Project Stage 2: PeersTor

Ferreira Ribeiro Ricardo, Petrescu Diana

January 13, 2021

1 Topic changes

As suggested, we decided to concentrate on the implementation of a basic version of The Onion Routing network [1]. On top of that we implemented hidden services that allow peers to uses services within the network provided by other peers.

2 Introduction

One of the major drawback of Peerster is its lack of security and anonymity. Those are the problems we want to tackle in our extension. More security through end-to-end encryption and authentication of all the communications among peers of the network would already be a good improvement, but we really believe that the world would be a better place if there were more privacy. Therefore, we want to use the Peerster network to add an anonymity layer. We want to create a Tor-like [1] network called PeersTor and take advantage of one of the most amazing features of Tor, the hidden services.

3 Related work

Obviously our work is closely related to Tor [1], however there exists other anonymous networks. More precisely, there are two kinds of anonymous networks. The low latency networks such as Tor and the high latency networks.

We decide not use an high latency network such as **Babel** [2] or **Mixmaster** [3]. Even tough those networks provide strong guarantees of anonymity being resistant to global adversaries, they introduce large and variable latencies. We found those latencies not suitable to a instant messaging program.

Tor is not the unique low latency network, but it is the most known and well documented one. Still we looked up for other networks on which we could base our implementation of PeersTor.

We tried to find Peer-to-Peer anonymous networks. We found **Crowds** [4] which has a major drawback. It assumes a very weak adversary and does not use public-key encryption which allows any node on a circuit to read user's traffic. Then we found **Tarzan** [5] and **MorphMix** [6] which are designed to scale decentralized Peer-to-Peer systems with thousands of nodes. Their main idea is to generate and relay traffic for other nodes. This allows to hide the creation of a new request within all the relays. This has a major drawback, the

network is flooded by dummy packets which leads to a waste of the resources. Tor seems to be the anonymous network that suits our needs the best. It is to adapt to Peerster and does not add to much burden on the network while it still has strong anonymity and security guarantees. Furthermore, it has a very interesting feature that will be very helpful to build our char room service, the hidden services.

4 System Goals and Functionalities and Architecture

4.1 System Goals

The goal of our project is to add an anonymity layer on Peerster. We want to implement an overlay network that is similar to the Tor network. To be able to build the overlay network we need first to implement point-to-point reliable and secure messages. Once we have this feature we can use it to implement an adapted version of the onion routing protocol. Our system is built on top of Peerster. Namely we needed all messaging and gossiping features of Peerster to be able to build this system.

4.2 Threat model

The threat model of our system is a weaker version of Tor's threat model. This is due to the fact that we will base our implementation on the design of Tor with simplifications that may weaken the security provided by Tor.

4.3 Architecture

4.3.1 Cryptography

To achieve our goals for this system we need cryptography all over the place. To exchange the keys we used Diffie-Hellman. Since no standard go library for Diffie-Hellman(DH) [7] exists, we used monnand [8] open-source library that offers all the tool to compute the different parts of the key exchange. We used the standard go libraries to sign and encrypt the messages. All the RSA [9] keys in the system are 4096 bits long and the symmetric keys are 256 bits long. The signatures use PKCS#1 v1.15. Asymmetric encryption uses RSA-OEAP and symmetric encryption uses authenticated encryption with associated data that uses AE [10] in GCM mode.

4.3.2 Key Sharing and Peers Discovery

First of all, every peer running a Peerster node is considered as a potential Onion Router (OR). We decided not to implement a proper consensus. In Tor the consensus is created and shared by special nodes called directory authorities. In PeersTor we rather use a trusted central entity called the certificate authority (CA). The CA gathers the identities and the keys of the nodes. The CA periodically publishes these identities in a consensus. This CA is a separate server that handles HTTP requests from the nodes to get the consensus, join the network or to update their status when they have already joined.

When a node join the network, it must first create a pair of keys (or load them from a file). Once this is done, the node sends a descriptor to the CA. The descriptor is made of the

identity of the node, i.e., its name, its public key and the signature of its public key. When the CA receives a descriptor it first verifies the signature. This signature is only meant to prove to the CA that the machine who sent the descriptor really owns the pair of keys. This signature is not meant to prevent impersonation of a pseudonym. We decided not to protect against impersonation because all nodes can receive the consensus from the CA. Hence they can verify if their name was used by someone else and act against that. Once the signature has been verified the CA verifies that neither the identity or the public key was previously used. The CA periodically release a new consensus that can be queried by any node in the network. This consensus contains all the nodes that either joined the network or sent a keep alive message to the CA since the last release.

