**Team 17 Methodology, Anonychat**

Ben Korza, Mark Fitzgibbon, Mitchell Caisse

The Anonychat program will consist of a client/server program to be used by individual users, as well as a centralized nameserver to confirm peers to connect are online. The project will be written in C, for Linux Operating Systems. Inter-OS behavior is not a focus of the project. The central nameserver will allow joining clients to get a list of peers to enable broadcasting, as well as query for people to communicate with. Once clients are aware of other possible connections, communication with other clients can be accomplished as such:

If Alice wants to send a message to Bob, Alice’s client will:

1) Check the name server, and see if Bob is currently online

2) Check if Bob has published his public key on the name server.

3) Craft a message, encrypt it using Bob's public key.

4) Send the message to N peers

The N peers recieveing a message will need to determine if the encoded message they received was intended for them to read or to be sent out. When a message is received, a client will make an MD5 hash of the received message, and check against an internal list of hashes. If the hash does exist, then the message is not new to the client and nothing will be done, since it has already been processed. If the hash proves to be new, it is added to the list, along with a time to live, and will attempt to decode the message with its available private keys.

A successful decryption (likely determined by a sentinel string in the message) will imply that the client is meant to display the message to the user. If the message is undecipherable, it will be passed along to a new set of peers, possibly including itself and the peer the message was received from. The specific number of peers in broadcast range will need to be determined when a message is being sent based on what the client knows exists.

Client peers will be determined based on a list of peers it receives from the nameserver. Each possible connection on the list will be given a TTL, and when that time expires the peer is forgotten and the client requests new peers from the nameserver. If a peer goes down before its TTL expires, the client’s behavior may not need to change. A potential method may be to remove the peer from the list early if connection attempts fail multiple times, though this runs a risk of false positives.

Anonymity will be maintained by ensuring that the original source of a message will not be determinable without the ability to decrypt the messages. The broadcast nature of messages means that the target of a client’s communication will be obfuscated.