Further Java Ticklet 5*

In order to gain a star in the mark sheet you must complete this exercise. Completing the exercise does not gain you any credit in the examination.

In recent years we've seen Facebook's WhatsApp platform and Apple's iMessage system adopt end-to-end encryption. This offers a variety of security benefits for the user. For example, a well-designed end-to-end encrypted message platform will not reveal old message data if server infrastructure is compromised. In this exercise you will extend your implementation of the Java chat server from Workbook 4 and your Java chat client from Workbook 2 to support end-to-end encryption of message contents between clients.

A simple solution to provide message confidentiality and integrity is to use a symmetric block cipher coupled with a suitable mode of operation to encrypt the contents of the messages. The plaintext copy of all client-to-client messages can then be encrypted using this method, thereby preventing the server from reading the contents of the messages. There are however a number of downsides with this approach. A short (incomplete) list includes:

- The symmetric key material needs to be shared out-of-band with all clients. This is difficult to do securely if users do not meet in person regularly and is usability issue since users need to manage keys explicitly.
- The symmetric key needs to be changed whenever a user leaves the group or if the key is compromised on any of the clients.
- The solution offers no *forward secrecy*.

A more sophisticated design might make use of public-key cryptography. A public-key solution might make use of the chat server to act as a key server as follows. When a client connects, the client sends its public key to the server; this key can then be distributed to all clients by the server. Then, when a client wishes to send a message, it can encrypt the message under a new symmetric key, and then encrypt the symmetric key under the public keys of all clients. There remain a number of downsides with this approach however, including the lack of forward secrecy.

There are more sophisticated protocols, such as the Signal Protocol [https://en.wikipedia.org/wiki/Signal_Protocol] which provide better security and privacy guarantees. There is also a rich academic literature on on the subject. A good starting point is Unger et al, *SoK: Secure Messaging*, IEEE Symposium on Security and Privacy (S&P), 2015. [http://ieeexplore.ieee.org/abstract/document/7163029/]

To successfully pass this tick, you should ensure that all messages are encrypted such that the server owner cannot read them without compromising a client. A basic solution which uses symmetric-key cryptography is sufficient, although we encourage creativity and a desire to produce an implementation which is more secure and more usable than a simple solution.

Submission

Your submission *must* include a statement which records the fact that the software you have written is experimental in nature, does not guarantee to offer strong security properties, and should not be used in production. You can do this in a README file or similar.

When you are satisfied you have completed everything, you should commit all outstanding changes and push these to the Chime server. On the Chime server, check that the latest version of your files are in the repository, and once you are happy schedule your code for testing. You can resubmit as many times as you like and there is no penalty for re-submission. If, after waiting one hour, you have not received a final response you should notify `ticks1b-admin@cl.cam.ac.uk` of the problem. You should submit a version of your code which successfully passes the automated checks by the deadline, so don't leave it to the last minute!