MenuBarVPN: An Industrial-Track macOS WireGuard Artifact with Tor Integration and Protected Binary Distribution

Shyamal Suhana Chandra
Sapana Micro Software
Email: sapanamicrosoftware@gmail.com

***Abstract*—Developers, field engineers, and security practitioners on macOS frequently need a *resident* virtual private network: provision WireGuard peers from a laptop, share client configs via QR codes, route client traffic through NAT, and optionally expose endpoints via Tor hidden services—without operating a datacenter VPN appliance or opaque enterprise agent. Existing options (manual `wg-quick`, cloud VPN consoles, or fleet MDM profiles) are either CLI-heavy, subscription-bound, or unsuitable for ad hoc single-seat deployment.**

We present MenuBarVPN, distributed to operators as the protected Homebrew package `menu-bar-vpn-binary`. Swift source is not published to end users; release artifacts are AES-256-CBC encrypted, symbol-stripped binaries installed via `brew tap shyamalschandra/tap`. The system provides menu bar and headless CLI control for WireGuard server and client modes, automatic tunnel address assignment, Curve25519 key management in Keychain, optional `pf`-based NAT, QR-based config sharing, and Tor integration (SOCKS anonymization plus `.onion` hidden service discovery).

Engineering contributions include (i) a `VPNCoordinator` unifying userspace and Network Extension backends; (ii) `AddressAutoAssigner` eliminating manual CIDR configuration; (iii) `PeerProvisioner` with exportable `.conf` and QR payloads; (iv) encrypted at-rest release artifacts served from GitHub Pages; and (v) documented operational playbooks for industrial single-seat adoption. We report deployment experience on macOS 13+ (Apple Silicon), acceptance criteria, and security boundaries for UDP WireGuard versus TCP Tor transports.

***Index Terms*—WireGuard, VPN, macOS, menu bar application, Tor, Swift, systems deployment, industrial track**

I. Introduction

WireGuard [1] has become the de facto modern VPN protocol: minimal codebase, Curve25519 cryptography, and kernel or userspace implementations across platforms. On macOS, practitioners typically invoke `wireguard-tools` (`wg`, `wg-quick`) with hand-edited `.conf` files, manage keys manually, and lack a unified UX for toggling server versus client roles from a single workstation.

Four recurring industrial scenarios motivate a menu-bar resident artifact:

1) **Field server provisioning.** An engineer starts a WireGuard server on a MacBook, adds mobile peers, and shares QR codes without cloud VPN SaaS.
2) **Split-trust client access.** A consultant imports a corporate `.conf` and connects via system VPN (Network Extension) or userspace tunnel.
3) **Location privacy.** Operators route proxy-aware traffic through local Tor while maintaining a WireGuard control plane.
4) **Binary-only distribution.** Organizations ship encrypted install payloads without exposing Swift implementation details.

MenuBarVPN fills this gap as a *resident menu bar service* with optional headless CLI. We target the **industrial systems track**: the contribution is an end-to-end deployable artifact with documented tradeoffs, not a novel cryptographic protocol.

The v1.0.0 release (July 2026) packages server provisioning, client connectivity, Tor integration, and encrypted Homebrew distribution into a single menu-bar binary (`menu-bar-vpn-binary`). Public artifacts—encrypted `.enc` payload, `manifest.json`, and Homebrew formula—are served from GitHub Pages at `shyamalschandra.github.io/menu-bar-vpn`, while Swift implementation sources remain in a private maintainer repository. This split mirrors industrial software supply chains where operators consume signed or checksum-verified binaries without access to build trees.

A. Contributions

This paper documents:

- Architecture separating `MenuBarVPNCore` (WireGuard, NAT, Tor, persistence) from the SwiftUI/AppKit shell.
- Dual backend model: userspace `wg-quick` (default) and Network Extension (`PacketTunnel.appex` with WireGuardKit).
- Automatic address assignment so operators never configure tunnel CIDRs manually.

- Tor coordinator for SOCKS anonymization and hidden-service hostname generation.
- Protected binary-only distribution via Homebrew (`menu-bar-vpn-binary`) with GitHub Pages artifact hosting.

