## Literature Survey for Tor / Onion Routing.

Tor, or The Onion Router, is a browsing tool used to surf the web anonymously. By using the end-user’s IP address, websites could use identifying information to target advertising and collect data about the user’s browsing habits. In a normal internet connection, the end-user is directly connected to the website he is trying to visit.

The Tor network is made up of countless nodes, or relay points, that pass your data along using layers of encryption – hence the onion metaphor. Each node that the end-user’s data passes through peels off another layer of encryption. The last node the end-user’s data passes through is known as the exit node, and it peels off the final layer of encryption and then delivers your data to the intended server. The point of origin, and the intermediary nodes is unknown.

In our current implementation, we set up a browser interface with a series of nodes. By shuffling through the list, we initialize three intermediary nodes, with one node also acting as the Tor Directory Server.

The end-user would go to a website, for example, google.com. The end-user prepares a 3-layered encrypted HTTP request and sends it to the first node. The first node will decrypt the first layer, peeling the encryption layer off and passing the packet to the second node. The second node will do the same and pass to the third node. The third node will peel off the last layer and pass the raw HTTP request to the HTTP server of google.com.

The third node then receives the HTTP response, encrypts it and passes it to the second node. The client will receive a 3-layer encrypted HTTP response, which he then decrypts to get the raw HTTP response and pass it to the browser to process and render.

**How is it different from Tor and similar projects?**

|  |  |  |
| --- | --- | --- |
| Implementations | Our Tor Implementation | Main TOR Implementation |
| Language it is written in | Python with NodeJS front end | C |
| Byte Size | 4970 bytes | 512 bytes |
| Cells | Header: Contains checksum, length of relay payload, and relay command | Header: Contains padding, create and destroy (of circuit), streamID, checksum, length of relay payload, and relay command |
| Constructing a circuit | Nodes randomly shuffled, chosen. A create cell is sent along to the first node. Diffie-Hellman Handshake implemented. | Diffie-Hellman Handshake implemented. Circuit established first by sending the nodes along |
| Congestion Control | None | Circuit-level throttling and Stream-level throttling implemented. |
| Number of Nodes implemented | 3-5 | 3 |
| Rendezvous Point | None Implemented as it was not in our intended scope due to time constraint. | Provided for responder anonymity via location-protected servers. Meant for hidden services. |
| Localhost | Due to network constraints, we were unable to move between multiple computers and IP addresses. Thus we opened a series of ports on our local computer to operate as nodes | Global nodes. |

**Related works**

Mix Networks: Outlined by Chaum in 1981, it provides one-way or two-way anonymous communication. It is a process where a message gets encrypted using public key cryptography in a chain of gates or servers where the message has to go through. Each gate can decrypt the cipher using its own private key so that the last gate will be able to transfer the message to its destination. Public components transmit through the mix by creating a pair including public and private key, which are generated by the mix.

The end-user specifies the location of a message and selects an order of mixes. The encryption algorithm proceeds in reverse order of the end-user’s path, and encrypts the message with the public key of the next hop in the mix. Each mix then sends the message and detaches an encryption layer.

There are high-latency systems which use batching strategies that produce long time delay between sending and receiving messages. Such long time delay tolerable applications could be found in message-oriented systems such as remailer systems.

Type 1 remailers chain multiple remailers, hence the message remains anonymous to each remailer. A client would select an order of remailers and each remailer uses its private key to decrypt the message, strip the header away, and forward it to the next remailer.

Type II remailers improve on Type I by adding batching and random padding. To avoid replay attacks, the packet IDS of the headers are kept. This protocol involves dynamic pool flushing algorithm, and SMTP for transport messages. However, it does not support anonymous messages.

Type III improves on Type II by making use of smaller amount of synchronized redundant directory servers to present uniformly structured information about the network.

Links:  
<https://www.cse.wustl.edu/~jain/cse571-11/ftp/anonym/index.html>: A summary of the different anonymity systems in place.

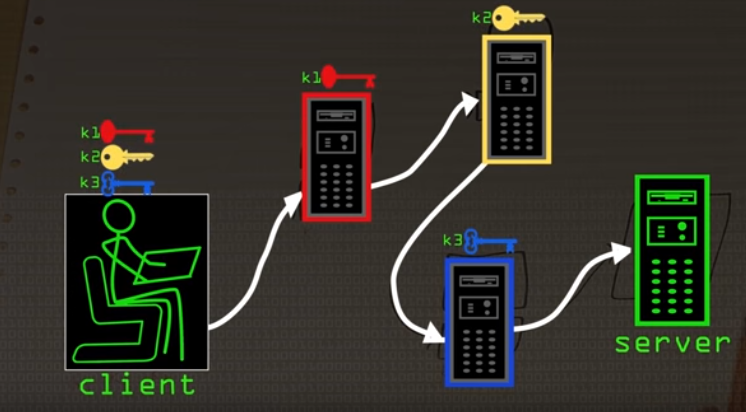
<https://www.torproject.org/about/over>: An overview on the Tor ([view using WayBack Machine at web.archive.org](http://web.archive.org/web/20181106224936/https:/www.torproject.org/docs/onion-services.html.en))

