WHITE PAPER

v1.0.22

# Warren

A no-log VPN built on QUIC: non-custodial identity, resistance to censorship, and port forwarding that we thought was doomed

| | |
|---|---|
| DATE | 14 July 2026 |
| SCOPE | Motivations, stack architecture, the WarrenGuard engine, network, identity, payment, applications, threat model, limits and trajectory |
| STATUS | Public |
| VERSION | v1.0.22 |
| AUTHOR | Warren Team |

# Summary

Warren is a consumer no-log VPN whose architecture is based on a radical decision: to make the QUIC protocol, and not a dedicated tunneling component, the foundation of the entire stack. From this choice follows the rest. The traffic blends in with regular HTTP/3, giving native rather than bolted-on censorship resistance. The user's identity is a public key derived locally from a recovery phrase, without an account or email address: the server holds no secrets and a theft of its database does not compromise any access. The exit servers operate on an immutable image, their execution state only lives in RAM, and there is no data partition: a physical seizure yields no activity history.

Confidentiality has changed in nature. Data collection has become a permanent and inexpensive infrastructure: what is captured is preserved, and remains usable years later, under other laws and by other actors than those of the moment. Faced with a memory of such scope, individual prudence no longer protects much as long as the data continues to exist somewhere. The only protection that lasts over time is the absence of data, and this is the principle that governs Warren: making sensitive data absent rather than promising its proper use.

Three beliefs guide the project. The first: no-log only has value if it is verifiable, not just promised. The second: port forwarding, which the main market players abandoned in 2023 under pressure from abuse, remains a legitimate feature: its mechanism must be hardened, it must stay open to any lawful use, and reported abuse must be dealt with. The third: each security guarantee must hold by construction, without a check box that the user would have to find. These protections form a unique offer, without paid tiers: what varies from one defense to another is its default state, adjusted according to its performance cost.

This document describes the entire stack, from the cryptographic engine to the control plane, remaining at the architectural level. It shows by what concrete mechanisms each guarantee holds, and where the boundary of what Warren protects is located.

# Contents

**Part I. Why Warren**

1. No-log, and why words are not enough
2. May 2023: when port forwarding became indefensible
3. Obfuscation as a condition of existence
4. Three properties, one conviction

**Part II. The genesis of the stack**

5. Why not WireGuard
6. The detour through Iroh, the pivot to QUIC
7. Why fork Mullvad
8. The QUIC bet, and its price

**Part III. WarrenGuard, the engine**

9. WarrenGuard: One Core, All Platforms
10. The QUIC datapath
11. Quinn’s fork
12. Native obfuscation
13. Multi-hop and blind relay
14. Defense against traffic analysis
15. Port forwarding, hardened
16. Kill-switch and fail-closed
17. Memory security and secrets hygiene
18. Wire format as a source of truth

**Part IV. The network**

19. Exit servers: no history
20. Anatomy of an exit server
21. The blind relay and the signed directory
22. The control plane
23. No-log, verified by the machine
24. Operate the fleet without breaking no-log
25. DNS blocking: opt-in, uniform, public, without hijacking

**Part V. Identity, payment and applications**

26. Identity: a wallet, not an account
27. Prove your subscription without an account
28. Payment, unrelated to usage
29. Anonymous session credentials
30. Applications: two paths, the same requirement for truth
31. The forum, without email or IP address

**Part VI. Safety and limits**

32. Synthetic Threat Model
33. Trajectory

**Conclusion**

**Appendix A. Glossary**

**Appendix B. References**

# Part I. Why Warren

Before architecture, reasons. What a no-log promise is really worth, what the sacrifice of port forwarding in 2023 says about the business, and why stealth is a condition of existence.

# 1. No-log, and why words are not enough

A consumer VPN sells a simple promise: your ISP, the public networks you use, and the sites you visit can no longer track you. This promise is based entirely on a rarely examined assumption: that the VPN provider keeps no trace. The entire industry relies on this word, “no-log”, and almost all players translate it into a confidentiality policy that no one can verify.

The weakness is structural. A policy is a text; a server is a system. Nothing requires the latter to respect the former, and the user has no way of observing what is actually happening inside an exit server. The VPN market is a market of declarative trust, and a market of declarative trust tends towards the lowest bidder: sponsored rankings, convenience audits, unverifiable promises.

The only way out of this trap is to replace trust with verification, and above all to ensure that sensitive data **does not exist**. Data that has never been written cannot be seized, requisitioned, leaked or sold. The recent history of the sector provides the clearest illustration of this: on April 18, 2023, at least six police officers from the Swedish National Police showed up at Mullvad’s offices in Gothenburg, armed with a warrant to seize machines containing customer data. This data did not exist. After Mullvad demonstrated to them how the service works, and after consulting the prosecutor, the police left without taking anything. Data minimization had functioned as a technical defense **and** legal at the same time.

Mullvad is the actor who has most seriously cleared this ground: his public decisions constitute the best available documentation of the pitfalls of the profession, and part of what Warren is doing is learning from them.

Warren takes this idea to its full consequences. No reusable server-side identity. No activity log, no source IP address, no DNS query, no lasting association between payment and usage. Where a residual trace is technically inevitable, it is limited in time, encrypted at rest using a key not present in the database, and its lifespan is precisely fixed. A credible no-log is measured by what it makes impossible, and this document describes the mechanisms that guarantee it.

## 2. May 2023: when port forwarding became indefensible

Port forwarding allows a machine behind the VPN to accept incoming connections: sharing peer-to-peer instead of just downloading, hosting a service accessible from outside, obtaining an open NAT for online gaming. It's a feature that technical users have been asking for and has long distinguished good vendors from the rest.

She was abandoned. On May 29, 2023, Mullvad announced the complete removal of port forwarding, with existing ports ceasing to function on July 1, 2023. The rationale, in his own words: *"Individuals have frequently used this feature to host unwanted content and malicious services from open ports on our servers. This has led to law enforcement contacts, blacklisting of our IPs, and termination of our contracts by hosts. »* One month later, on June 29, 2023, IVPN followed suit, withdrawing completely by September 30, observing that the influx of customers from another supplier, following a similar policy change, had increased the risk for them. The abuse always focuses on the last provider holding the door open.

The dilemma is real: the cost of abusing a feature can exceed its value, becoming a matter of survival for the infrastructure. But the market response, outright suppression, punishes the overwhelming majority of legitimate uses to protect against a harmful minority. Warren is making the opposite bet, by documenting it: we restore port forwarding, we **harden the mechanism** against technical abuse and deanonymization leaks, we keep it open to **any lawful use**, and we **act on reported abuse**: a reported port is closed, and the account that held it receives a warning. Section 15 describes the mechanism; section 32, limits.

A VPN should serve its users rather than protect itself from them, and the right response to abuse combines technical and legal measures instead of cutting functionality.

# 3. Obfuscation as a condition of existence

Obfuscation is associated with state censorship: Iran, China, Russia carry out deep traffic inspection (DPI) to detect and block VPN protocols. It's true, but it's reductive. The blocking of UDP, or that of an overly recognizable protocol signature, happens well before the state apparatus: in the Wi-Fi of a café, a train, a hotel, a company. A VPN whose protocol can be identified at the first packet is a VPN which falls on a significant part of everyday networks.

Most tunnel protocols expose a trivial signature. The WireGuard handshake, for example, fits into a fixed-size packet with a characteristic-type byte: filtering equipment recognizes it effortlessly. Making such a protocol stealthy requires grafting a layer of obfuscation on top, to be maintained separately, activated after the fact, often reserved for users who think to look for it.

We wanted stealth to be a property of the transport itself. This is the underlying reason for choosing QUIC (sections 5 to 8): a QUIC tunnel resembles, on the wire, a browser that loads a site in HTTP/3. A censor who wanted to block it would have to block HTTP/3, that is to say a major part of the modern web, with the collateral damage that this supposes. First-level obfuscation, that which erases the visible signals of the protocol, is at Warren **active by default**, without adjustment, for everyone. The heavier defenses, those which distort the very shape of the traffic, are paid for in throughput and latency: they are there, they are available to everyone, but they only turn on if the user requests them. This dividing line, between what is permanent because there is no compensation and what is optional because it has one, structures our entire approach to censorship (sections 12 and 14).