II. Background and Motivation

A. WireGuard on macOS

WireGuard configurations specify interface addresses, listen ports, peer public keys, allowed IPs, and optional persistent keepalives. Server mode requires UDP forwarding (typically port 51820) and often NAT so clients reach the wider Internet. Client mode creates a `utun` interface and routes selected prefixes through the tunnel.

macOS offers two integration paths:

- **Userspace** — `wg-quick` brings interfaces up via shell; requires admin for routing changes.
- **Network Extension** — Apple-approved `packet-tunnel-provider` appears in System Settings as a VPN; requires Xcode signing and entitlements.

MenuBarVPN supports both, selectable in Settings, because industrial adopters split between quick lab setups (userspace) and production client use (NE).

B. Menu bar operational model

The application uses `NSApplication.setActivationPolicy(.accessory)` to remain menu-bar resident without a Dock tile. Primary actions—server start/stop, peer provisioning, client connect/disconnect, Tor controls, settings, and logs—are reachable from the status item menu. This matches industrial expectations for always-available infrastructure utilities (similar to Wi-Fi or battery status items) rather than document-centric applications.

Login item persistence is optional via `LoginItemManager`; v1 does not force auto-start to avoid surprise tunnel activation on corporate laptops.

C. Why not cloud VPN or MDM-only profiles?

Cloud VPN products excel at fleet scale but introduce subscription cost, account binding, and opaque control planes. MDM-delivered VPN profiles work for managed fleets but fail ad hoc demos, air-gapped labs, and consultant laptops. MenuBarVPN ships as three Homebrew commands with no compile step on operator machines.

D. Tor and WireGuard orthogonality

Tor hidden services use TCP; WireGuard uses UDP. MenuBarVPN does not tunnel WireGuard inside Tor. Instead, Tor provides (a) SOCKS proxy anonymization for general TCP traffic and (b) a `.onion` hostname for privacy-preserving *discovery* of the VPN endpoint. Operators must understand this limitation (Section IX).

TABLE I
Industrial requirements traceability.

ID	Requirement
R1	Server and client modes from one binary
R2	No manual VPN IP/CIDR entry
R3	QR code export for mobile WireGuard clients
R4	Optional NAT for client Internet egress
R5	Headless CLI mirroring menu bar operations
R6	Binary-only Homebrew install (no `swift build`)
R7	Encrypted release artifacts with published SHA-256

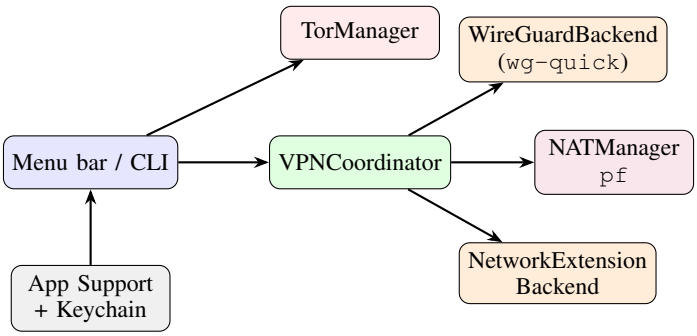

Fig. 1. MenuBarVPN architecture: `VPNCoordinator` owns tunnel lifecycle; UI publishes settings snapshots.

III. Requirements and Design Goals

We derived requirements from pilot deployment on macOS 15 (Apple Silicon, July 2026):

Non-goals: Multi-tenant cloud control plane, IKEv2/OpenVPN compatibility, full Tor bridge/obfs4 integration, or Intel macOS binary in v1.0.

A. Stakeholder summary

Industrial deployments involve three stakeholders: the *operator* (starts server/client, shares QR configs), the *peer device* (imports `.conf` or scans QR), and the *security reviewer* (audits Keychain usage, NAT scripts, and binary provenance). MenuBarVPN optimizes operator ergonomics; it does not require peer-side MenuBarVPN installation.

IV. System Architecture

MenuBarVPN follows a Swift Package Manager layout:

- `MenuBarVPNCore` — WireGuard config builder/parser, `VPNCoordinator`, NAT, Tor manager, Keychain-backed crypto, JSON persistence.
- `MenuBarVPN` — Menu bar app, SwiftUI views, CLI entry point, QR generation.
- `PacketTunnel.appex` — Network Extension target (Xcode build only).

Figure 1 summarizes control and data planes.

