Anonymity and Onion Routing

First Assignment Report

Tiago Almeida, Yves Bonneau, Diogo Leandro

November 3, 2025

Contents

1 Introduction **2**

2 History of Tor **2**

3 Tor network design **3**
- 3.1 High-Level Overview . . . 3
- 3.2 Onion Proxy and Onion Router . . . 4
- 3.3 Relay and Control Cells . . . 4
- 3.4 Circuits . . . 5
- 3.5 Streams . . . 7
- 3.6 Directory Servers . . . 7
- 3.7 Hidden Services . . . 8

4 Improvements Tor Adds Over the Original Onion Routing Design **9**

5 Tor's Anonymity Assessment **10**

6 Plan for Assignment 2 **12**
- 6.1 Experiment 1: Client Anonymity to a Clearnet Server . . . 12
- 6.2 Experiment 2: Hidden Service Anonymity . . . 13
- 6.3 Expected Outcomes . . . 14

7 Conclusion **14**

8 Bibliography **14**

1 Introduction

This report, developed for the course *Network Security [CC4031]*, presents an analysis of the Tor network. The primary objective is to elucidate the functioning of Tor, a system designed to enable low-latency, anonymous Internet communication. We will examine its core mechanisms, including onion routing, circuit construction, relay operations, encryption layers, and traffic management. By examining how these components interact, this project aims to clarify how Tor protects user privacy, resists network surveillance, and enables secure, decentralized communication.

Additionally, this work provides the plan for testing the software, collecting performance statistics, and analyzing the data for the subsequent assignment.

2 History of Tor

Tor (The Onion Router) origins began in the mid-1990s as a U.S. Naval Research Laboratory project led by Paul Syverson, Michael G. Reed, and David M. Goldschlag. Their goal was to develop a way to protect online communications against traffic analysis, techniques that can reveal who is communicating with whom, even if the message contents are encrypted. This early research culminated in the 1996 paper *"Hiding Routing Information"* [1], which introduced the concept of onion routing: wrapping messages in multiple layers of encryption that are peeled away at each relay, concealing both source and destination.

In the early 2000s, the Tor Project was created to make this technology publicly available and to encourage widespread, civilian use, an essential factor for maintaining anonymity. Roger Dingledine and Nick Mathewson, working with Syverson, released the first public version of Tor in 2002, emphasizing open-source development and volunteer-run relay nodes.

Over time, Tor evolved from a U.S. government–funded research prototype into a global anonymity network used by journalists, activists, researchers, and ordinary citizens seeking privacy. The Tor Project, Inc., a nonprofit organization founded in 2006, now maintains and develops the software, supported by grants and donations.

Modern Tor has expanded beyond simple web browsing anonymity. It now supports onion services (websites accessible only within the Tor network, also known as hidden services), improved encryption protocols, performance optimizations, and defenses against traffic correlation attacks. Despite challenges such as censorship and malicious relays, Tor remains one of the most

influential and widely used tools for online privacy and freedom of expression.

3 Tor network design

During this section we will give a detailed breakdown of the core components and operational principles of the Tor network. The following subsections will explore the high-level architecture and the specific roles of its key elements, including the Onion Proxy and Onion Router, the structure of relay and control cells, the lifecycle of circuits and streams, the critical function of directory servers, and the mechanics of hidden services. It is important to note that the majority of the technical information presented in this section is derived from the foundational paper, *"Tor: The Second-Generation Onion Router"* [2].

3.1 High-Level Overview

Tor network enables anonymous communication by routing internet traffic through a series of volunteer-run servers called Onion Routers (ORs), also called relays or nodes. It involves two endpoints, the client (called the Onion Proxy or OP) and the destination server, with several onion routers forwarding encrypted messages between them.

To begin, the client contacts directory servers, trusted servers, each owned by a different independent entity, to obtain a list of available ORs and randomly selects a set of them to form a circuit. Typically, this circuit consists of 3 different relays, an entry relay, a middle relay, and an exit relay. It then establishes a separate encryption key with each node, known only by the client and the corresponding node.