**NOTE · A TECHNICAL DIVIDING LINE, NEVER COMMERCIAL**

Warren has no tiers, no paid options, no features reserved for those who pay more. There is a single offering, and everything described in this document is part of it, from multi-hop to port forwarding to defense against traffic analysis. The only thing that changes from one feature to another is its default state, and the only criterion is the trade-off between protection and performance. What costs nothing in terms of flow runs constantly for everyone; what costs in performance remains a choice of the user and never weighs on the bill.

# 4. Three properties, one conviction

From these motivations arise three structuring properties, which sum up Warren before any technical detail.

**No reusable server-side identity.** The user is a public key that he generates himself, never an account that we hold. The server only knows public keys, useless alone, and single-use signatures. A theft of our database does not give access to any accounts, because there is nothing to steal that is a secret.

**No logs on exit servers.** Active sessions, port allocations, address translation tables only exist in RAM. A machine that is turned off or seized contains nothing about its users. The no-log is due to the system itself, designed not to be able to log sustainably.

**Port forwarding restored.** A first-class feature, with a hardened mechanism, open to any lawful use, and rules against abuse.

The underlying belief is that security and privacy should be **construction properties**. A kill switch protects because a socket is dead, not because a firewall rule has had time to install. A no-log is guaranteed because the data is never written, not because a script purges it later. An identity carries no name because the protocol does not transport one, not because we promised not to look at it. Everything else in the document declines this requirement.

# Part II. The genesis of the stack

How we arrive at a VPN built on QUIC: the protocols excluded, the costly detour, and what this choice entails.

# 5. Why not WireGuard

WireGuard is an excellent protocol: small, audited, fast, integrated into the Linux kernel. It was the starting point for our reflection.

Three reasons kept us away. First, a bare WireGuard tunnel gets blocked or throttled on networks that filter UDP or recognize its signature, and making it stealthy imposes an additional layer of obfuscation to maintain. Next, WireGuard does not offer any native path to the functionalities that are the heart of Warren: end-to-end encrypted multi-hop, in-band port forwarding, defense against traffic analysis. Each would have to be piggybacked on top, with its own control channel and maintenance, because WireGuard does not offer capacity negotiation, application control channels, or handshake extensibility to rely on. Finally, WireGuard's identity is based on a key specific to the tunnel, distinct from the user's cryptographic identity: two identities to synchronize, two correlation surfaces, where we wanted a single key.

These three reasons converge towards the same conclusion: the required properties are not grafted onto WireGuard, they must be carried by the transport itself.

## 6. The detour through Iroh, the pivot to QUIC

The reasoning led us to QUIC, the modern transport protocol on which HTTP/3 is based. The first attempt relied on Iroh, a peer-to-peer stack built on QUIC, attractive because it aligned node identity to a public key and promised NAT traversal and multipathing. In use, it mainly brought weight: thousands of unused lines for our case, an interface that moved every week, a class of unresolved NAT traversal bugs, and a long tail of transitive dependencies.

In May 2026, we made a clear pivot: abandon Iroh for Quinn, the reference Rust implementation of QUIC, without the peer-to-peer overlay. Its flagship functions, NAT crossing and multipath, were of no use in Warren's topology, where a client addresses a known exit server: it was dead weight for this product. The migration took a few days and produced measurable results: throughput variability greatly reduced, binaries reduced by more than half, dependency tree reduced from eighty crates to around twenty-five.

The rule we learned from it applies everywhere: prefer a modest and understood foundation to an ambitious and opaque foundation. A dependency that we cannot control is a security debt as much as a technical debt.

# 7. Why fork Mullvad

Building a production-quality VPN client on three desktop systems and two mobile systems takes years of work: kill-switch management at the firewall, DNS leak prevention, policy routing, cross-platform GUI, integration with iOS network extensions and Android VPN service. All of this already exists, matured by years of production, in Mullvad's open source application.

Our desktop and mobile application is a fork of this. We inherit the difficult and proven part (operating system control, anti-leak discipline) and we replace the tunneling component there: where the original customer mounts a WireGuard tunnel, ours mounts a QUIC WarrenGuard tunnel. We don't rewrite what works; the effort concerns our own value, transport and the control plane.

# 8. The QUIC bet, and its price

QUIC is not just faster transportation. It is the primitive from which the entire Warren stack derives, and it provides several properties that a VPN can directly exploit.

Its handshake combines connection establishment and TLS 1.3 negotiation in a single round trip. It integrates TLS 1.3 natively, which provides access to mutual authentication by raw public keys (Raw Public Keys, RFC 7250) without having to operate a public key infrastructure. It carries unreliable datagrams (RFC 9221), ideal for carrying IP packets that do not need to be reordered by transport. It knows how to migrate a connection from one network to another, allowing you to switch from Wi-Fi to cellular without breaking the session. And, most importantly, it merges on the wire with HTTP/3.

QUIC alone is not a VPN. Warren adds a clean application layer on top: a handshake format, wallet identity, multi-hop encryption, defense against traffic analysis, obfuscation. QUIC is the foundation on which everything else is built.

This choice has a price. A user-space QUIC stack encrypts where WireGuard relies on the core, and QUIC adds packet number encryption, header protection, and acknowledgment machinery. This is the cost of HTTP/3 mimicry, i.e. the property that carries everything else in the stack. The engineering of the datapath is catching up: the tunnel supports several gigabits per second on suitable hardware (section 10).

**KEY POINT · FIVE PROPERTIES THAT WARREN HOLDS IN TRANSPORTATION**

The choice of QUIC is due to five properties on which Warren directly depends, which a WireGuard tunnel cannot support without being grafted onto it.

- **Native stealth.** Traffic merges with HTTP/3 on port 443, and this stealth is a permanent property of the transport. A WireGuard tunnel instead exposes a recognizable signature from the first packet, and only achieves the same stealth by receiving a separate obfuscation layer to maintain.
- **In-band control channel.** QUIC provides reliable flows, capacity negotiation and extensible handshake. Sealed multi-hop, port forwarding, traffic analysis defense, and session tokens travel in the same connection. WireGuard does not offer any of these support points, and each of these functions would require a separate channel.
- **One identity.** TLS 1.3 is integrated into the transport: mutual authentication by raw public keys, signed channel binding, certificate of coverage and post-quantum sealing result, all backed by the user's key. No key specific to the tunnel needs to be synchronized in parallel.
- **IP packets in datagrams.** Unreliable datagrams (RFC 9221) carry IP packets one for one, without unnecessary reordering, while reliable flows only carry the application handshake.
- **Connection migration.** A network change, from Wi-Fi to Ethernet or to cellular, is detected and the QUIC connection switches to the new path without a new handshake, the time of a round trip for revalidation. The connection identifier, which replaces the address quadruplet, is renewed on the new path, so an observer does not connect the two paths by this identifier.

# Part III. WarrenGuard, the engine

The technical heart: a single datapath, in Rust, shared by all platforms, and the mechanisms that make it stealthy, waterproof and difficult to trap.

# 9. WarrenGuard: One Core, All Platforms

WarrenGuard is the engine of Warren: the software stack that implements VPN tunneling over QUIC, independent of anything specific to the Warren product. It is an open source component (AGPL license) designed to be generic: it carries no subscription policy, no signing key, no Warren directory, and leaves these decisions to the deployer. A third party could use it to operate their own VPN on QUIC.

This separation is a law of architecture, strictly applied. The stack reads in layers, and dependencies never move up.

**TECHNICAL DETAIL · THE FOUR LAYERS AND THE LAW OF DEPENDENCE**

The generic engine (WarrenGuard) is on the base. Above, two layers independent of each other: the **Warren client** (identity, selection of servers, application facade) and the **Warren server** (the control plane, subscription application, private keys). At the top, the **product application**. The rule: the engine knows nothing about Warren; the client and the server depend on the engine, never the other way around; the client and the server never know each other directly; their only bridge is the network. This direction of dependency is structural: the engine lives in its own repository, compiles on its own, and does not reference any Warren code, so a reverse dependency would not compile. It ensures that sensitive code, the one that holds the keys and enforces policy, cannot leak into client code.

