PRIVACY GUIDE

Tor in depth

Complete guide

This guide explains how Tor works, the network that lets you use the internet without revealing who you are or where you connect from: circuits, layered encryption, the directory, onion services, bridges, the browser and the real limits of the protection.

From the onion network to traffic analysis

Contents

INTRODUCTION AND BASICS

- How to use this guide 3
- The basic concepts 3

CHAPTERS

01 What Tor is and what problem it solves 5

02 Threat model: what Tor protects and what it does not 5

03 Three-hop circuits 6

04 Cells and layered encryption 8

05 The ntor handshake 9

06 Directory and consensus 11

07 Onion services 12

08 Bridges and pluggable transports 14

09 Tor Browser 15

10 Attacks and limits 16

11 Tor, VPNs and combinations 18

12 Tails and Whonix 19

13 Good practice 19

14 How the protocol is evolving 20

15 Community, funding and legality 21

16 Sources 21

INTRODUCTION

How to use this guide

Audience and prerequisites. No knowledge of networking or cryptography is needed. The "Before you start" section explains the basic concepts with everyday examples, and the guide gradually rises in level until it reaches the details of the protocol.

State of the network. The facts describe Tor as of mid-2026: the classic implementation in C ("C Tor"), the Rust rewrite called Arti, the new Counter Galois Onion encryption and the defences against denial of service on onion services.

How to read it. If you are starting from zero, begin with "Before you start". Chapters 1 to 3 give the overview and can be read on their own. Chapters 4 and 5 contain the two key formulas of the protocol, explained step by step with small examples. Chapters 6 to 9 describe the pieces of the network and the browser. Chapters 10 to 13 deal with limits, combinations with other tools and good practice. Chapter 14 describes how the protocol is evolving and chapter 15 who sustains it.

BEFORE YOU START

The basic concepts

To understand Tor, four ideas are enough: what an IP address is, who sees what when you browse, what an intermediary (proxy) does and what it means to encrypt in layers.

The IP address

Every device connected to the internet uses an **IP address**, a number similar to a postal address. When you open a website, your data packet carries two addresses: the destination's and yours, so that the reply knows where to return. That address usually reveals your country, your approximate city and your internet provider.

Who sees what

With no protection at all, each part of the path sees something different:

WHO	WHAT THEY SEE
Your internet provider	Which servers you connect to and when, even if the website uses HTTPS
The website you visit	Your IP address and what you do on its site
A public Wi-Fi network	Your unencrypted traffic and the destinations of everything else
Advertising companies	Your IP, your browser and a trail of the websites you visit

HTTPS encryption protects the *content* of what you send, but it does not hide who you talk to. That information about who talks to whom is called **metadata**, and it is exactly what Tor tries to hide.

Intermediaries

A **proxy** is an intermediary: you talk to it and it talks to the website on your behalf. The website sees the proxy's address, not yours. The problem is that the proxy does see you and what you ask for, so you have only changed whom you trust. A VPN works on the same idea, with the encrypted tunnel between you and the intermediary.

Tor avoids that single point of trust by spreading the work across several intermediaries, so that none of them alone knows both who you are and what you do.

Encrypting in layers

Imagine you want to send a letter without anyone knowing who sends it and who it is for. You put it in an envelope addressed to the recipient, that envelope in another addressed to a second intermediary, and that one in a third addressed to a first intermediary. Each intermediary opens only its own envelope and finds inside the address of the next one. None of them sees the complete letter or knows both the sender and the recipient.

The Tor network does exactly that with encryption: three layers, one per intermediary. That is why it is called *onion routing*.

CHAPTER 1

What Tor is and what problem it solves

Tor is a network of thousands of volunteer computers, called **relays**, that forward internet traffic in encrypted layers. Its name comes from *The Onion Router*. The idea of onion routing was born in the 1990s in a United States naval research laboratory, and the current Tor design was presented in an academic paper in 2004. Today it is maintained by a non-profit organisation, The Tor Project, with a community of volunteers who run the relays.

Tor solves a specific problem: that the website you visit does not know your IP address and that those who see your connection do not know which website you visit. It does not encrypt the content by itself (that is what HTTPS and end-to-end encryption are for), it does not make you invisible on a website where you log in with your name and it does not turn a compromised computer into a secure one.