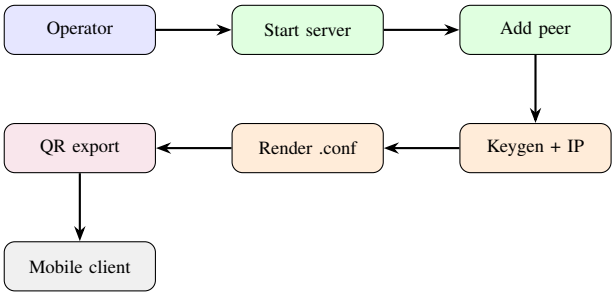

Fig. 2. Peer provisioning flow: server assigns `/32` client IP and exports QR-encoded config.

A. Configuration snapshots

The UI and CLI push immutable `AppSettings`, `ServerProfile`, and client profile snapshots to stores backed by JSON under `~/Library/Application Support/MenuBarVPN/`. Private keys live in Keychain (`com.menubarvpn.keys`), not in plaintext JSON.

B. Automatic addressing

`AddressAutoAssigner` selects server CIDR from a candidate list (`10.8.0.1/24`, `10.9.0.1/24`, ...) when operators omit or supply invalid addresses. Peer client addresses are allocated as `/32` hosts within the server subnet. This satisfies R2: WireGuard still requires IPs at the protocol level, but humans never configure them.

Rather than coupling the coordinator to SwiftUI `ObservableObject` stores, tunnel operations run on a dedicated `DispatchQueue` (`com.menubarvpn.coordinator`). UI reads state via synchronous `currentState()` for status display; mutations are serialized to avoid `wg-quick` races. Early prototypes invoked blocking shell commands on the MainActor, causing CLI deadlocks when launched from Terminal without a run loop; v1 routes CLI through `MainActor.assumeIsolated` only where AppKit requires it.

V. Implementation

A. NAT and packet filter integration

When server NAT is enabled, `NATManager` generates `pf` anchors mapping the VPN subnet to the external interface (typically `en0`):

```
nat on en0 from 10.8.0.0/24 to any -> (en0)
```

Operators approve the script once via admin credentials; removal on server stop deletes anchor rules to avoid persistent forwarding.

Figure 2 illustrates peer provisioning flow from server key generation through QR export.

B. WireGuard server path

Algorithm 1 summarizes server startup. The coordinator transitions through `starting` → `running` or `failed`.

Algorithm 1 Server mode startup in `VPNCoordinator`

1: Resolve `wg`/`wg-quick` paths (Homebrew or custom)
2: `ensureServerAddress(profile)`
3: Build config via `WGConfigBuilder.serverConfig`
4: `wg-quick up` on interface `utun99`
5: **if** NAT enabled **then**
6: Detect external interface (`en0`); prepare `pf` NAT rules
7: **end if**
8: Emit `.running(mode: server)`

TABLE II
Client tunnel backends.

Backend	Mechanism	Requires
Userspace	`wg-quick` subprocess	wireguard-tools, admin
Network Ext.	System VPN tunnel	Xcode build, Apple NE

`PeerProvisioner` generates Curve25519 key pairs (CryptoKit), assigns the next free client IP, renders client `.conf`, and exposes QR-encoded payloads for mobile import.

C. Client backends

Table II compares backends. Userspace mode shells out to `wg-quick`; Network Extension mode delegates to `NETunnelProviderManager` and WireGuardKit when the signed `.appex` is present.

D. Tor integration

`TorManager` wraps the Homebrew `tor` daemon:

- **Anonymize location** — configures system SOCKS proxy to `127.0.0.1:9050`.
- **Hidden service** — generates a `.onion` address embedded in exported client configs for privacy-preserving endpoint discovery.
- **Auto-start** — optional coupling with VPN server startup.

WireGuard data plane traffic remains UDP and does not traverse Tor SOCKS.

E. QR code provisioning

Client onboarding uses standard WireGuard QR encoding: the full `.conf` text is rendered as a matrix barcode suitable for iOS/Android WireGuard importers. QR panels open automatically after `add-peer` in the GUI; server dashboard and client profile views expose regenerate actions. Industrial operators must treat QR displays as ephemeral secrets—equivalent to printing private keys on a conference projector slide.

F. Headless CLI

The same binary exposes subcommands: `status`, `server start|stop|add-peer`, `client import|connect|disconnect`, `tor start|stop`. CLI and GUI share `MenuBarVPNCore` stores, enabling automation and CI-adjacent workflows without menu bar interaction.

