* [Skip to main content](https://restoreprivacy.com/anonymity-networks/#genesis-content)
* [Skip to footer](https://restoreprivacy.com/anonymity-networks/#footer)

**Additional menu**

[**Restore Privacy**](https://restoreprivacy.com/)

Your online privacy resource center

* Privacy Tools
  + [Secure Browser](https://restoreprivacy.com/browser/secure/)

* + [VPN](https://restoreprivacy.com/vpn/)

* + [Ad Blocker](https://restoreprivacy.com/ad-blocker/)

* + [Secure Email](https://restoreprivacy.com/email/secure/)

* + [Private Search](https://restoreprivacy.com/private-search-engine/)

* + [Password Manager](https://restoreprivacy.com/password-manager/)

* + [Tor](https://restoreprivacy.com/tor/)

* + [Privacy Tools](https://restoreprivacy.com/privacy-tools/)

* Research
  + [Privacy News](https://restoreprivacy.com/category/news/)

* Reviews
  + [Email Reviews](https://restoreprivacy.com/email/reviews/)
    - [ProtonMail](https://restoreprivacy.com/email/reviews/protonmail/)

* + - [Tutanota](https://restoreprivacy.com/email/reviews/tutanota/)

* + - [Mailfence](https://restoreprivacy.com/email/reviews/mailfence/)

* + - [CTemplar](https://restoreprivacy.com/email/reviews/ctemplar/)

* + - [Mailbox.org](https://restoreprivacy.com/email/reviews/mailbox-org/)

* + - [Posteo](https://restoreprivacy.com/email/reviews/posteo/)

* + - [Fastmail](https://restoreprivacy.com/email/reviews/fastmail/)

* + - [Hushmail](https://restoreprivacy.com/email/reviews/hushmail/)

* + [VPN Reviews](https://restoreprivacy.com/vpn/reviews/)
    - [ExpressVPN](https://restoreprivacy.com/vpn/reviews/expressvpn/)

* + - [NordVPN](https://restoreprivacy.com/vpn/reviews/nordvpn/)

* + - [Surfshark](https://restoreprivacy.com/vpn/reviews/surfshark/)

* + - [Perfect Privacy](https://restoreprivacy.com/vpn/reviews/perfect-privacy/)

* + - [CyberGhost](https://restoreprivacy.com/vpn/reviews/cyberghost)

* + - [IPVanish](https://restoreprivacy.com/vpn/reviews/ipvanish/)

* + - [Private Internet Access](https://restoreprivacy.com/vpn/reviews/private-internet-access/)

* + - [Netflix VPN](https://restoreprivacy.com/vpn/best/netflix/)

* + - [Best VPN for Torrenting](https://restoreprivacy.com/vpn/best/torrenting/)

* + - [NordVPN vs ExpressVPN](https://restoreprivacy.com/vpn/comparison/expressvpn-vs-nordvpn/)

* + [Cloud Storage Reviews](https://restoreprivacy.com/category/cloud-storage-reviews/)
    - [pCloud](https://restoreprivacy.com/cloud-storage/reviews/pcloud/)

* + - [Nextcloud](https://restoreprivacy.com/cloud-storage/reviews/nextcloud/)

* + - [IDrive](https://restoreprivacy.com/cloud-storage/reviews/idrive/)

* + - [SpiderOak](https://restoreprivacy.com/cloud-storage/reviews/spideroak-one/)

* + - [Sync.com](https://restoreprivacy.com/cloud-storage/reviews/sync-com/)

* + - [MEGA Cloud Storage](https://restoreprivacy.com/cloud-storage/reviews/mega/)

* + - [Tresorit](https://restoreprivacy.com/cloud-storage/reviews/tresorit/)

* + [Secure Messenger Reviews](https://restoreprivacy.com/category/secure-messenger-reviews/)
    - [Telegram](https://restoreprivacy.com/secure-encrypted-messaging-apps/telegram/)

* + - [Signal](https://restoreprivacy.com/secure-encrypted-messaging-apps/signal/)

* + - [Wire](https://restoreprivacy.com/secure-encrypted-messaging-apps/wire/)

* + [Password Manager Reviews](https://restoreprivacy.com/category/password-manager-reviews/)
    - [KeePass](https://restoreprivacy.com/password-manager/reviews/keepass/)

* + - [NordPass](https://restoreprivacy.com/nordpass/)

* + - [1Password](https://restoreprivacy.com/password-manager/reviews/1password/)

* + - [Dashlane](https://restoreprivacy.com/password-manager/reviews/dashlane/)

* + - [LastPass](https://restoreprivacy.com/password-manager/reviews/lastpass/)

* + - [Bitwarden](https://restoreprivacy.com/password-manager/reviews/bitwarden/)

* VPNs
  + [What is VPN](https://restoreprivacy.com/vpn/)

* + [Best VPN Services](https://restoreprivacy.com/vpn/best/)

* + [VPN Router](https://restoreprivacy.com/vpn-router/)

* + [VPN Coupons](https://restoreprivacy.com/vpn/coupons-discounts-deals/)
    - [ExpressVPN Coupon](https://restoreprivacy.com/vpn/coupons-discounts-deals/expressvpn/)

* + - [NordVPN Coupon](https://restoreprivacy.com/vpn/coupons-discounts-deals/nordvpn/)

* + [Free VPN](https://restoreprivacy.com/vpn/free/)

* + [Free Trial VPN](https://restoreprivacy.com/vpn/free-trial)

* + [Cheap VPNs](https://restoreprivacy.com/vpn/cheap/)

* + [VPN for Firestick](https://restoreprivacy.com/vpn/best/firestick-fire-tv/)

* [Mission](https://restoreprivacy.com/mission/)

No Ads: You can support us by [donating](https://restoreprivacy.com/donate/) or purchasing items through our links. [[Learn more]](https://restoreprivacy.com/mission/)

**Anonymity Networks: VPNs, Tor, and I2P**

*FEBRUARY 3, 2020 By*[**MIRIMIR**](https://restoreprivacy.com/author/mirimir/)[**4 COMMENTS**](https://restoreprivacy.com/anonymity-networks/#comments)



Online privacy and anonymity depend on preventing your ISP and other local adversaries from seeing communication content or metadata, and on preventing remote adversaries from seeing information about your identity or location. The first step is [securing your Internet connection](https://restoreprivacy.com/controlling-communication-channels/). Basically, you don’t connect devices directly to the Internet. Instead, you connect them through a standalone modem/router using Ethernet, and interpose a hardware firewall between the modem/router and your devices. That helps prevent leakage of information about your identity and location, and also protects your devices from compromise by external adversaries.

While that’s a typical professional setup for broadband, virtually nobody does it for smartphones. But now, at least, it’s feasible with the [Librem 5](https://puri.sm/products/librem-5/) and [PinePhone](https://www.pine64.org/pinephone/).