PIECE	WHAT IT IS
Tor (the network)	The volunteer relays and the rules they coordinate with
Tor Browser	A Firefox-based browser that comes already configured to use the network
Onion services	Sites and services reachable only through Tor, whose address ends in .onion
Bridges	Relays not listed publicly, for people who have Tor blocked
Arti	The reimplementation of Tor in the Rust language, under active development

CHAPTER 2

Threat model: what Tor protects and what it does not

A **threat model** answers three questions: what you want to protect, from whom, and what that person or entity can do. Tor was designed with a specific adversary in mind.

What Tor hides. Your IP address from the websites and services you visit. The sites you visit from your internet provider and from anyone watching your local network. The link between both ends from any individual intermediary.

What Tor does not hide. That you use Tor, because your provider sees that you connect to known relays (bridges reduce this). What you do inside a website if you log in or fill in forms with data that identifies you. The content of your traffic if the website uses unencrypted HTTP: the exit relay sees it. What your own device does if it is infected.

The adversary Tor was not designed for. Someone able to observe both the entry and the exit of your circuit at the same time can compare traffic patterns and link both ends, even without being able to read anything. Tor's designers acknowledged this from the start: it is a limit of the idea, not a flaw that a patch can fix. Chapter 10 covers it in depth.

That is why Tor works very well against commercial tracking, local surveillance and censorship, and works only to a limited extent against adversaries with a view of a large part of the internet, whose attack also demands skill and resources.

CHAPTER 3

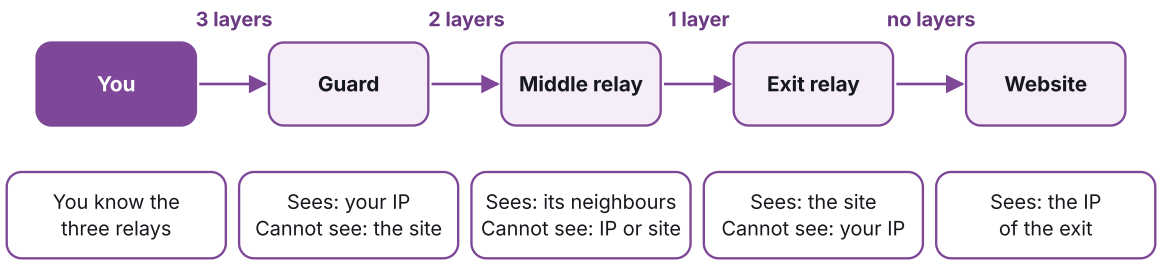
Three-hop circuits

When you open Tor Browser, your client picks three relays and builds with them a **circuit**: a three-hop path through which all your traffic passes. Each relay only knows the previous one and the next one.

Guards, middle and exit relays

ROLE	WHAT IT KNOWS	WHAT IT DOES NOT KNOW
Guard (entry)	Your IP address and that you use Tor	Which website you visit
Middle relay	Only the guard and the exit relay	Who you are and which website you visit
Exit relay	The destination website and, without HTTPS, the content	Who you are

The **guard** is not changed on every circuit: your client picks one and keeps it for weeks or months. If it picked a new one each time, sooner or later you would fall into the hands of a malicious relay with growing probability. With a fixed guard, either that guard is honest and your entry is protected, or it is not and the risk is concentrated in a single place. The **exit relay** is the one that sees the destination website; that is why exit operators receive the most legal complaints.



Each relay only knows the previous and the next one

Each relay only sees its neighbour; the layers are removed along the circuit.

How a circuit is built

The client builds the circuit like a telescope, hop by hop:

1. It connects to the guard over a TLS connection and negotiates a secret key with it (the handshake in chapter 5).
2. Through the guard, already encrypted, it asks the middle relay to join the circuit and negotiates another key with it. The guard only forwards bytes it cannot read.
3. Through the two previous ones, it repeats the process with the exit relay.

At the end, the client shares a different key with each relay and no relay knows the keys of the others. Tor reuses the same circuit for a few minutes and then opens another; Tor Browser also uses separate circuits for different sites, so that two websites cannot tell that you are the same person.