There is **a single implementation** of the datapath, in Rust, reused everywhere. The flagship application (the Mullvad fork) and the SDK family (Rust, Dart, TypeScript) consume the same native engine and the same QUIC fork. SDKs for other languages wrap this native datapath core rather than rewriting it. A direct consequence for censorship resistance: all Warren applications, regardless of their origin, emit exactly the same QUIC handshake signature. There is not a recognizable official application and distinguishable third-party applications; there is only one behavior on the thread.

**TECHNICAL DETAIL · QUIC AS A VPN TRANSPORT: WIDESPREAD USE, AND WARREN'S CHOICE**

Running a VPN through QUIC has become common, in two ways separate from Warren's. The first is the coating: the tunnel remains WireGuard, wrapped in QUIC to hide when the network blocks. Transport remains WireGuard, and the envelope is paid for in functions. Mullvad, who added this QUIC obfuscation in 2025, requires giving up multi-hop and defense against traffic analysis to enable it.

The second is account-based relaying. Apple iCloud Private Relay since 2021, Cloudflare WARP switched to MASK by default at the end of 2024, and Google IP Protection in Chrome tunnel natively over QUIC, but remain relays linked to a platform account, single-hop for WARP, limited to the browser for the other two, without identity by key, without port forwarding, without defense against traffic analysis. Their MASK handshake leaves the domain name readable in the first packet, which Warren's fragmentation defeats (section 12).

Warren makes QUIC the foundation of a complete VPN on a single datapath: non-custodial identity by key, multi-hop sealed to the egress key, hardened port forwarding, and defense against traffic analysis that runs on the QUIC tunnel itself.

The boundary between open and closed follows a simple line. All code that the user executes, the engine and the client kit, is public and auditable by anyone. What remains is the control plane, the signature keys and the fleet of exit servers, that is to say the operation of the network.

# 10. The QUIC datapath

The WarrenGuard transport relies on Quinn, the Rust reference implementation of QUIC, of which Warren maintains a targeted fork, detailed in section 11. This section describes the datapath it carries: session establishment, traffic structure, congestion control and MTU management.

**Handshake and authentication.** The tunnel uses TLS 1.3 exclusively; TLS 1.2 is refused. 0-RTT is disabled on both sides, deliberately: reusing a session secret would allow tunneled IP packets to be replayed, an unacceptable flaw for a VPN. The generic engine authenticates with Ed25519 raw public keys, without X.509 certificate or PKI chain. The user's identity is not presented during the TLS handshake itself: since version 5 of the protocol, the client signs the cryptographic channel link (a value derived from the TLS session, RFC 5705) to prove possession of its key, and since version 6, the egress server does the same in return. There is therefore no request for a client certificate on the wire, a signal which would betray the protocol.

In production, exit servers present a valid X.509 certificate for an innocuous coverage domain, making the handshake indistinguishable from a regular HTTP/3 connection to an observer. A server configured for a censored market refuses to start without this certificate rather than silently falling back on the raw keys: the stealth posture is a startup invariant.

**Traffic structure.** QUIC's reliable flows only transport the application handshake (negotiation of the tunnel address, MTU, functionalities). The user's IP packets travel in QUIC's unreliable datagrams one-for-one without unnecessary reordering. The encoding of control messages is strict: an unknown field causes a rejection, there is no approximate backward compatibility. Protocol versions evolve in clear steps, never by mutation of an existing format.

**Congestion control and MTU.** The default congestion controller is BBR, chosen because it holds up much better than traditional algorithms in the presence of losses. With two percent loss injected onto a stream, a typical controller collapses to a few megabits per second where BBR maintains several hundred. Active queue management, applied per flow to the tunnel datagrams, limits the latency when the load increases: rather than letting a buffer swell and the delay increase, excess packets are discarded early, and the transmission window is sized on the actually available flow. The tunnel MTU is conservatively initialized (1280 bytes) to avoid fragmentation black holes on intercontinental paths, then dynamically probed upwards.

**TECHNICAL DETAIL · DYNAMIC ADAPTATION OF THE MTU**

A tunnel crosses paths of varying sizes, and certain networks (encapsulated Wi-Fi, satellite link, railway network) impose a reduced MTU below the usual value. Warren adapts to it on two layers, without ever resizing the tunnel interface, which remains at 1280 bytes.

The first layer is the discovery of the MTU of the outer path. Starting from the conservative threshold of 1280 bytes, it probes upwards when the native path allows it and never drops below 1200 bytes, the minimum that QUIC guarantees end-to-end. On TCP fallback, where the stack can fragment on its own, this discovery is disabled and the internal budget is capped at 1100 bytes, the size that passes through this transport without falling into a black hole.

The second layer directly adapts the useful space offered to the user's packages, based on this living budget. For TCP flows, it rewrites the MSS option of connection opening packets, so that both ends size themselves to fit in the tunnel, transparently and without relying on ICMP. This same adjustment is applied on the exit server side, which fixes the collapsed paths even for a client that has not yet been updated. For other flows, a packet that is too large is discarded and its sender receives a real ICMP message ("fragmentation required" in IPv4, "Packet Too Big" in IPv6) bearing the right size, so that its own MTU discovery can adjust. Finally, the application indicates, for purely informative purposes, when the effective MTU falls below its nominal value; the MSS adjustment and the synthetic ICMP message do the job whether this indication is displayed or not.

**Measured performance.** On a bare-metal 10 GbE server (AMD EPYC Zen4 processor), between two machines in the same data center, a single tunnel stably supports on the order of 5.5 to 7 Gbps, or 55 to 70 percent of the line throughput, without saturating the processor on either side. This is the capacity of the datapath, and it far exceeds what real use requires. In terms of load, a server holds more than five thousand simultaneous sessions at less than ten percent CPU load: the limiting factor of an exit server is the bandwidth of its link, a choice of network sizing rather than a stack limit.

# 11. Quinn’s fork

QUIC is a large protocol, and rewriting an entire stack would be as much an engineering pitfall as it would be a security regression. Warren therefore starts from Quinn, the reference Rust implementation of QUIC, and diverges it with a small number of targeted modifications. The database faithfully follows the official repository: the fork is aligned with the current published version of Quinn and resynchronized when the upstream progresses, each divergence being isolated into a documented patch that can be read, applied or removed separately.

**TECHNICAL DETAIL · THE FORK'S ANTI-REGRESSION SAFEGUARD**

Silently going back from the fork to the upstream version would lose the performance gain and obfuscation. This risk is neutralized by a compilation safeguard: the code calls settings that only exist in the fork, so that a return to the upstream version causes an immediate compilation error instead of a discrete regression. The fork cannot be abandoned by accident, only by explicit decision. Each new version of the fork is also validated by an A/B comparative test bench before being adopted.

The differences fall into three families.

**Handshake obfuscation.** The fork adds two settings that govern the shaping of the first exchange: a padding floor on the initial packet, and a cap on the size of the first fragment of the TLS handshake, so that the handshake spans at least two UDP datagrams. This is what defeats the domain name extractor of the Great Chinese Firewall (section 12), according to the same idea as the anti-ossification protection of the Chrome browser. These settings are inert by default upstream; the motor activates them.

**Datapath performance.** Three modifications, with no effect on the visible protocol. Batch broadcast is enlarged, to push more datagrams per system call. Socket buffers are sized automatically upon creation, which avoids losses above the gigabit that the small default size of Linux causes, and in the process fills a gap upstream on Windows. A batch emission path specific to Apple platforms leverages their batch processing system calls.

**Congestion control and latency.** It’s the most substantial family, and it corrects real flaws. Two of them were inherited from upstream in the BBR congestion controller: on a lightly used connection, which is the case of an idle tunnel, the congestion window could grow without limit, observed at half a gigabyte on production exit servers. The first fix is a single comparison term, aligned with the Chromium implementation; the second reestablishes the rule for admitting bandwidth measurements. On top of this, active queue management of the FQ-CoDel type replaces the default deep queue on the tunnel datagrams: it bounds the latency under load and fairly distributes the throughput between the internal streams multiplexed in the tunnel, at the cost of a fraction of the through-

put in ideal conditions. A final adjustment reduces the size of the transmit buffer to the actual measured bandwidth-delay product of the path, instead of a constant calculated for the worst case, which avoids accumulating queue seconds on a slow link.