However, your ISP and other local adversaries can still see IP addresses that you access, and all communication content and metadata. And remote adversaries can still see content and metadata, and your ISP-assigned IP address. Although apps with end-to-end encryption will hide content, and some metadata, they don’t hide local or remote IP addresses.

**OK, so how can we fully hide content, metadata and IP addresses?**

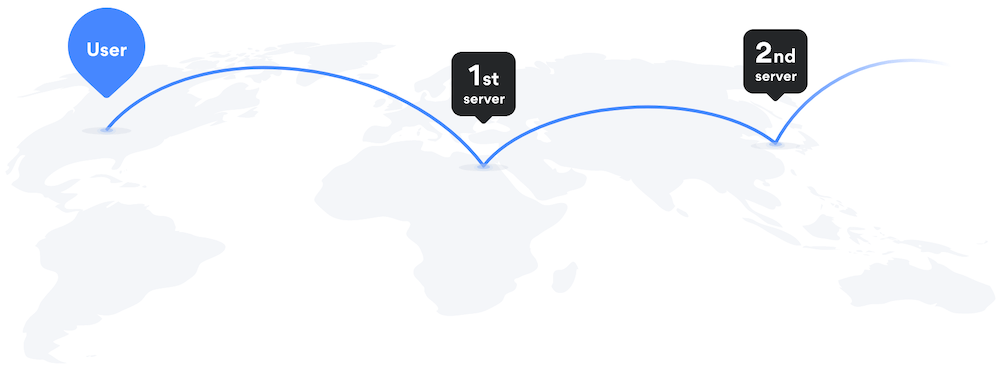
Basically, there must be an intermediary. It must strip out IP addresses. And all traffic between our devices and the intermediary must be securely encrypted. That protects almost everything from local adversaries, except for time and traffic pattern, and of course the IP address of the intermediary.

It also hides your ISP-assigned IP address from remote adversaries. However, they can still see content and metadata, unless you use apps with end-to-end encryption. Even then, metadata that’s required for addressing and transport can’t be hidden. So it’s best to avoid metadata that creates associations with our meatspace identities or locations. That is, we must compartmentalize our online personas from our meatspace identities. I’ll cover that in [How to Really Be Anonymous Online].

**VPN Services and Anonymous Proxies**

Anyway, such intermediaries are generally called “anonymous proxies”. Now, that typically means VPN services. Some people still use HTTPS and SOCKS proxies, but those are iffy, because they may leak users’ IP addresses to sites. And even Cloudflare’s VPN Warp does that, by design, because its focus is protection from local adversaries.

Some VPN services offer [multi-hop routing](https://restoreprivacy.com/vpn/multi-hop/) to protect against traffic analysis and infrastructure compromise. That is, the proxy isn’t just a single server. Traffic gets routed through multiple servers, often on different continents. So an adversary couldn’t see traffic by pwning just one data center or ISP.



Some emphasize that all traffic between VPN servers is [securely encrypted](https://www.perfect-privacy.com/en/features/multi-hop-vpn). And Perfect Privacy further claims that its [NeuroRouting](https://restoreprivacy.com/neurorouting/) option minimizes Internet exposure of users’ traffic, by selecting an entry server near the user, and an exit server near the site being accessed, perhaps in the same data center.

**But here’s the problem: You can’t rely on an anonymous proxy to actually hide anything.** That’s because there’s no way to know what information it — or an adversary with access to its infrastructure — is retaining. Or who it’s cooperating with, or compromised by. You’re stuck with trusting it, or at least, trusting that its infrastructure is secure.

**Distributing Information and Trust**

There is indeed no protection against adversaries that are omniscient and omnipotent. However, you can reduce the risk of compromise by distributing information and trust. To my knowledge, it was David Chaum [who first noted that](https://www.cs.utexas.edu/~shmat/courses/cs395t_fall04/chaum81.pdf) in “Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms”, published in 1981:

*This paper presents a solution to the traffic analysis problem that is based on public key cryptography. Baran has solved the traffic analysis problem for networks, but requires each participant to trust a common authority. In contrast,****systems based on the solution advanced here can be compromised only by subversion or conspiracy of all of a set of authorities****. [emphasis added]*

Although the details of his implementation aren’t relevant here, all systems that provide substantial anonymity rely on distributing information and trust.

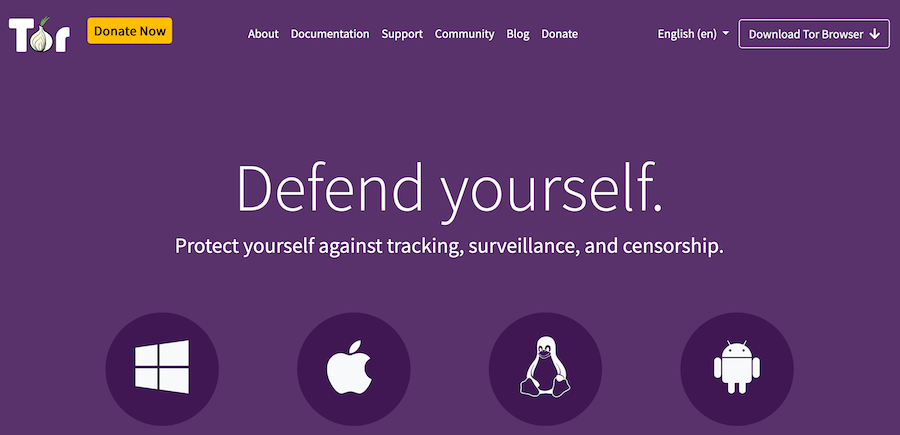
* remailer mix networks
* Tor
* I2P
* nested VPN chains
* possibilities for smartphones

**Remailer Mix Networks**

Chaum’s approach for “untraceable electronic mail” was first implemented in the Mixmaster and Cypherpunk remailer networks. It’s an elegant approach, and potentially very effective against traffic analysis. Nodes introduce latency by caching messages. And they introduce uncertainty by mixing messages before forwarding.

However, they were complicated, and so not many ever used them. And that kept them too small to effectively resist traffic analysis. They still exist, though.

**Tor**



Tor was designed to provide users with low-latency access to Internet resources. And so it didn’t implement caching or mixing. While it’s somewhat decentralized, it’s not a P2P network. It currently [comprises](https://metrics.torproject.org/relayflags.html) about 6300 servers aka relays. About 3000 are trusted as guards, and about 1000 as exits. A small group of trusted relays cooperatively provides various directory and authentication services. And there are currently over two million [users](https://metrics.torproject.org/userstats-relay-country.html).

So basically, after getting information about relays from directory servers, Tor clients build circuits to use for accessing stuff. For each circuit, a client chooses three relays: an entry guard, a middle relay, and an exit relay. Each client chooses a few entry guards to use consistently. That helps protect clients against mass surveillance and deanonymization attacks by malicious relays.