[Tor: The second-generation onion router](https://scholar.google.com/scholar?oi=bibs&cluster=14435376457342883792&btnI=1&hl=en) P Syverson, R Dingledine, N Mathewson - Usenix Security, 2004: an updated version of the previous Tor implementation as held in 1999

[Onion routing](https://scholar.google.com/scholar?oi=bibs&cluster=7502327163795732821&btnI=1&hl=en): D Goldschlag, M Reed, P Syverson - Communications of the ACM, 1999

**Slides:**

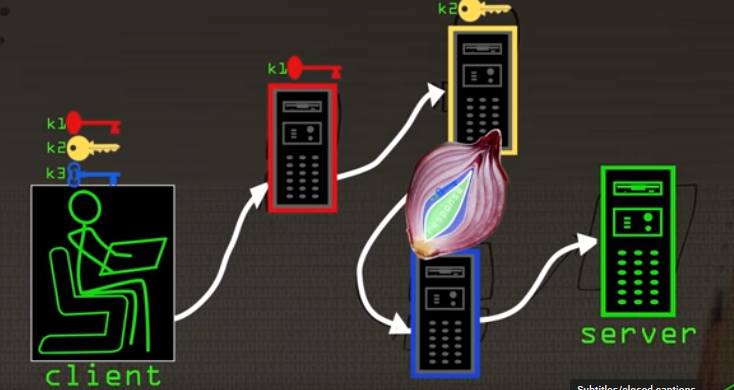
*What’s clever about onion routing is that the respective nodes don’t know the entire connection. They just know what’s before them, and what’s after them. Shared symmetric keys have been established between the three nodes. We use an AES Key exchange, issuing the keys in a central directory server.*

*Each node has their own public/private key. If I picked something with K1, only the first node can see it. Onion routing sends messages encrypted multiple times in encrypted layers.*

**

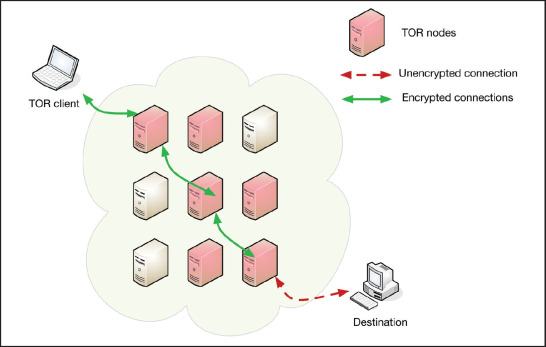
*Only the first node can unlock k1, with node 2 unlocking k2 and so on. Node 1 can’t understand anything because it is rubbish, but it can forward it to node 2. It can strip off its layer. This continues on to node 3, where it gets the original, unencrypted server, and the node, known as the exit node, can immediately pass on the request to the respective server.*

*On the way back, this happens:*

**

*Node 3 will add its encryption of k3, and forward the message onto node 2, who will add k2, and node 1 will add k1, and only the client/end-user can decrypt the whole message. Node 2 does not need to know anything except the address of node 1, all it knows that based on the protocol, it has to pass the message to node 1 after applying its encryption/decryption, depending on whoever is doing it.*

