CSCE 438 – HW4 Design Document

Dainel Tan, Gregory LaFlash, Brad Coffman, Andy Hampton

1 RPC Design

We decided to use the callbacks in conjunction with the RPC code in order to facilitate the type of communication that we needed for this assignment. This required an extensive use of timers that check every two seconds if a callback needs to be called due to a received RPC call. The callbacks give us the flexibility to communicate bi-directionally from both the server and clients in a reliable way. This was important in our decision as the other methods we were investigating involved only being able to call the server from the clients and return whatever result was to be communicated to the client severely hampering how the previous lsp code would work with it. Communication with clients required assigning program numbers to each client when they are initiated so that they can be identified later when the server calls a client's RPC function.

The major changes from the last project were the removal of protobuf, sockets, and marshaling is no longer handled. RPC handles all three of those aspects for this project automatically. The major hurdle for this design was in the svc_run function which will never return and became blocking to our server. In order for the server to respond on its own to each client instead of using the return value for the RPC function, the svc_run function cannot be blocking. To overcome the svc_run design issue, we implemented our own svc_run that is non-blocking and allows for the RPC implementation to work by returning after the first successful RPC call.

One other major hurdle that we over came involved using a global variable to indicate that data was successfully received. Once the global variable has been set, the client or server can retrieve the message from the RPC function call using a pair of broad getter functions specifically created for this data. This does not break the client and server distinction, but simply allows for the LSP portions of the code base to work accordingly.

2 Password Cracker Design

We are using the same password cracker design as that which was provided to us for this assignment. We are simply modifying how the lsp code communicates so as to hide these changes from the actual server, requester, and worker code.

3 How to Compile and Run

We have included a makefile that facilitates all compilation needed to run this program. Typing “make server” will make the server, and “./server” will run the server application. Typing “make request” will make the requester, and “./request host:port sha1_hash password_length” will run the requester application. Note: These are the only two running portions of the code base and fully demonstrate a working RPC application.