When sending data, the client first wraps the message in multiple layers of encryption, one for each node. Each router only knows the node that sent it a packet and the node it should forward to next, it never sees the entire route. As the data travels through the circuit, each router decrypts only its layer, revealing the next destination, until the exit node removes the final layer and sends the intended TCP message to the target. The response follows the same path back, with layers of encryption added at each step, which the client then removes in reverse order.

Because no single node knows both the source and the destination, Tor effectively separates identity from activity, ensuring strong anonymity and privacy for users.

3.2 Onion Proxy and Onion Router

We have already mentioned the names onion proxy and onion router, but what exactly are they and what is their function?

The Onion Proxy is the client-side Tor software that users run to connect to the network. It fetches information about available relays from the directory servers, selects a sequence of nodes to form a circuit, performs the necessary key exchanges, and manages all encrypted communication between the user's applications and the Tor network.

The Onion Router, also called a relay or node, is a volunteer-operated server responsible for forwarding Tor traffic. When participating in a circuit, an Onion Router establishes a TLS-encrypted connection to its immediate neighbor relays in that circuit. These connections are created on-demand for circuit construction and are not maintained permanently between all possible relay pairs, which would be infeasible at scale. The TLS layer provides a secure transport channel, protecting the traffic from external eavesdropping on the links between nodes. Each relay maintains a pair of long-term keys:

- **Identity Key:** used to sign the relay's descriptor and verify its long-term authenticity to directory authorities and clients.
- **Onion Key:** used during the initial circuit creation step to perform a Diffie-Hellman key exchange with the Onion Proxy, establishing a temporary, ephemeral session key.

The Onion Proxy likewise negotiates unique ephemeral keys with each Onion Router in the circuit. These session keys are then used to encrypt and decrypt data passing through the network, ensuring that each hop can only access its own layer of encryption.

3.3 Relay and Control Cells

In the Tor network, all data is transmitted in fixed-size packets called cells, each exactly 512 bytes long. This fixed size helps prevent traffic analysis by ensuring that all traffic looks identical in size, regardless of its content or purpose.

Each cell is divided into two parts: a header and a payload. The header contains control information used by Tor routers, while the payload holds the actual data or instructions to be processed.

The header includes two main fields. The Circuit ID identifies the circuit to which the cell belongs, allowing routers to handle multiple circuits

simultaneously. The Command field specifies how the payload should be interpreted and processed. Based on this command, a cell is classified as either a Control Cell or a Relay Cell.

Control Cells are used to manage and maintain circuits between nodes (also known as onion routers). They are not forwarded through the entire circuit but processed only by the node that receives them.

Relay Cells, in contrast, are used for communication between the client and the destination through the established circuit. Each relay cell contains an additional relay header within its payload. This relay header includes a Stream ID, which identifies which stream or TCP connection the data belongs to; a Relay command, which specifies the purpose of the cell; and a checksum that ensures data integrity.

Commands sent by the Control Cell are:

- **Create / Created:** used to establish a new circuit and perform key exchange between the client and a Tor node.
- **Destroy:** used to close an existing circuit.
- **Padding:** used to send empty cells that help obscure traffic patterns.

Commands sent by the Relay Cell are:

- **Data:** carries user data through the circuit.
- **Begin / Connected:** start and confirm a TCP connection through the exit node.
- **End:** close a stream (end the TCP connection).
- **Extend / Extended:** add a new node to the circuit and confirm it.
- **Truncate / Truncated:** shorten or tear down part of a circuit

3.4 Circuits

In the Tor network, communication between a client and a destination occurs through virtual paths called circuits. A Tor circuit is a sequence of encrypted connections established through a small number of randomly selected Tor relays. Each relay in the circuit knows only its immediate predecessor and successor, ensuring that no single relay can identify both the source (client) and the destination (server). This layered encryption process, known as onion routing, forms the foundation of Tor's anonymity.