This [Tor.SE answer](https://tor.stackexchange.com/questions/12100/understanding-the-tor-circuit#12101) explains how circuit construction works. Starting with the entry guard, a Tor client extends circuits by successively negotiating forward and reverse encryption keys with each relay, relying on the relay’s public key for authentication and encryption. So bottom line:

* “Each relay strips a layer of encryption using the shared forward key on the way forward.”
* “Each relay adds a layer of encryption using the shared reverse key on the way back.”
* “The same path (circuit) is used for both forward and reverse.”

Given the multi-hop routing with layered encryption, this is called [“onion routing”](https://www.onion-router.net/). And it provides anonymity by distributing information and trust among the three relays in each circuit:

* An entry guard sees the IP addresses of the client and middle relay, but only encrypted content.
* A middle relay sees the IP addresses of the guard and exit, but only encrypted content.
* An exit relay sees the IP addresses of the middle relay and destination, and plaintext content.

Tor also supports [onion services](https://community.torproject.org/onion-services/overview/), which are accessible only through Tor clients. Like clients, onion services normally build three-relay circuits. At startup, onion services establish persistent circuits to relays that will serve as introduction points. They also publish those anonymous introduction points to a distributed hash table.

To initiate connections with an onion service, a client first learns its introduction points from the distributed hash table. It then picks a rendezvous point, and builds a circuit to it. And then it builds a circuit to one of the onion service’s introduction points, and requests a connection through the rendezvous point that it’s picked. And finally, the onion service builds a circuit to the rendezvous point. That way, the client and onion service are mutually anonymous. However, there’s an option for onion services to opt out of anonymity, and connect directly to introduction and rendezvous points.

To better protect against deanonymization attacks, there’s the [vanguards](https://github.com/mikeperry-tor/vanguards) addon. It uses two entry guards in “double-hop” mode, instead of just a single guard. Although it’s mostly for onion services, it also works for clients. It’s not yet part of Tor itself, and is implemented in Python.

**In theory, Tor is among the most anonymous of workable options. But there’s**[**cause for distrust**](https://restoreprivacy.com/tor/)**.** Realistically, there are many malicious relays. At least some have just avoided detection through discretion. But there’s reason to suspect that the US government and its friends may operate malicious relays with the tacit approval of at least some Tor Project staff. Or that Tor developers may delay fixing vulnerabilities, in deference to government interests. And even if Tor itself isn’t fatally compromised, the NSA and other global adversaries may be logging connections to all Tor entry guards and exit relays.

**Even so, in my opinion, Tor is theoretically too anonymous to ignore completely.** Even at worst, it likely provides better anonymity than a random VPN service does. Otherwise, an adversary would need complete oversight for all Tor traffic. At best, it may provide better anonymity than nested VPN chains (and perhaps even Orchid) do. But in any case, connecting directly to the Tor network is arguably too risky. I always use Tor through nested VPN chains.

As [@thegrugq](https://medium.com/@thegrugq/tor-and-its-discontents-ef5164845908) says:

*VPNs provide good cover that Tor simply cannot – “I was using it to watch Hulu videos” is much better than – “I was just trying to buy illegal drugs online”*

**I2P**



I’m not as familiar with I2P. That’s basically because I2P doesn’t work well without open ports, and that gets complicated while using VPN services. But I’ll say more about that below.

I2P focuses on [anonymous peer-to-peer (P2P) communication](https://geti2p.net/en/docs/how/intro), rather than on anonymous Internet access. However, there are some [Internet outproxies](https://geti2p.net/en/faq), and that justifies covering it here.

I2P began as a [fork](https://geti2p.net/en/docs/how/intro) of [Freenet](https://en.wikipedia.org/wiki/Freenet), which was arguably the first P2P “platform for censorship-resistant communication”:

*I2P initially began in Feb 2003 as a proposed modification to Freenet to allow it to use alternate transports …*

Freenet is entirely about P2P content, and there’s no mechanism for Internet access. But it’s worth mentioning, if only to distinguish it from I2P and Tor, and as a warning.

Like many early P2P networks, Freenet relies primarily on “plausible deniability”, rather than on anonymity. That is, all peers can connect to each other directly. Given that, malicious peers can cache illegal content, and then log the IPs of peers that access it.

Freenet does employ a sophisticated encryption and forwarding strategy, to obfuscate senders and recipients from innocent intermediaries. So arguably, any peer can plausibly claim that they’re just an intermediary, handling end-to-end encrypted content.

However, given logging data from malicious peers, adversaries can discover peers that handle illegal content. Then they can attempt to distinguish senders and recipients from innocent intermediaries, through statistical traffic analysis. And then they can arrest and prosecute them.

This is [not theoretical](https://web.archive.org/web/20190226201043/https:/www.deepdotweb.com/2015/11/27/police-log-ips-making-arrest-by-planting-own-nodes-in-freenet/). Police used a modified version of the Freenet client, [Black Ice](https://retro64xyz.gitlab.io/assets/pdf/blackice_project.pdf). The Freenet Project [calls bullshit](https://freenetproject.org/police-departments-tracking-efforts-based-on-false-statistics.html):

*Documents initially made public by the Missouri police department describe their efforts on tracking Freenet usage. Using a simple scheme, they claim a near zero false positive rate for tracking the originator of a download. While we applaud all public documentation on attacks, we have to point out that the claimed effectiveness of their attacks is based solely on flawed mathematics. In reality, the false positive rate of their method is at least 83%, and close to 100% in real world scenarios.*

Maybe so, but establishing that in court depends on having an expert witness who can convince a jury that the prosecution’s case relies on “flawed mathematics”. That’s likely not so easy, and certainly expensive. It’s far better to avoid the attention. **I do not recommend using Freenet, except on a thoroughly anonymous VPS, managed and accessed through nested VPN chains and Tor.** I’ll cover the general approach in [How to Really Be Anonymous Online], but not specifically Freenet.

Anyway, I2P is also a P2P network. And so it’s potentially vulnerable to such peer discovery, and bullshit claims about traffic analysis.

However, I2P provides anonymity through multi-hop [“garlic routing”](https://geti2p.net/en/docs/how/garlic-routing), which Michael Freedman defined as an extension to [“onion routing”](https://www.onion-router.net/). Indeed, I2P is somewhat like Tor onion services, although the specifics are not at all similar. One key difference is that multi-hop I2P tunnels are unidirectional, whereas multi-hop Tor circuits are bidirectional.

I2P “messages go out K hops through the outbound tunnel and another K hops through the inbound tunnel, with K no longer than 3.” Each peer can [specify](https://blokt.com/guides/what-is-i2p-vs-tor-browser) how many hops its inbound and outbound tunnels will use. Basically, [more hops means better anonymity](https://geti2p.net/en/docs/how/tunnel-routing):

* 0-hop tunnels … very basic plausible deniability
* 1-hop tunnels … plausible deniability and basic anonymity
* 2-hop tunnels … costs of mounting the traffic analysis attack increases
* 3-hop tunnels … recommended for the highest level of protection

So for both I2P and Tor onion services, connections involve 4-6 hops.