Not everything is forked. Several properties that one might believe to be specific to Warren relate to pure upstream, which the engine simply configures: neutralization of the spin bit, deactivation of 0-RTT, recovery after MTU black hole. The engine chooses the settings without touching the code.

Two of these fixes are intended to go back upstream: the fragmentation of the initial package and the repair of the BBR, both conforming to the specification and useful to any Quinn user. They are prepared for a proposal in advance. The rest concerns the setting specific to the case of a VPN tunnel, which the upstream would not activate by default, and therefore remains in the fork.

# 12. Native obfuscation

Warren's first-level obfuscation erases the signals by which inspection equipment would recognize a VPN protocol. It is active by default, for everyone, without adjustment, and its bandwidth cost is low.

Several signals are neutralized simultaneously. The application protocol announced in the handshake is only `h3`, that of HTTP/3. The server name (SNI) and the first part of the TLS opening message are fragmented over at least two UDP datagrams: this is a precise defense against a documented SNI extractor from the Great Chinese Firewall (work presented at USENIX Security 2025), which does not reassemble a message spread over several datagrams. The QUIC spin bit, which could serve as a fingerprint if it were constant, is neutralized. And an active sounder that tries to provoke the server to identify it only receives a generic transport closure, never a Warren-specific error code: the protocol-specific bytes are only sent after a first valid establishment message, which a sounder is incapable of producing.

These properties are automatically tested invariants: a modification that breaks them causes continuous integration to fail.

**TECHNICAL DETAIL · ENGINEERING REASONING, TRACED IN ARCHITECTURAL DECISIONS**

Each obfuscation choice is backed by a decision document (ADR) which sets out the threat and the arbitration. The selected adversary model explicitly includes an active state-class prober, not just passive inspection. A discovery was structuring: classic TLS mutual authentication allowed a client certificate request to pass over the wire, invisible to passive inspection but detectable by a prober; version 5 of the protocol removed it in favor of in-band signature. Another: the raw public key identity of the server was itself a detectable signal, upstream of any other defense, which motivated the move to an X.509 coverage certificate in version 6. The target bar is precise: indistinguishable from HTTP/3 in the face of a realistic adversary combining deep inspection, active probing and fingerprinting. Exact parity against a global adversary equipped with machine learning, which compares byte by byte of traffic to torrent volume, is the subject of the fingerprint project described in section 33.

The obfuscation doesn't stop at this first layer. When the tunnel is idle, rather than a regular beat of hold messages (a cadence that an observer easily relates), Warren emits covering traffic at varying intervals and sizes, which drowns out this rhythm. And the X.509 coverage certificate comes with a decoy HTTP service on the exit servers: a pollster querying the server like a website receives a regular web server response, instead of an error that would betray a VPN. These defenses rotate without the user having to search for them.

There remains the case of networks which not only filter a signature, but block the entire UDP: corporate Wi-Fi, captive portal, locked network. A purely QUIC tunnel over UDP would be unreachable there. Warren then switches the **same** QUIC tunnel to a TCP connection protected by TLS 1.3, to the **same** coverage domain and the same port 443. This fallback is armed by default: the client tries the UDP path and, if it does not respond very quickly, opens the TCP backup channel in parallel, without adjustment or long wait. On the wire, the fallback connection looks like a regular HTTPS session, just like the nominal path. Where UDP passes, it remains preferred, faster; where it is cut, the tunnel still holds.

# 13. Multi-hop and blind relay

Multi-hop passes traffic through two successive servers rather than one, so that no single point knows both who you are and what you do. The first node (the relay) sees your IP address but not your destination; the second (the exit) sees your destination but not your IP address.

We discarded the naive solution, a QUIC tunnel within another QUIC tunnel, because it stacks two congestion controls, two window managements and two MTU adjustments, with a performance loss of around half. Warren instead uses two independent QUIC connections, with the relay copying datagrams from one to the other without decrypting them.

The relay is **blind by construction**. The client seals the content of its traffic to the public key of the exit server, according to the HPKE hybrid encryption standard (RFC 9180). The relay never holds the decryption key from the exit server; it only handles opaque bytes. An attempt to divert a datagram to another exit server cleanly fails decryption, because the destination server identifier is cryptographically linked to the message: there is no exploitable “confused MP”.

**TECHNICAL DETAIL · PACKET KEYS ON AN UNRELIABLE CHANNEL**

HPKE encryption usually assumes an orderly and reliable flow. However, QUIC datagrams can be lost and arrive out of order. Rather than suffer sequence counter desynchronization, Warren derives a unique per-packet key from the HPKE context, indexed by epoch and sequence number, and encrypts with a fixed nonce: this is secure precisely because the key is unique for each packet, the same technique that QUIC employs internally. The associated data links each datagram to the identifier of the intended exit server, which prevents any hijacking.

A two-tier key infrastructure (one offline root, one operational key) signs the identity of servers, with version rollback protection. The sealing context of a session rotates frequently, every thirty minutes, so that the window during which the same key protects traffic remains short by construction.

The sealing itself is **post-quantum hybrid**. A post-quantum mechanism (ML-KEM) is added to the classic X25519 key exchange, without replacing it: confidentiality can never be worse than with X25519 alone, even if one of the two mechanisms proves weak one day. This combination responds to the adversary who archives encrypted traffic today to decrypt it the day a quantum computer allows it. The post-quantum choice is locked by a signature, never by an unauthenticated bit that an intermediate device could erase to force a fallback, and this is the default behavior.

# 14. Defense against traffic analysis

Encrypting the content does not hide the **shape** of the traffic. The packet sizes, the intervals between them, the bursts draw a fingerprint that machine learning techniques know how to link to a visited site, even through encryption. This is an angle of attack against which a simple encrypted tunnel does nothing.

Warren defends himself against this with Maybenot, the framework resulting from the work of Karlstad University and sponsored by Mullvad, who uses it under the name DAITA. The principle: probabilistic state machines which add padding (dummy packets) and delay to blur the shape. This defense is plugged into the QUIC datagram stream and negotiated at each session. Warren includes a set of defenses from the literature (constant flow padding, random noise, and variants). When a client requests it, the egress server draws one for that connection, announces it to the client, then applies it to the traffic it sends while the client applies the advertised defense to the traffic it sends. Because the draw is independent at each connection, two clients of the same egress server rarely exhibit the same form of traffic, which broadens the range of behaviors that an observer sees occurring.

This defense is paid for in bandwidth: scrambling the shape of traffic and maximizing throughput are two opposing objectives. Warren therefore makes it a user choice, on a single datapath and in a single offer: a switch, in the application, activates the defense against traffic analysis at the cost of throughput. Turning it off retains all the protocol obfuscation of section 12, which costs almost nothing and runs constantly. The only sharing criterion is the trade-off between protection and performance.

**KEY POINT · THE TRADING SIGNAL DOES NOT LIE**

A defense against traffic analysis that thinks it works without running is worse than no defense: the user adapts his behavior to absent protection. This is why activation is not a simple local setting. The client requests the defense, the exit server responds by announcing the machine it is applying, and the application only displays “active” on this confirmation. If the server does not grant it, the interface clearly shows it, on the dedicated page as well as on the connection screen, instead of suggesting that protection is not being implemented.

# 15. Port forwarding, hardened

Warren restores port forwarding via a full NAT-PMP server (RFC 6886) running on the egress server, accessible only from inside the tunnel. The contract is the one set out in section 2: any lawful use, a hardened mechanism, rules against abuse.

Hardening against technical abuse is applied in the code. Each tunnel address is limited to five simultaneous redirects. The allocation rate is capped at twelve new ports per sixty-second rolling window. A freed port waits for five minutes before another client can reuse it. The lifespan of a redirection ranges from sixty seconds to one hour, renewable. All of these limits fail in a closed manner: a request beyond the quota is explicitly refused. Mappings between a port and a session are ephemeral and exist only in memory, with one exception: abuse reported on a port that is still open.

