

An anonymity network you can check for yourself

HidraNet is an open-source overlay network: onion routing over a post-quantum handshake, a Sybil-resistant DHT, self-certifying `.hidra` services and end-to-end encrypted applications. This paper describes what is built, how it works, and, just as plainly, what it does not do yet. Every claim in it points at code anyone can run.

Authors João Brandão, Leonardo Mello Licence MIT

Source github.com/NeurexBK/HidraNet Supersedes v1.0, September 2026

17.7k lines of Rust in the network engine	6 applications built into the browser	243 engine tests, attack scenarios included
0 scripts from a CDN: every library is served locally	1024 ML-KEM-1024, hybrid with X25519	MIT open source, every line

ABSTRACT

What this paper is

HidraNet combines onion routing with a hybrid post-quantum key exchange (X25519 + ML-KEM-1024), a Kademlia DHT hardened with the S/Kademlia defences, self-certifying hidden services and a suite of applications that encrypt on the device. The reference implementation is a Rust network engine (`hidra-node`) and a desktop browser.

We state one thing first, because a whitepaper that overstates is worse than none: **today, exit anonymity in HydraBrowser is provided by the Tor network**, which the browser bundles. The HydraNet protocol layer is implemented and tested; its relay population is not yet large enough to provide anonymity on its own. Section 11 sets out how that changes.

1

Status at a glance

COMPONENT	STATE	WHERE
Hybrid post-quantum handshake	BUILT	crypto/handshake.rs
Onion circuits, three hops	LAYERING BUILT; TELESCOPING NOT YET	onion/, client/
Sybil-resistant DHT (S/Kademlia)	BUILT	p2p/dht/
Signed, expiring .hidra service records	BUILT	p2p/dht/descriptor.rs
Exit anonymity for the browser	VIA TOR TODAY	bundled Tor
Independent external security audit	NOT YET	§11

Architecture

Two programs, each doing one job. The browser is what a person opens. The engine carries the network.

