Secure Point-to-Point Messaging

Earl Lee, Peter Nguyen, Marvin Qian

Introduction

Our current Peerster has good functionality. It can broadcast messages across a network of nodes, allowing connected nodes to be involved in an open chat room of some sort. It also allows for direct peer-to-peer messaging using a decentralized message transfer scheme and file sharing in a similar manner. However, in order for our Peerster to meet industry standards and be a viable tool in the real world, we must incorporate some form of encryption to secure direct messages. Note that open messages spread across the network using rumor mongering do not need encryption, since they are meant for every receiving node to read. However, direct messages serve a different use case. They are meant to be private and should not be exposed to anyone but the two parties involved.

Thus, we propose to adapt a version of Pretty Good Privacy (PGP) to encrypt direct messages. The primary components of PGP include public-key and symmetric-key encryption, digital signatures, and web of trust, each of which we will implement in our Peerster.

Key Transfer

One of the most critical tasks in securing point-to-point messages in Peerster is to establish a secure way to advertise public keys. The system will use a simple web of trust mechanism that relies on direct connections and manually selecting trusted peers. Peerster will display all peers (IP:port pairs) that are directly connected to the user's node and the user can select those that it trusts and directly notifies those nodes requesting their public key. Those nodes will then respond with their public key and it will be stored by the node as a trusted key. From then on, trusted peers can send a encrypted and signed, direct message with contents that are a public key and origin name pair, and the receiving node will decrypt and verify that message. If the message is intact then the receiving node will trust that the public key and origin name pair is correct and begin accepting secure private messages from that origin.

Once initial, direct trust links are established, further trust can be established transitively. For example, when Alice wants to send Charlie a secure private message, they need to be connected by some path through the web of trust for them to share keys. If Alice and Charlie have both established trust with Bob then Alice can send her signed and encrypted public key and origin name pair to Bob. Bob will then decrypt and verify the message and if that succeeds, he will encrypt and sign the message with his own public key and send it to the peers that he trusts. Thus, Charlie will receive a message from his trusted peer, Bob, then Charlie will decrypt and verify the message and add Alice's public key and origin name pair to his trusted origins. Now, Charlie can respond with his public key through a similar process. Once that finishes, Charlie and Alice will be able to communicate privately through secure point-to-point messages.

While this approach is simple, it has a few drawbacks. One is that broadcasting one node's public key and origin name pair will have to span that entire connected component of the web of

trust. Furthermore, it will trigger responses from all newly trusted nodes which will also increase network traffic. Instead of doing this any time a node wants to connect to a new, untrusted node, the nodes could trigger this process periodically (e.g., once per minute) and discover new, trustable nodes that way. Another way to resolve this is to try to accomplish the transfer through direct messages instead of broadcast along the web of trust. The next hop entry for each origin is not a feasible approach since there is no guarantee that that hop is a trusted peer and, even if it were trusted, there is no guarantee that the closest trusted hop has a path to the target peer. One possible solution is to keep a list of all next hops that are trusted regardless of latency. This could significantly reduce network traffic, but may not be part of the initial implementation.

Sending and Receiving Messages

Once two parties have each other's respective keys, they can begin to send secure private messages. Suppose Alice is trying to send a private message to Bob. She will first generate a random key for symmetric cryptography. She will then use the random key to encrypt her message to Bob. After that, she'll encrypt the random key with Bob's public key. Next, she'll hash the original, unencrypted message to create a digest. Finally, she'll encrypt the digest with her private key to create a signature. The encrypted digest and encrypted random key will be additional fields in the message datagram that is sent to Bob.

When Bob receives the message, he will first decrypt the random key with his private key. Then he will use the random key to decrypt the message. Following that, he'll hash the message to create a digest. Next, he'll decrypt the digest from the datagram with Alice's public key and compare it to the digest he generated from the message. If they match, then the message from Alice reached him intact. If not, he'll assume that someone tampered with the message and discard it.

Suppose a malicious third party, Mallory tries to intercept the message along the path from Alice to Bob. There is no way for her to discover the modify the contents of the message. She can't decrypt the random key since she does not have Bob's private key so she cannot read the message. If she tries to modify the message, the digest won't match. If she tries to modify the digest too, it still won't match since it needs to be signed by Alice's private key which she also does not have. Thus, she cannot create a man-in-the-middle attack except for throwing away all or some of the messages.

Conclusion

In our naive implementation of PGP, we can successfully prevent third-party nodes from reading or successfully tampering with messages sent between two nodes engaging in direct messaging. Hostile nodes will not be able to make sense of encrypted data, and any tampering would be evident upon decryption—so long as the hostile node does not have access to private keys.

We are still struggling to find a good way to build a web of trust given our limited resources. Manually selecting trusted nodes requires our initial trusted nodes to truly be trustworthy, but there is no easy way to confirm that in our small test environment.