Tor: The Second-Generation Onion Router

Thomas Maier Tom Petersen

January 16, 2017

The paper *Tor: The Second-Generation Onion Router* [6] was written by Roger Dingledine, Nick Mathewson and Paul Syverson in 2004. It introduces a system for anonymous communication connections preventing traffic analysis or surveillance, which can be used to anonymize the traffic of a lot of real-world internet protocols like HTTP to request websites.

The first section presents related work Tor is based on as well as other systems dealing with the same problem field. The following section describes how Tor works and what improvements Tor has implemented. Afterwards attack types against Tor are described. The final section shows the influence the discussed paper had to science and the Tor network has to the internet and our society.

1 Previous work

This chapter will cover the previous work which has lead to the development of Tor. In [3] David Chaum presented the mix concept which used the at that time freshly invented priciple of public key cryptography to build the theoretical foundation of untraceable electronic communication.

In 1995 the work on developing onion routing began - a technique to offer anonymous connections[1] using some of the concepts of Chaum [4, 1, 5]. In the onion routing network an initiating application makes connections through a sequence of machines called onion routers. The data sent by the application consists of layers of encrypted data (like an onion) containing the next destination of the message in the network. Each layer of the onion is decrypted by one onion router and contains the next hop in the route. So each onion router knows only its adjacent onion routers. Because of that an onion router is not able to know who is communication with whom.

There are some applications which have implemented the mentioned concepts. They can be differed in high-latency networks, which try to maximize anonymity requiring higher latencies, and low-latency networks, which enable the processing of interactive web traffic at the cost of lower anonymity. Eavesdropper at the end of connections are able to correlate ingoing and outcoming traffic of the network easier. Examples for high-latency networks are Mixmaster or Mixminion, which offer a service to send anonymous emails. Examples for Pre-Tor low-latency network applications are JAP, which is based on fixed mix cascades instead of variable circuits, or MorphMix, a peer to peer based anonymity network.

In 2004 the original Tor paper was published (including one of the developers of the original onion routing technique as an author) and the network was deployed.

2 Tor

The details mentioned in this section are taken from [6] if not stated otherwise.

Tor is a circuit-based low-latency service providing anonymous communication improving the concepts of the already explained onion routing. It is designed to anonymize TCP connections. This enables Tor to work with most standard applications of the net like HTTP to visit websites and other application layer protocols.

Users of the network use an **onion proxy** to connect their TCP connections to the Tor network. They establish a circuit through the network containing several nodes, so-called **onion routers**. Every onion router just knows his predecessor and successor in the circuit. How this works in detail is covered below. An overview about the exchanged messages can be found in figure 1.

2.1 Creating circuits

Tor uses TLS connections between the onion routers to prevent message tampering by external adversaries and providing perfect forward secrecy (see next section). A client who wants to use the network has to establish a circuit consisting of several onion routers incrementally.

[1] The communication using these connections does not have to be anonymous. The goal is to limit the traffic analysis in a way that makes it impossible to determine who is communicating with whom through a public network.

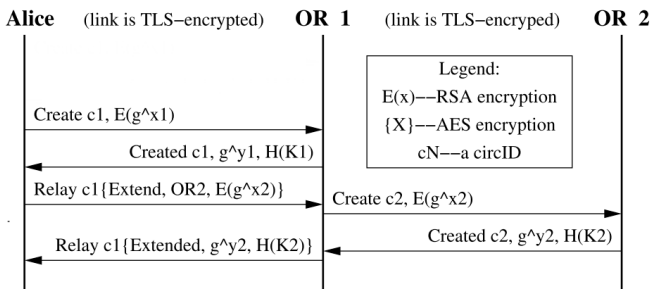

Figure 1: Creating a circuit with two onion routers. Taken from [6] and slightly edited.

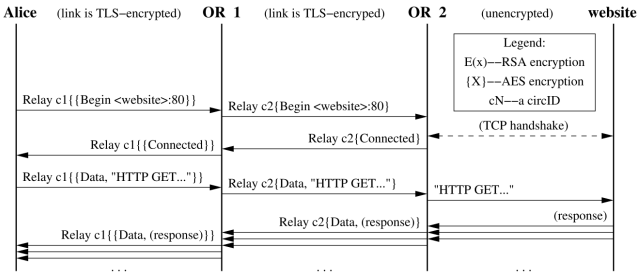

Figure 2: Requesting the content if a website via HTTP over an established circuit. Taken from [6] and slightly edited.

Let's call the client Alice and the first onion router on her chosen circuit Bob[2]. Bob (as well as every other onion router) owns a long-term identity key pair to sign TLS certificates and his router description and a short-term onion key pair used when setting up circuits.
Alice sends Bob a *create* control cell[3] containing a unused circuit id and a DH[4] key exchange parameter g^x encrypted with Bob's onion key. Bob decrypts this parameter, computes the shared key $K = g^{xy}$ and sends Alice a *created* cell containing the circuit identifier and the DH parameter g^y so that Alice can compute K too.

