Introduction to Java
CS9053 Section I
Thursday 6 PM – 8:30 PM
Prof. Dean Christakos
April 8th, 2023
Due: April 14th, 2024 11:59 PM

Networks: Build a Chat Application

This assignment is going to require a bit more development than usual.

You are going to use the network to build a simple chat application, using your knowledge of the Java networking API.

Most design issues are up to you, but you will be creating a ChatServer class that creates a window with a JTextArea that displays debug info including who has connected. You will then be creating a ChatClient

What is going to happen is that you are going to start a ChatServer that accepts connections on port 9898 (make sure you don't have another application accepting connections on this port). This server should be able to accept multiple connections.

Each Chat client should be able to connect to a server on port 9898. Once the chat client is connected, anything a user types in the chat client should be able to be seen by all other connected clients.

The tricky part is this: each connection has a handshake protocol:

- When the ChatClient connects to the ChatServer, the ChatClient sends a message "HELLO".
- When the ChatServer receives a "HELLO" message, the ChatServer Sends a reply message to the ChatClient that says "CONNECTED."
- When the ChatClient receives the "CONNECTED" message, it sends an "AES Seed" to the ChatServer Encryption with the ChatServer's public key
- When the ChatServer receives the encrypted "AES Seed", the ChatServer decrypts the encrypted message with its Private Key
- The ChatServer and ChatClient both generate an AES Key and use the AES Key to send encrypted messages to each other[1]
- This AES Key is the encrypted channel for communications between the ChatServer and ChatClient

[1] Actually, the "AES Seed" IS the key. But there are two steps in the process. Normally the "AES Seed" would be used in an algorithm that both sides already agreed upon to generate the AES Key but I didn't put things through this extra step. You just take the "Seed" and pass it to the `Encryption.generateAESKey` method and it returns an AES Key using the "AES Seed" bytes

All the encryption methods are taken care for you and form a group of static methods in the encryption.Encryption class:

`public static String encrypt(Key key, String input)` – given an encryption key and a plaintext String, returns an encrypted String
`public static String decrypt(Key key, String cipherText)` – given an encryption key and encrypted text, returns a decrypted (plaintext) String

`public static byte[] pkEncrypt(Key key, byte[] plaintext)` – given an asymmetric key and an array of 16 plaintext bytes, returns an array of 128 encrypted bytes

`public static byte[] pkDecrypt(Key key, byte[] ciphertext)` – given an asymmetrc key and an array of 128 encrypted bytes, returns an array of 16 plaintext bytes

`public static Key generateAESKey(byte[] sequence)` – Given a 16 byte sequence, returns an AES (symmetric) encryption key

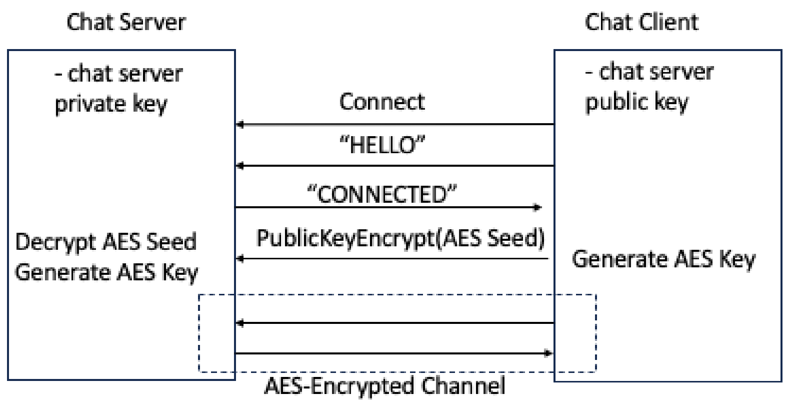
`public static byte[] generateSeed()` – generates a random 16 byte sequence that can be used as a seed to generate an AES encryption key

`public static PrivateKey readPrivateKey(String filename)` – reads an RSA PKCS8 file and returns a private key

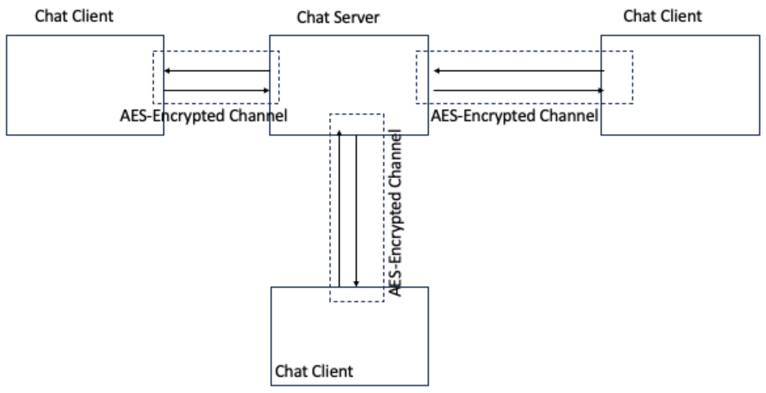
`public static PublicKey readPublicKey(String keyString)` – takes an RSA keyString and returns a public key

To make life a little easier for you, the code in the ChatServer already reads in its Private Key and the code in the ChatClient already reads in the ChatServer's Public Key.