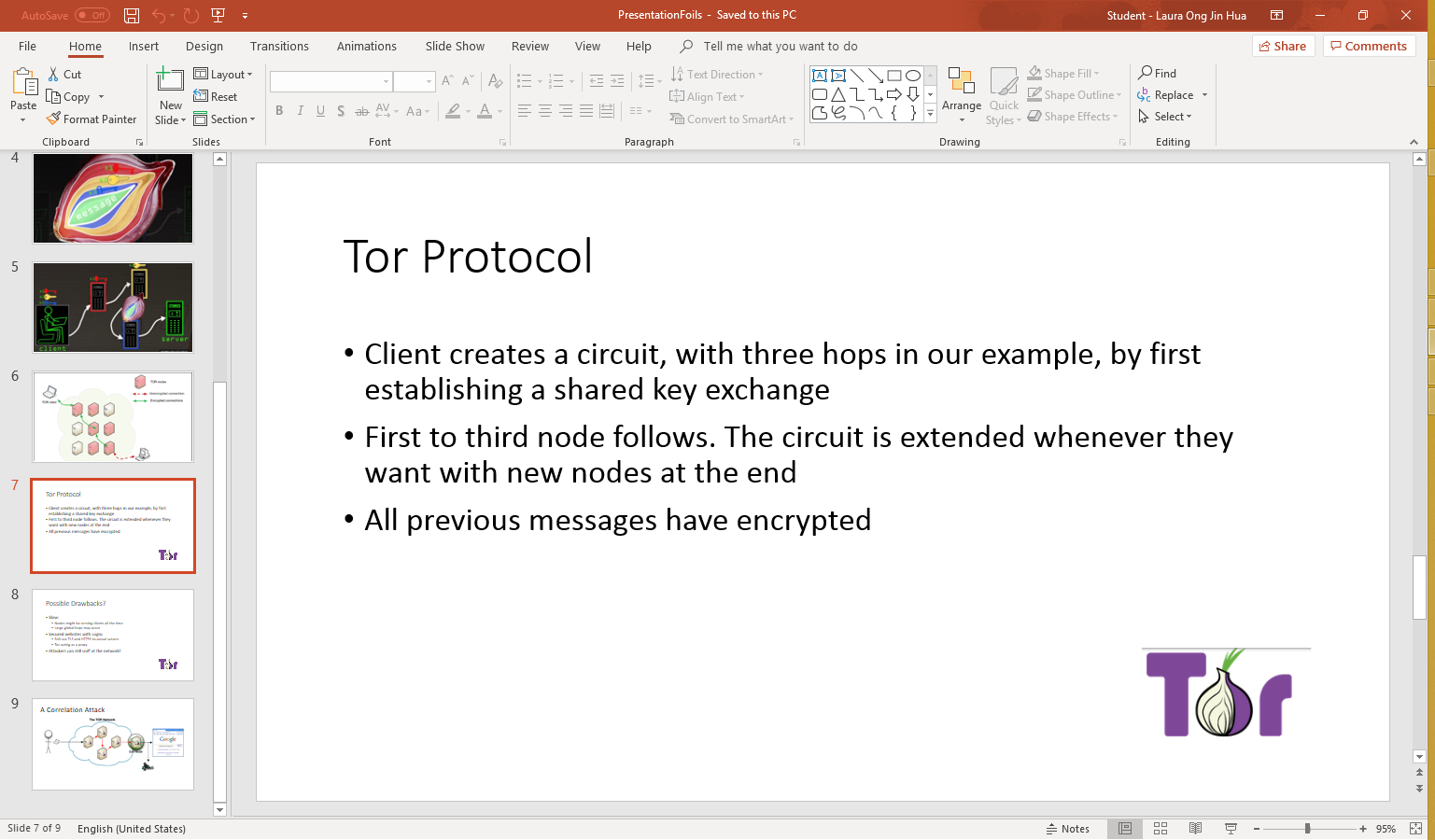
*An attacker would know that the nodes are Tor nodes or onion routing nodes. They haven’t learnt the identity of the user, don’t know which server is being accessed because both times it is layered with encryption that it cannot remove.*