The key distinction is that about 95% of I2P users reportedly [route each other’s traffic](https://geti2p.net/en/faq), and doing so is recommended to “help the network”. And so **virtually all I2P nodes are potentially discoverable by malicious peers**, just as with Freenet. But conversely, a given Tor client connects only with directory servers, and just a few entry guards. And so mass surveillance of Tor clients is harder.

Comparisons of I2P and Tor — or often, garlic and onion routing — sometimes note that garlic routing implements caching and mixing, whereas onion routing does not. That did indeed drive the decision to make I2P tunnels unidirectional. That way, there’s no need to negotiate common forward and reverse keys, and tunnels can [bundle](https://geti2p.net/en/docs/how/garlic-routing) multiple messages, “each with its own delivery instructions”.

**However,**[**I2P v9.44**](https://geti2p.net/en/download)**does not actually**[**implement**](https://geti2p.net/en/docs/how/intro)**caching or mixing of message content:**

*The two main mechanisms for allowing people who need strong anonymity to use the network are explicitly delayed garlic routed messages and more comprehensive tunnels to include support for pooling and mixing messages. These are currently planned for release 3.0, but garlic routed messages with no delays and FIFO [first in, first out] tunnels are currently in place. Additionally, the 2.0 release will allow people to set up and operate behind restricted routes (perhaps with trusted peers), as well as the deployment of more flexible and anonymous transports.*

It’s somewhat ambiguous to compare Tor and I2P by size. Tor does have over 200 times more users: over two million for [Tor](https://metrics.torproject.org/userstats-relay-country.html) vs “tens of thousands” for [I2P](https://blokt.com/guides/what-is-i2p-vs-tor-browser). However, reportedly 95% of I2P users route each other’s traffic, and there are only about 6300 Tor [relays](https://metrics.torproject.org/relayflags.html). Also, while Tor is arguably larger, perhaps I2P is harder to [compromise](https://blokt.com/guides/what-is-i2p-vs-tor-browser), in that I2P tunnels are unidirectional and independently routed, whereas Tor circuits are bidirectional.

Another issue is that I2P developers are mostly [anonymous](https://geti2p.net/en/about/team), whereas the Tor Project is not. That does arguably reduce the risk that they’ll be pressured. But it also means that we have no clue who they are. So there’s no basis for trusting them any more than that for trusting the Tor Project. And both I2P and Tor software are open-source.

**Bottom line, as with Tor, connecting directly to the I2P network is arguably too risky. And as with Tor, it’s prudent to connect through nested VPN chains.**

But here’s the thing: That’s more complicated with I2P.

I2P works best when peers can connect to your node, because then you’re not just using tunnels that your node builds to peers. And that requires allowing inbound TCP and UDP connections to your node’s [Internet-facing port](https://geti2p.net/en/faq#ports). When you’re just connecting through your broadband router, you can easily configure port forwarding through its WebGUI interface.

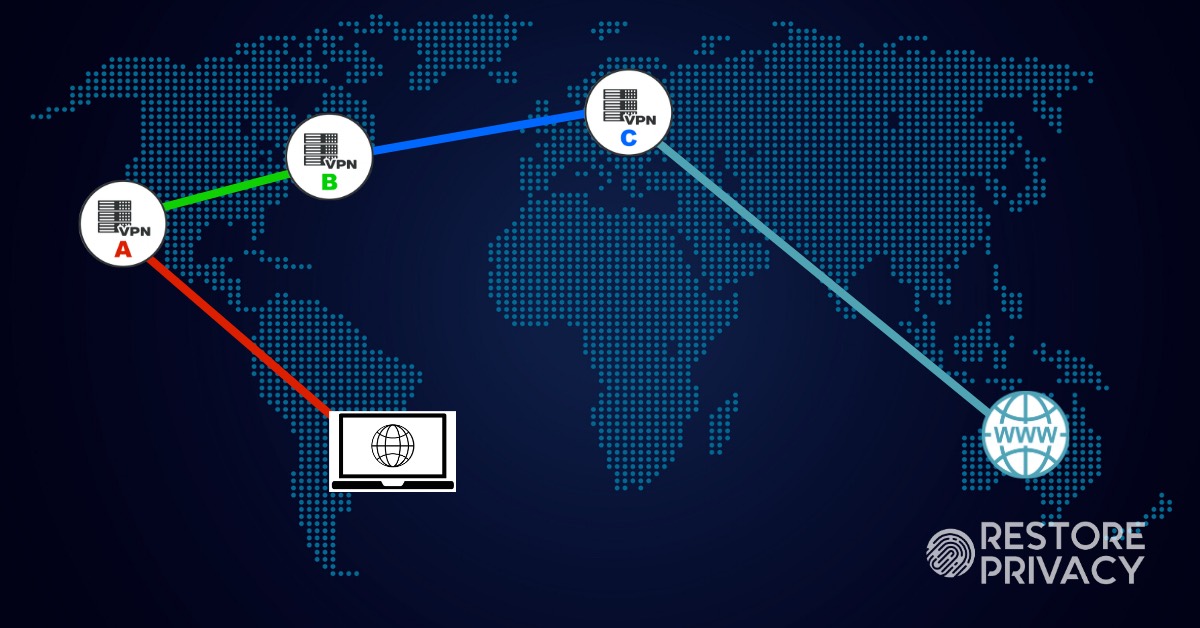
When you’re connecting through a VPN service, the I2P port must be open on the VPN server, and forwarded to your computer. However, many VPN services will configure port forwarding for their users.

In order for an I2P node (aka router) to connect to peers, it must know (among other data) their IP addresses. Routers upload their contact information to a [distributed database (netDb)](https://geti2p.net/en/docs/how/network-database) which makes it available to other routers. And they can also update netDb as needed, and flag their contact information as changed.

However, routers don’t necessarily check very often for [nedDb updates](https://geti2p.net/en/docs/how/network-database). They check frequently at startup, but only until they have information about enough reachable routers.

As long as your node’s VPN exit IP address is stable, this should work well enough. But if the it changes too often, the node could become less reachable by peers.

**Nested VPN Chains**



As noted above, you can’t rely on individual VPN services to actually hide anything. Multi-hop routing helps protect against adversaries. But bottom line, you’re stuck trusting the provider, just as you’d otherwise be trusting your ISP.

And it’s no better if you run your own VPN. You’d just be stuck trusting the data center that hosts your VPS or server, and the ISP that provides Internet connectivity. Also, you would likely be the VPN’s only user, and that greatly simplifies traffic analysis.

However, as with Tor and I2P, you can reduce the risk of compromise by distributing information and trust among multiple VPN services. That is, you can use nested chains. As with Tor circuits and I2P tunnels, each VPN in the chain sees only where traffic is coming from, and where it’s going. So in order to locate and identify you, and associate you with your online activity, adversaries would need data from most or all of the VPNs in your chain. In my experience, nested chains with 4-5 VPN services work reliably. And deanonymizing such nested VPN chains would arguably require considerable effort.