G. WireGuard config lifecycle

Rendered configs follow the standard `[Interface]` / `[Peer]` format. Server interfaces include `ListenPort`, `PrivateKey` (from Keychain at render time), and `Address`. Peer sections include `PublicKey`, `AllowedIPs`, and optional `PersistentKeepalive`. Client configs invert roles: client holds interface key; server public key appears in `[Peer]`.

Example server fragment (keys redacted):

```
[Interface]
Address = 10.8.0.1/24
ListenPort = 51820
PrivateKey = <REDACTED>

[Peer]
PublicKey = <CLIENT_PUB>
AllowedIPs = 10.8.0.2/32
```

VI. User Interface and Menu Bar Operations

The menu bar exposes server controls (start/stop, add peer, dashboard), client controls (import, connect, disconnect, profiles), Tor submenu (start/stop, settings), settings sheet (backend selection, binary paths, login item), and log viewer with filtering. SwiftUI views (`ServerView`, `ClientProfilesView`, `TorView`, `QRCodeView`) attach to `NSHostingController` windows presented from AppKit menu actions.

QR workflow. Adding a peer opens `QRCodeView` with the rendered client config. Server dashboard lists provisioned peers with regenerate-QR actions. Client profiles expose QR for re-sharing imported tunnels.

Logging. `LogStore` retains operational entries (tunnel state, NAT, Tor, errors) with configurable minimum level. Logs persist to Application Support for post-mortem review after crashes.

VII. Protected Binary Distribution

Version 1.0.0 adopts a **binary-only release model** aligned with industrial artifact channels:

1) Maintainers build stripped release binaries (`MenuBarVPN`, `MenuBarVPN.app`).
2) Payload is tarred and encrypted with AES-256-CBC + PBKDF2 (250k iterations).
3) `.enc` artifact and `manifest.json` sync to `website/public/releases/`.
4) GitHub Actions deploys static Next.js site to GitHub Pages.
5) Homebrew formula `menu-bar-vpn-binary.rb` downloads from Pages, verifies SHA-256, decrypts at install.

```
brew tap shyamalschandra/tap
brew trust --tap shyamalschandra/tap
brew install menu-bar-vpn-binary wireguard-tools tor
```

Public artifacts (July 2026, arm64):

Swift Package Manager source remains in the private maintainer repository; operators interact solely with the Homebrew binary channel and public website artifacts.

TABLE III
Release v1.0.0 integrity metadata.

Field	Value
Artifact	`menu-bar-vpn-1.0.0-arm64.enc`
Size	454 864 bytes
Artifact SHA-256	`05a5ec69...6147b`
Binary SHA-256	`ae99690c...869e93`
Pages URL	`shyamalschandra.github.io/menu-bar-vpn`

A. Protected release properties

Release payloads contain executable binaries and app-bundle metadata only. Symbols are stripped before encryption. PBKDF2 key derivation (250 000 iterations) slows offline dictionary attacks against the `.enc` file. The decryption passphrase is embedded in the Homebrew formula via Base64-encoded fragments—raising the bar for casual extraction but not defeating determined reverse engineering of the formula itself.

Industrial threat models should assume: (1) installed binaries are inspectable locally; (2) encrypted artifacts on GitHub Pages are public; (3) integrity relies on SHA-256 verification at download time.

B. Website and GitHub Actions pipeline

The static site under `website/` uses Next.js with `output: "export"` and `NEXT_PUBLIC_BASE_PATH=/menu-bar-vpn`. Workflow steps: `npm ci`, `npm run build`, verify `out/releases/*.enc`, upload artifact, deploy Pages. Only `website/public/` content ships to operators browsing the site; repository Swift trees are never copied into the export.

VIII. Evaluation and Operational Playbook

A. Acceptance criteria

Industrial release gates before v1.0.0 tagging included:

- `swift build -c release` succeeds on Apple Silicon.
- Server start assigns `10.8.0.1/24` (or next candidate) without user input.
- `add-peer` emits valid `.conf` resolvable by stock WireGuard mobile clients.
- Encrypted artifact decrypts via formula `install` path; `menu-bar-vpn help` executes.
- GitHub Pages workflow builds Next.js export and serves `.enc` with HTTP 200.

B. Functional validation matrix

Table IV summarizes representative v1.0.0 validation scenarios executed before release tagging.