The other problem that arises is the public key exchange between the peers. Indeed, As defined in the Tor protocol [1], each peer that wants to use PeersTor will need to have a pair of keys. To distribute the public keys in a secure and decentralized way would require a whole project. Fortunately, we can use the CA and the consensus for that too. Indeed the peers in the network can trust the consensus they received because it was signed by the CA. The public key of the CA is hard coded in all machines that run PeersTor.

The last task of our CA is to act as the distributed hash table needed in the hidden services to share the onion addresses. We will address that in section 4.3.6.

We are aware that the CA, with respect to security and decentralization, is not an optimal solution. However, with respect to anonymity, this solution does not weaken the assumption made in the design of Tor. Indeed, this CA mimics the publicly available consensus [11] created by the directory authorities of Tor.

4.3.3 Secure Messages

The main difference between our PeersTor and the Tor network is that the communication between between the onion routers is not routed using the same transport layer protocol. Whereas Tor uses TLS between the ORs we had to use UDP to reuse Peerster's existing infrastructure. Therefore, we decided to build our own secure message protocol based on the Peerster's private messages [1]. This protocol is meant to be as close to TLS as possible.

The secure message provides unique a bidirectional secure tunnel between two nodes of our system. In the tunnel the messages are encrypted, FIFO-ordered and reliably delivered. Only one connection is created between two nodes and it is the role of the upper layer to deal with the incoming data. We have built three functions [2] that allow the application layer and the secure layer to communicate. Those functions should be the only way the two layers communicate.

As TLS, the secure message connections start with a handshake (see figure 1). We decided to keep the same names as in the TLS handshake. The client is the initiator of the communication and the server is the peer on the other end of the tunnel.

The message in the handshake must be sent in this exact order. Any out of order message is discarded at this point (except for the messages coming from the client. Those are stored and sent later). Let's say that Alice and Bob are communicating. The purpose of each message is the following:

[1] Disclaimer: You should never roll your own crypto, designing and implementing a cryptographic protocol is a hard task. We decided to do it anyways since it is a fun exercise in the context of a university project whose goal is not to be deployed as is.

[2] SecureBytesConsumer and the SecureToPrivate/Tor functions

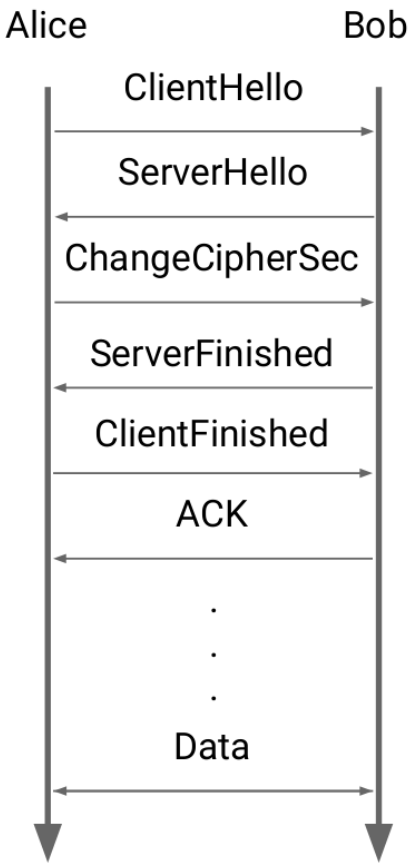

Figure 1: Handshake at the beginning of secure message connection.

1. **ClientHello**: This message is used to bootstrap the connection. It tells Bob to open a channel. It also sends a nonce. At this point a timer is started. This timer is reset each time a new message arrives. If no message is received and the timer expires, the connection is aborted.

2. **ServerHello**: Bob opens a connection and starts a timer too. He answers with a nonce, his part of the Diffie-Hellman key exchange and a signature of it.

3. **ChangeCipherSec**: Alice first verifies the signature in the ServerHello message and then sends her part of the key exchange and with her signature.

4. **ServerFinished**: Bob computes the shared key and sends to Alice the first encrypted message containing all messages of the handshake.