*­*

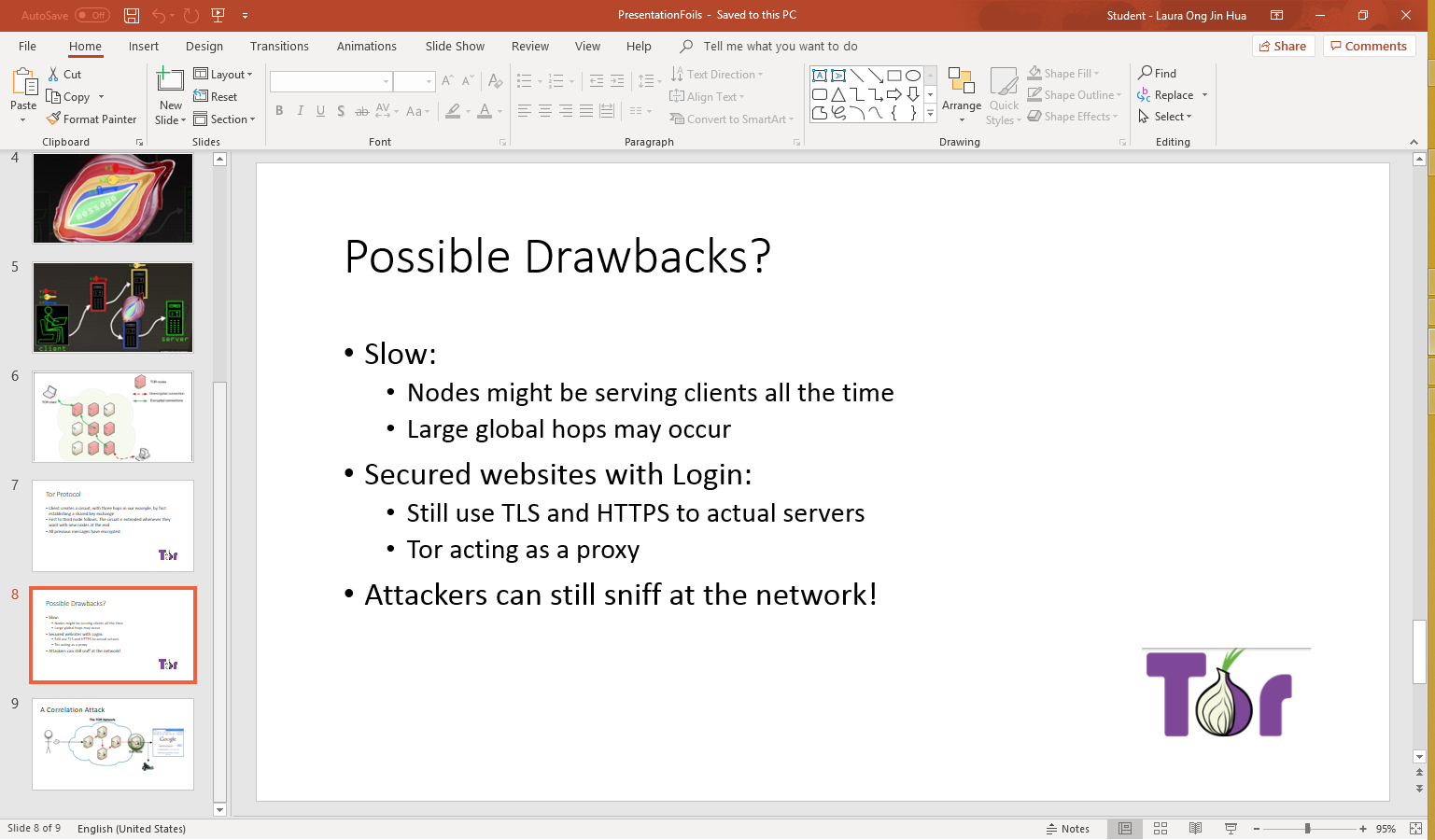
*Taken from https://www.sciencedirect.com/science/article/pii/S1353485816300289*

*But wait, the node 3 knows the server! However, node 3 does not know the client. It doesn’t know how many more layers there are; onion routing could have multiple servers, but Tor fixes it at three. Output node knows the server, and similarly, input node knows that the client is using Tor.*