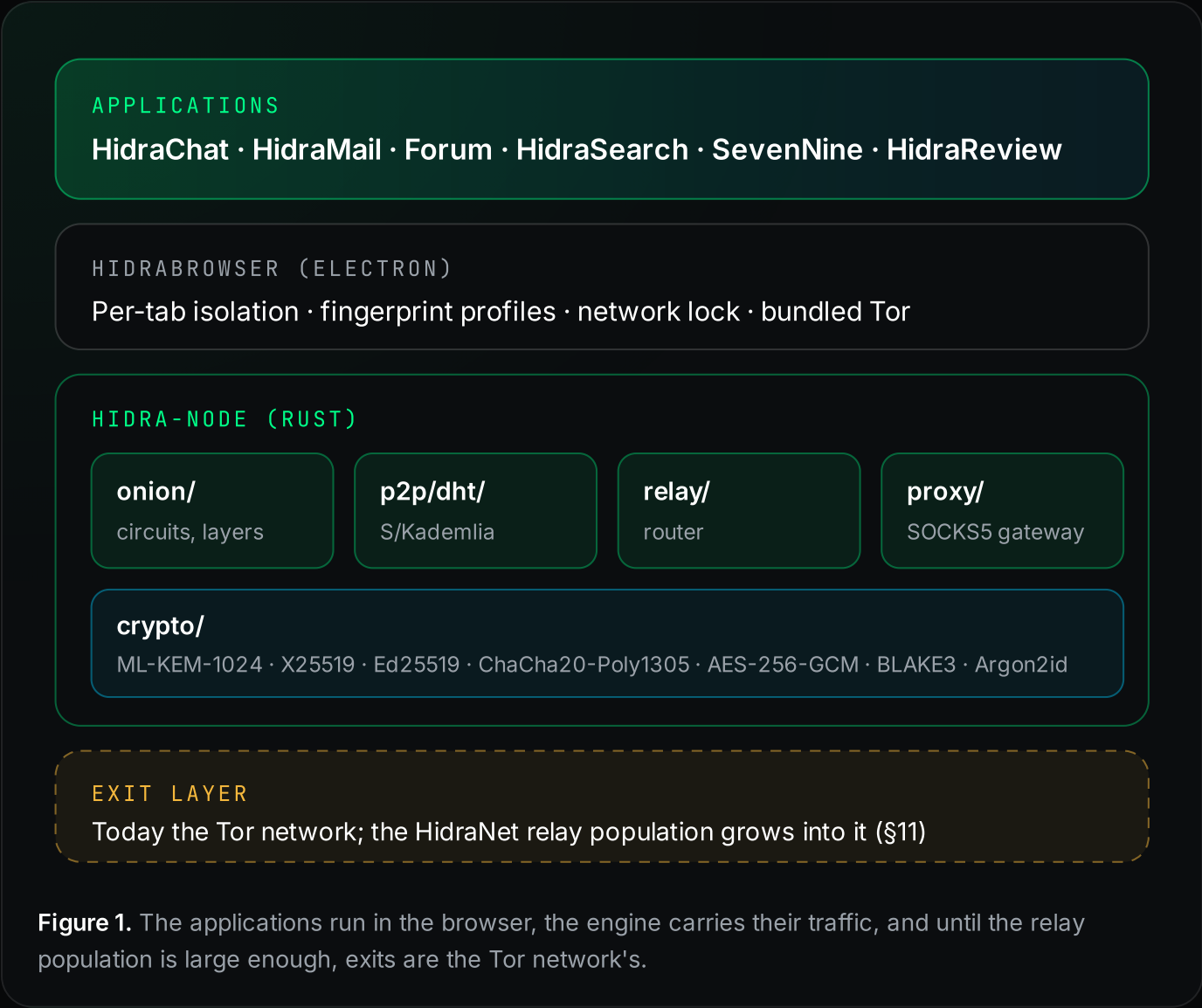


Figure 1. The applications run in the browser, the engine carries their traffic, and until the relay population is large enough, exits are the Tor network's.

COMPONENT	LANGUAGE	JOB
hidra-node	Rust 2024	Circuits, DHT, relay, SOCKS5 gateway, the apps' local servers
hidra-browser	JavaScript, Electron	Interface, tab isolation, fingerprint profiles, Tor management

COMPONENT	LANGUAGE	JOB
Applications	JavaScript, Web Crypto; Rust	Encryption on the device; the network carries ciphertext

