ChatGPT

Tor Browser: Architecture and Anonymity Mechanisms

Tor (The Onion Router) anonymizes Internet traffic by routing it through a three-hop overlay network with layered encryption (onion routing). A Tor client (embedded in the Tor Browser bundle) first downloads the network **consensus** from Tor's directory authorities, which lists all active relays. It then builds a multi-hop **circuit** (usually three relays: an *entry/guard* node, a *middle* node, and an *exit* node). In *telescoping path-building*, the client negotiates a fresh symmetric key with each hop in sequence [1]. Application data is then encapsulated in fixed-size encrypted cells and sent through the circuit: each relay peels off one layer of encryption with its key and forwards the data [2]. Thus no single node knows both the origin and destination – only its predecessor and successor [2]. By encrypting in layers and deleting session keys after use, Tor achieves **perfect forward secrecy**: compromising a node later cannot retroactively decrypt past traffic [1].

- **Circuit construction and use:** A new circuit is built incrementally. The first hop (entry/guard) is selected from a small set of stable guards, the second is a randomly chosen middle relay, and the third is an exit relay that connects to the final destination. The guard node is kept fixed (typically for months) to reduce the chance that a malicious relay will be chosen as entry [3] [4]. (All Tor Browser streams to one first-party site reuse the same circuit; connections to different first-party domains use distinct circuits [5].) The exit relay peels the final layer and makes the actual request on behalf of the user; only the exit sees the destination, and only the entry sees the user's IP.

- **Encryption (onion layers):** Traffic is split into constant-size Tor "cells" (padding any shorter data) [2]. Each cell is encrypted in layers: the client encrypts the payload first with the exit's key, then with the middle's, then with the guard's. At each hop one layer is removed. This makes the "onion" – layers of encryption – visible only to the client. Nodes cannot decrypt beyond their layer or link input data back to the user. This layered design (from the Tor specification) ensures that relays only see neighboring IPs and an indecipherable interior, providing anonymity in transit [2] [4].

- **Stream multiplexing:** Multiple TCP streams (e.g. tabs) can share a circuit to reduce overhead [6]. Tor Browser reuses the same circuit for a given first-party site for efficiency, but rotates circuits between different sites to avoid linking them [5].

- **Directory and consensus:** The Tor client regularly fetches an authenticated consensus document (via HTTPS from directory authorities) to know the set of relays, their public keys, and capabilities. Circuits are built using this up-to-date view. Each relay advertises exit policies (which ports it allows) so that the client picks an exit node permitted to reach the desired target.

Bundled Privacy Extensions and Hardening

The Tor Browser is a heavily patched **Firefox ESR** tailored for anonymity. It includes several built-in tools and settings:

- **HTTPS Everywhere:** By default Tor Browser bundles the EFF's HTTPS Everywhere extension [7]. This forces encrypted HTTPS connections wherever possible, reducing exit-relay eavesdropping. (Modern Tor Browser also enables Firefox's built-in HTTPS-Only mode by default.)

- **NoScript (JavaScript control):** Tor Browser includes NoScript (as a WebExtension) for controlling scripts [8]. The "Security Slider" interface in Tor Browser is implemented using NoScript to disable JavaScript, Java, Flash, and other active content at higher security levels. Users can manually raise the security level or disable scripts on untrusted sites. Disabling JavaScript helps prevent drive-by exploits and many forms of active fingerprinting.

- **Pluggable Transports:** To evade censorship, Tor Browser ships with transport plugins (obfs4, meek, FTE, etc.) that obfuscate Tor traffic [9]. These are used when bridges are configured, making Tor traffic look like random data or allowed protocols.

- **Tor Launcher and Torbutton:** On startup, Tor Launcher configures and launches the Tor background process and displays a connection status. Torbutton (now largely integrated) provides the interface for "New Identity" and "New Circuit" and for showing the current Tor circuit. Torbutton also manages disabling unsafe plugins by default [10].

State Isolation, Cookies, and Fingerprinting Protections

Tor Browser implements strong defenses against stateful tracking and fingerprinting:

- **Proxy obedience and first-party isolation:** All browser traffic is forced through the Tor proxy (SOCKS), with DNS resolution via Tor (see below) [11]. Tor's patched Firefox sets `network.proxy.socks_remote_dns=true` and disables DNS prefetch [11] so that all DNS lookups go over Tor. No traffic (HTTP, FTP, DNS, OCSP, etc.) bypasses the proxy [12]. In the Firefox configuration, *Proxy Obedience* is mandatory [13]. Similarly, *State Separation* is enforced: the browser "MUST NOT provide the content window with any state from any other browsing mode" [14]. In practice, this is implemented as first-party isolation: cookies, localStorage and other data are kept separate per first-party domain, so that third-party trackers cannot link you across sites.

- **Session and disk avoidance:** By default Tor Browser runs in a permanent "private browsing" mode: it does not save history, and it avoids writing caches, cookies or storage to disk between sessions [15]. (All such data is kept in-memory and cleared on exit.) This follows Tor's *disk avoidance* design requirement [15]. The "New Identity" action explicitly clears all session data: it stops all page activity, clears HTTP auth, SSL state, HSTS, cookies, caches, DOM storage, etc., closes tabs, and then instructs Tor to build new circuits [16] [17]. This prevents linkability over time.