Abuse handling follows the same discipline. When a port is opened, the application hands the exit server a tag containing the account key, encrypted under a key only the control plane holds; the server keeps it in memory while the port is open. When a third party reports abuse coming from a port, the port is closed, and the tag is decrypted only if the current holder already held the port at the time of the abuse. The account then receives a warning; three warnings within ninety days revoke it for twelve months. These records are the only ones that link an account to an event after a session ends. They are declared in the privacy policy, with their retention: warning ninety days, revocation twelve months, case file twelve months and then anonymised.

The crackdown on deanonymization targets a classic attack, known as “Port Fail”. If the input address and output address of a server are the same, an attacker subscribing to the same service can, via port forwarding, leak the real IP address of a victim whose traffic to that address bypasses the tunnel. The root cause is that the route exclusion to the VPN server is destination based.

**TECHNICAL DETAIL · CORRECT THE SOCKET, STAY IN SINGLE-IP**

Mullvad was immune to Port Fail as early as 2010, by separating the input address from the output address of its servers. Warren attacks the root cause rather than the symptom: instead of excluding from the tunnel *anything* that goes to the server address, which causes any packet to leak to that address, the exception is restricted **to only the tunnel socket itself**, by system primitives. This is the approach of WireGuard and the recommendation of the TunnelCrack work (USENIX 2023); it closes Port Fail and the TunnelCrack-ServerIP variant in the same gesture, without the cost of a second IP address, and Warren remains single-IP. As we control all of our applications, a client-side patch is sufficient where a generic VPN should patch on the server side, and the egress server also adds its own defense-in-depth barrier.

macOS requires special care, because several network interfaces often coexist (Wi-Fi and Ethernet, for example) and binding a socket to one of them can, in certain configurations, cause it to lose all exit routes. The socket link is therefore coupled with egress monitoring: if the linked socket really stops transmitting, the application switches itself to a dedicated route fallback, rather than leaving the traffic in a black hole. Protection self-corrects instead of depending on an ideal network configuration.

# 16. Kill-switch and fail-closed

A kill switch prevents traffic from leaking out of the tunnel when it falls. The major architectural lesson from documented industry incidents (TunnelVision, TunnelCrack) is that you should never rely on the routing table as a security mechanism: fail-closed must rely on a firewall that blocks anything that does not pass through the tunnel.

Warren makes it a construction property rather than a reaction. As long as the tunnel carries traffic, the OS firewall is in denial by default: the rule is set before the first packet, and nothing leaves the tunnel because nothing else is allowed to leave.

Failure of the software itself doesn't reopen anything. The blocking rules live in the kernel and survive the process: a crash leaves the host silent instead of leaking, then the service restarts itself, restarts in the blocked position, and reestablishes the tunnel without intervention. And because a protection that would sequester one's own machine would be an engineering fault, the application keeps an explicit output in all circumstances: a click, under the native elevation of the system, restores access to the Internet without the VPN, firewall and DNS restored.

For those who need it, **persistent kill-switch** takes the next step. Once armed, the blocking survives the stopping of the daemon, its crash and the restart of the machine: the failure may be total, the host remains silent until the user disarms the protection himself. Warren does not impose it by default: an imposed persistent block transforms the slightest software incident into a machine without a network, without recourse, including uninstalling. On the exit server side, where no human is there to disarm anything, the arbitration is reversed: their binaries are compiled in such a way that a fatal failure **interrupts** the execution without carrying out any cleaning.

On desktop systems in full tunnel mode, the OS firewall (nftables on Linux, pf on macOS, the equivalent on Windows) is denied by default, allowing only loopback, tunnel device, and daemon traffic to the egress server. The Port Fail fix is one step lower in routing: the tunnel socket itself is bound, instead of excepting a destination, which covers the nominal UDP path like the TCP fallback in section 12. DNS is pushed into the tunnel, with no path to the host resolver.

The level of guarantee differs depending on the mode, and Warren distinguishes between the two. The fail-closed applied by the OS firewall falls under privileged tunnel mode. The non-privileged proxy mode offers a fail-closed of another nature, structural but limited to the configured application (section 30). A promise such as "zero packets outside the tunnel" only makes sense under a specific level of guarantee: certain operating systems exempt themselves from application firewalls.

# 17. Memory security and secrets hygiene

The engine is written in Rust with the prohibition of `unsafe` code set at the compilation workshop level. Only three crates relax this ban, each for a specific and documented reason: the socket binding system call, the interface with the Windows network API, the opening of the privileged tunnel device. Everywhere else the code is without `unsafe`. This discipline eliminates entire classes of memory vulnerabilities by construction.

The default cryptography is entirely in Rust (`ring` library), without a C compilation chain in the current profile. Secret materials (seeds, signing keys, recovery phrases) are erased from memory upon release, and no type carrying a secret exposes any debug representation that would reveal it: at best the public address. The no-log discipline is applied down to error messages and logs: never a full public key, a source IP address, a nonce or a seed in plain text, on either side of the language boundary. An identifier, if a fragment is really necessary for diagnosis, is truncated.

# 18. Wire format as a source of truth

Warren's client protocol is implemented once in Rust, then re-exposed to other languages via SDKs. For a TypeScript or Dart kit to speak the exact same language as production servers, binary compatibility must be strict, byte for byte.

We guarantee this by **golden vectors**: a set of shared reference files which freeze each format (identity derivation, address, request signature, handshake frames, multi-hop frame, signed directory) to the bit. Each SDK replays these same files. A format is anchored by exact bytes rather than described by prose subject to interpretation. Modifying a vector means changing the wire format, which requires mounting a version of the diagram, never mutating the existing one. We never modify a vector to pass a test: a vector divergence is a real protocol regression. This rigor is what allows multiple implementations to coexist without ever drifting, and therefore all applications to present the same signature on the wire.

Protocol versioning and capacity negotiation follow the same logic. The functionalities are announced by a bit mask, the versions coexist properly during a migration, and a node which does not understand a new version clearly refuses it rather than attempting an approximate interpretation.

# Part IV. The network

What runs on the other side: servers that keep no history, a blind relay, and a control plane that knows as little as possible.

# 19. Exit servers: no history

An exit server is the most sensitive place in the entire system: it is there that the traffic becomes clear again and where, for the duration of a session, the user's real IP address and their destination coexist. The whole design aims for a seizure of this machine to yield no activity history.

Each production exit server runs on an immutable Alpine Linux image, installed by a tool that takes control of a bare metal server and rebuilds it into a read-only system. The root file system is a read-only compressed image (squashfs), layered with a write layer (an overlay) that only exists in RAM. The running system is disposable: a reboot wipes it completely.

**KEY POINT · WHAT PERSISTS, AND WHAT DOES NOT PERSIST**

There are **no data partitions**. The only thing that needs to survive a reboot is the node's identity (its signing key), and it survives not by a writable disk but because it is burned read-only into the image and rewritten identically with each update. Everything else (active sessions, port allocations, DNS cache, logs) lives in the memory layer and disappears on reboot. A machine turned off and then seized makes the operating system read-only and its own identity key, and nothing about the users. A machine turned on and entered renders, at worst, the living sessions present in memory at the moment of entry, never a history. The kernel does not write any core dump: when stopping a process on error, the system normally saves all of its memory in a file, which here would contain the addresses of the users and their sessions; this writing is disabled. The service logs live in RAM and never touch the disk.

The deployment of updates follows the same logic. The system uses two root slots (A and B). An update is done without interruption or restart, in two stages: first the hot replacement of the binary for the current process, then the reconstruction of the inactive slot so that it serves the same version at the next startup. A fallback mechanism in the initramfs automatically switches to the old slot if a new one fails to start. This immutability has a valuable corollary for verifiability: what rotates is exactly what was constructed, and any alteration requires reconstructing an image.

# 20. Anatomy of an exit server

The VPN plane of a production server fits on the single port **443**: the UDP socket which carries QUIC, and the TCP socket which completes the fallback of section 12. Behind them runs a unified splitter which is both the multi-hop relay and the egress terminator. Each incoming QUIC connection carries, in plain text, a routing label indicating the targeted exit server: if it is the node itself, it terminates the connection locally; otherwise, it retransmits it blindly to another node. This design folds the single jump and the double jump on the same datapath, in all modes. On this port, an exit server looks like an HTTP/3 server: 443, protocol `h3`, a plausible server name.