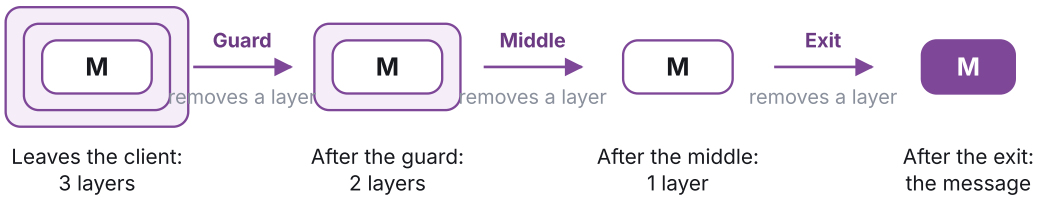
CHAPTER 4

Cells and layered encryption

Everything that travels through a circuit is packed into fixed-size **cells** of about 512 bytes, with about 498 bytes of useful data. Having all cells the same size stops an observer from telling a short request from a long one by the size of the packets.

The layers, one per relay

The client encrypts each cell three times, once per relay, like the layers of an onion. Each relay removes its layer and forwards the result. Only the last relay sees the original content, and only the first sees your IP address.



Only the relay that shares its key with the client can remove each layer

The message travels wrapped in three layers and each relay removes its own.

Tor's classic encryption (called *tor1*) uses AES in counter mode. This mode generates a pseudorandom sequence of bytes, the *keystream*, and mixes it with the data using the XOR operation. That choice is what makes the layers possible.

KEY FORMULA.

$$C = M \oplus S_{\text{guard}} \oplus S_{\text{middle}} \oplus S_{\text{exit}}$$

In words: the message leaving the client is the original message mixed, one after another, with three secret sequences; each relay undoes only its own, and what remains stays unreadable until the last one.

- `M` is the original message, the cell's data.

- `S` is each relay's keystream, derived from the key the client shared with it. Only the client and that relay know it.
- `⊕` is bitwise XOR: it gives 1 when the bits differ and 0 when they are equal. Applying it twice with the same sequence returns the original value.
- `C` is the encrypted cell the client sends to the guard.

Example with bits. Let `M = 1011`, the guard's `S = 0110`, the middle relay's `= 1100` and the exit's `= 0011`.

1. The client computes `1011 ⊕ 0110 ⊕ 1100 ⊕ 0011`, which gives `C = 0010`.
2. The guard applies its sequence: `0010 ⊕ 0110 = 0100`. It does not know what this is.
3. The middle relay applies its own: `0100 ⊕ 1100 = 1000`. Neither does it.
4. The exit relay applies its own: `1000 ⊕ 0011 = 1011`, the original message.

Why it works. XOR is commutative and each sequence cancels itself out, so the order in which the layers are removed does not matter. And since each relay only knows its own keystream, nobody can reconstruct the message before the exit.

This scheme has a known weak point. Because XOR authenticates nothing, a malicious relay can alter bits of a cell without it being noticed at that hop; if another of the attacker's relays, further along the circuit, sees the same pattern of alteration, both can conclude that they are part of the same circuit. This is the *tagging attack*. In addition, tor1 only adds 4 bytes of integrity checking per cell and uses the same key for the whole life of the circuit. Chapter 14 explains the replacement that fixes both problems.

