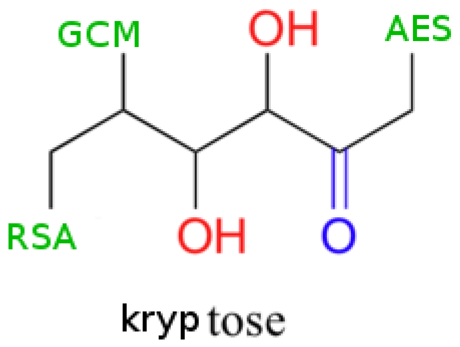
Kryptose™

Assurance Document: Final Release



Personnel: Jonathan Shi (js2845), Antonio Marcedone (am2623), Alexander Guziel (asg252), Jeff Tian (yt336)

JUnit Tests: We wrote thorough automated JUnit tests for several parts of our system. On the client side, we have tests for all of the client-side cryptography components, as well as the request handler (which sends requests to and receives responses from the server). Since there is really no way to unit test the GUI, we did exhaustive manual testing, trying to simulate every distinct sequence of commands that can come up to exercise all branches of our code.

On the server side, we wrote JUnit tests for all of the major components, including tests for the user authentication table, tamper-proofed logging, server request handling along with the reference monitor, and parts of the server SSL socket connection handler. In terms of SSL connection testing, we focused on both integration and unusual context testing for the SSL communication code (weak SSL versions are not accepted, certificates are actually checked, we only accept our own certificates and exclude the trust anchors from the OS store). The request handler tests tested a variety of scenarios including ones that were designed to crash the system.

We estimate that the JUnit tests achieve >70% coverage of the code. We accounted for the missing parts (such as the GUI) with manual inspection. This includes testing end-to-end integration manually, checking that every action the user takes with the GUI triggers the correct response from the server and is displayed back to the client properly.

Regression testing was run on these unit tests whenever further changes were made to any of the code that they tested.

In terms of client interfact testing, We manually verified that user stories worked as expected from the client, with a server running on localhost. This included testing every command, and combinations of commands that interact with each other, as well as testing access to the same server from different client sessions. It also involved testing invalid commands. Rigorous manual testing was employed on the client in a variety of scenarios. For example, we tested all the commands when the user had no passwords, many passwords, and many passwords then deleted them.

In terms of **code reviews**, We often reviewed each others' code especially when testing and when updating code that someone else wrote. We should consider more disciplined review, as our code reviews were more ad-hoc and on an as-needed basis rather than systematic.

We checked the **FindBugs scan** and fixed any serious errors. After 2-3 iterations, we were able to get the code in a state such that no more FindBugs errors were reported.

We employed thorough **logging** of unexpected conditions in the code, and used this to find errors that might not otherwise have been noticed. This involved logging both server and client-side events to console so that we could observe that the correct state transitions were happening at the correct times and that the expected functions did indeed execute properly and terminate without exception.

Overall, through the testing strategy we've developed over the course of these three sprints, we believe that by the end we became quite effective in pinning down bugs, identifying and responding to the sources of new bugs whenever code was changed, and using a mixture of a variety of methods such as regression testing and peer reviews to ensure that a minimal number of new bugs are introduced to the system in any given commit. We believe the stable state of the final product reflects our successes in this area.