5. **ClientFinished**: Alice also computes the shared key, checks that the ServerFinished message is correct and then also sends an encrypted message containing all messages of the handshake.

6. **ACK**: Finally, Bob verifies the last message and then sends an ACK to tell Alice that she can start sending the Data messages.

At this point the connection is established between Alice and Bob. Alice can start to send all messages that were pending. Each time that a Data message is received, its payload

is decrypted and the receiver checks that it is correct, i.e., that the encrypted nonce and the sequence number are the same of the ones in the message header. If the message is correct an ACK is sent. The messages are delivered to the the upper layer in the same order they were sent. If no ACK is received, there is a timeout and the message is sent again. The timeout duration is doubled. This can happen at most four times in a row. After the fourth timeout, the connection is aborted.

4.3.4 Path Selection

Before being able to construct a circuit through the PeersTor network we need to select the nodes that are part of the path. In comparison to Tor path Selection is simplified. We decided to implement a very basic path selection where, for each connection, we draw two nodes at random from the set of all nodes in the most up to date consensus. You may ask why not three as in Tor? The messages never leave the PeersTor network, therefore two intermediate nodes and the final node being the destination is sufficient to have the same anonymity as in Tor.

4.3.5 Onion Routing

We adapted the Onion routing protocol of Tor to the Peerster network. All the point-to-point connection uses secure messages. We added a new type of messages called Tor messages. Those message have different fields:

- **Create**: Tells the OR that it is part of a circuit that is being created. Alice's DH key part is encrypted with the OR public key. On the way back the OR's DH part is not encrypted but an hash is used to prove the integrity of the key.
- **Extend**: This field tells an OR that it must forward a create packet to the next hop. The next hop is also specified in the extend packet.
- **Relay**: This tells an OR to decrypt the payload of a packet and process its content.

For ease of comprehension, let's say that Alice wants to talk to Bob. She selects two nodes OR1 and OR2. Alice must first exchange a pair of key with each node in the circuit(as shown on figure 2).

Once the keys have been exchanged Alice can start to send her messages to Bob (see figure 3). One thing that is worth noticing is that Alice can hide is identity (an option is given to her to tell Bob who she is). Even though Bob does not not who Alice is, he can reply to her using the same circuit. The circuits are bi-directional. The message are encrypted with three layers of encryption from Alice to Bob. On the way back each node adds its layer of encryption.

As for secure messages, we used timers to destroy inactive circuit. We try to detect crashes in the path with the help of the consensus. If a node has crashed, it eventually disappears from the consensus. When Alice notices that a node in one of the path she created is missing in the consensus, she destroys this path and build a new one. The first thing she does with this new path is to send again all messages that were not delivered yet.

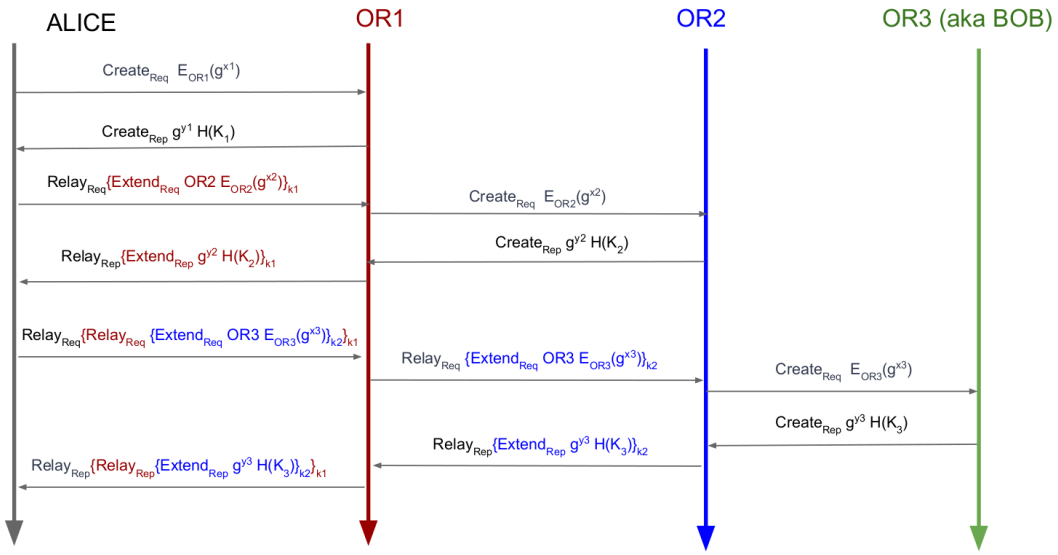

Figure 2: PeersTor key exchange. E() is a RSA-encryption and {} is an AES encryption.