CHAPTER 5

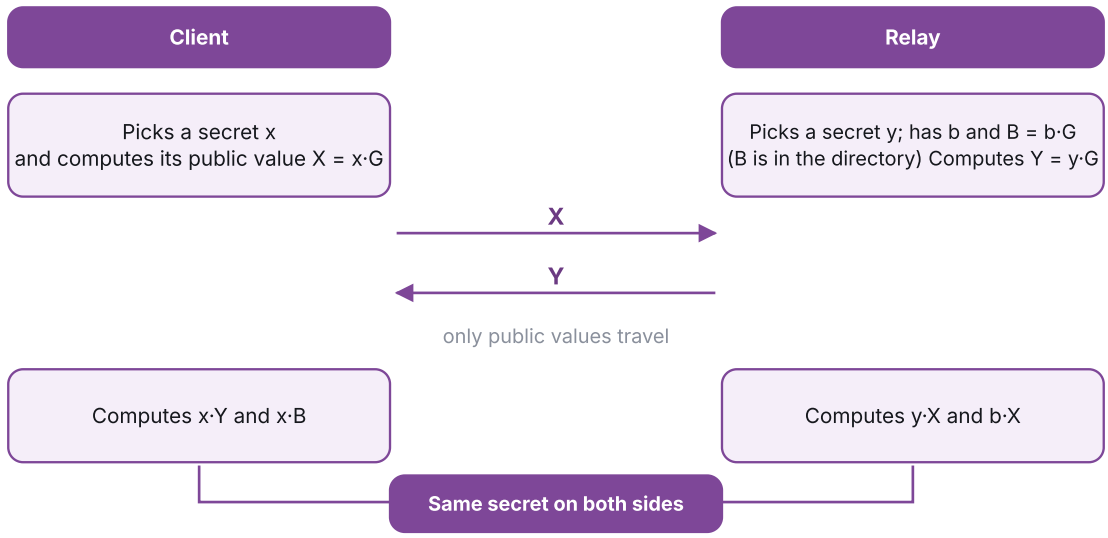
The ntor handshake

One question remains: how do the client and a relay agree on a secret key if they talk over a network that may be watched. Tor uses a protocol called **ntor**, based on Diffie-Hellman key exchange over the Curve25519 curve.

Diffie-Hellman with paints

The classic analogy is paints. You and the relay agree on a public colour, for example yellow. Each of you secretly mixes your own private colour with the yellow and sends the other the mixture. Then each adds their own private colour to the mixture received. Both end up with the

same final colour, and whoever only saw the mixtures cannot obtain it, because separating mixed colours is practically impossible.



Client and relay exchange only public values and obtain the same secret.

KEY FORMULA.

$$x \cdot Y = x \cdot (y \cdot G) = y \cdot (x \cdot G) = y \cdot X$$

In words: if you combine your secret number with the relay's public value, and the relay combines its secret number with your public value, both obtain the same result without ever having sent it.

- `G` is a fixed, public point on the curve, known to everyone.
- `x` is a secret number the client picks for this connection and discards afterwards.
- `X = x · G` is the client's public value. It can be sent: deducing `x` from `X` is infeasible.
- `y` and `Y = y · G` are the same for the relay.
- The `·` operation is the multiplication of a curve point by a number. It is easy in one direction and practically impossible to reverse.

Example with small numbers. The real curve works with enormous numbers, but the idea can be seen with 23-hour clock arithmetic and base 5.

1. The client picks `x = 6` and sends `5^6 mod 23 = 8`.
2. The relay picks `y = 15` and sends `5^15 mod 23 = 19`.
3. The client computes `19^6 mod 23 = 2`.

4. The relay computes `8^15 mod 23 = 2`.

Both obtain 2, and whoever only saw the 8 and the 19 would have to guess `x` or `y`.

Why it works. Multiplication is commutative: `x · (y · G)` and `y · (x · G)` are the same point. And the difficulty of recovering `x` or `y` from `X` or `Y` (the discrete logarithm problem) is what protects the secret.

ntor adds two things. First, the relay also has a long-term key `b` with public value `B = b · G`, published in the directory; the client also computes `x · B`, which the relay recovers as `b · X`. This proves that whoever answers really is that relay and not an impostor, because only it knows `b`. Second, the resulting secret, together with the identities and the public values, goes through a key derivation function (HKDF) that produces the circuit's encryption and verification keys. There is a variant, ntor v3, that lets the client send additional data inside the handshake itself.

CHAPTER 6

Directory and consensus

To choose relays, your client needs a reliable list of which ones exist, where they are and which public keys they have. That list cannot come from a single place, because whoever controlled it could lead you to relays of their choosing.

The solution is a handful of **directory authorities**, servers run by independent volunteers in different countries (on the order of ten). Every hour, each authority publishes its own opinion on the state of the network and all the others read it. Then they vote and produce a single document, the **consensus**, which is only valid if a majority of the authorities sign it. Clients have the authorities' public keys built in, so they check the signature and do not have to trust whoever hands them the document.

CONSENSUS DATA	WHAT IT IS FOR
List of relays and their keys	So the client can build circuits
Flags (Guard, Exit, Stable, Fast, HSDir)	Deciding which relays can be guards, exits or onion-service directories
Measured bandwidth	Spreading the load according to real capacity, measured by the authorities themselves
Network parameters	Adjusting the behaviour of the whole network at once

An attacker who wanted to falsify the network would have to control a majority of the authorities, not just one. That is one reason why their small number and diversity are a design decision rather than a shortcoming.

