

Tor: The Second-Generation Onion Router

Tor is a circuit-based low-latency anonymous communication service. It adds perfect forward secrecy, congestion control, directory servers, integrity checking, configurable exit policies, and a practical design for location-hidden services via rendezvous points. Requires little synchronization or coordination between nodes, and provides a reasonable tradeoff between anonymity, usability, and efficiency.

Tor, a protocol for asynchronous, loosely federated onion routers that provides the following improvements over the old Onion Routing design:

- **Perfect forward secrecy:** Tor now uses an incremental or *telescoping* path-building design, where the initiator negotiates session keys with each successive hop in the circuit.
- **Separation of “protocol cleaning” from anonymity:** Tor uses the standard and near-ubiquitous proxy interface, allowing us to support most TCP-based programs without modification.
- **No mixing, padding, or traffic shaping (yet)**
- **Many TCP streams can share one circuit:** Tor multiplexes multiple TCP streams along each circuit to improve efficiency and anonymity.
- **Leaky-pipe circuit topology:** Through in-band signaling within the circuit, Tor initiators can direct traffic to nodes partway down the circuit.
- **Congestion control:** Tor’s decentralized congestion control uses end-to-end acks to maintain anonymity while allowing nodes at the edges of the network to detect congestion or flooding and send less data until the congestion subsides.
- **Directory servers:** Tor takes a simplified view toward distributing this information. Certain more trusted nodes act as *directory servers*: they provide signed directories describing known routers and their current state. Users periodically download them via HTTP.
- **Variable exit policies:** Tor provides a consistent mechanism for each node to advertise a policy describing the hosts and ports to which it will connect.
- **End-to-end integrity checking:** Tor hampers attacks by verifying data integrity before it leaves the network.

- **Rendezvous points and hidden services:** Tor provides an integrated mechanism for responder anonymity via location-protected servers.

Design goals and assumptions

Goals: Deployability, Usability, Flexibility, Simple design

Design

The Tor network is an overlay network; each onion router (OR) runs as a normal user-level process without any special privileges. Each onion router maintains a long-term identity key and a short-term onion key. The identity key is used to sign TLS certificates, to sign the OR's router descriptor (a summary of its keys, address, bandwidth, exit policy, and so on), and (by directory servers) to sign directories.

The onion key is used to decrypt requests from users to set up a circuit and negotiate ephemeral keys. Short-term keys are rotated periodically and independently, to limit the impact of key compromise.

Onion routers communicate with one another, and with users' OPs, via TLS connections with ephemeral keys. Traffic passes along these connections in fixed-size cells. Each cell is 512 bytes, and consists of a header and a payload.

Cells are either *control cells*, which are always interpreted by the node that receives them, or *relay cells*, which carry end-to-end stream data. The control cell commands are: *padding*, *create* or *created*, and *destroy*, or Relay cells have an additional header (the relay header) at the front of the payload, containing a streamID (stream identifier: many streams can be multiplexed over a circuit); an end-to-end checksum for integrity checking; the length of the relay payload; and a relay command.

In Tor, each circuit can be shared by many TCP streams. To avoid delays, users construct circuits preemptively. To limit linkability among their streams, users' OPs build a new circuit periodically if the previous ones have been used, and expire old used circuits that no longer have any open streams. Because circuits are built in the background, OPs can recover from failed circuit creation without harming user experience.

Constructing a circuit

To begin creating a new circuit, the OP (call her Alice) sends a create cell to the first node in her chosen path (call him Bob). (She chooses a new circID CAB not currently used on the connection from her to Bob). Once the circuit has been established, Alice and Bob can send one another relay cells encrypted with the negotiated key. Alice sends a relay extend cell to Bob, specifying the address of the next OR (call her Carol). Bob copies the half-handshake into a create cell, and passes it to Carol to extend the circuit. When Carol responds with a created cell, Bob wraps the payload into a relay extended cell and passes it back to Alice. Now the circuit is extended to Carol, and Alice and Carol share a common key. To extend the circuit to a third node or beyond, Alice proceeds as above, always telling the last node in the circuit to extend one hop further.

This circuit-level handshake protocol achieves unilateral entity authentication. We use PK encryption in the first step because a single cell is too small to hold both a public key and a signature.

Closing a Tor stream is analogous to closing a TCP stream: it uses a two-step handshake for normal operation, or a one step handshake for errors. If the stream closes abnormally, the adjacent node simply sends a relay teardown cell. If the stream closes normally, the node sends a relay end cell down the circuit, and the other side responds with its own relay end cell.

Congestion control

If enough users choose the same OR-to-OR connection for their circuits, that connection can become saturated.

- Circuit-level throttling: To control a circuit's bandwidth usage, each OR keeps track of two windows
- Stream-level throttling: The stream-level congestion control mechanism is similar to the circuit-level mechanism

Rendezvous Points and hidden services

Rendezvous points are a building block for location-hidden services (also known as responder anonymity) in the Tor network. This type of anonymity protects against distributed DoS attacks.

Goals

Access-control: Bob needs a way to filter incoming requests, so an attacker cannot flood Bob simply by making many connections to him.

Robustness: Bob should be able to maintain a long-term pseudonymous identity even in the presence of router failure.

Smear-resistance: A social attacker should not be able to "frame" a rendezvous router by offering an illegal or disreputable location-hidden service and making observers believe the router created that service.

Application transparency: Although we require users to run special software to access location-hidden servers, we must not require them to modify their applications.

Open Questions in Low-latency Anonymity

- How often should users rotate to fresh circuits?
How should we choose path lengths?
Does the hydra topology (many input nodes, few output nodes) work better against some adversaries?
- Are we going to get a hydra anyway because most nodes will be middleman nodes?
- Will users abandon the system because of this brittleness?
- If affected users rebuild circuits immediately, how much anonymity is lost?
-

Closure

Tor works on the real-world Internet, requires no special privilege or kernel modifications, requires little synchronization or coordination between nodes, and provides a reasonable tradeoff between anonymity, usability, and efficiency.