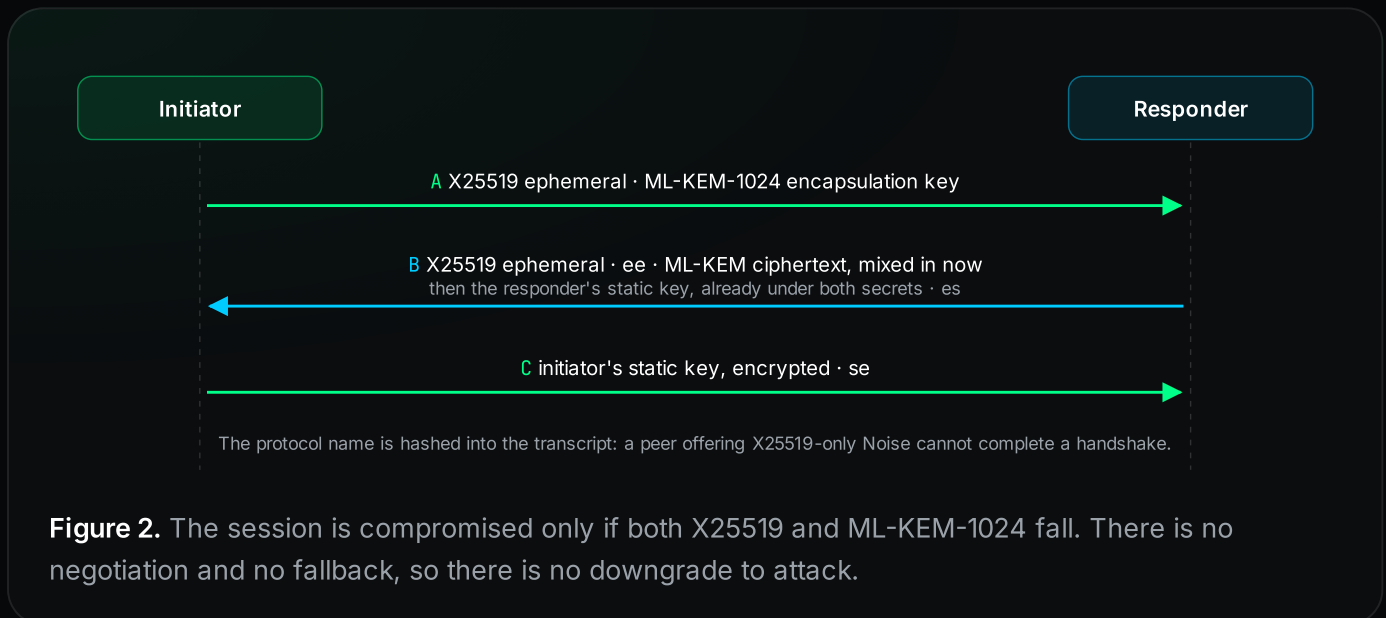
Post-quantum transport

Traffic recorded today can be stored and decrypted the day a cryptographically relevant quantum computer exists. The only defence that works against *harvest now, decrypt later* is a key exchange that is already quantum-resistant when the recording is made.

HidraNet's is.

The hybrid is the transport handshake itself, not a layer beside it:

`Noise_XXhfs_25519+MLKEM1024_ChaChaPoLy_BLAKE3`. The first message carries an ephemeral ML-KEM-1024 encapsulation key next to the X25519 ephemeral; the second carries the ciphertext, and the shared secret is mixed into the chaining key right after `ee`, before the responder's static key is encrypted. An adversary with a recording and a quantum computer breaks the X25519 half and still cannot read the static key, let alone the traffic.



Primitives in use

FUNCTION	ALGORITHM	STANDARD
Post-quantum key encapsulation	ML-KEM-1024 (<code>ml-kem</code>)	NIST FIPS 203
Classical key agreement	X25519	RFC 7748
Signatures	Ed25519	RFC 8032

FUNCTION	ALGORITHM	STANDARD
Authenticated encryption	ChaCha20-Poly1305, AES-256-GCM, XChaCha20-Poly1305	RFC 8439
Hashing	BLAKE3, SHA-256	
Password-based derivation	Argon2id	RFC 9106

i Why ML-KEM and not Kyber. The engine used `pqc_kyber` until an audit of the dependency tree found two unfixable advisories: an AVX2 backend that skipped implicit rejection, handing a chosen-ciphertext attacker the decapsulation key, and KyberSlash timing leaks. The crate was unmaintained, so it was replaced with the FIPS 203 implementation rather than pinned. `cargo audit` runs in CI on every change.

Onion routing

Traffic crosses a circuit of three relays. The client wraps each message in one layer per relay (ChaCha20-Poly1305, a fresh key per hop from that hop's handshake), and each relay removes its layer. In the design, a relay learns only the hop before it and the hop after it, so no single relay knows both who is talking and what they are talking to.

The circuit builder does not deliver that yet. Today the client opens a direct connection and a separate handshake to each of the three relays, with the same client static key on every hop. So every relay on the circuit sees the client's address, and the shared key lets the three relays link their views of the same client. The layers still keep the payload from all but the last relay, but the unlinkability the design promises is not there. The fix is telescoping construction, as in Tor: the client connects only to the entry and extends through it, hop by hop, with a fresh ephemeral key for each hop. It is the first item on the roadmap (§11).