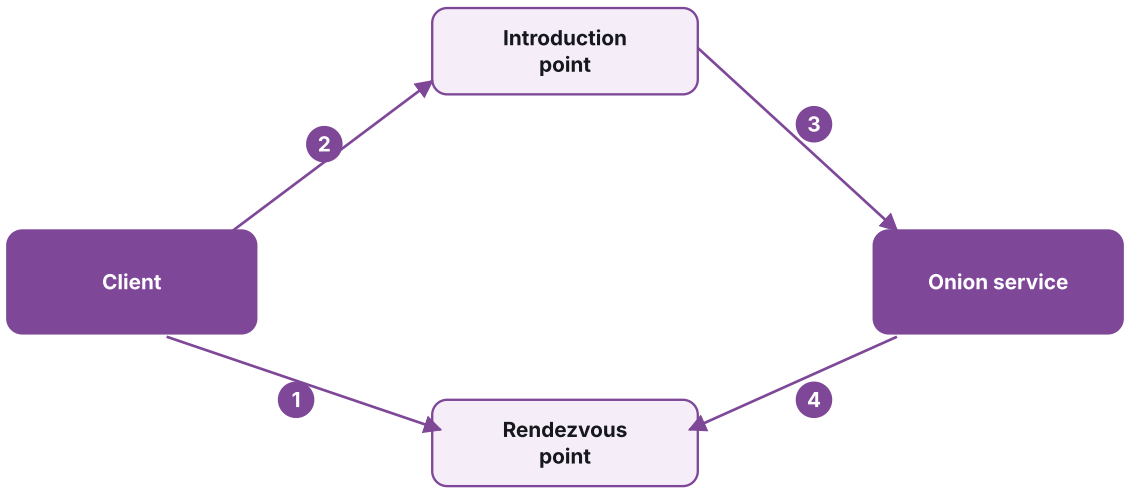
CHAPTER 7

Onion services

Tor does not only hide the visitor. An **onion service** is a site or service whose location is also hidden: nobody, not even whoever visits it, knows which server it is on or where. Its address ends in `.onion`, is 56 characters long and is, at the same time, its public key. That means the address *is* the identity of the service and no external authority is needed to certify that it is authentic.

Who does what

A connection to an onion service is assembled from four pieces:



1. The client builds a circuit to a rendezvous point.
2. Over another circuit, it asks an introduction point for a meeting there.
3. The introduction point notifies the service, without knowing who it is.
4. The service builds its circuit to the rendezvous point; the two circuits join.

Each arrow is a three-hop circuit; the rendezvous point joins the two sides.

1. **Introduction points.** The service picks a few relays and builds circuits to them. They are its "mailboxes": they receive connection requests without knowing who the service is.
2. **Descriptor.** The service publishes, on special relays (the HSDirs), a signed document saying what its introduction points are. The descriptor is encrypted with keys derived from the address itself, so the HSDirs cannot read it.
3. **Rendezvous point.** The client that wants to connect picks a relay and builds a circuit to it. It is the meeting place.
4. **Introduction and rendezvous.** The client sends an introduction point the message "meet me at this rendezvous point". The service receives it, builds its own circuit to that rendezvous point and the two circuits are joined.

The result is six hops between client and service, three on each side, with the rendezvous point in the middle. No relay knows both the client and the service, and the traffic is encrypted end to end with a key that only they share.

The .onion address

The v3 address is not an arbitrary hash: it contains the service's public key, an error check and a version number.

Secondary formula. The address is assembled like this:

$$\text{address} = \text{base32}\big(K \,\|\, \text{checksum} \,\|\, \text{version}\big)$$

> **In words:** the address is the service's public key with a small check code, so that a mistake when copying it is detected.

The keys published in the directory are renewed every day from the main key through a process called *blinding*, so that a relay cannot follow a service from one day to the next. Since 2021 only v3 addresses work, much more secure than the old 16-character ones.

Against flooding attacks, onion services have had a **proof-of-work** defence since August 2023: under normal conditions it goes unnoticed, but if the service receives an avalanche of requests, clients must solve a small computational puzzle before being served. A legitimate client takes from milliseconds to a minute depending on the attack; an attacker sending thousands of requests pays that cost thousands of times.