When a Tor client (the Onion Proxy, or OP) needs to send traffic through the network, it first retrieves the list of available relays and their public keys from the directory authorities. Using this information, the client builds a circuit typically consisting of three relays:

1. **Entry (Guard) node:** the first relay in the circuit, which knows the client's IP address but not the destination.

2. **Middle relay:** forwards traffic within the Tor network, linking the entry and exit nodes without knowing either endpoint.

3. **Exit node:** the last relay, which decrypts the final encryption layer and connects to the destination server on the open Internet.

Each circuit is constructed incrementally through a series of cryptographic handshakes. The client begins by establishing a secure session with the entry node using a Diffie–Hellman key exchange, creating a shared symmetric key. Next, it extends the circuit to the middle relay by asking the entry node to forward an encrypted "extend" request. The middle node performs a new key exchange with the client, resulting in another shared key. This process repeats for the exit node. In the end, the client possesses a unique symmetric key for each relay, while each relay only shares its own key with the client.

When the client sends data, it encrypts the message in multiple layers, once for each relay, starting from the exit node's key and ending with the entry node's key. As the message passes through the circuit, each relay "peels off" one layer of encryption, revealing only where to send the next packet. The exit node then forwards the plaintext data to the destination. The return traffic follows the same circuit in reverse, with each relay successively adding its encryption layer until it reaches the client, which removes all layers to recover the original response.

Circuits are short-lived for security reasons, typically lasting about ten minutes before being replaced. Multiple TCP streams can share the same circuit, reducing latency and overhead while maintaining anonymity. If a stream requires a different destination or anonymity context, the client can build a new circuit.

Through this design, Tor achieves a balance between anonymity, efficiency, and performance, ensuring that while messages traverse a distributed network of relays, no single node can fully reconstruct the path or link sender to recipient.

3.5 Streams

A stream in Tor represents a data connection between the client and a destination that is guided by an established circuit. The circuit provides the encrypted pathway across the Tor network, while the stream carries the actual communication and data flow between both endpoints.

Tor supports multiplexing of streams, allowing multiple independent connections to be transmitted simultaneously through a single circuit. Each stream is assigned an unique Stream ID, which enables Tor to distinguish and correctly route the data belonging to each connection, ensuring efficient and organized communication over shared circuits.

3.6 Directory Servers

Directory servers are specialized Onion Routers that maintain a consistent, signed view of the Tor network. These servers, known as directory authorities, are run by independent and geographically distributed entities. This decentralization is crucial for security, as it makes it difficult for a single actor to control a majority of the authorities and thus manipulate the overall view of the network. Each active relay periodically uploads a signed descriptor containing its public keys, IP address, and exit policy. The servers aggregate these descriptors to create the network consensus, which lists all recognized relays and their assigned flags (Guard, Exit, HSDir).

Every Tor client includes the public keys of all directory servers. When Tor starts, it downloads the latest signed consensus via HTTPS and verifies the signatures. This prevents forged network views or fake relay announcements during the bootstrap process.

The servers operate a simple multi-round consensus protocol. Each collects relay descriptors, shares its local view with the others, and computes a majority decision about valid relays. Only after a quorum of signatures is attached is the consensus accepted as authoritative. This distributed signing model ensures that no single compromised directory can alter the network map.

By replacing the old flooding system with this coordinated set of trusted servers, Tor gains scalability, consistency, and verifiable trust. The directory system filters out unapproved or malicious relays and provides clients with a uniform, cryptographically authenticated snapshot of the network.

3.7 Hidden Services

Hidden services, now officially known as onion services, are a fundamental feature of the Tor network that enable users to host and access services anonymously. Unlike ordinary Internet servers, a hidden service does not reveal its IP address or physical location. Instead, both the client and the server communicate exclusively through the Tor network, ensuring that neither party learns the other's network identity.

When a hidden service (the server) starts, it generates a pair of long-term cryptographic keys that define its permanent identity. The public portion of this key is hashed to produce the service's unique .onion address (for example, exampleonionaddress.onion, where the "exampleonionaddress" part would be the hash of the public key created). This address acts as a self-authenticating identifier: if the public key inside the service descriptor matches the address, the client can be confident it is connecting to the genuine service.