Several security mechanisms lie in the details that make or break a no-log VPN.

**Anti-correlation address translation.** The server hides user addresses behind its own, and it does so with a **completely random** source port allocation. The kernel does not preserve the ephemeral source port chosen by the client in the output quintuplet: a client which reuse a fixed source port would not be able to bring out a stable output port that an observer would connect to its sessions.

**No-log DNS resolver.** The exit server does not log any DNS queries, by design. A recursive resolver runs as a separate service, resolving names itself with query minimization (no third party sees the full query), its cache pinned to RAM so resolved names never hit disk, no logging module, core dumps disabled.

**Privilege model.** The exit server runs as a non-privileged user. The system capabilities it needs (administer the network, bind to port 443) are granted to it as a set of ambient capabilities (*ambient capabilities*) after relinquishing root privileges, never as capabilities attached to the binary file.

**TECHNICAL DETAIL · WHY AMBIENT CAPACITIES, NOT FILE CAPACITIES**

The kernel clears the ambient capability set when executing a binary that carries file capabilities; and on the read-only overlay file system, file capabilities apply unreliably. Attaching capacities to the binary therefore broke the link to port 443 intermittently. The ambient capabilities model, transmitted through privilege relinquishment, is both more secure (the binary carries no capabilities at rest) and more robust on this type of image.

# 21. The blind relay and the signed directory

The relay role (the first hop of multi-hop) consists of transmitting opaque ciphertext from the client to the chosen exit server, without ever holding the latter's decryption key. What the relay can see: the client's source IP address (it is the network peer), the clear routing label, and opaque bytes. What it can't see: the clear text between the client and the egress server, the destination, or the decrypted traffic. The HPKE seal is the true security boundary.

A node learns about the existence of the other nodes in the fleet through a **signed directory**, retrieved from an authenticated access point reserved for registered exit servers. Each directory entry is checked against the operational key before a relay agrees to dial to it: a compromised control plane can neither inject a malicious exit server nor let a simple subscriber key enumerate the addresses of exit servers.

The list of servers published to applications follows the same principle of minimization. It is signed, and it only reveals what an application needs to choose a server and compose it: the node identifier, its location (country and city, what the user sees in the list), a selection weight, its entry points and the capabilities it announces. It was deliberately slimmed down to no longer contain the roles of the nodes nor the **exit address**, as many levers as a censor would use to enumerate and block the fleet. An application learns where to enter, never where to exit.

# 22. The control plane

The control plane is the single HTTP service that exit servers and applications talk to. It manages subscriptions, vouchers, payments, the register of exit servers and their heartbeats, the multi-hop directory, the issuance of anonymous session tokens. It is never exposed directly: it is protected by a reverse proxy.

The knowledge asymmetry between the control plane and the exit servers is the core of the privacy model.

The control plane knows: which public key is subscribed, and until when. This is an unavoidable authentication fact, but without any activity history. He knows the accounting references of payments, the register of exit servers, aggregated statistics. It does not know which exit server a subscriber used, nor its destinations, outside a port-forwarding abuse case (section 15).

An egress server knows: the live sessions in memory, the tunnel address allocations, the traffic it sends out. None of this is written to disk.

The relay, finally, only sees the client's IP address and opaque bytes.

No single component holds the complete chain linking an identity to an activity. This is the property that Section 29 takes to its conclusion with anonymous session credentials.

# 23. No-log, verified by the machine

On the server side, no-log is a set of automatic checks.

Each service carries a test that runs through its own source code and **fails continuous integration** if a log line were to interpolate a full public key, client IP address, nonce, or secret. The no-log is thus guarded by the tools, not by human vigilance.

The reverse proxy filters its access logs to remove the client's IP address and all headers, keeping only the method, URL, status and duration; it disables logging entirely for URLs that might contain a secret. Where a service does not need the client address, the passed address header is pinned to a null value, so that no downstream component learns it; where the API needs it to cap throughput, it keeps it in memory, never logged.

What is stored in the database is inventoried in a retention matrix, itself guarded by a continuous integration test: a new durable table cannot be delivered without a line documenting what it contains, why, and for how long. The data deliberately never stored is explicit: traffic, DNS, connections, source IP addresses, bandwidth per user.

> **NOTE · TWO TABLES, ZERO ROWS**
>
> The schema contains two tables capable of logging activity: migrations create them, so they are present in the production database. Their writing is behind a disabled switch, and that switch is not armed. A listener who opened the database would see two empty tables. It is a verifiable property, and it is the strongest form that a no-log can take: not data that we promise not to use, but data that is never written.

# 24. Operate the fleet without breaking no-log

Updating a fleet of exit servers without downtime or vulnerability windows is an engineering problem in its own right. Warren solves it with a deployment control plane whose security property lies in the signature chain: the version catalog is signed offline, with protection against rollback (a monotonic counter) and against replay (an expiration date).

Deployment follows one state machine per node (drain, failover, check, return to service) and a strict rule: a single canary server, the least loaded, goes first; its health is compared to the rest of the fleet left intact, and the analyzer that renders the verdict is separated from the actuator that applies it. It is only once the canary has been validated that the rest follows. Draining is a true “make-before-break”: the server being retired reports its in-band state, applications exclude it from their selection, and the application supervisor establishes a new session before closing the old one. An update does not interrupt users logged in to it.

# 25. DNS blocking: opt-in, uniform, public, without hijacking

Warren offers DNS-level content blocking (ads, trackers, malware, and optional categories). Its design keeps it away from anything that would make a filter a surveillance or censorship tool.

It is **opt-in** and disabled by default. The user chooses the categories, and their choice travels encoded in the address of the resolver they use in the tunnel, without the server having to maintain a profile per user. It is **uniform and public**: the blocking lists are identical for everyone and published in an open repository, and no targeting by user is even expressible in the system, due to the lack of any user dimension in the resolver. A list change imposed under duress would appear in the public filing history. It is **hijack-free**: a blocked domain receives a response "this domain does not exist" (NXDOMAIN), and not a silent redirection to a server that would observe the attempt. And it's **logless**, like everything else.

# Part V. Identity, payment and applications

How to prove a subscription without an account, pay without being tied to its use, and what the applications really guarantee on each system.

# 26. Identity: a wallet, not an account

The identity of a Warren user is a pair of Ed25519 keys that he generates himself on his device, from a recovery phrase of twelve words in the BIP39 standard (the *mnemonic*, 128 bits of entropy). The public key, encoded in a readable address that begins with `wb`, is its identifier. The private key never leaves the device, where it is kept encrypted at rest by the operating system's secret store: keychain on macOS and iOS, without syncing to the cloud; DPAPI on Windows; hardware keystore on Android; machine sealed key on Linux.

There is no account, no email address, no password. This model is **non-custodial** in the cryptographic sense of the term, a property that neither Mullvad, nor IVPN, nor Proton offer to date.

**IN SHORT · WHAT "NON-CUSTODIAL" REALLY MEANS**

Take Mullvad's sixteen-digit account number, which is the best thing on the market for an email-free account: it's generated by the server, stored by it, and compared with each connection. It is, cryptographically, a bearer token that the provider holds, and a theft of its base would expose all accounts in the clear. At Warren, the key pair is generated locally, the private key never leaves the device, and the server only sees public keys (useless on their own) and single-use signatures. A theft of our infrastructure does not give access to any account, a signature cannot be replayed, and the user proves their identity without ever revealing their secret. The downside is that there is no recovery on the server side: losing the twelve words means losing the subscription. No one else has them, not even us.

The `wb` address borrows an encoding format from the world of blockchains (the SS58 format), but there is **no blockchain** in Warren, no smart contract, no subscription registered on a distributed ledger. The component that we call "contract" is a contract in the software sense, that of the exchange format between the client and the server; it's a library, not a token. Subscriptions live in a classic database, off-chain.

