did a good amount of interactive user testing to ensure our code works. We used multiple laptops for different clients and tried talking back and forth. We also tried edge cases, such as unexpected quits and sending long messages, to get an idea of how well our project deals with such scenarios. We've resolved all bugs we found through interactive testing.

For the RSA module we were able to write test cases to ensure the messages were encrypted and decrypted properly. These tests focused on the decrypt_message and encrypt_message methods because those were most important to our project. Because the helper methods are encapsulated by RSA.mli, we had to remove those test cases from our test file. However, before we deleted them we ensured the helper methods work, and since encryption and decryption rely on these helper methods by testing decrypt_message and encrypt_message we also tested the helper functions. We tried to test edge cases such as the empty string and ensure that if invalid strings were passed in to decrypt that an exception was successfully raised.

Division of Labor:

In the beginning we all worked on the prototype which included the Client and Server modules. During this time we worked on Atom using Teletype, so even though Fletcher was pushing most of the changes everyone was working simultaneously on the code. After the prototype we split up into pairs, with Fletcher and Arshi focusing on the Server and Client modules and Nandini and Adit creating on the RSA module.

Fletcher focused on the Server and Client modules. In the beginning this consisted of learning more about the Async module and how deferreds worked. Once the prototype was complete he worked along with Arshi on implementing the built-in commands, such as “//leave”, “//quit”, “//help”, and “//block”. Much of the work was including a number of if-else statements which determined which command the user typed in and then execute the appropriate action. He spent about 25 hours on the assignment.

Arshi focused on the Server and Client modules. After the prototype, she worked with Fletcher on the Server and Client. They were using teletype portal shared by Fletcher so you can only see the git commits by Fletcher and not her. She worked on making the Server and Client asynchronous which included figuring out the Async module and understanding how the deferred objects work. She also worked on adding features such as blocking people and displaying online users. She worked on this assignment for about 25 hours.

Adit focused on RSA encryption and testing. He initially worked with everyone on the prototype. This involved understand the libraries and understanding how deferred objects work. He worked on setting up the the server when a client connects like setting up unique usernames and setting up their message buffers. And then he worked with Nandini to understand and implement RSA. RSA required conceptual understanding and implementing encryption by avoiding overflow. Implemented the algorithms that RSA requires to get the keys required. Integrated the RSA into the Client Server model. Wrote the test file for checking if everything is working fine. Worked on the assignment for about 25 hours.

Nandini initially worked with everyone on the prototype. This involved understanding the Async modules and producing a simple server to client and client to client connection. After the prototype, she worked with Adit to understand and implement RSA encryption for the chat. The implementation involved exploring some additional libraries and looking at online resources to

strengthen the level of encryption. They used teletype to pair code. It also involved understanding and implementing the inverse function necessary to implement RSA. She also wrote test cases to check the encryption and decryption. She worked on the assignment for about 25 hours.