When Alice wants to extend this circuit to the next onion router Carol she sends a relay *extend* cell to Bob, which contains the address of Carol as well as a DH parameter g^{x_2} encrypted with Carols onion key. This data is encrypted symmetrically with AES-128 and the already negotiated key K between Alice and Bob.
Bob decrypts this information, chooses a unused circuit identifier between Carol and him and forwards the information received from Alice to Carol in a new *create* cell. The same process as already described takes place on Carols side and she sends g^{y_2} back to Bob after computing the key $K_2 = g^{x_2 y_2}$. Bob encrypts this information with K and sends it back to Alice who can also compute K_2 then. The circuit is extended by another onion router now and Alice has a key shared with Bob and one shared with Carol.

2.2 Sending TCP data through a circuit

After having established a circuit Alice can start to send data over the Tor network. This is illustrated below for the simple case of requesting a website. An overview can be found in figure 2.

In the first step Alice sends a relay begin cell to open a new TCP stream. The content of the cell (the begin command, a stream identifier, the website address and a digest of the message) is encrypted multiple times with the keys obtained when creating the circuit in order of the onion routers on the circuit. So Alice encrypts the message with Bob's key first and then with Carol's key and sends it to Bob.

Bob decrypts the information the first time and forwards it to Carol. Carol decrypts it the second time, gets the original request from Alice and performs the TCP handshake with the requested webserver. After a successful connection she answers with a *connected* relay cell encrypted with the with Alice negotiated key. Bob encrypts the answer a second time and returns it to Alice who can then decrypt it twice and knows the handshake was successful. Now she can start to send a HTTP request over the established TCP stream packed in relay *data* cells in the same way as before.

But how does Bob know he is not the addressee of the cell during the handshake? This is where the message digest comes into play. Every onion router checks if the message digest he obtains after decrypting a cell is valid for the cell data. If this is the case the cell should be received by him and he can perform the required task (opening a TCP connection in this case). If the message digest is not valid, he forwards the message to the next receiver on the circuit. This topology is called leaky pipe - enabling a user to choose different exit points for streams using just one circuit.

2.3 Improvements to onion routing

While the approaches described in the previous subsection are quite similar to the original onion routing techniques, there are several improvements in Tor which this subsection will cover.

[2] Traditions have to be preserved!

[3] The protocol data units of Tor - called cells - have a fixed size of 512 bytes and always contain circuit identifier, the command and a payload. The command can be *control* for cells which are interpreted by the receiving onion router or *relay* for carrying end-to-end data.

[4] The Diffie-Hellman key exchange is a method to exchange keys over a public network using public-key cryptography. Details can be found in [7].

Directory servers: Both - onion routing and Tor - have to give users access to state information about the network (known onion routers and their current states). In the onion routing design the network is flooded with these state informations (similar to normal routing protocols), which can be unreliable, complex and give attackers opportunities to exploit different client knowledge. Tor uses so called *directory servers* to give clients access to the signed current network state. The users get a predefined list of directory servers with their Tor application and contact them periodically via HTTP to update their known network state.

Rendezvous points and hidden services: Previous sections all were dealing with how to prevent traffic analysis through the network to a well-known server. But Tor also has the ability to offer hidden services - servers which real addresses are hidden and which can just be contacted via so-called rendezvous points. This can for example prevent precise DoS-attacks.

An operator can announce several onion routers as contact points for his hidden service and builds circuits to them from his service. A client, who knows about the hidden service, can lookup the contact points and inform a contact point of a chosen onion router where the circuits should connect - the rendezvous point. The connection of both circuits is done by using a rendezvous cookie which is sent to the location point of the hidden service. Once again the Diffie-Hellman key exchange is used to negotiate a shared key between client and server to prevent even the rendezvous point to look into the message content.

The onion routing design used long-living "reply onions", which were used to build a circuit to a hidden server. This approach failed if any onion router on the preestablished path went down or updated his keys.

Perfect forward secrecy: Perfect forward secrecy describes the property of a key exchange protocol. By using ephemeral session keys, which cannot be derived from long-term keys and are deleted when not being required anymore, older messages cannot be decrypted even if the long-term key gets compromised.

The original onion routing suffers from the problem that a single malicious node can record traffic and by compromising following nodes in the circuit making them to decrypt the recorded traffic. To avoid this Tor uses an incremental path building design. The initiator negotiates session keys with each node in the circuit. When these keys are deleted after a circuit has been destroyed, the compromised nodes are not able to decrypt the recorded traffic.

Another advantage is the usage of TLS between the onion routers enabling perfect forward secrecy on connection level.

Congestion control: If many users choose the same connection between onion routers in their circuits, it is possible to overload a connection. To avoid this, Tor is able to control the circuit bandwidths in each onion router. It does this by using a TCP like window approach for incoming and outgoing connections.

Integrity checking: Tor uses an end-to-end integrity checksum in its messages, which prevents altering messages in between. Additionally using TLS between the onion routers hampers the altering of data in connections between onion routers.