*With regards to the protocol being used, it establishes how you send the keys, how you establish contact, how each node should decrypt them and forward them to the next one.*



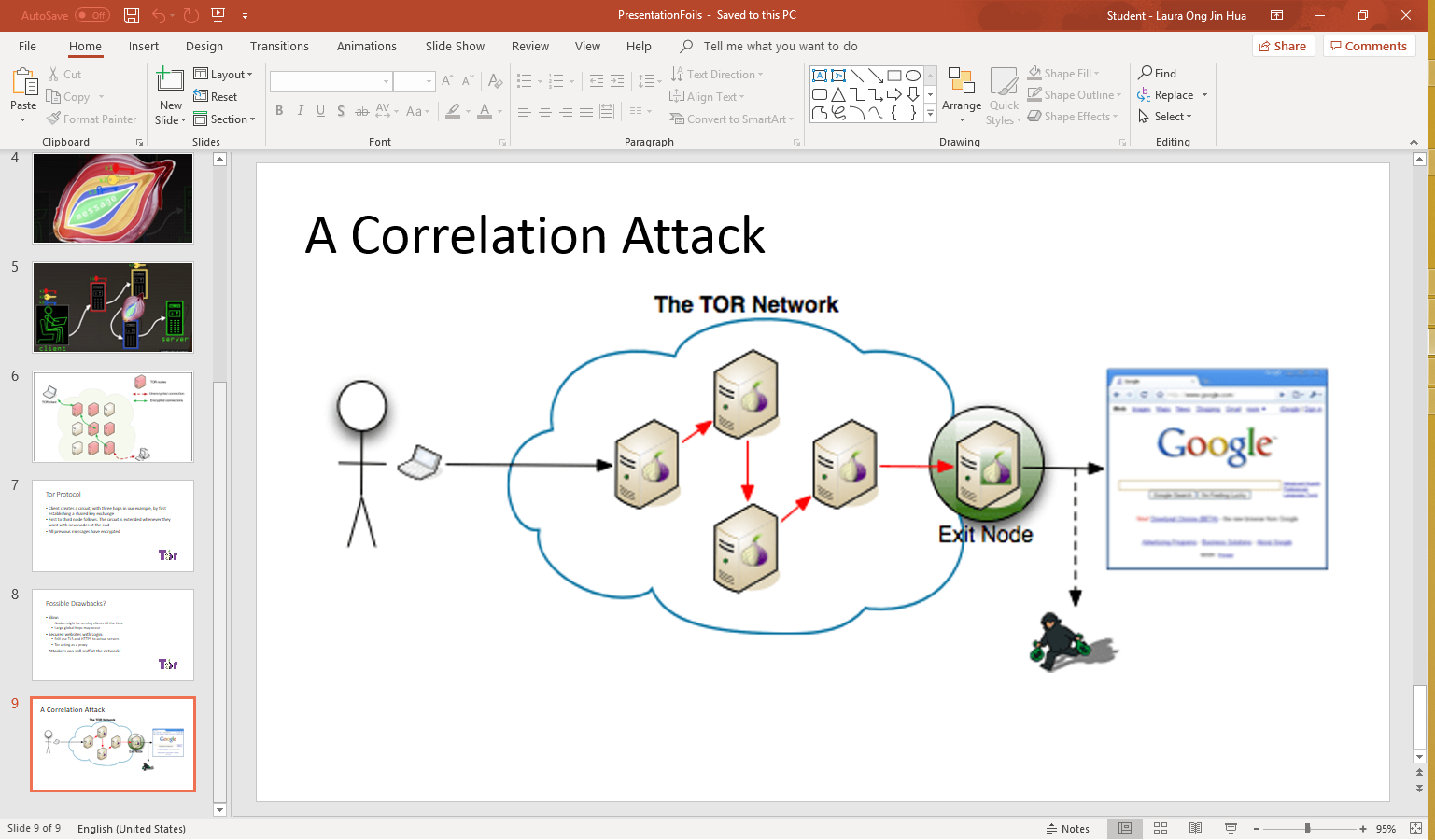
*The client will create a circuit, with three hops in our example, by first establishing a shared key of him and then instructing him to do the same and then the third node follows. The circuit is extended each time they want with new nodes on the end and every time this is done, all previous messages have been encrypted.*



