The goal of this assignment is to add encryption to your chat client application for pseudosecure client-client communication.


3. [10] Create a new Eclipse project named ChatClientEncrypted. Create a ChatClientEncrypted class for the project with all of the capabilities of your ChatClient class from Homework 6.

4. [30] Add a transform() method to your ChatClientEncrypted class which takes two characters (i.e., type char) and a 2x2 integer matrix (i.e., type int[2][2]) as input and returns a String. This method will perform the matrix multiplication step of the Hill Cipher, but unlike the traditional Hill Cipher which operates on only the 26 case-insensitive alphabetic characters, our modified Hill Cipher will operate on the 128 standard ASCII characters. This modification will allow the chat client to send/receive encrypted case-sensitive alphabetic characters, digits, and punctuation. Here’s pseudocode for the transformation:

   Create an int[2] using the input characters a and b and multiply this array by the input matrix using standard matrix multiplication to determine the characters c and d (you’ll need to implement the matrix multiplication):

   
   
   
   
   
   The method will perform the matrix multiplication step of the Hill Cipher, but unlike the traditional Hill Cipher which operates on only the 26 case-insensitive alphabetic characters, our modified Hill Cipher will operate on the 128 standard ASCII characters. This modification will allow the chat client to send/receive encrypted case-sensitive alphabetic characters, digits, and punctuation. Here’s pseudocode for the transformation:

   Create an int[2] using the input characters a and b and multiply this array by the input matrix using standard matrix multiplication to determine the characters c and d (you’ll need to implement the matrix multiplication):

   
   
   
   
   
   Calculate the values of c and d modulo 128 (e.g., c %= 128).

   If c or d is negative, add 128 (e.g., if (c < 0) c += 128).

   Return a new String containing c and d.

5. [10] Add an encrypt() method which takes a String as input and returns an encrypted String. The method should first check to determine if the number of characters in the input String is odd, and if so pad the string with a trailing space (i.e., “ ”) character. The modified Hill Cipher used for this assignment only works on even-length strings, as you’ll see next. Next, loop through the String calling transform() on each pair of characters and adding the result to the output stream. Use the following encryption key as input to transform(): [[4, 3], [5, 4]]. You may define your own encryption key, but you’ll need a paired decryption key and generating such a key is nontrivial and beyond the scope of this course (see below).
6. [10] Add an decrypt() method which takes a String as input and returns a decrypted String. Similar to encrypt(), loop through the String calling transform() on each pair of characters and adding the result to the output stream. Use the following decryption key as input to transform(): [[4, -3], [-5, 4]]. Again, you may define your own decryption key, but you’ll need a decryption key D paired with your encryption key E such that \( D = E^{-1} \mod 128 \) (we’ll discuss in class).

7. [10] Set up ChatClientEncrypted to encrypt all output sent to the serve and to decrypt all input received from the server.

8. [+20] (Extra credit) Create a new server which requires and login and password to send/receive chat messages. This new server must be able to communicate with your encrypted chat client. First create a new Eclipse project named ChatServerEncrypted. Create a ChatServerEncrypted class for the project with all of the capabilities of your ChatServer class from Homework 6. Make the following improvements to ChatServerEncrypted:

   - Add your encryption/decryption methods from above so that the new server class can understand messages from clients and send its own encrypted messages to the clients.
   - Modify the server to store state information along with each Socket which implies whether the client has just connected, has supplied a valid username, or has supplied a valid username and password. It’s critical that this state is stored per socket so that multiple logins can be in progress simultaneously. Hint: you’ll probably want to create a new class which is a wrapper around a Socket with the additional state variables added.
   - When a client connects, send a message to the client asking for a username.
   - Treat the first message from the user as the username. If the username was valid, send a message to the client asking for the password. If the username was not valid, alert the client by asking for another username.
   - Treat the next message after the valid username as the password. If the password was valid, send a message to all connected clients confirming their login (e.g., “eric has joined.”). If the password was not valid, alert the client by asking for another password.
   - Only send messages from clients to other clients which have successfully logged in. Also prepend the client’s username plus some formatting characters in front of any messages sent (e.g., “eric-> Hello!”).
   - You can specify a list of valid users/passwords in any way that you’d like, but recall that we created a PasswordList class early in the course.
   - Hint: your client classes currently send a “Joined” message. This will need to be disabled or it will be interpreted by the server as a login attempt.
Zip the relevant folders in your Eclipse workspace directory along with your problem-set solution document (PDF or TXT) and upload the .zip file to Blackboard (see Assignment 8 assignment in the Course Documents area).