Virtual machines (VMs) are essential for this, because you can create isolated environments for different purposes. This is another aspect of compartmentalization (which I’ll cover fully in [How to Really Be Anonymous Online]). [Qubes](https://www.qubes-os.org/) is arguably the most secure compartmentalization option. However, there are restrictive hardware requirements, and it’s arguably overkill, unless you’re targeted by resourceful adversaries.

Using VirtualBox, you can create router VMs for VPN services, Tor and I2P. And by daisy chaining the router VMs, you can route traffic through the Internet however you like. You can also create multiple workstation VMs. So each of your personas would have its own workstation VM. And each of those VMs would reach the Internet through its own routing chain.

For better privacy, you would typically use some flavor of Linux or BSD in your workstation VMs. But you could also use Windows, Android, or even (with some effort) MacOS. Diversity also protects against a pernicious vulnerability: WebGL fingerprinting.

Basically, the WebGL fingerprint depends on the host’s graphics hardware, and also on the VM’s graphics driver. So if you have Debian and Ubuntu VMs running in the same host, browsers in both will have the same WebGL fingerprint. That’s because they use the same virtual graphics driver, and the same graphics hardware.

For the same reasons, that’s also true for two Centos VMs, two TrueOS (FreeBSD) VMs, two Windows VMs, two MacOS VMs, and so on. However, for any of those operating system (OS) types, VMs on different hosts will have different WebGL fingerprints. And on a given host, each OS type will also have a different WebGL fingerprint. Even so, you’re likely safe enough running multiple VMs with a given OS type, as long as you disable WebGL in browsers.

It’s crucial to keep in mind that the host machine is necessarily most trusted. VMs (aka guests) are not at all protected from host machines. They’re just software, alter all. And conversely, host machines are relatively well protected from VMs. So given that, host machines must be [protected from the Internet](https://restoreprivacy.com/controlling-communication-channels/). And it’s risky to actually use them for anything except running VMs.

Just as with machines generally, adding a firewall/router VM makes the workspace VM less untrusted, creating a DMZ between it and the Internet. But even so, VMs are always less trusted than host machines.

Also, there are exploits to break out from VMs to the host. So if you’re doing anything particularly iffy, such as playing with malware, or hanging out with people who play with malware, it’s best to isolate those VMs on a different host.

Anyway, some years ago, I wrote a [series of guides](https://www.ivpn.net/privacy-guides/advanced-privacy-and-anonymity-part-1) about using nested VPN chains and Tor. Those guides employ VirtualBox internal networks of pfSense router VMs, each serving as the gateway for a VPN service. Basically, recursive NAT forwarding creates nested VPN chains. For Tor, the guides recommend using Whonix. Whonix comprises two Debian VMs, a Tor gateway VM, and a workstation VM with Tor browser and other apps.

Although I still use that approach, the guides have become somewhat outdated, and I will revise them soon, after finishing this series of posts. In particular, I’ll be incorporating the option to include hardware routers running pfSense, which I’ve already covered in a [standalone guide](https://www.ivpn.net/setup/router-pfsense.html).

I’ll also be adding the option to change VPN chains periodically, as with Tor circuits and I2P tunnels. I’d always wanted to do that, but never figured out how to coordinate the activity of multiple pfSense VMs. However, I’ve recently [managed it](https://github.com/mirimir/vpnchains) by doing dynamic routing and NAT forwarding within a single Debian router VM. There are routing rules that direct one VPN through another, and iptables rules that restrict traffic to the chain. So even if a chain fails partially or completely, there will be no DNS or traffic leaks. There’s just a brief interruption in traffic.

This approach with dynamic nested VPN chains arguably provides better anonymity than the approach using pfSense VMs as VPN gateways. Because the chain changes periodically, perhaps every few minutes, rather than being static. And it works just as easily with Whonix. Also, although it does require some tedious massaging of OpenVPN configuration files, it’s less complicated overall than configuring multiple pfSense VMs. While each OpenVPN client isn’t compartmentalized in a separate VM, you can daisy chain multiple Debian router VMs, each with a short dynamic VPN chain.

As with the pfSense approach, **you should use a different VPN service for each level of the nested chain**. If you like, you can consistently use the same first and last VPN servers in your chains, because that may attract less attention. And you can select exit VPNs that sites don’t typically block.

You could even include Perfect Privacy with [NeuroRouting](https://restoreprivacy.com/neurorouting/) activated, perhaps using a pfSense gateway VM. And perhaps even combined with local dynamic nested VPN chains. That would give you quite the complexly changing Internet routing.

Although I won’t be getting into specifics here, I will update this post after revising the IVPN guides, and perhaps moving implementation details to GitHub.

**Even so, no matter how you route your traffic, it’s crucial to keep in mind that adversaries with global-scale intercept capability can identify and locate traffic endpoints.** As long as an adversary controls one end of a connection — for example, a site that you connect to, or a device that connects to your site — it can create unique traffic patterns. And then it can search available intercepts, looking for those patterns.

Intercepts generally capture traffic between the various networks that comprise the Internet. So once the adversary identifies a large-scale source and/or destination network for your traffic, it can search intercepts of traffic among parts of that network. By drilling down the routing hierarchy, it can eventually find your ISP. And if it has internal intercepts from that ISP, it can eventually find you. Or otherwise obtain the requisite data from the ISP.

However, that’s arguably not something that even the NSA will do for every anonymous coward.

**Possibilities for Smartphones**



Many VPN services provide clients for Android and iOS. And [Tor browser](https://play.google.com/store/apps/details?id=org.torproject.torbrowser&hl=en_US) is finally available for Android. That’ll never happen for iOS, because it only allows Safari code.

Alternatively, there is the long-awaited [Orchid app](https://www.orchid.com/) for Android (and perhaps soon for iOS). It’s a multi-provider VPN network, with dynamic multi-hop routing, and reputation-based route selection. Users buy bandwidth with supposedly anonymous Etherium-based cryptocurrency.

Orchid may provide privacy and anonymity that’s at least comparable to do-it-yourself nested VPN chains, and perhaps comparable to Tor. In that you’re distributing information and trust among multiple providers, who are paid anonymously. And it’s obviously far less work than nested VPN chains. But as with Tor, there’s the downside of trusting a complex system, which you likely won’t understand well.

**However, current smartphones are so horribly insecure that using such tools arguably provides merely an illusion of security, privacy and anonymity.** But there’s hope. With the [Librem 5](https://puri.sm/products/librem-5/) or [PinePhone](https://www.pine64.org/pinephone/), you can run a mobile Linux distro, and [secure the Internet connection](https://restoreprivacy.com/controlling-communication-channels/). Basically, you kill the device’s cellular and WiFi connectivity, and use an Ethernet-tethered WiFi router, or possibly even a tethered LTE modem.

Given Linux on secure hardware, you can use some of the approaches, discussed above, for robust privacy and anonymity. However, you can’t just install whatever Linux distro on smartphones. That’s because Linux was developed for x86-64 CPUs from Intel and AMD, whereas smartphone SoCs have ARM CPU cores. Both the Librem 5 and PinePhone SoCs have four 64-bit Cortex A53 cores.