*Is it slower? It’s a lot slower, because these nodes might be serving clients all the time, so that might be shared keys with other nodes, with high bandwidth needed due to the additional traffic. Furthermore, they not be located close together, large global hops that take a little bit longer for messages to get through.*

*How about secured websites like Facebook? I still need to give my username and password, so I still have to use transport layer protocols such as TLS or HTTPS connections to the actual servers. Once circuit is set up, you send HTTP requests normally, just that instead of it going through your normal internet connection, and ISP router, it goes through this circuit. So Tor works like a proxy; with the client talking to this proxy server which handles all this and responses come back. Web connection is slower, but it’s still a normal connection to the server.*

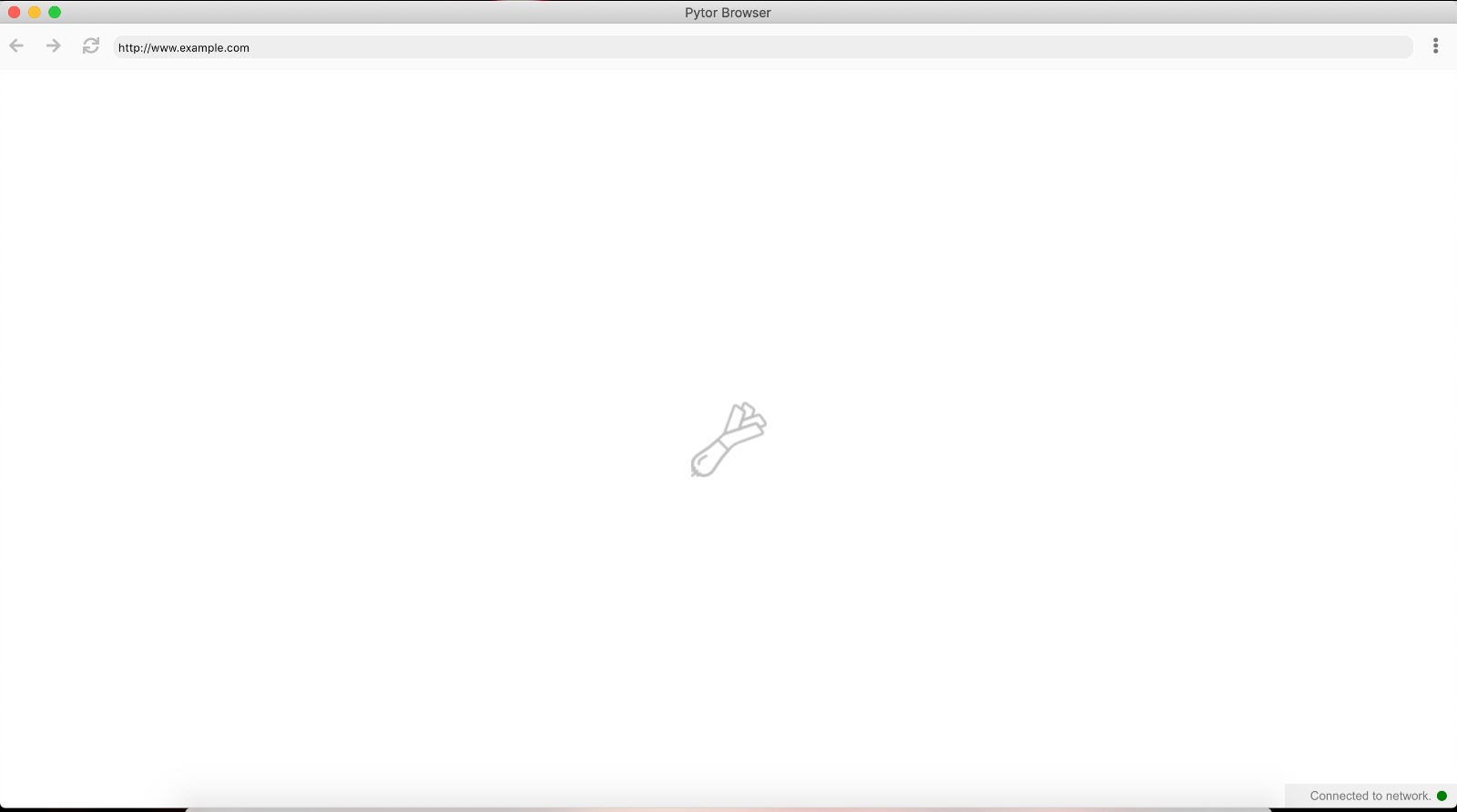
*But a few downsides. Besides the speed, attackers can sniff on this network and start to understand which nodes are talking to one another, their approximate positions.*