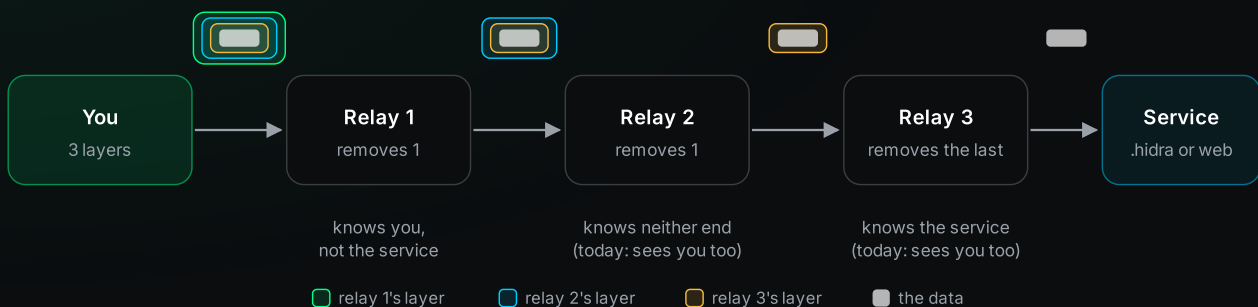


Figure 3. The layering, and what each relay should learn. Until circuits are built by extending through the entry, every relay also sees the client's address. In HidraBrowser today, web traffic leaves through Tor. HidraNet's own circuits carry `.hidra` traffic, and are what the volunteer relay population grows into.

⚠ Not yet. Circuits are not yet extended through the entry, as described above. Cells are not padded to a fixed size on the wire and there is no cover traffic, so the size and timing of a page load cross the network largely intact. The padding code exists and is tested, and is not yet on the data path. Circuit construction does not yet pin each relay's static key against a published one; announcing that key through the DHT and checking it is on the roadmap.

A DHT that resists Sybils

Peers and services are found through a Kademlia DHT, with no central server. A plain Kademlia has a well-known weakness: node IDs are free, so one operator can mint thousands of them right next to a service's key, own the set of nodes closest to it, and hide the service or point it somewhere dead. HydraNet puts the S/Kademlia defences on the wire.

IDs cost work, bound to a key

A node ID is the first 20 bytes of `BLAKE3("hidra/node-id/v1" || public key || nonce)`, valid only when the rest of that hash starts with 24 zero bits. The ID is unknown until the work is done, so closeness cannot be tested for free.

Signed, and bound to the signer

Every DHT message is signed with Ed25519. The sender's key must be the signing key and its ID the one that key and nonce derive. Every node named in an answer is checked the same way before it is asked anything.

Seated only where it answers

A node that writes first is seated after it answers a fresh ping at the address it wrote from. A signed packet replayed from elsewhere seats nothing. Checks are budgeted per network and backed off per host.

Three disjoint paths

Lookups are dealt to three paths that never ask the same node, seeded one network at a time. Service lookups walk until the 60 closest nodes they have seen have answered, within fifteen seconds.

One network, a few seats

At most two nodes from one IPv4 /24 (or IPv6 /48) per bucket and ten per table, counted at the address the node answered from, not the one it claims.

Records signed by the service

A `.hidra` descriptor carries the service's own signature, expires an hour after signing and is republished every fifteen minutes. A node in the path can withhold a record; it cannot forge one.

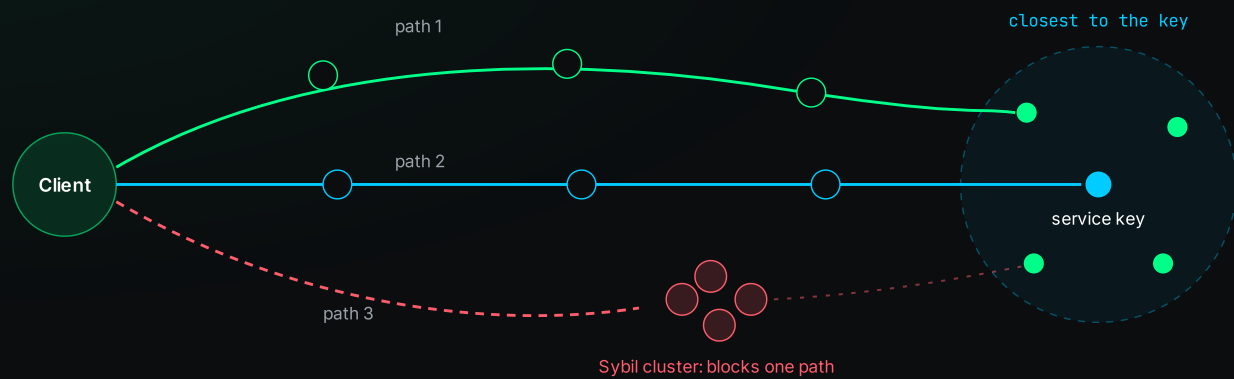


Figure 4. Paths never share a node, and each keeps its own closest responders. A cluster closer to a service than every honest node has to win on every path before a lookup stops reaching honest storers.

