**Report**

High Level Overview:

We implemented the initial log in for client to server using RSA to encrypt the login messages. A nonce was sent over the sockets to guarantee freshness and so that the client may SHA256 hash its password that it enters with the salt and send it back to the server. The server then checks that hash by running it through a hash verification routine in the code to see if it matches the clients digest stored in the database. The database was configured off site. After they were successfully authenticated, they were able to continue communication with the server, at which time a shared AES and CMAC key with the server was generated and sent across the socket to the client. The client then recovered it and all communication from there on out with the server was encrypted using this AES and CMAC key. The client then requests, in code, to get its buddy list and the server responds with the client’s buddy list. The client was then able to request to speak to another client. At this point, a new shared secret key needed to be generated between the client and the specific client requested and sent over the socket along with the requested client’s username (for reassurance to our client that they are getting back a key for who they requested) and a ticket encrypted with the requested client’s AES and CMAC key. The clients are then able to talk to each other once the ticket has been sent to the requested client.

Challenges:

We faced many, many challenges. We decided to write this in C++ since that is the most comfortable language between the two of us. This was a bad choice from the beginning. We had to learn how to use the Crypto++ library in order to do the project. This library was very poorly documented with not many people asking and solving questions about it over the internet for us to have a reference to go off of. We had an extremely steep learning curve that took about 2 weeks. The biggest challenges we faced were buffer issues over the sockets such as the buffer containing NULL characters, not being of the right padding length for RSA (which you cannot just simply add more bytes), having to learn how to Base64 encode the messages after encrypting to send over the socket correctly and decrypt on the other side after receiving it, and many others. We had to deal and cope with many data transformations that were happening and it caused much confusions in lots of places. AES had many issues as well. We kept having issues with the cipher text length not being a multiple of the key size. We had to run it through a stream transformation filter in order to decrypt it and who knows what it was actually doing. We also had many concurrency issues on the side with the sockets being too fast or too slow and losing data in between, even though it is supposed to be a TCP connection. We were forced to go directions we had not ever previously thought would arise.

We solved our buffer issues by Base64 encoding our encrypted messages. We also made sure to memset our buffers to 0. This solved our RSA issues as well. Socket synchronization had a lot

to do with solving our AES invalid ciphertext length issues, but we still were never able to definitively put a point on that. It mostly boiled down to a client closing the connection while the server was waiting to decrypt a message from a client. In the end, we still had this problem. The Crypto++ had a nice API that allowed us to pass data through a pipeline, as a source, which would then go through a filter or encoder and come out the other end inside the sink as a string or perhaps SecByteBlock. This was extremely convenient. Otherwise, I do not think this would be possible to do in C++ with the given time period. We learned how difficult it is to do cryptography over a network. We had to build a general cryptographic API to allow us to quickly write code.

James’ Contributions:

I wrote all of the log in code and did all of the cryptography. This involved SHA256 hashing of the username’s password along with a randomly generated number to ensure rainbow table attacks could not be performed. I also implemented the nonce business to ensure freshness. It took me quite a long time to learn the library. The number of bugs that I was coming across that really stumped me was much more than I could handle. Joel helped me so much with that, I appreciate his work. I was stuck on bugs for days that were not trivial and had to do with very subtle things going on with sockets and cryptography mixed together. I handled the protocol implementation up to the point of generating tickets and shared secret session keys between clients. The rest was Joel’s. I wrote the server and client back end, cryptographic code. I also spent a lot of my time just figuring out how to transform data into things that I needed without losing information in between, which was challenging.

Joel’s Contribution:

Joel wrote all of the client application level code. This involved writing asynchronous code to get clients talking with each other, printing text to the screen while a user was typing, etc. He helped me do many of my tasks since crypto was the hardest. He did a LOT of work as well. He used my cryptographic API that I built up for him to do the job of talking. The main piece he was working on was client-client asynchronous communication, which was the heart of the front end.