Names:
Michael Grandel (mg3856)
Elysia Witham (ew2632)
Federick Gonzalez (fag2113)

Procedure

1. Server creation
    a. Develop root certificates and intermediate certificates using open SSL
    b. Develop Server certificates and store on a server side subdirectory
    c. Use intermediate CA to develop client certificates and allow trust chain verification.
    d. Use TLS encryption to connect server to client and send data over HTTPS
2. Sendmsg
    a. Client sends server a certificate
    b. Server verifies the certificate with trust chain verification
    c. If the certificate is accepted, Client sends a Get request stating there is a message to be sent, along with the messages to be sent
        i. This message will be parsed to determine a list of users
    d. If valid cert, server sends back requested certificate of the included lists of users
    e. Client takes certificate, signs messages using openssl, and sends these signed messages back to the server
    f. Server adds signed messages to pending mailbox
3. Receive msg
    a. Client sends server certificate and request to read messages
    b. If valid cert, server accesses the server-side mailbox for the client
    c. The server encrypts the messages and sends them over TLS to the client, where it is displayed.
    d. After this is sent, the server clears the file.
4. `getcert` - user will get their certificate
    a. We will store the certificate on the server via OpenSSL
    b. We will return a copy of the certificate to the client
5. Optional: `changepwd` - user will be able to change their password
    a. IF there are no pending messages (directory with the certificate name is empty), then proceed. Else, do not allow changing the password.
    b. What really happens is they get a new certificate, unless somehow you can change the password of the current certificate via OpenSSL
    c. Old will be deleted, new will be stored and returned as copy

Architecture

__ getcert.cpp

```
\__ certs
    \__ ca
        \__ certs
        \__ crl
        \__ intermediate
            \__ intermediate CA data
        \__ newcerts
        \__ private
            \__ private keys
\__ server
    \__certificates
    \__crypto_lib
    \__csr
    \__private
    \__mail
        \__ user mailboxes
    \__ Makefile and Source Code
\__ client
    \__certificates
    \__crypto_lib
    \__csr
    \__private
    \__mail
        \__ user mailboxes
    \__ Makefile and Source Code
\__ Makefile
\__ README.md
```

Files: getcert.c/cpp, changepw.c/cpp, sendmsg.c/cpp, recvmsg.c/cpp, openssl_setup.sh, gen_cert.sh, backups.txt, Makefile

- server.cpp will generate the server for this system, accepting incoming connections, verifying their certificates, and either adding to files within each mailbox or reading from files within each mailbox.
- client.cpp will generate the connection to the server, sending its certificate along with a get request to either send a message or receive messages.
- getcert.c/cpp will take in a username and password, verify the username and password using its hashed version, and then call gen_cert.sh to generate a certificate for the user which we will also store on the server
- changepw.c/cpp will take in a username and password, verify the username and password, and changes the password. If a user has unreceived messages, the program quits. If it continues, it deletes the pre existing cert, and then calls getcert.
- openssl_setup.sh will set up a certificate authority (only needs to be run once, from Makefile)

- gen_cert.sh will use server certs/keys to generate user certs, will be called by getcert only, and will be the only program with access to server certs/keys (aside from the setup)

Planned Testing protocol

- Test if at any point buffer overflows occur
- Test if a user can send a message to an invalid user
- Test to see if an invalid user can get into the system
- Test to see if passwords are at all able to be accessed by outside users, likewise for messages
- Test if users can change other users passwords
- Test if at any point messages can be corrupted or removed
- Test if the program is secure against issues in the network connection (if connection fails, what data is potentially lost? Back it up, etc)
- Make sure only valid users can generate certs
- Make sure if a user tries to use a cert that does not belong to them that is caught
- Make sure expired certs are not accepted
- Make sure all root certs are read/write protected and not accessible by users
- Make sure we catch any interceptions on network via integrity checking
- Test if multiple messages can be sent with invalid recipients mixed in
- Test if users are able to access other pending mailboxes
- Test if users are able to remove other users pending mail