To accept incoming connections without exposing its location, the hidden service establishes encrypted circuits to several Tor relays called introduction points. It then creates a signed document known as a service descriptor, which lists these introduction points along with temporary encryption keys. The descriptor is uploaded to a small set of special Tor relays called Hidden Service Directories (HSDirs). Each HSDir stores only a tiny portion of all descriptors, selected by a hash of the service's public key and a time period. This distributed, time-varying storage design prevents enumeration of all onion services and provides redundancy if some nodes go offline.

When a client (such as Alice) wants to contact a hidden service (Bob), she first obtains Bob's .onion address through some external means. Using that address, her Tor software computes where in the network Bob's descriptor is stored and downloads it from the appropriate HSDirs. The descriptor tells her which introduction points to use. Alice then selects a random Tor relay to serve as a rendezvous point and sends a message, via one of Bob's introduction points, inviting Bob to connect there. Bob builds a Tor circuit to the rendezvous point, and once both sides connect, encrypted communication flows between them, without either knowing the other's IP address.

Hidden services use both long-term identity keys (which define the persistent .onion address) and short-term ephemeral keys (which change periodically to protect past sessions if a server is compromised). This dual-key structure provides forward secrecy and limits the effects of key exposure.

In practice, onion services can host any TCP-based application, from websites and chat servers to software update channels. Beyond anonymity for publishers, they also offer integrity and censorship resistance, making them a crucial component of Tor's goal to enable private, secure, and uncensorable

communication on the Internet.

4 Improvements Tor Adds Over the Original Onion Routing Design

Early versions of Onion Routing showed that people wanted private communication online. Even though the prototype only ran on one machine and often broke, more than sixty thousand users connected to it. That success also exposed a lot of problems: the old design didn't have forward secrecy, circuits were unreliable, every new request needed expensive key exchanges, and node operators had no way to control what kind of traffic went through their servers. Tor was created to fix those issues and make anonymous networking actually usable across the Internet.

In Tor, circuits are built using a method called telescoping. Instead of creating one big, pre-encrypted "onion" all at once, the client, as explained in earlier sections, builds the path one hop at a time, making use of ephemeral keys with each OR. Once a key is used, it's deleted, so even if someone later hacks a relay, they can't decrypt old traffic. This approach also helps with reliability, if a connection to one node fails, the client knows exactly which one caused it and can try another without rebuilding the whole circuit.

Tor also makes it easier for different programs to use the network. The old system needed a separate proxy for every application type, like one for web traffic and another for email. Tor instead uses the standard SOCKS interface, which most software already supported at the time. That means any TCP-based application, like a browser or SSH client, can send its data through Tor without changes. Tor itself doesn't try to clean up application data, like removing cookies or tracking headers, that job is left to outside tools such as Privoxy (still actively maintained in 2025).

The designers were pretty practical in regards to bandwidth-heavy protections like padding or mixing. Earlier ideas suggested adding fake traffic or delaying messages to hide timing patterns, but those methods wasted a lot of bandwidth and didn't stop advanced timing attacks anyway. So Tor had left those features out at the time of release, focusing on efficiency and proven protections. In 2025, circuit-level padding is now a thing. From the tor specification "At present, Tor uses this system [circuit padding system] to deploy two pairs of circuit padding machines, to obscure differences between the setup phase of client-side onion service circuits, up to the first 10 relay cells."

Tor's circuit structure is also more flexible. Data doesn't have to exit at

the final relay, it can leave from any hop in the path. This topology is called "leaky-pipe". This design makes it harder for observers to figure out which node is actually contacting the destination. To reduce overhead, it also lets many connections share one circuit, so multiple browser tabs or sessions can reuse the same path instead of creating new ones each time.

Earlier systems ignored bottlenecks. Standard load-balancing methods require global coordination, which breaks anonymity. Tor uses end-to-end acknowledgments (RELAY SENDME cells) so nodes slow down automatically when congestion appears. This is decentralized and privacy-preserving.