What an attack costs

Landing one identity among the $K = 20$ closest to a key in a network of N nodes costs about $(N / 20) \times 2^{24}$ hashes; the whole closest set, twenty times that. On one server core, one identity takes about 1.2 seconds. Proof of work alone is therefore cheap for a patient attacker, and it is not what carries the weight: hiding a service also takes addresses in more than sixty distinct networks, seated in honest tables, winning on all three paths.

PARAMETER	VALUE
ID puzzle difficulty	24 leading zero bits
Disjoint lookup paths	3
Service lookup width	60 closest nodes
Seats per /24 network	2 per bucket, 10 per table
Descriptor lifetime	1 hour after signing; republished every 15 minutes
New services one node may introduce	32 per hour

PARAMETER	VALUE
Wire format	HDT2, incompatible with earlier nodes on purpose



What this does not do. An operator with enough hardware and addresses in many networks can still hide a service, from some clients or from all; it cannot forge one. A client whose every path starts at attacker nodes can be misled. DHT queries still leave from the user's own address; routing them through a circuit is on the roadmap.

Each defence is covered by a test that fails when the defence is removed: 15 DHT integration tests, attack scenarios included, run against real sockets.

6

.hidra services

A **.hidra** address is the 160-bit BLAKE3 hash of the service's public key. The name is the key: there is no certificate authority to trust and no registrar who can take it away. Resolution is checked by the client itself: a descriptor that the service's key did not sign is refused, and the client keeps the highest revision it can verify. A descriptor names up to eight introduction points, so the host's own address is never published. SevenNine builds and hosts the sites; deleting one withdraws its record, which is gone from the network an hour after it was last signed.

Applications

Each application encrypts on the device. The network carries ciphertext and has no key to read it with.

APP	HOW IT PROTECTS YOU	NOT YET
HidraMail	Self-certifying @hidra addresses derived from two keys: ECDH P-256 for encryption (ECIES, AES-256-GCM, a fresh ephemeral key per message) and ECDSA P-256 for signatures, so the recipient can prove who sent a message. The client refuses a key that does not hash to the address.	Delivery between nodes connects directly to the recipient; delivery through a circuit is open.
HidraChat	Rooms sealed with a key derived from a shared password. The engine-served chat uses Argon2id (64 MiB, 3 passes) and ChaCha20-Poly1305.	The browser's chat page still derives its key with PBKDF2 and a fixed salt; moving it to Argon2id is open. Messages travel through public MQTT brokers: content is encrypted, timing is visible to the broker.
Forum	Public discussion on the network.	Same transport note as chat.
HidraSearch	Meta-search through privacy-respecting engines, no query log.	
SevenNine	Builds and hosts .hidra sites; publishing signs the site's record.	
HidraReview	Reviews of local businesses, each with its own .hidra address.	

Protecting the machine

- **Local services cannot be reached by web pages.** Everything listens on loopback and screens `Host` and `Origin`, so a page cannot reach it by DNS rebinding.
- **Keys at rest.** Secret keys are created readable by the owner only. With a passphrase set, they are sealed with XChaCha20-Poly1305 under a key from Argon2id.
- **The browser's own secrets** live in a vault on the operating system's keychain, one compartment per app.
- **Logs that cannot betray.** Peer addresses, destinations, mail addresses, nicknames and payloads are scrubbed before they reach a log; a test fails if any log line would write one.
- **Fingerprinting.** Each tab gets one coherent profile (system, GPU, screen, timezone, language chosen together), deterministic canvas noise, masked WebGL renderer, and no client-hint headers. WebRTC is held to the proxy in the network stack. The profile is injected when the page's DOM is ready, not yet at document start; that is on the roadmap.
- **Downloads you can check.** The SHA-256 of every package is published beside it, in `SHA256SUMS.txt`.