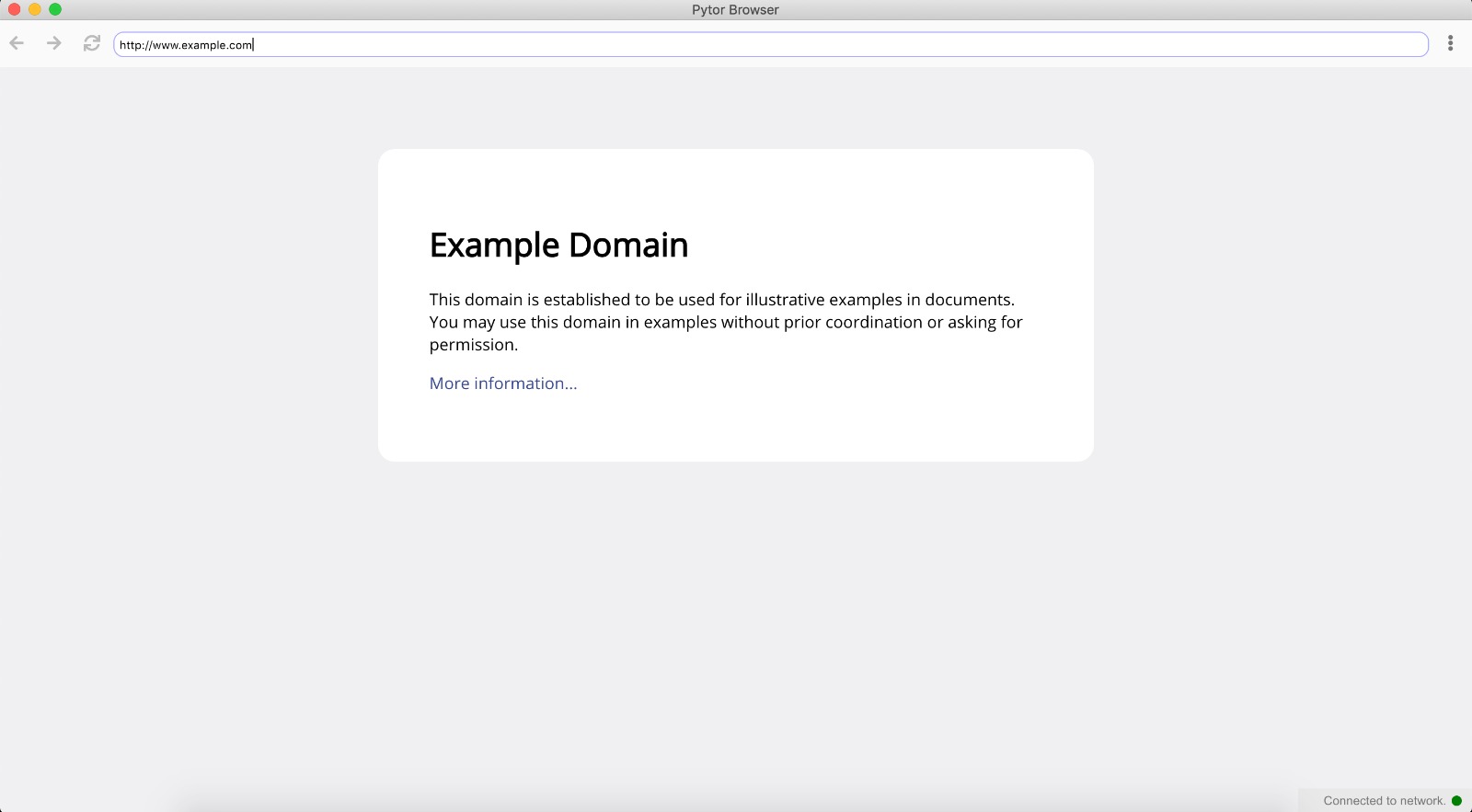
*Say that you are sniffing at nodes 1,2,3. You can only see that you are using Tor. But you can’t see anything else. The nodes are stored in the Tor directory server, there’s no point in trying to find the location. However, between the server and node 3, if I find out that someone on Tor is visiting this website, if it’s encrypted, then I can sniff before node 1 and node 3. That’s the essential weakness of Tor, so the only solution is to be very careful. Node 1 does not have only one user, it might act as node 2 or exit node, it might be talking to different clients. So, if you look at the traffic coming to this router, and got the time signatures, then you can find the time occurrences that match up between node 1 and client; and between node 3 and server.*