CHAPTER 8

Bridges and pluggable transports

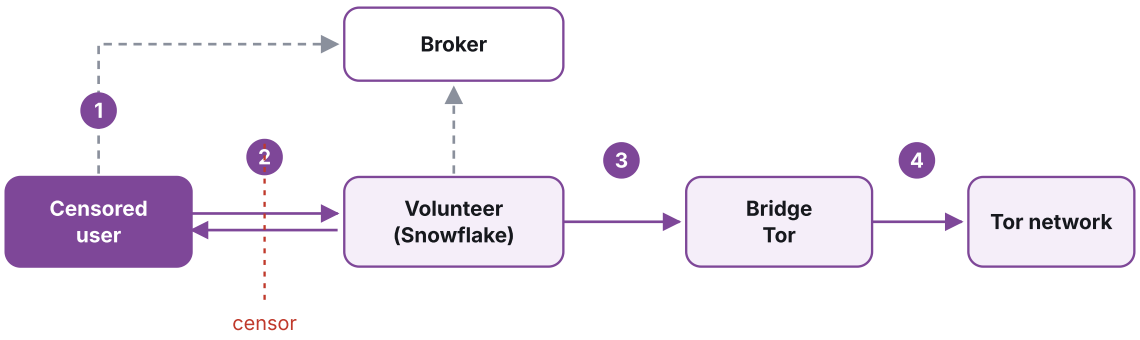
Relays are public, so a government or a network can block Tor simply by preventing connections to those addresses. **Bridges** are relays that are not on the public list. They are requested one by one through channels such as the project's bridges site, so a censor cannot obtain them all at once.

But hiding the address is not enough: Tor's traffic has a recognisable shape. That is why **pluggable transports** are used, which disguise the connection:

TRANSPORT	HOW IT DISGUISES ITSELF
obfs4	Makes the traffic look like random noise with no recognisable structure
Snowflake	Uses WebRTC connections through volunteers who lend their browser as a temporary bridge
WebTunnel	Imitates a normal HTTPS web connection to a common site

Snowflake: a borrowed bridge

With **Snowflake**, anyone can leave a tab open or install an extension and become an ephemeral bridge for someone who is being censored. An intermediary service, the *broker*, connects each user with a volunteer, and the connection is encrypted between the two. The traffic looks like a video call, something censors hesitate to block because it has many legitimate uses. Tor Browser includes a connection assistant that, if Tor does not connect, suggests suitable bridges depending on the country.



1. User and volunteer meet through the broker.
2. They connect over WebRTC, encrypted end to end: it looks like a video call.
3. The volunteer forwards the traffic to a bridge.
4. The bridge feeds it into the Tor network.

A volunteer lends their browser as an ephemeral bridge; the censor only sees a video call.

None of these measures guarantees that Tor will work in every country: it is a continuous race between those who block and those who invent new disguises.

CHAPTER 9

Tor Browser

Using Tor with an ordinary browser is of little use: your usual browser has cookies, extensions, fonts and settings that identify you even if the IP is different. **Tor Browser** is a hardened version

of Firefox, configured so that all its users look the same.

MEASURE	WHAT IT PREVENTS
All traffic through Tor	Any request leaving with your real IP
A different circuit per site	Two websites discovering that you are the same person
Isolation of cookies and data per site	Cross-site tracking with third-party cookies
Uniform fingerprint	Your browser standing out by screen, fonts or language
Window size adjustment	Your screen size working as a fingerprint
NoScript and security levels	Limiting JavaScript and dangerous features
No saved history or cookies	A local trace remaining when you close it

Tor Browser offers three security levels. In the standard one almost everything works, in the safest one features that have been used for attacks are disabled, at the cost of some websites not displaying properly, and the intermediate level sits between the two. There is a version for Windows, macOS, Linux and Android. On iOS there is no official Tor Browser; the project recommends Onion Browser.

The central idea is **uniformity**: the more users are identical, the harder it is to single you out. That is why it is best not to add extensions, resize the window by hand or install new fonts: every change makes you more unique.