Exit policies: Tor allows onion routers to provide a policy of allowed outgoing connections. This can be used to prohibit or just allow connections to special hosts or to limit the allowed protocols. Because Tor is a volunteer-based network these policies are required to give operators control over routed traffic.

Leaky-pipe topology: A final advantage of Tor over onion routing is the already explained leaky pipe topology which allows requests leaving the Tor network in the middle of circuits. This makes traffic analysis even harder for adversaries, because just observing the begin and end of a circuit is not enough anymore.

Many TCP streams can share one circuit: Tor can use one circuit for multiple TCP connections to improve efficiency and anonymity of the connections.

Usage of a standard proxy interface: Onion routing required a seperate application proxy for each application protocol. Tor uses the standard SOCKS proxy interface. Therefore it is possible to support most of TCP-based applications without modification.

2.4 Non-goals of Tor

The developers from Tor have explicitly deferred several possible goals, either because they are solved elsewhere, or because they are not yet solved. Tor doesn't support secure peer-to-peer connections and is not secure against end-to-end attacks. Tor doesn't provide protocol normalization or steganography.

3 Attacks against Tor

The original Tor paper includes a chapter, named *Attacks and Defenses*, which lists a variety of attacks, and discusses how well Tor withstands them. It describes passive attacks, active attacks, directory attacks and attacks against rendezvous points.

Passive attacks like exploiting end-to-end timing correlations are attacks, in which the attacker just passively observes traffic and tries to obtain information about a users connection by this.

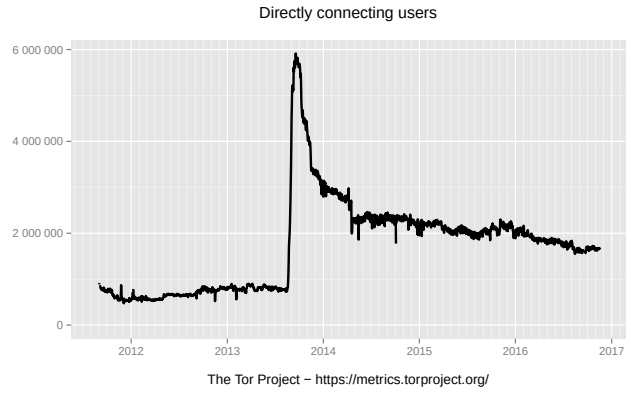

Figure 3: Estimated number of users connecting to Tor. Taken from [8].

In active attacks the attacker alters some information of messages or keys or runs his own onion proxies or routers - so he takes an active role in some way. An example for this type of attacks would be the attempt to compromise keys of an onion router.
Directory attacks try to trick the client to choose an attacker controlled path through the Tor network or decrease the availability of the network by attacking the directory servers, e.g. by subverting existing directory servers.
Finally, attacks against rendezvous points describes attacks against the confidentiality or availability of hidden services.

4 Importance and impact

A wide area network was briefly deployed after the invention of onion routing but it was not really used because it required a seperate "application proxy" for each supported application protocol and most of them were never written. Tor on the other hand uses the standard and near-ubiquitous SOCKS proxy interface, which supports most TCP-based programs without modification.

Because of that Tor is used in the world wide web since 2004 for anonymous connections. Individuals use Tor for censorship circumvention to reach othervise blocked content, to keep websites from tracking them or their family members, to publish services without needing to reveal the location of them or for sensitive communication. Tor is also used by companies or Journalists to communicate with activists or whistleblowers[9]. Currently there are about two million people using Tor as figure 3 states. The peak in August if 2013 was probably caused by a botnet using Tor for the communication between the bots and the command and control server[10]. This is an example for the downside of enabling anonymous connections. Tor can also be used by criminals to sell drugs, child porn, weapons or other illegal things.

Apart from the practical impact of the Tor network the published paper also has a big impact on computer science and was cited more than 3000 times by other papers dealing with improvements for or attacks against Tor, analysis of practical effects of the Tor network or using the developed approach to enable anonymous connections in other fields like VoIP systems.

References

[1] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *International Workshop on Information Hiding*, pp. 137–150, Springer, 1996.

[2] "Introduction to latency on computer networks." `https://www.lifewire.com/latency-on-computer-networks-818119`. Accessed: 2016-11-17.

[3] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[4] "Onion routing." https://www.onion-router.net/Summary.html. Accessed: 2016-11-20.

[5] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous connections and onion routing," *IEEE Journal on Selected areas in Communications*, vol. 16, no. 4, pp. 482–494, 1998.

[6] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, (Berkeley, CA, USA), pp. 21–21, USENIX Association, 2004.

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

[8] "Tor metrics." https://metrics.torproject.org. Accessed: 2016-11-20.

[9] "Tor: Overview." `https://www.torproject.org/about/overview.html.en`. Accessed: 2016-11-17.

[10] "How to handle millions of new tor clients." https://blog.torproject.org/blog/how-to-handle-millions-new-tor-clients. Accessed: 2016-11-20.