*If you found out the 15 messages that was sent in was created and sent out at the exit node a while later, you can de-anonymize the client.*

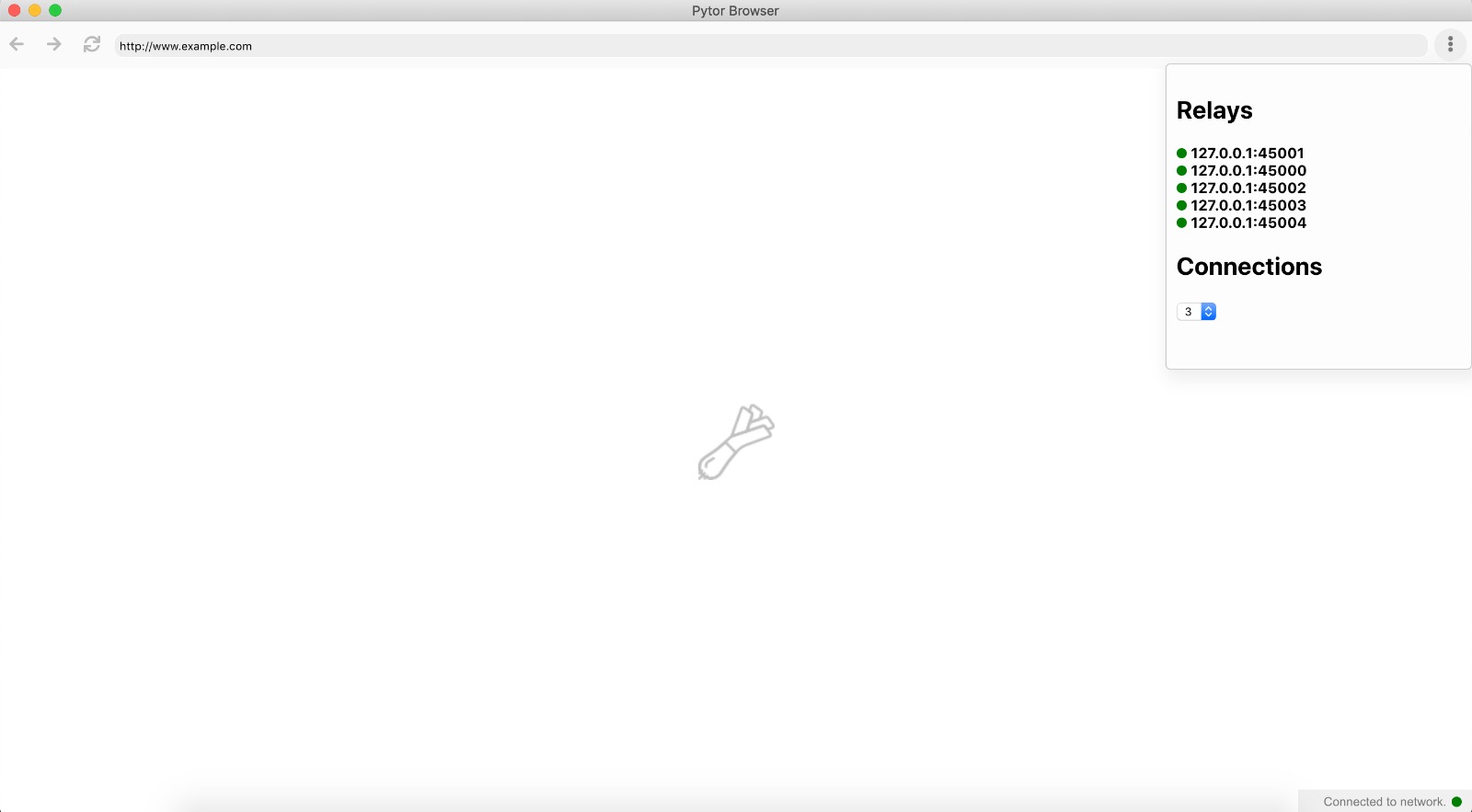
**Our Implementation**

****

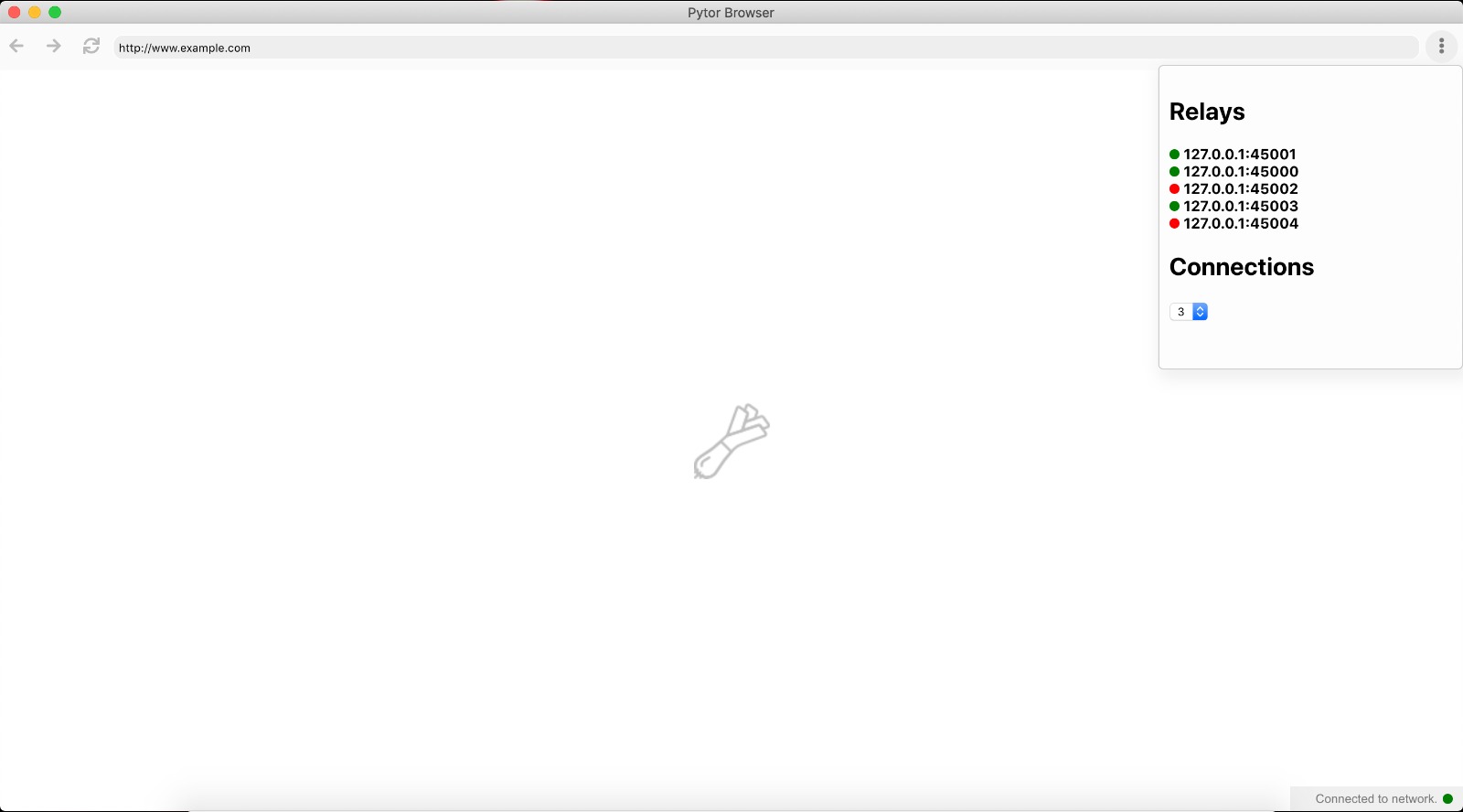
Our setup page.

****

We have implemented our browser using an electron interface. By entering the URL, we are able to access simple HTML pages.



We can find the number of relays online.

Once the node is offline, the network reassigns new nodes for the relay to take place.