And so everything must be compiled for ARM, after tweaking the source code. That’s a lot of work, and developers are still dealing with the basics. Although many single-board computers (such as the Raspberry Pi) also have ARM cores, there are enough differences that we can’t just use apps built for them. But at least we know that it’s doable.

**One limitation is clear: these ARM SoCs**[**don’t support virtualization**](https://forums.ubports.com/topic/3174/ubuntu-touch-libraries)**.** So if you want to compartmentalize VPN and Tor gateways from your workspace, you’d need to use Ethernet-tethered routers. That would be more stuff to lug around. But with small routers, in some sort of attached case, you’d basically just have a thicker phone.

Also, it does look like you can dual boot the PinePhone, at least, by [switching the SD card](https://forums.ubports.com/topic/2403/pinephone/443). So you could compartmentalize personas in different SD cards, and use a different anonymization setup for each.

**I can’t write anything reliable about smartphones with ARM SoCs until I have one. So for now, I’ll just suggest possibilities.** If I do manage to get one without blowing my cover, I’ll expand this into a full guide.

It does look like Ubuntu Touch includes OpenVPN, which you can [setup in terminal](https://forums.ubports.com/topic/918/solved-openvpn-vpnsecure-connected-but-no-network). However, not surprisingly, it also seems to have network manager. And that will likely be problematic, for doing anything fancy with VPNs. So it may be necessary to disable network manager.

I see that many users are asking for Tor browser in Ubuntu Touch. But developers are saying that it’s [“not seen as a core aim”](https://ubports.com/fr_FR/blog/notre-blog-1/post/ubuntu-touch-q-a-30-143). However, some years ago, I managed to compile Tor browser for the Raspberry Pi, so I’m pretty sure that it’s doable. After all, the Tor Project has managed it for Android on ARM SoCs.

But there are at least 25 mobile Linux and BSD distros under development for the PinePhone. I don’t have the patience now to research them all for OpenVPN and Tor browser. And there’s also the fact that things are totally in flux.

* [AOSC OS](https://aosc.io/projects/aosc-os)
* [Armbian](https://www.armbian.com/)
* [Batocera Linux](http://batocera-linux.xorhub.com/)
* [DietPi](https://dietpi.com/)
* [HardenedBSD](https://hardenedbsd.org/)
* [KDE Neon](https://neon.kde.org/)
* [Lakka](http://www.lakka.tv/)
* [LuneOS](https://en.wikipedia.org/wiki/LuneOS)
* [LibreELEC](https://libreelec.tv/)
* [Maemo Leste](https://maemo-leste.github.io/)
* [Manjaro](https://manjaro.org/)
* [NEMS Linux](https://nemslinux.com/)
* [NetBSD](https://www.netbsd.org/)
* [NextCloudPi](https://ownyourbits.com/nextcloudpi/)
* [OpenBSD](https://www.openbsd.org/)
* [OpenHAB](https://www.openhab.org/)
* [Open Media Vault](https://www.openmediavault.org/)
* [Plasma Mobile](https://www.plasma-mobile.org/)
* [PostmarketOS](https://postmarketos.org/)
* [Q4OS](https://q4os.org/)
* [Recalbox](https://www.recalbox.com/)
* [Retro Arena](https://odroidarena.com/)
* [SailfishOS](https://sailfishos.org/)
* [UBPorts Foundation](https://ubports.com/)
* [Volumio](https://volumio.org/)

***VPN Services***

If your mobile Linux distro includes OpenVPN, you could use any OpenVPN-based VPN service: one-hop, multi-hop, or Perfect Privacy with NeuroRouting. You could even use Orchid, once they release an app for Linux on 64-bit Cortex A53 cores.

It’s crucial to have iptables rules that prevent traffic from bypassing the VPN tunnel. If your mobile Linux distro includes the package iptables-persistent, it’s the easiest approach. First install it:

# apt -y install iptables-persistent

Unless you need IPv6, it’s best to just block it. You edit the IPv6 ruleset like this:

# nano /etc/iptables/rules.v6  
\*filter  
:INPUT DROP [0:0]  
:FORWARD DROP [0:0]  
:OUTPUT DROP [0:0]  
COMMIT  
  
And then reload it:

# ip6tables-restore < /etc/iptables/rules.v6

The VPN tunnel interface will likely be tun0. And the LAN interface will likely be enp0s3, or eth0 in older releases. But do check:

# ip a

Also get the VPN server’s IP address:

# netstat -natp | grep -e "Proto" -e "openvpn"

Then create an iptables ruleset that allows output via the LAN interface only to the VPN server (w.x.y.z), with everything else using the VPN tunnel:

# nano /etc/iptables/vpn-rules.v4  
\*filter  
:INPUT ACCEPT [0:0]  
:FORWARD ACCEPT [0:0]  
:OUTPUT ACCEPT [0:0]  
-A INPUT -s 127.0.0.1/32 -d 127.0.0.1/32 -j ACCEPT  
-A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT  
-A INPUT -j DROP  
-A FORWARD -j DROP  
-A OUTPUT -s 127.0.0.1/32 -d 127.0.0.1/32 -j ACCEPT  
-A OUTPUT -m conntrack --ctstate ESTABLISHED -j ACCEPT  
-A OUTPUT -o enp0s3 -d w.x.y.z -j ACCEPT  
-A OUTPUT -o tun0 -j ACCEPT  
-A OUTPUT -j DROP  
COMMIT  
  
And then load the VPN ruleset:

# iptables-restore < /etc/iptables/vpn-rules.v4

Whenever you reboot the device, the default iptables rules (rules.v4 and rules.v6) will be loaded. So you’ll need to load the VPN ruleset.

***Nested VPN Chains***

Given openvpn, you could also implement [nested VPN chains](https://github.com/mirimir/vpnchains), either static or dynamic. You could do that in the phone itself, or in a separate router, which is more secure. That way, exploited apps can’t readily bypass the VPN chain, and phone home.

Before implementing nested VPN chains in the phone, you’ll likely need to disable network manager, because otherwise it would interfere. You’ll also want to install iptables-persistent and block IPv6 traffic, as described above for VPN services.

You’ll also need to remove the forwarding rules from vpn-rules-base.v4, as well. For chains with two VPN services, you’d use this:

# nano /etc/iptables/vpn-rules-base.v4  
\*filter  
:INPUT ACCEPT [0:0]  
:FORWARD ACCEPT [0:0]  
:OUTPUT ACCEPT [0:0]  
-A INPUT -s 127.0.0.1/32 -d 127.0.0.1/32 -j ACCEPT  
-A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT  
-A INPUT -j DROP  
-A FORWARD -j DROP  
-A OUTPUT -s 127.0.0.1/32 -d 127.0.0.1/32 -j ACCEPT  
-A OUTPUT -m conntrack --ctstate ESTABLISHED -j ACCEPT  
-A OUTPUT -o enp0s3 -d VPN0 -j ACCEPT  
-A OUTPUT -o tun0 -d VPN1 -j ACCEPT  
-A OUTPUT -o tun1 -j ACCEPT  
-A OUTPUT -j DROP  
COMMIT  
  
Please note that VPN0 and VPN1 are merely placeholders. Scripts use the file to create the actual iptables ruleset, for a given combination of VPN servers. I’ve tested this in a Debian VM, so it ought to work in mobile Linux distros that include openvpn.