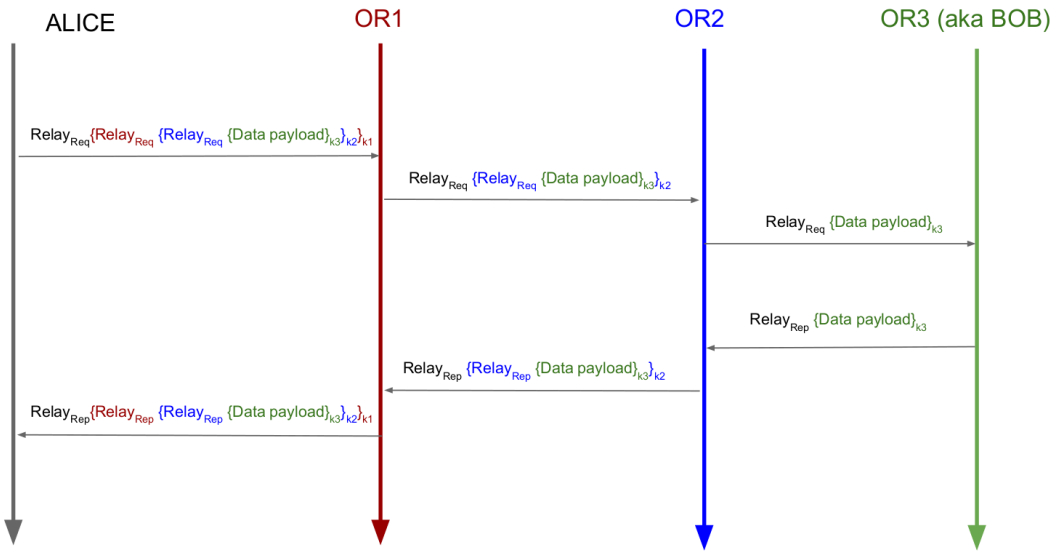

Figure 3: PeersTor Data exchange.

4.3.6 Peerster Services

One important part and interesting use of Tor are the hidden services, also called onion services. They allow the client and the server to be anonymous so that the client and the server talks to each other without knowing who they respectively are (server and client anonymity). These services are hidden as the name indicates it and can not be easily discovered, especially as network becomes bigger. There are some vulnerabilities but they aren't common. Some other problems happen for example, when services are reachable through Tor onion services and the public Internet. They are therefore susceptible to correlation attacks and thus not perfectly hidden (but this will not be our case). Other pitfalls include misconstrued services

(e.g. identifying information included by default in web server error responses), uptime and downtime statistics, intersection attacks, and user error. We will keep this in mind during our implementation. Hidden service description are used for this protocol and they contain the service public key and IP addresses of the introduction points. The descriptor is normally published to a distributed hash table. Therefore, all routers hold some part of the information of the hidden services. The onion address (.onion URL) that is derived from the public key of the server isn't publicly available, the clients have to know this address in order to contact the servers. The client will choose a rendez-vous point (therefore establishing a circuit through it) and will make contact with the server through one of the introduction point (sending it the address of the rendez-vous point). The client sends an identifier and the server responds with the same identifier to the rendez-vous point (establishing a second circuit) so that the rendez-vous point knows that they are talking with each other.

