An example of secured and short lived content sharing in peer-to-peer network through an authentication server

Lerato Mosegedi
Banele Matsebula
Thuto Aphiri

P2P Infrastructure

We sought to set up a peer-to-peer network that used an authentication server to attain shared keys to facilitate a secure flow of data between nodes. To do this we chose python as the python libraries are relatively rich in terms of tools. For this particular task we used the Twisted framework to connect to endpoints (servers and clients). As with most p2p networks, each node is required to be a listener (server) and simultaneously to be a client. We implemented this using twisted.protocols where we could define a listening protocol and as well as a client protocol connecting to a peer. We also attached a third protocol to read in user input (see figure 1 below).

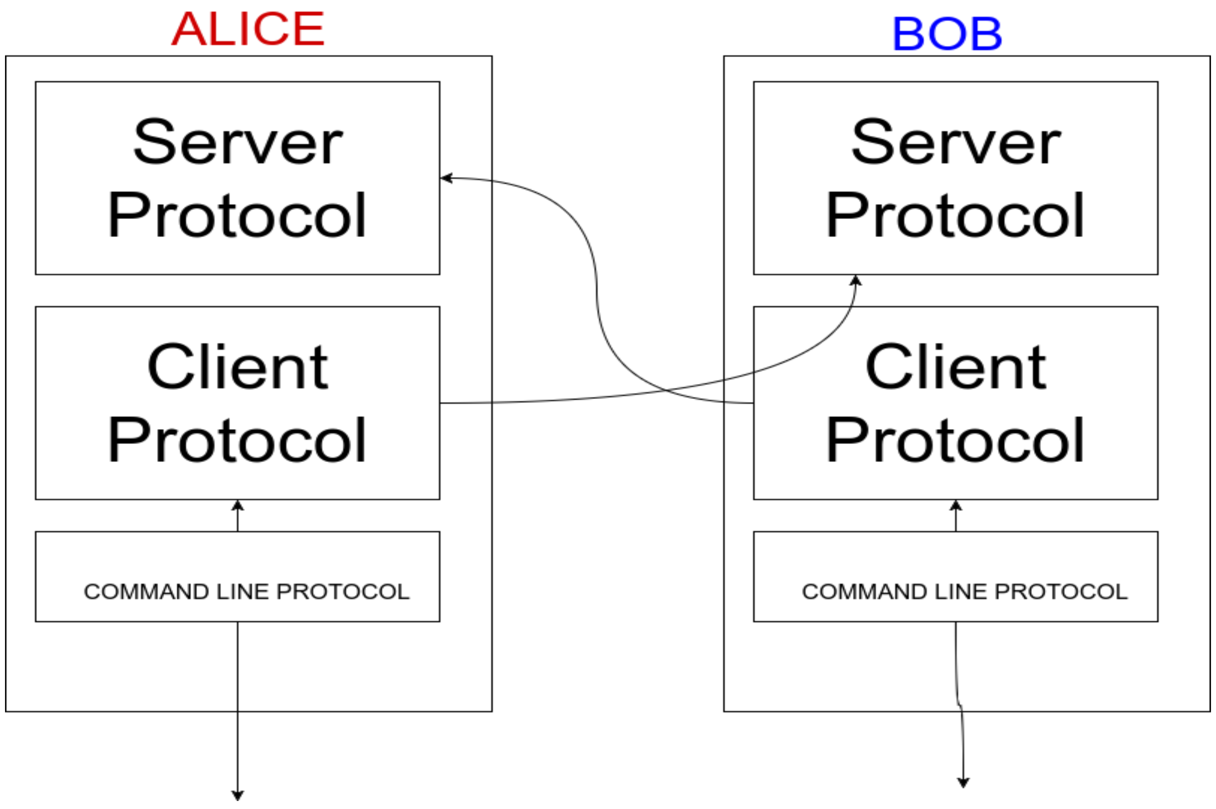

Figure 1

Authentication

We created an authentication server that generates and encrypts shared keys that two peers can use to secure their content. The authentication server acts as a trusted authority that has the keys of all nodes on the network. If Bob wants to communicate with Alice, he will ask for a shared key that they can both use. But first, Bob must authenticate himself on the authentication server, meaning he must provide his password (which he must have encrypted) along when requesting the shared key. Since the authentication server has Bob's private key, it will decrypt the encrypted password and verify it against the one the server knows about. Once verified, the authentication server will SHA a combination of Alice and Bob's private keys along with a 16 character salt. To minimize the chances of running successful collision attacks, we added a ttl to the shared key, essentially making it a session key (once the key expires, the connection between the two ends). The authentication server will return an encrypted message that Bob will use to verify the server. This message is encrypted with the Bob's private key (on the authentication server) and decrypted (by Bob since he has his key). This is a message that the Bob knows and can verify to be correct. This 2 way encryption is meant to lower the risk of having a man in the middle attack or a intercepter spoofing and mimicking the server, as we saw in Lab 1. In summary the authentication server will return 3 items

1. The server's authentication to Bob
2. The shared key encrypted using Bob's private key
3. The shared key encrypted using Alice's private key

If Bob fails to authenticate on the server, a generic message saying "Service not found" is returned". This is meant to minimize the chances of a brute force attack where attacks can use large dictionaries to try and access the shared key. The "service not found" message is intentionally ambiguous (meaning the server might not be available, the requested node might not be available or the password is wrong or it could be that the key used is wrong). This is done to demotivate attackers.
Bob will pass on the shared key encrypted with Alice's private key to Alice. Alice will decrypt the shared key and from then on, both nodes will have the shared key and can thus sign every message thereafter. On

FTP

The network can also handle file transmissions albeit at a very slow rate. Because of the library we are using, which returns characters encoded with non utf-8 characters, we are obligated to encode the encrypted messages in utf-8 before transmitting them over to another node. On the other node, they are decoded then decrypted. Our network is based on json form messages, this was another limiting factor that meant we had to send our files in 2048 byte chunk sizes per second, so as to not flood the tunnel.