Joey Gomez-Benito

904140064

[jgbenito7@gmail.com](mailto:jgbenito7@gmail.com)

Patrick Killian Jackson

Lab 4 File Encryption Design Problem

Specification:

Our goal was to make peer to peer file transfer more secure by encrypting file uploads and downloads. The contents of the file should be encrypted on a byte by byte basis and should prevent network snoopers from viewing the file contents.

In order to separate the code we added from the main functionality of the peer2peer network, we added a new flag ‘-e’ to run the program in encryption mode. We also edited the help section to reflect this new mode and added an entry for ‘make run-e’ in the make file.

In order to easily run our encryption program, a user can simply type ‘make run-e’.

We also added the option for a user to change their password before the program runs. This mode assumes that you are also running the program in encryption mode. We included an option to run the flag ‘-ec’ to run the program in both encryption and change password mode. This brings up a new option to change the password. For security purposes, the old password must be entered beforehand. The default password is ‘pass’.

In order to easily run our encryption program in change password mode, a user can simply type ‘make run-c’

Plan for Implementation:

The first step for implementation was to setup a repository that could be used to push code changes and make working together easy. We made a new github repository and were both added as collaborators. We first had to research file encryption and decide how we were going to encrypt the files. We decided on using at xor bit shift for every byte in the file, because the same function could be used for both encryption and decryption. When uploading a file, our function would open the file using the filename and create temporary blank file, loop through to the end of the file, apply ^200 to each character byte, append that byte to the temporary file, and then send the temporary file instead of the original file. When downloading a file, assuming a unified encryption system, the same method would be used to decrypt the contents of the file. Once we had this working, we then had to work on asking for a password while trying to download a file from a peer. For the purposes of this project, we did not create a database for storing passwords. If we were developing an actual password system we would have stored user names and passwords in a database and referenced the database for password lookups and changes. We instead have a default password that is restored every time the program runs. We would then have to create a method for

Summary of Results:

Who did what:

Joey started the implementation by adding a simple passkey global variable and by implementing in the main function a check to see if the passkey was correct. Joey then began implementing an encryption function and a corresponding decryption function. Killian did some additional research on encryption and modified the implementation strategy for the encryption function. After finalizing the encryption function to use a bitwise xor operation to encrypt files byte by byte the decryption function was removed as the encryption function served to both encrypt and decrypt. Killian then added a character array member to task\_t to store the peer's password and moved the password check from the main function into the task\_download function. Killian also added in the procedure for calling Encrypt(char\* filename) into both task\_download and task\_upload to decrypt and encrypt respectively. Joey then added the implementation allowing the option to change the password, via the -c flag, and also implemented per peer decryption passwords adjusting the previous password check in task\_download. Joey then added a flag, -e, to enable encryption mode so the application may run normally or with encryption enabled. We collaborated on the design problem write up.