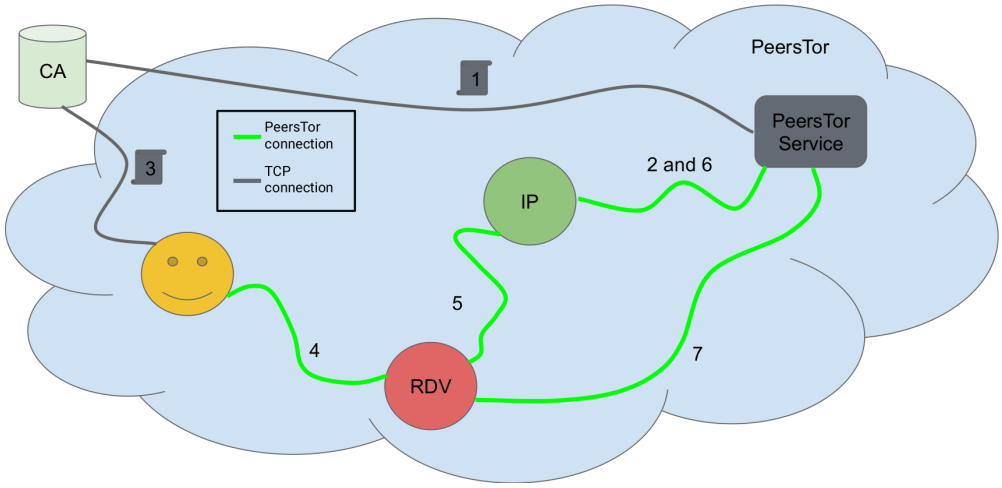

Figure 4: Overview of the PeersTor Services.

The implementation of the PeersTor Services is really close to the one of the hidden services. First, the node in PeersTor that wants to run a new PeersTor Service must create a service. For this purpose it needs to create a server that runs its service. Then it generates a pair of keys. From the public key it generates the onion address [3]. It selects a peer in the network, the Introduction Point (IP). This special node will be used by other peers to introduce themselves to the service. Finally, it sends a descriptor to the CA (step 1) containing its public key, IP's name, the onion address and the signature of the descriptor. The hidden service create a PeersTor path between him and the IP and ask the IP to be its the introduction point (step 2). Once the service is up, the other nodes in PeersTor can access it thanks to the onion address. They need to query the CA to retrieve the descriptor. Once they have the descriptor, they verify it is genuine, select a rendez-vous point and generate a one-time password called the cookie (step 3). After that, the client sends a packet to the rendez-vous point via a PeersTor connection. This packet contains the session cookie and the identity of the introduction point (step 4). The rendez-vous starts a new PeersTor connection to the introduction point and sends a message through it containing its own name and the cookie sent by the client (step 5). The introduction point forwards the packet sent by the rendez-vous point to server via the previously built channel (step 6). Once the server

[3]The onion address is the half of the base32-encoded SHA-1 hash of the public key.

has received the cookie, it creates a PeersTor connection to the rendez-vous and send him the cookie. The rendez-vous point notifies the client that he can start to send data to the server through him. From now on, the rendez-vous point only acts as a bridge that forwards the packets from the client to the server and vice-versa (step 7).

If we would stop here, the rendez-vous point would be able to peek into the conversion. To avoid that the client and the server need to exchange a key using Diffie-Hellman and then encrypt all messages. Once the key is shared, the client can interact with the server with HTTP as a normal client/server interaction. As a proof of concept we implemented a PeersTor service that runs an exciting website about unicorns.

5 Work Repartition

The design of PeersTor has been discussed at length together and the implementation choice were first approuved by both of us. The different parts of the project were closely linked. Therefore, most of the work was done together. However, Diana worked more on the CA and the onion routing while Ricardo worked on the secure messages and on the hidden services.

References

[1] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pp. 303–320, 2004.

[2] C. Gülcü and G. Tsudik, "Mixing email with babel," in *1996 Symposium on Network and Distributed System Security, (S)NDSS '96, San Diego, CA, USA, February 22-23, 1996*, pp. 2–16, 1996.

[3] "Mixmaster." http://mixmaster.sourceforge.net. Accessed: 2019-11-22.

[4] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM Trans. Inf. Syst. Secur.*, vol. 1, no. 1, pp. 66–92, 1998.

[5] M. J. Freedman and R. T. Morris, "Tarzan: a peer-to-peer anonymizing network layer," in *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pp. 193–206, 2002.

[6] M. Rennhard, *MorphMix - a peer to peer based system for anonymous Internet access*. PhD thesis, ETH Zurich, 2004.

[7] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[8] monnand, "dhkx." https://github.com/monnand/dhkx. Accessed: 2020-11-22.

[9] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[10] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography, Springer, 2002.

[11] "Consensus health." https://consensus-health.torproject.org/. Accessed: 2019-11-22.