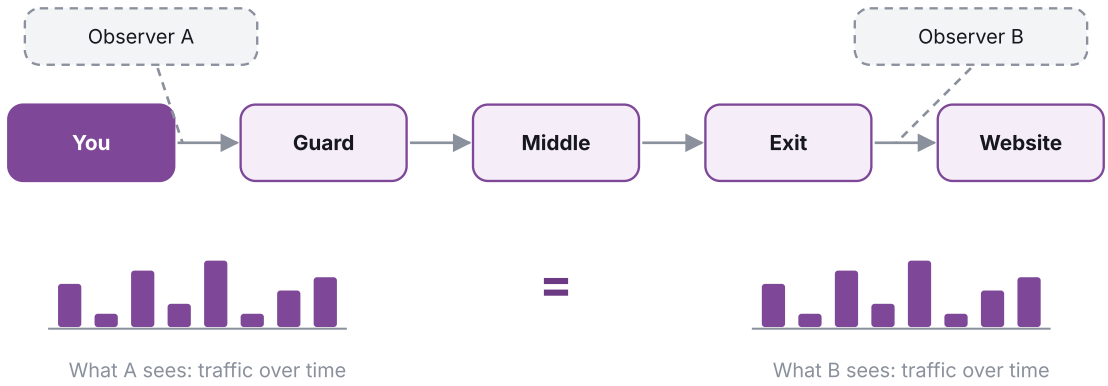
CHAPTER 10

Attacks and limits

Tor is not a guarantee of anonymity, but a considerable difficulty for whoever wants to track you. These are the known attacks.

The best-known limit: correlation

End-to-end correlation. If an adversary sees the traffic entering the network on your side and the traffic leaving on the other, it can compare when and how much is sent and deduce that they are the same conversation. It does not need to break any encryption. It is Tor's best-known theoretical limit.



Same timing and volume pattern: same conversation, with no encryption broken

Whoever sees both ends can match traffic patterns without decrypting anything.

Malicious relays. Anyone can run relays, and operators have been discovered who controlled a large portion of the network for unclear purposes. The project detects and expels them, and the use of guards reduces the risk of falling into their hands again and again. Exit relays can be malicious and alter or spy on unencrypted traffic, which is why it is important to use HTTPS.

Attacks on onion services. In 2024 the case was documented of an onion-service user identified by the authorities (according to the press, German ones). According to the project's analysis, the victim was running an old version of a chat program without the most recent protections; the attacker would have located their guard and then cross-referenced timing data. The project itself concluded that Tor Browser remains safe if kept up to date, and that users with very long connections are more vulnerable.

Website fingerprinting. An adversary who only sees your entry can try to guess which website you visit from the shape of your traffic: how many cells, in what order. It works best with sites that are few in number; the defences being studied are adding padding and varying the patterns.

User mistakes. Most real identifications did not break Tor, but exploited slips: logging in with a personal account, using the same alias on several sites, opening a downloaded document outside the browser or installing an add-on.

Browser flaws. Tor Browser is based on Firefox and inherits its flaws. In the past, browser vulnerabilities have been exploited to reveal the real IP; that is why it is essential to keep it updated and, if the risk is high, to use the strictest security level.

CHAPTER 11

Tor, VPNs and combinations

Tor and a VPN are alike in that they hide your IP, but they solve different problems.

	TOR	VPN
Whom you trust	Nobody in particular: the traffic is split between three relays	The company that runs the VPN
Who knows your IP and your destination	No single relay at once	The provider, always
Speed	Slower, because of the three hops	Almost that of your connection
Conceals that you use Tor	Only with bridges	Not applicable
Cost	Free	Usually paid

A VPN protects well against your internet provider and local networks, and shifts the trust to the VPN provider. Tor distributes that trust, but it is slower and some sites block it. They are not substitutes: they are tools for different threats.

Tor over VPN. You first connect to the VPN and from there open Tor. Your internet provider only sees the VPN, and the Tor guard sees the VPN's IP instead of yours. It can be useful for hiding that you use Tor, at the cost of the VPN provider knowing that you do. An example is a VPN such as Mullvad, which does not ask for personal data.

VPN over Tor. You connect to Tor and from there to a VPN. It is uncommon and complex to set up. It helps in specific cases, such as reaching a service that blocks Tor exits, but the VPN identifies you to its provider.

For most people, using only Tor Browser is the simplest and most reasonable option. Combining tools adds complexity and every new layer can introduce mistakes that reveal more than they hide.

CHAPTER 12

Tails and Whonix

Tor Browser only protects the browser. If you open another program, the rest of your computer keeps connecting directly. Two complete systems solve this problem.

