Assignment 3 - Cryptography

SEng 360: Security Engineering
November 9th, 2017

Scott Goerzen V00844662
Meagan Pal V00849477
Joss Vrooman V00813897

Overview and Contents

For this assignment, we created a collection of directories and Java files to simulate a secure Instant Messaging (IM) program between a client and a server. Along with this PDF, we submitted the package GrootChat containing the source files for the IM, as well as a JavaDocs documentation for our files and the methods within. Here is a description of its contents (note that, in theory, the contents of HiddenClient and HiddenServer are access controlled directories):

- **ClientOperation.java** - a java program representing the client; begins the IM session by initiating communication with the server
- **ServerOperation.java** - a java program representing the server; receives the client's request
- **RMIInterface.java** - an interface for ServerOperation.java
- **RMICInterface.java** - an interface for ClientOperation.java
- **MD5Hash.java** - a hashing program to hash strings for authentication purposes
- **doRSA.java** - a java program to encrypt/decrypt with public/private keys for asymmetric cryptography
- **JavaDocs** (directory):
    - **index.html** - an html file to explore our java documentation, not actually used within the program
- **Public** (directory)**:**
    - **publicClient.key** - the client's public key
    - **publicServer.key** - the server's public key
- **HiddenClient** (directory)**:**
    - **ClientPassword.txt** - a text file with the hash of the client's password.
    - **privateClient.key** - the client's private key
- **HiddenServer** (directory)**:**
    - **ServerPassword.txt** - a text file with the hash of the server's password
    - **privateServer.key** - the server's private key

Executing the Program

This Program uses 3 terminal windows.

1. Open all windows to the downloaded folder and in the first window, run the following command.

```
rmiregistry
```

If this does not work you may want to specify the port it is being run on, include the port number as the argument.

2. On one terminal window, compile all the files using the command

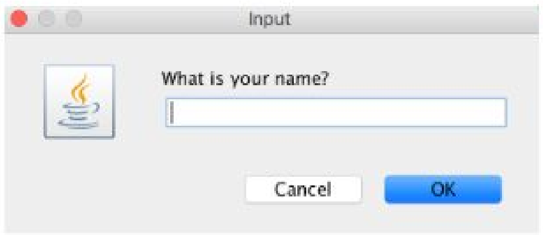
```
javac GrootChat/RMIInterface.java GrootChat/RMICInterface.java
GrootChat/ServerOperation.java GrootChat/ClientOperation.java
```

3. On the same window, run

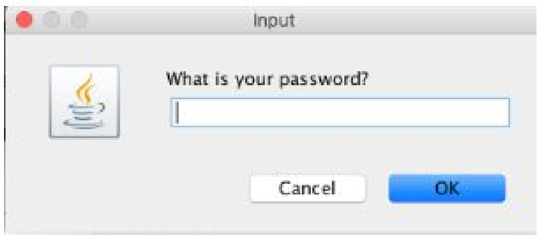
```
java GrootChat/ServerOperation
```

Note that the server must be run before the client. However, the client can be run at any time after and can reconnect any number of times.
Multiple clients can be connected, but the server will only communicate with the last one to connect.

4. Enter the password 'GROOT' before selecting the desired security options. You will have 6 attempts to enter the correct password, otherwise the program will quit.

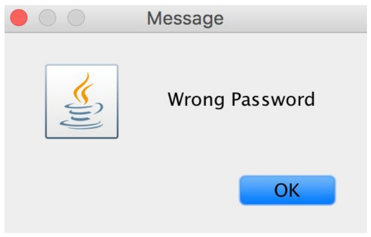


What is your name?

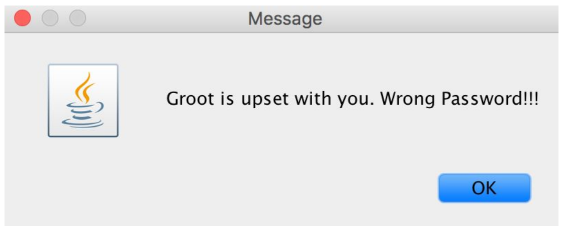


What is your password?

Sample outputs after running out of attempts.



Wrong Password



Groot is upset with you. Wrong Password!!!

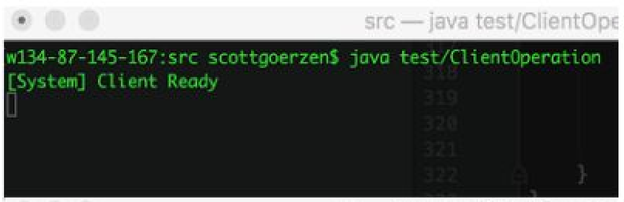
5. On the last window, run

```
java GrootChat/ClientOperation
```

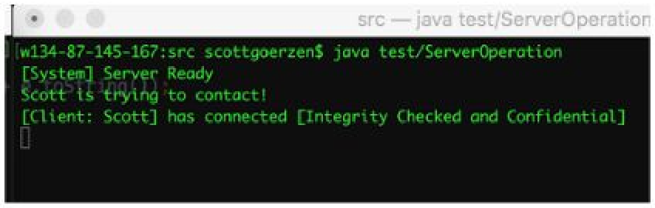
6. Enter an arbitrary Username and the password 'Rocket' to authenticate the user before selecting the desired security options. Again, there are 6 possible attempts to get the correct password.

   Note: The Username is arbitrary because for one client, we don't necessarily need two 'things you know.' The username is simply a way of personalizing the Client's interaction with the server whilst the password Authenticates the person using the Client.
   For more users, a dictionary can be used to relate passwords to users, but with only one client, it does not matter as much.