## 27. Prove your subscription without an account

Since there is no session or bearer token, how does a client prove to the server that it has the right to act? By a signature. Each request to the control plane carries four headers: the public key, an Ed25519 signature, a timestamp, and a nonce. The signed message covers the method, path, timestamp, nonce, and a body print; it does not contain the public key itself. The server verifies the signature, rejects a timestamp outside of a sixty-second window, and rejects a previously seen nonce using an anti-replay table.

A database theft yields, at best, all subscribed public keys and their expiration date: no email address, no name, no IP address, no log, no time series of activity per account. There is no way to reconstruct a private key, since none ever passes through the server.

Control of the number of devices is not a durable connection but a simultaneity limit: a ceiling of devices **connected at the same time** by subscription, applied via a register of short-lived leases which regenerate on their own. Nothing is permanently recorded; reinstalling costs no space. A random, memory-only device identifier is drawn on each run.

# 28. Payment, unrelated to usage

Paying often involves revealing a real identity (a credit card, an email). Warren separates payment from usage through an airlock, whatever the means of payment.

All payment verification is **self-hosted**: we verify the signature of each payment event ourselves, without going through a third-party intermediary who would see both the payment and the identity. Each provider (card, blockchains, mobile stores) is reduced to a neutral event carrying only an amount, a currency and an opaque reference: it is a type of data whose absence of personal data is verified at compilation.

The airlock mechanism is based on a voucher. The payment produces a voucher of which only the hash is stored; this voucher is then exchanged for a public key, the exchange itself requiring no authentication since the voucher is the only proof. When paying, the app pulls a random purchase ID; the payment provider only sees this identifier, never the user's public key; the payment site never sees the public key; and the control plane only learns the public key at the moment the voucher is exchanged.

**TECHNICAL DETAIL · THE JOINT AT REST, LIMITED AND HARDENED**

An at-rest join links a payment reference to a public key, so that a refund or dispute can effectively cut off access (without this link, a refunded voucher would still be usable). It is hardened in three ways. It is encrypted at rest under a key that is **not in the database**, so a raw theft of the database cannot read it. The line is deleted after twenty days (ninety for mobile payments). Timestamps are truncated to day and expiration rounded to the next midnight, so no duration reveals the second of a trade. Combined with the total absence of an IP address log, this join does not provide any usable correlation under normal operating conditions.

Card payment is in service; cryptocurrency payments rely on the same voucher, and each means of payment is offered only once it is actually in service. It is the airlock, not the means of payment, that separates the purchase from the usage. Automatic renewal is controlled by the customer, the card remaining with the payment provider and never with us.

# 29. Anonymous session credentials

The payment gate prevents a real identity from being linked to a public key. There remains one last join to break: that between the subscribed public key and the use of the tunnel. As long as the exit server sees the user's public key at login time, a link exists, even if it is not logged.

Warren breaks it with anonymous session credentials based on Privacy Pass (RSA blind signature tokens, RFC 9578 and RFC 9474 standards). The principle: the control plane issues tokens to a subscribed public key, but these tokens are blinded, so that the finalized token that the user spends is cryptographically impossible to link to the issuance. Issuance keys change every hour, derived from a seed, which ensures that a token is only spendable in the hour for which it was signed.

**In normal operation, no Warren component records a link between a subscribed public key and an exit server, a session, or a destination.** The control plane only sees the emission, authenticated by the wallet. The egress server only sees an anonymous token and a single-use serial number. The relay only sees the IP address and some ciphertext. One exception, described in section 15: abuse reported on a port that is still open records a warning on the account that holds it.

This mechanism is in production on all platforms: the token issuer runs, exit servers accept the protocol, and desktop and mobile applications open their sessions with anonymous tokens. Tokens are minted in advance, in the background and at a fixed rate, never at connection time. The application thus maintains a local reserve of tokens valid for the hours to come; the broadcast keys change every hour, it keeps a few per time slot. If the broadcast schedule followed that of connections, it would reveal a link between subscription and usage; this is why the two are decoupled. And if the reserve is exhausted, for example after a long period offline, the session is opened by signing with the wallet (section 27) instead of failing: the connection always succeeds, availability never depends on the stock of tokens. On this fallback path, the exit server knows the account key for the duration of the session, without writing it.

# 30. Applications: two paths, the same requirement for truth

Warren runs on macOS, Windows and Linux (desktop application, Mullvad fork), on Android and iOS (native applications), as a command line for interface-less servers, and as a browser extension. Beneath all these surfaces, a single native heart.

Two data paths coexist, with different security properties.

The first is an **unprivileged datapath proxy**. A user-space TCP/IP stack, exposed locally as a SOCKS5 or HTTP CONNECT proxy, terminates local application flows and pushes them as QUIC datagrams to the egress server. It does not request any privileges on any system. Its fail-closed property is structural, but of a particular nature.

> **KEY POINT · FAIL-CLOSED AS A LIFECYCLE PROPERTY**
>
> The local proxy port only exists **as long as the tunnel is up**. The application is pointed at this port. If the tunnel fails, the port dies, and application connections fail instead of falling back to the direct route. No firewall rules to install on time: it's a lifecycle property, no tunnel, no port, no leak. This guarantee applies per application, for the configured application: an unconfigured application, or a component that bypasses the proxy, may leak. This is a separate level of assurance from the full fail-closed firewall.

The second is the privileged **system tunnel path**, which captures all device traffic through a true tunnel device with split default routing, DNS pushing, and a kill-switch at the OS firewall. This is the path of desktop application and mobile applications. Its fail-closed guarantee is that of the firewall (section 16), applied by the system.

DNS does not leak in either mode: in proxy mode, resolutions are made to the exit server in the tunnel, without going through the host resolver; In tunnel mode, the firewall policy only allows port 53 to the tunnel resolver, whose address is only routable through the tunnel.

**"Connected" is not "protected."** The distinction comes from a concrete case: an egress server being withdrawn can continue to respond to tunnel maintenance packets, so that the QUIC connection appears perfectly alive even though it is no longer transmitting anything. The interface would display "connected" even though the user no longer has any access to the Internet.

Monitoring transport is therefore not enough; you have to exercise the datapath from end to end. The application periodically opens a real connection **through** the tunnel, to an external target: it can only succeed if the datapath actually carries bytes from one end to the other. When it fails several times in a row, the session is declared dead and the application reconnects elsewhere. This is a principle that governs the entire design of Warren applications: the displayed state comes from the traffic that actually passes, measured, and not from a flag that the software simply positions.

## 31. The forum, without email or IP address

Warren's community forum takes the same logic to the point of authentication. You prove your identity by signing the wallet, exactly the same canonical request as everywhere else, the browser never touching the key. The forum receives a matched ID, a synthetic email, and never a password, real email, or client IP address. The public username is a pronounceable string derived from a fingerprint of the key, deterministic so that the medium can recalculate the link but not invertible and not correlable to the public address. The column which contained the public key in plain text has also been removed from the diagram.

# Part VI. Safety and limits

What Warren protects, what he doesn’t protect, and who controls what.

# 32. Synthetic Threat Model

A security system is judged by what it protects, what it partially protects, and what it does not protect. Here's the summary for Warren.

**Established guarantees.** Your service provider and the networks you cross only see traffic mistaken for HTTP/3 towards an exit server, never your destinations. Sites you visit see the exit server address, not yours. A theft of the control plane database provides no access and no history. A physical seizure of an exit server yields no history about users. A multi-hop relay is cryptographically blind to content. Sessions are opened with anonymous tokens, on all platforms: in normal operation, no component records a link between your subscription and your usage. The server-side no-log is guarded by tests that block continuous integration.

**Partial or conditional guarantees.** Obfuscation defeats realistic deep inspection and active probing, but not yet exact fingerprint parity against an adversary who compares traffic byte-by-byte to that of a real browser (section 33). Traffic Analysis Defense protects the shape of traffic when the user activates it; off, the protection concerns the content and recognizability of the protocol, not the form. No-log is guaranteed by the architecture, open code and tests that guard each service.