***Tor Browser***

If your mobile Linux distro includes Tor browser, installation is [very easy](https://www.torproject.org/download/). You just download, and extract the archive wherever you like. Then open a terminal in the tor-browser\_[locale] folder, and run ./start-tor-browser.desktop --register-app to add Tor browser to the applications menu.

If Tor browser isn’t available, you could build it from source. But that’s more than I want to get into here.

In any case, there are limitations and vulnerabilities:

* the Tor client runs only while Tor browser is running
* other apps aren’t configured to use Tor
* there’s no protection against apps bypassing Tor

You can configure other apps to use Tor, typically using the torsocks package.

Normally, one would use iptables to block all outgoing traffic, except for that from the Tor client. However, that requires that the Tor client be run as its own user, such as debian-tor. And Tor browser runs the Tor client as the login user.

But if your mobile Linux distro includes the package iptables-persistent, there is a workaround, albeit somewhat crude and fragile. First install iptables-persistent and block IPv6 traffic, as described above for VPN services.

Then, after installing and starting Tor browser, check what relays the Tor client is connecting to:

# netstat -natp | grep -e "Proto" -e "tor" | grep -v "127.0.0.1"

You’ll probably see as many as six foreign IP addresses (a.b.c.d, e.f.g.h, etc), which are Tor relays. Then create an iptables IPv4 ruleset that allows output only to those Tor relays:

# nano /etc/iptables/tor-rules.v4  
\*filter  
:INPUT ACCEPT [0:0]  
:FORWARD ACCEPT [0:0]  
:OUTPUT ACCEPT [0:0]  
-A INPUT -s 127.0.0.1/32 -d 127.0.0.1/32 -j ACCEPT  
-A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT  
-A INPUT -j DROP  
-A FORWARD -j DROP  
-A OUTPUT -s 127.0.0.1/32 -d 127.0.0.1/32 -j ACCEPT  
-A OUTPUT -m conntrack --ctstate ESTABLISHED -j ACCEPT  
-A OUTPUT -d a.b.c.d -j ACCEPT  
-A OUTPUT -d e.f.g.h -j ACCEPT  
...  
-A OUTPUT -j DROP  
COMMIT  
  
And then load the Tor rules:

# iptables-restore < /etc/iptables/tor-rules.v4

Tor browser should still work, but no other apps should reach the Internet.

Whenever you reboot the device, the default iptables rulesets (rules.v4 and rules.v6) will be loaded. So you’ll want to load tor-rules.v4 before using Tor browser. If the entry guard that Tor browser picked becomes unreachable, you’ll need to add the new one to your Tor ruleset. Just load the default IPv4 ruleset:

# iptables-restore < /etc/iptables/rules.v4

With Tor browser running, see what relays the Tor client is connecting to:

# netstat -natp | grep -e "Proto" -e "tor" | grep -v "127.0.0.1"

And then add any new ones to your Tor ruleset.

***VPN + Tor Browser***

It’s also pretty easy to route Tor browser through a VPN. If you’re running just a single VPN client, the VPN tunnel interface will likely be tun0. And the LAN interface will likely be enp0s3, or eth0 in older releases. But do check:

# ip a

Also get the VPN server’s IP address:

# netstat -natp | grep -e "Proto" -e "openvpn"

First install iptables-persistent, block IPv6 traffic, and get the VPN service working, as described above. Then, after installing and starting Tor browser, determine what relays the Tor client is connecting to, as described above.

Then create an iptables ruleset that allows output only to the VPN server (w.x.y.z) through the LAN interface, and only to the Tor relays (a.b.c.d, e.f.g.h, etc) through the VPN tunnel:

# nano /etc/iptables/vpn-tor-rules.v4  
\*filter  
:INPUT ACCEPT [0:0]  
:FORWARD ACCEPT [0:0]  
:OUTPUT ACCEPT [0:0]  
-A INPUT -s 127.0.0.1/32 -d 127.0.0.1/32 -j ACCEPT  
-A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT  
-A INPUT -j DROP  
-A FORWARD -j DROP  
-A OUTPUT -s 127.0.0.1/32 -d 127.0.0.1/32 -j ACCEPT  
-A OUTPUT -m conntrack --ctstate ESTABLISHED -j ACCEPT  
-A OUTPUT -o enp0s3 -d w.x.y.z -j ACCEPT  
-A OUTPUT -o tun0 -d a.b.c.d -j ACCEPT  
-A OUTPUT -o tun0 -d e.f.g.h -j ACCEPT  
...  
-A OUTPUT -j DROP  
COMMIT  
  
And then load the VPN+Tor ruleset:

# iptables-restore < /etc/iptables/vpn-tor-rules.v4

Whenever you reboot the device, the default iptables rules (rules.v4 and rules.v6) will be loaded. So you’ll need to load the VPN+Tor ruleset. If the entry guard that Tor browser picked becomes unreachable, you’ll need to add the new one to the VPN+Tor ruleset. Just load the default IPv4 ruleset:

# iptables-restore < /etc/iptables/rules.v4

With Tor browser running, see what relays the Tor client is connecting to:

# netstat -natp | grep -e "Proto" -e "tor" | grep -v "127.0.0.1"

And then add any new ones to the VPN+Tor ruleset, and load it:

# iptables-restore < /etc/iptables/vpn-tor-rules.v4

***Nested VPN Chains + Tor Browser***

You could also route Tor browser through [nested VPN chains](https://github.com/mirimir/vpnchains). If you use a separate router (which is more secure) you can just set it up as described.

If you want everything in the phone, first get nested VPN chains working in the phone, as covered above. Then, after installing and starting Tor browser, determine what relays the Tor client is connecting to, as described above.

Then stop Tor browser and nested VPN chains, and tweak vpn-rules-base.v4 to only allow output to the Tor relays (a.b.c.d, e.f.g.h, etc) through tun1:

# nano /etc/iptables/vpn-rules-base.v4  
\*filter  
:INPUT ACCEPT [0:0]  
:FORWARD ACCEPT [0:0]  
:OUTPUT ACCEPT [0:0]  
-A INPUT -s 127.0.0.1/32 -d 127.0.0.1/32 -j ACCEPT  
-A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT  
-A INPUT -j DROP  
-A FORWARD -j DROP  
-A OUTPUT -s 127.0.0.1/32 -d 127.0.0.1/32 -j ACCEPT  
-A OUTPUT -m conntrack --ctstate ESTABLISHED -j ACCEPT  
-A OUTPUT -o enp0s3 -d VPN0 -j ACCEPT  
-A OUTPUT -o tun0 -d VPN1 -j ACCEPT  
-A OUTPUT -o tun1 -d a.b.c.d -j ACCEPT  
-A OUTPUT -o tun1 -d e.f.g.h -j ACCEPT  
...  
-A OUTPUT -j DROP  
COMMIT  
Please note that VPN0 and VPN1 are merely placeholders. Scripts use the file to create the actual iptables ruleset, for a given combination of VPN servers.