- **Uniform settings (fingerprint minimization):** Tor Browser spoofs or normalizes many browser attributes to minimize fingerprint diversity [18] [19]. For instance, all Tor users share the same *User-Agent string* (e.g. "Windows NT 6.1; Firefox/60.0" on desktop) irrespective of true OS [20]. Other headers (Accept-Language, platform) are also fixed to a small set of values. Screen

dimensions and DPI are made uniform by “letterboxing” windows into 200×100px buckets [21]. Fonts are limited and fallback fonts standardized. Canvas and WebGL APIs are disabled or return blank values by default, preventing canvas fingerprinting [22]. As a result, every Tor Browser instance presents essentially the same fingerprint. For example, in Tor Browser we see the same spoofed Windows UA, a UTC+0 timezone, a 1000×900 window size, and “WebGL: Not supported” for all users [18] [20] (see figure below).

Attribute	Value
User agent	Mozilla/5.0 (Windows NT 6.1; rv:60.0) Gecko/20100101 Firefox/60.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Content encoding	gzip, deflate
Content language	en-US,en;q=0.5
List of plugins	
Platform	Linux x86_64
Cookies enabled	yes
Do Not Track	NC
Timezone	0
Screen resolution	1000x900x24
Use of local storage	yes
Use of session storage	yes
Canvas	
WebGL Vendor	Not supported
WebGL Renderer	Not supported

Tor Browser standardizes its fingerprint across all users. This example fingerprint (user-agent, time zone = 0, fixed resolution, disabled WebGL) is identical for every Tor Browser instance [20] [23].

- **JavaScript-hardening:** Beyond blocking scripts, Tor Browser also patches or configures Firefox to reduce fingerprint leaks. WebRTC is disabled (both at compile time and via `media.peerconnection.enabled=false`) to prevent IP leaks [24]. Plugins (Flash, Silverlight, etc.) are disabled by default and require click-to-play [10]. Telemetry, experiments and SafeBrowsing URLs are disabled to avoid unwanted network calls. The `Referer` header is stripped on onion-to-clearnet transitions to avoid leaking source domains.

Privacy Mechanisms and Security Details

- **DNS Resolution:** Tor does not perform DNS lookups locally. When the browser requests a hostname, the Tor client sends the hostname inside the encrypted circuit to the exit node. The exit node performs the DNS resolution and returns the IP within the Tor protocol [25]. Tor Browser’s configuration (`socks_remote_dns`, `dns.disablePrefetch`) ensures no DNS leaks. Thus all DNS queries travel through the Tor network and are seen only by the exit relay [25].

- **Traffic-correlation resistance:** Tor’s low-latency design does not hide timing or volume. If an adversary can observe both the traffic entering the Tor network and the traffic exiting it, they can correlate flows to deanonymize users [26]. To mitigate this, Tor uses **entry guards**: each client chooses a few guard relays and consistently uses them as the first hop [4]. By reusing guards for extended periods (months [3]), Tor minimizes the chance that a malicious node will ever occupy the entry position on a user’s path. Although Tor cannot stop end-to-end timing attacks in principle, this guard strategy greatly reduces the probability that an adversary simultaneously

controls both entry and exit for the same user [4]. (Tor also randomizes circuit paths periodically and limits concurrency to increase anonymity.)

- **Update mechanism:** Tor Browser uses Firefox's built-in updater to fetch new releases, but in a privacy-preserving way. All updates are downloaded over Tor and are cryptographically signed by the Tor Project. Tor Browser enforces certificate pinning (`security.cert_pinning.enforcement_level=2`) for the update server [27]. The individual update files (.mar files) are signed with Tor's GPG keys (rotated annually) [27]. The updater is patched to **not leak OS or kernel versions** during update checks [28]. It also uses Tor's SOCKS credentials to force each update fetch onto a new Tor circuit, preventing a malicious exit from suppressing updates [29]. If an automatic check indicates the browser is outdated, Torbutton displays a prompt (without revealing user info) and points to the official download page. Users can also manually verify the Tor Browser package signatures with the published GPG keys.

Tor's design balances practical performance with strong anonymity guarantees: multi-hop onion encryption prevents any single node from linking source to destination, while Tor Browser's hardened configuration closes known protocol and browser leaks. Combined, these systems provide layered privacy and make all Tor users appear nearly identical to web servers and network observers [4] [18].

Sources: Tor Project documentation and research on Tor's network protocols and Tor Browser design [2] [4] [25] [20] [30] [11] [27], etc.

---

[1] [2] [6] svn.torproject.org
https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf

[3] [5] [16] Managing identities - Features - Tor Browser — Tor
https://support.torproject.org/tor-browser/features/managing-identities/

[4] What are Entry Guards? - How Tor works - About Tor — Tor
https://support.torproject.org/about-tor/how-tor-works/entry-guards/

[7] [8] [9] [10] [11] [12] [13] [14] [15] [17] [24] [27] [28] [29] [30] The Design and Implementation of the Tor Browser [DRAFT]
https://2019.www.torproject.org/projects/torbrowser/design/

[18] [22] [23] Browser Fingerprinting: An Introduction and the Challenges Ahead | The Tor Project
https://blog.torproject.org/browser-fingerprinting-introduction-and-challenges-ahead/

[19] [20] [21] Fingerprinting protections - Features - Tor Browser — Tor
https://support.torproject.org/tor-browser/features/fingerprinting-protections/

[25] protocol - How does Tor route DNS requests? - Tor Stack Exchange
https://tor.stackexchange.com/questions/8/how-does-tor-route-dns-requests

[26] Traffic correlation using netflows | The Tor Project
https://blog.torproject.org/traffic-correlation-using-netflows/