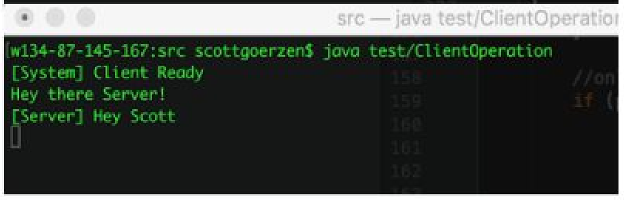
You can now type freely as the Client or Server to send messages to the other party.



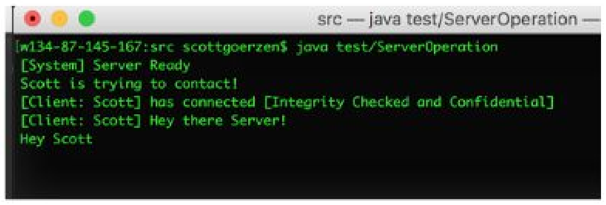
```
w134-87-145-167:src scottgoerzen$ java test/ClientOperation
[System] Client Ready
```



```
w134-87-145-167:src scottgoerzen$ java test/ServerOperation
[System] Server Ready
Scott is trying to contact!
[Client: Scott] has connected [Integrity Checked and Confidential]
```



```
w134-87-145-167:src scottgoerzen$ java test/ClientOperation
[System] Client Ready
Hey there Server!
[Server] Hey Scott
```



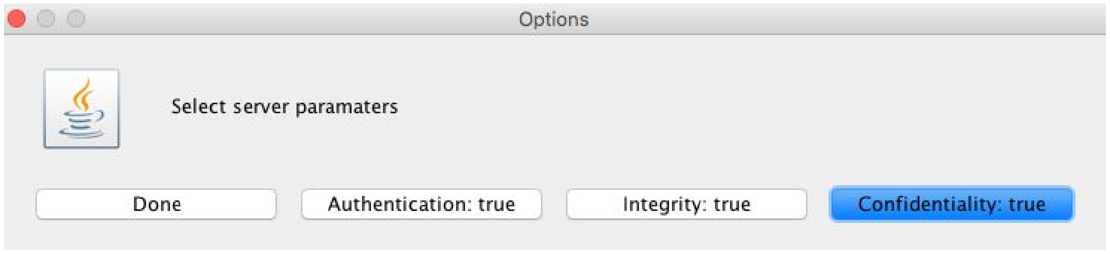
```
w134-87-145-167:src scottgoerzen$ java test/ServerOperation
[System] Server Ready
Scott is trying to contact!
[Client: Scott] has connected [Integrity Checked and Confidential]
[Client: Scott] Hey there Server!
Hey Scott
```

Security Options

Security options can be changed by the server *only* when there is no client connected. In order for the Client to change their security options, the Client must quit and reestablish their connection. This is done by typing a '-' followed by a command into the input field.
Commands include:

-ct	Confidentiality true: Turns on encryption of outgoing messages
-cf	Confidentiality false: Turns off encryption of outgoing messages
-it	Integrity Checks true: Turns on checking of message integrity via message authentication codes (MAC)
-if	Integrity Checks false: Turns off checking of message integrity via message authentication codes (MAC)
-at	Authentication true: Turns on checking of real users via public/private key encryption
-af	Authentication false: Turns off checking of real users via public/private key encryption

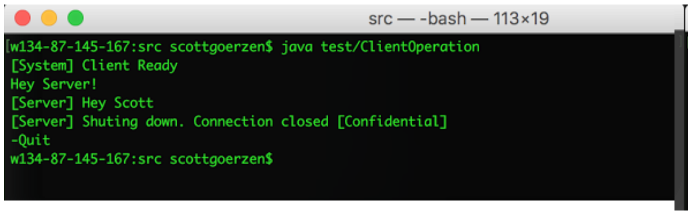
The security options are enabled by default, but are selectable and set by the initiation sequence in the menu feature.



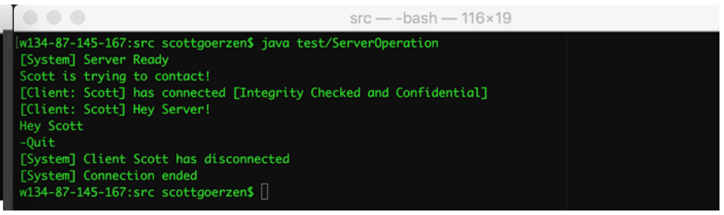
Authorization is checked at login on both Server and Client side to ensure **confidentiality**. This is done by means of comparing the hash of the input password to the hash saved in an access controlled directory. When the client and the server connect to each other, they send an authentication message via public/private keys in asymmetric cryptography by RSA 2048 bit. When the client contacts the server, a session key, established for symmetric cryptography with AES, is returned along with the name the client encrypted for the server, thus proving the server is who they say they are. All later encryption is done by AES encryption with the session key established at the first step.

Integrity is verified by means of passing a message authentication code (MAC) along with the message. The MAC consists of the message being hashed with MD5, a one way hashing scheme, encrypted with AES and then converted back to a string.

Both client and server connections are stopped by typing the '-Quit' command instead of a message into the command line. If the server quits with a client still connected, it kicks the client out.



```
w134-87-145-167:src scottgoerzen$ java test/ClientOperation
[System] Client Ready
Hey Server!
[Server] Hey Scott
[Server] Shuting down. Connection closed [Confidential]
-Quit
w134-87-145-167:src scottgoerzen$
```



```
w134-87-145-167:src scottgoerzen$ java test/ServerOperation
[System] Server Ready
Scott is trying to contact!
[Client: Scott] has connected [Integrity Checked and Confidential]
[Client: Scott] Hey Server!
Hey Scott
-Quit
[System] Client Scott has disconnected
[System] Connection ended
w134-87-145-167:src scottgoerzen$
```