Threat model

A threat model that does not say what it does not protect against is advertising, not engineering.

Protected against

ADVERSARY	WHAT THEY GET
Your local network or ISP	Encrypted traffic to an entry point; neither destination nor content
The site you visit	An exit's address, not yours; a fingerprint that changes per tab
One relay in the middle	By design, the previous and the next hop only. Today it also sees the client's address (§4)
Someone recording today for a quantum computer	Nothing: the handshake needs ML-KEM to fall too
A message transport operator	Ciphertext without a key
A Sybil operator in the DHT	The power to withhold a record, not to forge one
A web page attacking local services	A refusal: loopback services screen Host and Origin

Not protected against

THREAT	WHY
An adversary watching both ends	Timing correlation defeats every low-latency network, Tor included
A small network	Anonymity is a crowd; a few hundred users is a small one

THREAT	WHY
Relays comparing notes	Circuits are not yet telescoped: each relay sees the client's address and the same client key (§4)
Traffic shape	No fixed-size cells on the wire yet and no cover traffic
Relays run by one party	No directory authorities or consensus yet; the largest architectural gap
A malicious exit	It sees what is not end-to-end encrypted. Use HTTPS
A compromised device	The keys are on it
What you publish yourself	No network can hide what a person reveals



If being identified would put you in danger, use Tor. HydraNet is young and has not been audited by anyone outside the project. The browser ships with Tor built in for exactly this reason.

10

Verify it yourself

Nothing in this paper needs to be taken on trust. From a checkout of the repository:

```

cargo test --manifest-path hidra-node/Cargo.toml --no-fail-fast    # 243 tests
cargo test --manifest-path hidra-node/Cargo.toml --test dht_discovery  # the DHT attacks
cargo test --manifest-path hidra-node/Cargo.toml --test no_addresses_in_logs
node hidra-browser/test/anonymity-claims.js                        # every badge is backed by code

```

SUITE	RESULT, OCTOBER 2026
Engine (<code>hidra-node</code>)	243 tests passing, 15 of them DHT attack scenarios
Continuous integration	Format, lint, tests and <code>cargo audit</code> on every change

Roadmap

Public, with its state, because a security project's credibility is measured by what it says it will fix and then fixes.

ITEM	STATE
Network kill switch in the browser	DONE
Third-party libraries served locally, no CDN	DONE
WebRTC held to the proxy	DONE
Presence beacon without location or room	DONE
Keys sealed at rest under a passphrase	DONE
Sender signatures in HydraMail	DONE
S/Kademlia defences in the DHT	DONE
Telescoping circuits: connect to the entry only, extend through it, fresh ephemeral key per hop	OPEN, HIGH
Pin each relay's static key in circuit construction	OPEN, HIGH
DHT queries through a circuit	OPEN, HIGH
Message transport on HydraNet itself instead of public MQTT brokers	OPEN, HIGH
Fingerprint profile at document start, in every frame	OPEN, HIGH
Argon2id in the browser's chat page	OPEN, MEDIUM
Fixed-size cells and cover traffic on the data path	OPEN, MEDIUM

ITEM	STATE
Volunteer relay programme	NEXT
Independent external security audit, report published	BEFORE BROAD RELEASE

The audit is a condition for a broad public release of the browser. We do not announce a privacy tool to a wide audience before somebody outside has tried to break it.

12

Licence and governance

All of it is under the MIT licence. The protocol is a common good: anyone may implement it, audit it, fork it and run nodes. The company develops software and services and **does not operate exit nodes**; the network is meant to be run by volunteers, as Tor's is.

13

References

- NIST FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)
- Perrin, The Noise Protocol Framework; and the hybrid forward secrecy (hfs) extension
- Dingledine, Mathewson, Syverson, Tor: The Second-Generation Onion Router
- Maymounkov, Mazières, Kademlia: A Peer-to-peer Information System Based on the XOR Metric
- Baumgart, Mies, S/Kademlia: A Practicable Approach Towards Secure Key-Based Routing
- RFC 7748 (X25519), RFC 8032 (Ed25519), RFC 8439 (ChaCha20-Poly1305), RFC 9106 (Argon2)