**Out of scope by nature.** Warren does not protect against a global passive adversary capable of observing both ends and correlating by time. It does not protect against malware on your device. It does not protect against fingerprinting of your browser (the VPN hides the IP address, not the browser fingerprint). It does not protect against a coalition that would control both the relay and the exit server of your circuit. Nor does it protect against a targeted interception measure authorised by law, which Warren may be required to assist with its hosting providers. A request from the authorities receives the data that exists when it arrives: no activity history, since none exists.

There remains the question of who controls what. Are **centrally controlled**: the control plane, the version signing key (kept offline) and the coverage certificate. Are **truly decentralized**: the identity of the user (non-custodial, on their device), the identity of each exit server, and the datapath in RAM of the exit servers.

**KEY POINT · THE THREE-SENTENCE THREAT MODEL**

Warren guarantees that no activity history exists where it could be seized, and that traffic cannot be trivially identified or blocked. It does not guarantee absolute anonymity against an adversary who observes both your access to the Internet and your network exit and intersects the two, nor against software which controls your device. No VPN can, and a VPN that claims to is lying.

# 33. Trajectory

Warren is in production, and its architecture has been tailored to absorb new bricks without being rewritten. A privacy stack that stops moving forward is a stack that gets caught up: this is where the effort is going.

**Fingerprint parity.** The egress servers present a regular coverage certificate, the handshake is fragmented to thwart domain name extractors, and the tunnel footprint is monitored in continuous integration, at each commit, to prevent any drift. The next goal is the most demanding in the field: making Warren's QUIC fingerprint **byte-for-byte identical** to that of a consumer browser. No supplier has achieved it, and the necessary tooling does not yet exist in Rust: we are building it.

**Server image attestation.** The exit server image is immutable (section 19). We are working to provide enforceable proof: a fingerprint signature, checked at start-up, would result in any unofficial image being refused. It is this which will determine the opening of the network to third-party operators: without certification, "decentralized" would mean "unknown servers that cannot be verified", a step backwards disguised as progress. We prefer to decentralize when it is verifiable.

# Conclusion

Warren is the result of a series of convictions held to their technical consequences. The no-log is a construction property: from there come the non-custodial identity, the exit servers in RAM, the anonymous session credentials, the no-log kept by the tooling. Resistance to censorship is native, carried by the transport itself: hence the choice of QUIC as a foundation and the permanent obfuscation. Serving the user takes precedence over protecting yourself from them: hence the restored and hardened port forwarding, where Mullvad and IVPN removed it in 2023.

These properties have a cost. HTTP/3 mimicry requires a user-space QUIC stack, and we wrote it. Data minimization requires an infrastructure designed to hold nothing back, and we built it. What remains before us, perfect fingerprint parity, the architecture awaits without having to be redone.

The code is authoritative. The engine and the client kit are open and auditable, and the properties described here can be verified directly there. This is where what this document states is judged.

# Appendix A. Glossary

**BBR** : congestion control algorithm based on bandwidth and delay estimation, robust to losses.

**BIP39** : standard for representing a cryptographic seed in the form of a list of memorizable words.

**DAITA** : defense against traffic analysis by adding padding and delay, blurring the shape of the traffic.

**Datapath** : The path that the user's packets follow from the tunnel device to the egress server. Distinct from the control plane, which manages subscriptions and server discovery.

**DPI** (Deep Packet Inspection): inspection of the content of packets by network equipment, to identify or block a protocol.

**Ed25519** : Fast and secure elliptical curve signature scheme used for the Warren identity.

**Fail-closed** : behavior of a system which, in the event of a failure, blocks rather than lets it pass.

**Fingerprint** : observable signature of a protocol or software (packet sizes, order of TLS extensions, cadence), which allows it to be identified even when encrypted.

**Golden vectors** : reference files freezing a binary format to the nearest bit, replayed by each implementation to guarantee that they speak exactly the same protocol.

**Handshake** : Initial exchange that establishes a connection, negotiates encryption, and authenticates the parties.

**Heartbeat** : periodic message by which a server signals to the control plane that it is alive.

**HPKE** (RFC 9180): Hybrid public key encryption, used to seal end-to-end multi-hop traffic.

**Kill-switch** : mechanism that prevents any traffic from leaking out of the tunnel when it falls.

**Decoy** (*decoy*): fake service (here a real HTTP/3 server) presented to a pollster so that it does not distinguish the VPN from an ordinary site.

**MTU** : maximum size of a packet on a given network path.

**NAT-PMP** (RFC 6886): port opening protocol through address translation, used for port forwarding.

**No-log** : absence of logging of user activity.

**Non-custodial** : model where the identity secret is held by the user alone, never by the provider.

**Overlay** : write layer superimposed on a read-only file system. At Warren it only exists in RAM, so nothing persists when restarted.

**Padding** : Padding bytes added to a stream to hide the actual size of the data.

**Privacy Pass** (RFC 9578, RFC 9474): blind signature tokens allowing a right to be proven without being identifiable.

**QUIC** : modern transport protocol over UDP, foundation of HTTP/3, integrating TLS 1.3.

**Quinn** : Rust reference implementation of QUIC, of which Warren maintains a fork.

**Raw Public Key** (RFC 7250): TLS authentication by raw public key, without X.509 certificate or PKI.

**Reverse proxy** : server placed in front of a service to receive traffic for it (here, it filters access logs and never exposes the service directly).

**Slot A/B** : the two root locations of an exit server. We update the inactive location, switch to the next startup, and revert to the old one if the new one fails.

**Spin bit** : QUIC header bit intended for network measurement; Warren neutralizes it because its value could be used as a fingerprint.

**Squashfs** : Compressed read-only file system, used as the immutable root of exit servers.

**SS58** : address encoding format borrowed from the Substrate ecosystem, here used as simple encoding, without blockchain.

**Voucher** : a voucher produced by a payment, of which only the hash is stored, and which is then exchanged for a subscription; the link between the payment and the subscription serves only refunds and disappears after twenty days.

**WarrenGuard** : Warren's VPN engine on QUIC, generic and open source.

**Wire format** : the exact binary definition of a message exchanged on the network. It is the contract between implementations, fixed by the golden vectors.

# Appendix B. References

Primary sources of port forwarding removal:

- Removal of port forwarding at Mullvad, May 29, 2023, ports cut July 1, 2023. `https://mullvad.net/en/blog/removing-the-support-for-forwarded-ports`
- Gradual withdrawal of port forwarding at IVPN, June 29, 2023, complete withdrawal on September 30, 2023. `https://www.ivpn.net/blog/gradual-removal-of-port-forwarding/`
- Search with warrant at Mullvad headquarters, April 18, 2023, left without seizing anything. `https://mullvad.net/en/blog/mullvad-vpn-was-subject-to-a-search-warrant-customer-data-not-compromised`

Standards:

- RFC 6886, NAT Port Mapping Protocol (NAT-PMP)
- RFC 7250, Raw Public Keys in TLS
- RFC 5705, Keying Material Exporters for TLS
- RFC 9000 and RFC 9221, QUIC and unreliable datagrams
- RFC 9180, Hybrid Public Key Encryption (HPKE)
- RFC 9312, operational properties of QUIC (spin bit)
- RFC 9474 and RFC 9578, RSA blind signatures and Privacy Pass tokens

Academic and industrial work:

- Analysis of SNI QUIC Extractor of the Great Firewall of China, USENIX Security 2025
- TunnelCrack, deanonymization and VPN tunnel bypass, USENIX Security 2023
- TunnelVision (CVE-2024-3661), route hijacking by DHCP
- Maybenot, traffic analysis defense framework, Karlstad University
- Literature on website fingerprinting (PETS, CCS)

QUIC landscape (VPN transport):

- Mullvad, QUIC obfuscation (MASK) for WireGuard, 2025. `https://mullvad.net/en/blog/introducing-quic-obfuscation-for-wireguard`
- Cloudflare, WARP on MASK as default transport, 2024. `https://blog.cloudflare.com/zero-trust-warp-with-a-masque/`
- Apple iCloud Private Relay, HTTP/3 tunnel and MASK, 2021. `https://www.apple.com/icloud/docs/iCloud_Private_Relay_Overview_Dec2021.pdf`
- Google IP Protection (MASK and Privacy Pass tokens) in Chrome. `https://github.com/GoogleChrome/ip-protection`