Kryptose: Assurance Document

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

The cryptography and communication parts were unit tested by unit tests that were written to ensure that they worked individually according to their expected interfaces. Regression testing was done on these parts every time associated code was changed, to make sure these individual units stayed functional and correct. Everything else was integration tested, meaning that we hand tested and verified the behavior of the system in response to the different commands. This included running every command possible from the client's perspective and testing the system works from all of the distinct states we could think of, and also testing with simultaneous clients to ensure that concurrency issues were handled correctly, such as the versioning of our file and the rejection of stale writes.

We also ran invalid commands in an attempt to crash the client. We made sure to exercise every line of code we wrote, either through the aforementioned unit tests or making sure every possible sequence that the client can force the system to execute by integration testing by hand. As per requirement, we ran FindBugs on our final code version and ensured that no "scary" errors or warnings were raised from the static analysis of our source code.

Pair programming was done for the client read-eval-print-loop as well as client-server communication to make sure certain specifications were met. All code was reviewed by at least one other person who did not write the code, and many sections had multiple people working on it.

Integration testing was done locally and on a remote machine, which involved using the client to try every type of response in order to break the server or client. Sending packets to the server that are not well formed according to our communication protocol (or not valid TLS packets) will cause the thread dealing with that connection to terminate. This does not leave the server in an inconsistent state, as requests are handled independently, and the exception gets correctly logged. Multiple of these packets does not cause an issue because threads are expected to terminate anyway and a new thread will be spawned to handle upcoming work.

To make sure the various security requirements were met, we made sure that all of the cryptography was implemented correctly first and foremost through unit testing, and that when we ran our program the encryption actually was happening. Authentication was not required for this Alpha release, this will be built by the Beta

release. This provides the bulk of the assurance for the confidentiality and integrity requirements that we promised in our Requirements document, namely that the key derivation functions, communication channels, and cryptography methods are all safe and up to par with the current standards on protocols and key lengths. We ensured that the server never receives the client's file decryption key and never receives any part of the client's files in plaintext, thus ensuring the desired properties from any type of server compromise. In terms of the audit security requirement, we provided detailed server and user-specific logs. The server logs contain every major action and exception encountered by the server, and the user-specific logs detail every request type issued by the clients and the result of the request (success or failure and reason for failure), but we make sure that no information contained in the logs are too revealing or expose confidential information that the client has.

We maintain a server log that stores information about all the TLS connections (attempts) received. We look at this log to ensure that everything was working as expected, that each function expected to be executed was actually executing and hence writing to the log, while the functions not expected to execute indeed do not execute and hence no logging is produced from these functions in unexpected places.