The protocol process will look like this:



Ultimately, the ChatServer will Communicate with several ChatClients over encrypted AES Channels, each with their own AES key:

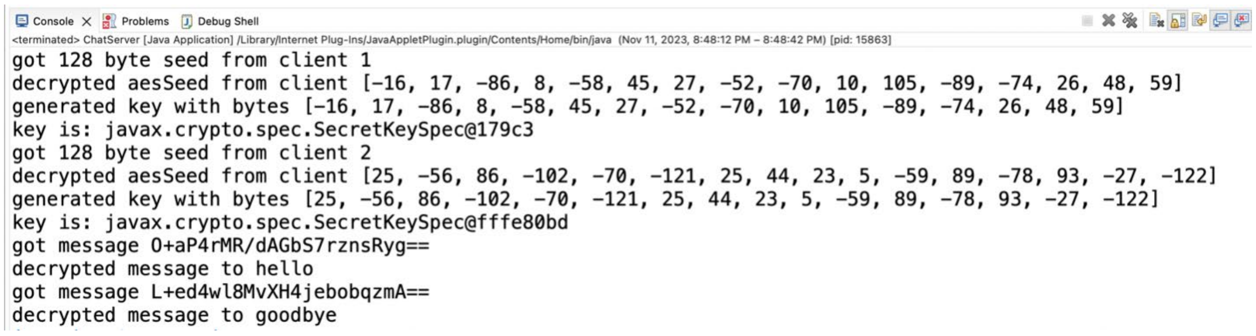


One last hint: When sending Strings back and forth between client and server, the input and output streams will be called with:

```
String inString = fromServer.readUTF();
toServer.writeUTF(outString)
```

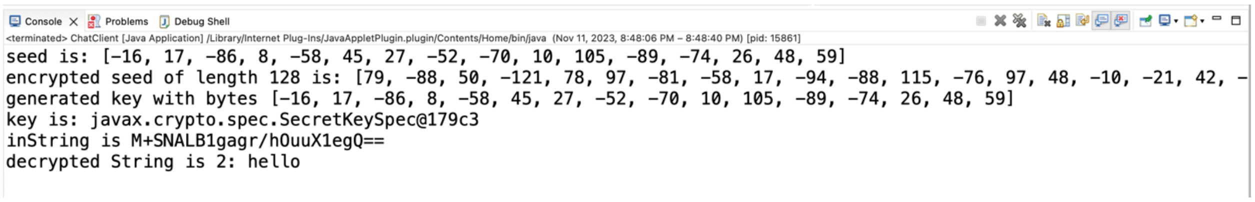
See the video demo to see how the chat server should work. I put some debugging info in the console on my version that describes the kind of behavior you should expect:

Server:



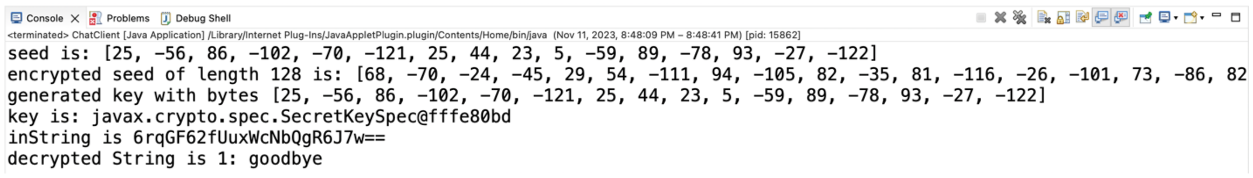
```
<terminated> ChatServer [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java  (Nov 11, 2023, 8:48:12 PM – 8:48:42 PM) [pid: 15863]
got 128 byte seed from client 1
decrypted aesSeed from client [-16, 17, -86, 8, -58, 45, 27, -52, -70, 10, 105, -89, -74, 26, 48, 59]
generated key with bytes [-16, 17, -86, 8, -58, 45, 27, -52, -70, 10, 105, -89, -74, 26, 48, 59]
key is: javax.crypto.spec.SecretKeySpec@179c3
got 128 byte seed from client 2
decrypted aesSeed from client [25, -56, 86, -102, -70, -121, 25, 44, 23, 5, -59, 89, -78, 93, -27, -122]
generated key with bytes [25, -56, 86, -102, -70, -121, 25, 44, 23, 5, -59, 89, -78, 93, -27, -122]
key is: javax.crypto.spec.SecretKeySpec@fffe80bd
got message O+aP4rMR/dAGbS7rznsRyg==
decrypted message to hello
got message L+ed4wl8MvXH4jebobqzmA==
decrypted message to goodbye
```

Client 1:



```
<terminated> ChatClient [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java  (Nov 11, 2023, 8:48:06 PM – 8:48:40 PM) [pid: 15861]
seed is: [-16, 17, -86, 8, -58, 45, 27, -52, -70, 10, 105, -89, -74, 26, 48, 59]
encrypted seed of length 128 is: [79, -88, 50, -121, 78, 97, -81, -58, 17, -94, -88, 115, -76, 97, 48, -10, -21, 42, -
generated key with bytes [-16, 17, -86, 8, -58, 45, 27, -52, -70, 10, 105, -89, -74, 26, 48, 59]
key is: javax.crypto.spec.SecretKeySpec@179c3
inString is M+SNALB1gagr/hOuuX1egQ==
decrypted String is 2: hello
```

Client 2:



```
<terminated> ChatClient [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java  (Nov 11, 2023, 8:48:09 PM – 8:48:41 PM) [pid: 15862]
seed is: [25, -56, 86, -102, -70, -121, 25, 44, 23, 5, -59, 89, -78, 93, -27, -122]
encrypted seed of length 128 is: [68, -70, -24, -45, 29, 54, -111, 94, -105, 82, -35, 81, -116, -26, -101, 73, -86, 82
generated key with bytes [25, -56, 86, -102, -70, -121, 25, 44, 23, 5, -59, 89, -78, 93, -27, -122]
key is: javax.crypto.spec.SecretKeySpec@fffe80bd
inString is 6rqGF62fUuxWcNbQgR6J7w==
decrypted String is 1: goodbye
```

You will get 2 points extra credit if you come up with some kind of timeout/failure handling for the initial connection handshake.