Tails is an operating system that starts from a USB stick. All its traffic goes out through Tor, it leaves no trace on the computer's disk and forgets everything when shut down, except what you save in an optional encrypted storage. It is useful for working on borrowed or shared computers and for journalists or activists at a high level of risk.

Whonix works with two virtual machines: a gateway that sends all traffic through Tor and an isolated workstation that can only talk to the gateway. If a program on the workstation is compromised, it cannot find out your real IP, because it never knows it. It is usually used inside Qubes OS.

	TOR BROWSER	TAILS	WHONIX
Protects	Only the browser	The whole system	All applications on the work machine
Persistence	Normal	None by default	Normal inside the virtual machine
Difficulty	Low	Medium	High

CHAPTER 13

Good practice

The tool matters less than the habit. These rules reduce the risk a lot.

1. **Download Tor Browser only from the official site** and keep it updated. Updates fix flaws that are actually exploited.
2. **Do not log in with your real identity** in a session you want to keep anonymous, and do not reuse aliases across sites.
3. **Always use HTTPS** or onion services, so the exit relay does not see your content.

4. **Do not change the browser's settings** or install extensions: every change makes you more recognisable.
5. **Do not open downloaded files** outside Tor Browser while you are connected, because they can connect to the internet on their own.
6. **Use bridges** if your environment blocks Tor or if hiding that you use it matters.
7. **Adjust the security level** to the risk: stricter the more you have at stake.
8. **Avoid very long sessions** with the same identity if the risk is high, and use "New identity" to separate activities.
9. **Do not mix identities:** what you do anonymously and what you do under your name must be kept apart, including in schedules and habits.

And one rule of common sense: before deciding how much protection you need, answer the three questions of the threat model. Against a weak adversary, Tor Browser is enough; against a very capable one, more measures and a lot of care are needed.

CHAPTER 14

How the protocol is evolving

Tor keeps changing. These are the most important improvements of recent years.

Counter Galois Onion (CGO). It is the new layered-encryption scheme that replaces tor1. It fixes the three weak points of chapter 4: each cell now carries a 16-byte integrity check instead of 4, the keys evolve with every cell (if someone obtains a key, they cannot read what came before) and the cells are chained, so altering one invalidates all the following ones. This closes the tagging attack. CGO debuts in Arti, where it has been considered production-ready since July 2026, and will reach the rest of the network as relays and clients adopt it.

Arti. It is a rewrite of Tor in the Rust language. This language structurally avoids whole classes of memory errors, which are a common source of vulnerabilities in programs written in C. Arti already includes congestion control enabled by default and is moving towards being able to work as a relay and as a directory authority.

Congestion control and conflux. Tor improved its performance with finer congestion control and with *conflux*, which splits one connection across two circuits to use the available bandwidth better.

Onion-service defences. The proof of work against flooding (chapter 7) and the protections called *vanguards*, which make it harder to locate a service's guard, are part of this effort.

CHAPTER 15

Community, funding and legality

Tor is sustained by The Tor Project, a non-profit organisation. For years it depended heavily on funding from government agencies, and it has worked to diversify it with individual donations, foundations and public fundraising campaigns. The code is open, the network is run by volunteers and anyone can review how it works.

Anyone can contribute without being an expert: by running a relay or a bridge, installing the Snowflake add-on, translating documentation, reporting bugs or donating. Exit relays are the ones that carry the most legal risk for their operator, because the traffic appears to originate from their IP address; middle and entry relays are much simpler.

Using Tor is legal in most countries. In a few it is restricted or blocked, and it is worth checking local rules before using it in a risky environment. Tor is a tool: it can serve to protect a journalist as well as for misuse, like a knife, and its design cannot tell intentions apart.

CHAPTER 16

Sources

- Tor Project, "Tor is still safe": https://blog.torproject.org/tor-is-still-safe/
- Help Net Security, on Counter Galois Onion: https://www.helpnetsecurity.com/2025/11/25/tor-project-cgo-encryption-update/
- CyberInsider, on Arti version 2.5.0: https://cyberinsider.com/tor-releases-arti-2-5-0-with-stable-cgo-encryption-and-security-fixes/
- Tor Project, proof-of-work defence for onion services: https://blog.torproject.org/introducing-proof-of-work-defense-for-onion-services
- Tor specifications, onion services: https://spec.torproject.org/rend-spec/overview.html
- Tor Metrics: https://metrics.torproject.org/