A small set of directory servers maintains a signed, global list of active relays. These directories replace the earlier flooding system and ensure all clients share a consistent, verified view of the network (see Directory Servers section). Each relay can also publish its own exit policy, listing which ports or sites it will allow connections to. That flexibility helps volunteers contribute without worrying about being blamed for unwanted traffic.

Tor improves security in other ways too. It checks the integrity of every data cell as it travels through the network, preventing any relay from secretly changing or tagging traffic. And it adds built-in support for hidden services: instead of the "reply onions" from older versions, Tor uses rendezvous points where clients and servers can meet through a third relay without ever revealing their IP addresses.

Finally, Tor is designed to be practical and portable. It runs entirely in user space and doesn't need any operating system modifications. That limits it to TCP traffic, but it also makes it easy to install and run on almost any platform.

Altogether, these improvements: forward-secure circuits, easy app integration, efficient traffic handling, flexible exits, decentralized control, directory servers, integrity checks, and hidden services turn Onion Routing from a fragile experiment into a real, scalable anonymity network that anyone can use.

5 Tor's Anonymity Assessment

Tor's anonymity relies on how its internal mechanisms behave under realistic attack conditions. Its design assumes that parts of the network may be watched or even controlled, but not the entire Internet at once. Within those limits, Tor's structure, its layered encryption, ephemeral keys, and decentralized directory system, ensures that identity and destination are never visible together on the same link. Each relay decrypts only one layer of the packet, learns where to send it next, and sees nothing else. Even if one relay is hostile

or compromised, it cannot see the whole circuit. This compartmentalization is the foundation of how Tor preserves anonymity in practice.

Forward secrecy reinforces that protection. The client establishes new, temporary Diffie-Hellman keys with every relay in the circuit, and deletes them after use. If an attacker later gains access to a relay's memory or keys, previously transmitted data cannot be recovered or linked to a user. This limits the damage of any breach and prevents long-term compromise of anonymity. Similarly, integrity checks built into Tor's cell structure stop any node from modifying traffic in transit. A relay cannot "tag" or replay packets to try to identify where they go downstream, because every cell is authenticated end-to-end. This protects against a broad class of active correlation and replay attacks that earlier Onion Routing systems could not prevent.

Tor's circuit management also contributes directly to anonymity. Because circuits are built incrementally and can be replaced or extended when a relay fails, the user controls which nodes participate at any time. If a node becomes unreliable or suspicious, the circuit can be rebuilt without revealing the user's identity or destination. The system's congestion control maintains steady traffic while hiding congestion signals that might otherwise leak timing information. By keeping data moving smoothly but independently across each link, Tor reduces opportunities for simple timing correlation between different parts of the circuit.

The network's directory infrastructure also plays an important role. The directory servers' signed consensus ensures clients choose paths based on verified relay information, reducing manipulation risks.

Anonymity also depends on diversity. Tor's random path selection spreads trust across many jurisdictions and operators, making it statistically unlikely that the same adversary controls both the entry and exit of a circuit. Even if some relays are malicious, they can only observe their immediate neighbors. The only way to definitively deanonymize a user is to control both ends of the same circuit and perform precise traffic correlation, which is a difficult and expensive attack given the network's size and turnover.

Yet Tor's anonymity has limits that stem from its low-latency nature. It cannot stop a global observer who can monitor all network edges simultaneously. Because Tor must preserve interactive performance, it cannot add enough padding or delay to fully disguise timing and size patterns. A sufficiently well-positioned adversary can still match traffic entering and leaving the network by analyzing these patterns. Likewise, exit relays remain a point of exposure for unencrypted traffic. If users connect to sites without HTTPS or other end-to-end encryption, hostile exits can observe or alter their data, even though they still cannot identify the user directly.

The same trade-off applies to hidden services. Rendezvous circuits conceal server locations by using third-party relays, but timing analysis can still link their traffic to an external observation if the adversary controls enough vantage points. Despite this, hidden services maintain strong practical anonymity because neither side ever learns the other's IP address, and the use of independent circuits for introduction and rendezvous limits correlation paths.