Start nested VPN chains, and once it’s working, Tor browser.

***Whonix***

In theory, you could configure an intervening router (between modem and phone) as a Whonix gateway, and the smartphone as a Whonix workstation. In the [Whonix](https://www.whonix.org/) setup, there’s no forwarding allowed on the Ethernet link between the Tor gateway and workstation. The gateway just exposes Tor SocksPorts for the workstation. It’s not just a transparent Tor proxy. That forces apps in the workstation to use Tor properly, and prevents them from bypassing Tor. Also, if the workstation gets compromised by malware, it can’t readily fsck with the Tor client.

Whonix developers provide instructions for installing the Whonix gateway on the [Raspberry Pi 3B](https://www.whonix.org/wiki/Dev/Build_Documentation/Physical_Isolation#How_To_Install_Whonix-Gateway_.E2.84.A2_on_the_Raspberry_Pi_3_B_.28RPI3.29). Something like that would likely work for small ARM-based routers. Or you could just use a Pi 3B.

Alternatively, you could use two intervening routers, the first connecting through nested VPN chains, and the second as a Whonix gateway. But then you’d have three little boxes, in addition to the smartphone (the modem, and two routers). I suppose that you could make a simple case, and velcro it to the phone.

The same [approach](https://www.whonix.org/wiki/Dev/Build_Documentation/Physical_Isolation#How_To_Install_Whonix-Gateway_.E2.84.A2_on_the_Raspberry_Pi_3_B_.28RPI3.29) supposedly also works for Whonix workstation, so perhaps you could install that on your smartphone. However, it’s my impression that smartphone ARM SoCs are more idiosyncratic than random ARM systems, so that’s a long shot.

Whonix workstation is far easier to use, and arguably far more secure, than a random Linux distro with Tor browser and other torified apps, which has merely been configured to use a separate Tor gateway.

Whonix developers do recommend against installing Whonix workstation directly on hardware, given that “hardware serials would be visible” to the system. However, I can’t imagine how that would be any worse than just running Tor browser on the phone.

*This post is part of the ongoing*[*Advanced Privacy Guides*](https://restoreprivacy.com/category/advanced-privacy/)*series.*



**About Mirimir**

My name is Vladimir (Dimi) Arseniev. I stand for freedom and privacy. I focus on persona management, technology and methods for privacy and anonymity, and associated OPSEC. I write for websites, and post mainly to Wilders Security Forums and Hacker News. My public GnuPG key is on Keybase.

**Reader Interactions**

**Comments**

1. PrivateRyan *says*

MARCH 4, 2020

Thank you for the post and your privacy guides on ivpn. You never mention having a vpn on a router/firewall which would be a guard vpn that the isp sees. The host vpn would be a 2nd step and each vm vpn would be a third step. What do you think?

Also do you think it’s important to try to hide the modem (private, not isp provided) metadata from isp also? All traffic from router/firewall would be through a vpn, but the modem itself connects to the internet for updates and time checks so the isp would know the make of the modem and maybe other things? Do you think this is worth trying to hide?

[**REPLY**](https://restoreprivacy.com/anonymity-networks/#comment-72938)

1. Anon *says*

FEBRUARY 21, 2020

“So if you have Debian and Ubuntu VMs running in the same host, browsers in both will have the same WebGL fingerprint. That’s because they use the same virtual graphics driver, and the same graphics hardware.”

Im interested in this question of making sure your different personas dont share the same hardware fingerprint.  
I get the impression the only way to be reasonably sure of this separation of personas, is to have them using different hardware, such as different VPSes in different places. Is that accurate?

[**REPLY**](https://restoreprivacy.com/anonymity-networks/#comment-71339)

1. Jon *says*

FEBRUARY 4, 2020

I came across this <https://github.com/PrivateMemberPP/PP_openVPN_cascade> and forked it for Qubes OS here <https://github.com/92VV3M42d3v8/PP_openVPN_cascade>. I think it can be used for cascading other VPN too.

[**REPLY**](https://restoreprivacy.com/anonymity-networks/#comment-68802)

* + Mirimir *says*

FEBRUARY 6, 2020

Thanks.  
I don’t know German, but it looks like it uses the same VPN for all hops. Is that right?  
And if so, could that be readily changed?  
Also, for better or worse, my GitHub hack changes the entire multi-hop path, rather than changing individual nodes.

[**REPLY**](https://restoreprivacy.com/anonymity-networks/#comment-69030)

**Leave a Reply**

Top of Form

Your email address will not be published. Required fields are marked \*

Comment

Name \*

Email \*



Bottom of Form

***\*****Unless you get an error message, your comment has been successfully submitted and should appear within 24 hours. You can use any name and email address for the comment form (real or fake).*

**Footer**

**Recommended Privacy Setup**

1. **Private and secure browser**: [Modified Firefox](https://restoreprivacy.com/firefox-privacy/) or [Iridium](https://iridiumbrowser.de/)  
2. **VPN service**: [ExpressVPN](https://restoreprivacy.com/go/expressvpn)  
(with [49% off coupon](https://restoreprivacy.com/go/expressvpn-discount)) or [NordVPN](https://restoreprivacy.com/go/nordvpn)  
3. **Ad blocker**: [uBlock Origin](https://github.com/gorhill/uBlock) or [AdGuard](https://restoreprivacy.com/go/adguard)  
4. **Secure email**: [Tutanota](https://tutanota.com/) or [ProtonMail](https://restoreprivacy.com/go/protonmail)  
5. **Secure Messenger**: [Signal](https://signal.org/)  
6. **Private search engine**: [MetaGer](https://metager.org/) or [Swisscows](https://swisscows.ch/)  
7. **Password manager**: [NordPass](https://restoreprivacy.com/go/nordpass) or [Bitwarden](https://bitwarden.com/)

**About**

Restore Privacy was created to provide you with honest, useful, and up-to-date information about online privacy and security topics. You can support our work through [donations](https://restoreprivacy.com/donate/) and sharing info from the site with others.

You can see our [mission here](https://restoreprivacy.com/mission/).

Restore Privacy is also on [Twitter](https://twitter.com/ResPrivacy)

COPYRIGHT © 2020 RESTORE PRIVACY, LLC · [PRIVACY POLICY](https://restoreprivacy.com/privacy-policy/) · [TERMS OF USE](https://restoreprivacy.com/terms-of-use/) · [CONTACT](https://restoreprivacy.com/contact/)