This is an outline of the client and server processes for our project.

The server will create a socket and listen for client connections. When a client connects a child process will be forked to handle the request to allow multiple simultaneous clients. The server will take the port to use as a command line argument.

The client program will connect to the server and submit requests. To make testing and development easier we can support simple command line handling and an interactive mode (reading commands from std in). The client will take the address and port of the server as command line arguments.

I did not include a third server to act as a CA. We could easily “pretend” to have a CA by simply generating a public/private key pair for the CA and generating a certificate consisting of a user’s public key encrypted using the CA’s private key.

We will need another small application (or an option on the client app) to generate public/private keys pairs (using BBS?). This could also generated the user’s CA certificate.

The server needs to support the following functions:

* User Management
  + add new user – adds a user and their public key. This will involve implementing a public key ring.
  + delete user – removes a user and all owned files (optional)
  + get a user’s public key (optional)
* File Transfer
  + Put file – adds a picture file, the matching signature file, and a list of authorized users
  + Get file – gets a file and verifies the signature. This will store on the client’s computer both the picture file and the signature file. The user can then verify the signature file manually later if desired (optional).
* Transaction Log
  + Log each operation
  + View transaction log

For the communication between the client and server there will be an API that includes send, receive, send\_encrypted, and receive\_encrypted functions. The unencrypted routines will only be used by the initial connection and the public key exchange routines. Once the session key is determined, all other communication will use the encrypted versions. OpenSSL has routines that will make the actual encryption process tractable.

In this outline I only describe the client and server processes, with the intent of including many of the topics we’ve gone over in class, such as encryption, public/private key exchange, key generation, master/session keys, and key management. I did not touch on Web access. If we want to include a Web component it should be able to use the client process (or libraries used to build the client process if that’s easier) to access the server’s functionality.

The high-level outline of the client and server looks like this:

parse command line

create socket

connect to socket

**Client**

repeat until done

close socket

exit

close socket

exit

repeat until done

call request specific server routine

call request specific client routine

receive request structure

fill in request structure

send request

get request from command line or std in

user authorization client

user authorization server

all communication from

here on is encrypted using the shared key

public key exchange server

public key exchange client

generate session key

connect to socket

Server child process

accept on socket

fork child

parse command line

create socket

listen on socket

**Server**

For example, the routines to put and get files could look like:

**Put file** - request includes

filename

list of authorized users

**Server**

**Client**

create signature hash file

encrypted with user’s public key

receive file

store file

send (picture) file

receive hash file

verify hash

store file

store authorization info

log transaction

send hash file

**Get file** - request includes

filename

**Server**

**Client**

verify user is authorized to get the requested file

receive file

store file

send (picture) file

send hash file

receive hash file

verify hash

store file

log transaction