Tor's defenses against denial-of-service and directory manipulation also support anonymity indirectly. By throttling new circuit creation and validating relay reliability through signed directories, Tor prevents attackers from flooding the network or filling it with fake relays to bias path selection. Signed software releases and open-source transparency reduce the risk of distributing modified clients that could leak identifying data.

Altogether, these mechanisms demonstrate how Tor's architecture enforces anonymity through design rather than obscurity. Even under partial compromise, its encryption layers, key management, and distributed trust model keep identity separate from activity. Only a global, coordinated adversary or careless application behavior, such as unencrypted traffic through an exit, can reliably break this separation. Tor therefore achieves a realistic form of anonymity: not absolute invisibility, but practical unlinkability and protection against the vast majority of surveillance and tracing capabilities that ordinary users and servers face.

6 Plan for Assignment 2

For Assignment 2, we will move from a theoretical analysis to a practical investigation of Tor's anonymity guarantees. Instead of a broad comparison, we will conduct two targeted experiments designed to probe the specific mechanisms that protect client and server anonymity. The setup for both experiments will involve virtual machines or Docker containers acting as clients, servers, and observers, with network traffic captured and analyzed using tools like Wireshark or `tcpdump`.

6.1 Experiment 1: Client Anonymity to a Clearnet Server

This experiment evaluates Tor's effectiveness in concealing a client's identity and activity from network observers when accessing a standard web server.

- **Setup:** A client machine and a clearnet web server will be deployed on separate networks. Two observer instances will be placed: **Observer**

A on the client's local network, and **Observer B** on the server's local network.

- **Procedure:** The client will access the web server through the Tor network.
- **Analysis:** We will analyze the captured traffic to determine:
  - What **Observer A** can see: Can it detect the destination server's IP? Is the traffic to the Tor network identifiable?
  - What **Observer B** can see: Can it identify the client's real IP address, or only the IP of a Tor exit relay?

6.2 Experiment 2: Hidden Service Anonymity

This experiment investigates the stronger anonymity properties of Tor onion services, focusing on the protection of both the client's and the server's locations.

- **Setup:** We will configure a hidden service (server) and a dedicated Tor relay to act as a rendezvous point. Three observer instances will be deployed: **Observer C** on the client's network, **Observer R** on the rendezvous point's network, and **Observer S** on the hidden server's network.
- **Procedure:** The client will connect to the hidden service using its `.onion` address.
- **Analysis:** We will analyze the traffic from all three vantage points:
  - **Observer C** verifies that the client only communicates with the Tor entry network.
  - **Observer R** is critical for testing the rendezvous protocol: Can it correlate the incoming and outgoing circuits to link the client and the server?
  - **Observer S** confirms that the server receives connections only from the rendezvous point, not the client's real IP.

6.3 Expected Outcomes

By comparing the data from these observers, we aim to provide empirical evidence for how Tor's layered encryption and circuit-based routing concretely implement anonymity. We will assess the visibility of IP addresses, the effectiveness of traffic analysis resistance, and the performance characteristics inherent in each scenario.

7 Conclusion

This report explored how the Tor network operates and the mechanisms it uses to protect user anonymity. We examined its layered encryption, circuit-based routing, and the roles of Onion Routers, Onion Proxies, and directory servers in maintaining secure and private communication. We also analyzed the structure of Tor cells, the functioning of hidden services, the concept of streams, and the improvements made over earlier versions of the network, concluding with an assessment of Tor's effectiveness in preserving anonymity.

Based on this understanding, the plan for Assignment 2 will put these concepts into practice by simulating clients, servers, and observers in virtual environments. By testing different communication scenarios, we aim to analyze how Tor affects network visibility, latency, and overall anonymity.

8 Bibliography

References

[1] Paul Syverson David M. Goldschlag, Michael G. Reed. Hiding routing information. *Information Hiding Workshop*, 1996.

[2] Paul Syverson Roger Dingledine, Nick Mathewson. Tor: The second-generation onion router. *USENIX Security Symposium*, 2004.