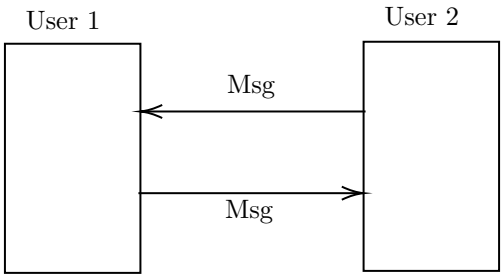
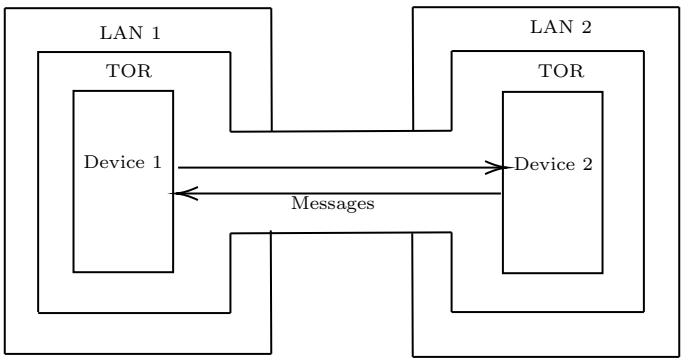
The fundamental design approach of this system is to layer several service on top of one another to add features. These layers are as follows; the database, the message server, the Tor hidden service, and the Ethereum authentication service. The goal of the message server is to receive and send http POST requests to other message servers on the network. Each device is used to act as its on message server and client. Using Tor hidden services to bypass the firewall on the local network allows for requests to be sent or received by other servers regardless of local network configurations [**?**]. To keep track of what address goes to what user an authentication service implemented on the Ethereum network is used to tie users to their TOR hidden service address. The database stores the information of the primary user, the user contacts, and the received messages. Each is designed to work independently of the system below it, allow for more flexibility while implementing the application. In the following sections, the specific design of each layer will be reviewed.

The Database contains several tables: Primary user, Contacts, Conversations, and messages. Primary user is the user of the current instance of the application. Information regarding the address of their TOR hidden server and their username is stored in the primary user table. The Contacts table is used to store the contacts of the primary user, their associated shared keys, and the last updated address of the contact. The Conversations table stores the active conversations, associating the users with the conversations they are apart of. Finally messages are stored in the message table which holds information on the conversation the message is apart of, the message itself, and a time stamp of when the message was received. The message server makes use of the database to store messages and look up conversations to validate received messages.

The Message Server is an http server which receives and sends HTTP post requests and uses SSL for further encryption of the data stream. Requests contain a JSON object with two fields: the conversation hash, and the hashed message. 256SHA is used along with a key shared by each user to decode/hash the data. The conversation hash is used to identify the conversation that the message belongs to. Using this the unhashed name of the conversation the message server then inserts the message into the database storing the associated conversation id and a timestamp of when the message occurred. To send messages the server again uses POST requests to transmit the data in the same fashion it received it, retrieving destination address from the database. One received the http server will return an OK as an acknowledgement Each user acts as both client and server which forms the peer to peer structure of the network.



Tor hidden services are used to facilitate connections to servers behind NATs without port forwarding in a method know as NAT hole punching [**?**]. NAT stands for natural address translation, and is used to hide network IP ranges behind a single network address. They are needed because IPV4 only has a set number of IPs that can be assigned in a network. So by splitting larger networks up into their own isolated networks IPs can be reused. Many NATs and firewalls do not allow servers from outside the network to initiate connections inside the network. So for most messaging servers like Imessage use centralized servers are used that devices can call out to retrieve data. Tor can be used to skip this intermediate step by acting as a rendezvous server [**?**]. Devices behind NATs can reach out to these servers which in turn are able to forward incoming connections to the device. This effectively is what as known as "TCP hole punching" and is what allows devices to act as their own message server. By making a TOR hidden service hosted on the device, incoming http post requests can be received by noctua message servers. The hidden service is separate from the message server and just transfers data between a listening port.



EAS(Ethereum Authentication service) will be used as an identity management and name service. Using smart contracts to allow users to create a username and set an associated address securely. The EAS stores the last update address of the

users Tor hidden service. Each username will be unique allowing for users to add contacts like they would with any other service. When a contact is added the application will attempt to initiate a transaction to find the user in the EAS. If found it will add its contact and address to the users hidden service. When ever a new hidden service is created, a new transaction will be initiated to update the users hidden service address. If a user sends a message to a contact and that contacts HTTP server does not respond another transaction will attempt to find the new address of the contact. Due to the nature of the Ethereum network transactions are automatically authenticated by the network using the users wallet address [**?**] meaning only the user can update their own addresses.