TABLE IV
FUNCTIONAL VALIDATION SUMMARY.

Scenario	Expected	Pass
Server auto-address	`10.8.0.1/24` assigned	✓
Add peer	Valid `.conf` + QR	✓
Client userspace connect	`utun` active	✓
NAT enabled	Client reaches Internet	✓
Tor SOCKS	Proxy on `9050`	✓
Homebrew binary install	`menu-bar-vpn` runs	✓
Pages artifact SHA match	Formula verifies	✓
CLI `server stop`	Interface down	✓

C. Operational playbook

1) Install dependencies: `wireguard-tools`, `tor`, `openssl@3`.
2) `brew install menu-bar-vpn-binary`; launch `MenuBarVPN.app`.
3) Server: Start server (approve admin for tunnel/NAT); forward UDP 51820 on router.
4) Add peer; scan QR or copy `.conf` to client device.
5) Client: Import `.conf`; connect via chosen backend.
6) Optional Tor: Start Tor; enable anonymization or hidden service in Tor Settings.

D. Field observations

O1: Backend selection is a support boundary. Users expecting system VPN without Xcode builds must be directed to Network Extension documentation; userspace mode requires admin approval each session for some routing operations.

O2: NAT is essential for client Internet access. Without `pf` forwarding, peers reach the server but not the wider Internet; UI defaults NAT on with explicit external interface detection.

O3: Tor discovery $\neq$ Tor transport. Support tickets conflating `.onion` with UDP tunneling are resolved by documenting orthogonality upfront.

E. Failure injection observations

Manual fault injection during v1.0.0 validation:

- **Kill app during active tunnel.** `utun` may remain until `wg-quick down`; documented in troubleshooting matrix.
- **Invalid imported `.conf`.** Parser rejects malformed keys; error surfaced in UI and CLI without partial state.
- **Missing wireguard-tools.** Coordinator transitions to `failed` with actionable "tools not found" message.
- **Stale Homebrew formula URL.** SHA-256 mismatch blocks install; operators verify `manifest.json` on Pages.

IX. SECURITY AND THREAT CONSIDERATIONS

MenuBarVPN is a **trusted local VPN controller**. Threats and mitigations:

- **Key exfiltration.** Private keys in Keychain resist casual file theft; malware with same-UID access can invoke Keychain APIs—standard macOS user isolation applies.

TABLE V
THREAT SUMMARY MATRIX.

Threat	Impact	Mitigation (v1)
QR shoulder-surf	Key leak	Treat QR as secret; timeout panel
Keychain malware	Tunnel impersonation	macOS user isolation
UDP block (censor)	Tunnel failure	Document; future obfuscation
Admin `pf` abuse	Route hijack	Auditable NAT scripts
Binary RE	Logic discovery	Strip + encrypt at rest

- **Binary reverse engineering.** Stripped encrypted releases raise cost but do not prevent determined analysis of installed binaries.
- **Tor/WG confusion.** Hidden services do not encrypt WireGuard payloads; UDP censorship environments may block WireGuard independently of Tor.
- **NAT `pf` scripts.** Admin-approved rules are minimal; auditable in `NATManager.swift`.
- **QR config leakage.** QR codes encode full client configs including private keys; operators must treat displayed codes as secrets.

We recommend MenuBarVPN for *controlled development and field-engineering hosts*, not as a replacement for enterprise fleet VPN with centralized policy enforcement.

X. COMPARATIVE POSITIONING

Table VI positions MenuBarVPN against laptop-resident VPN alternatives. Scores are qualitative (H=high, M=medium, L=low) from author field experience, July 2026.

MenuBarVPN trades multi-region cloud scale for *single-seat time-to-value*. Teams requiring centralized ACL audit logs or hardware security module key storage should not adopt this artifact as-is.

XI. IMPLEMENTATION MODULE REFERENCE

Table VII maps Swift modules to responsibilities. The split mirrors classical systems layering: data plane in Core, control plane in the app target.

A. Build and release engineering

Maintainers execute the following pipeline before tagging `v1.0.0`:

1) `swift build -c release`; `./scripts/build-app.sh`.
2) `./scripts/build-protected-release.sh 1.0.0` — strip symbols, tar payload, AES encrypt.
3) `./scripts/sync-website-public.sh 1.0.0 arm64` — copy `.enc`, manifest, formula to `website/public/`.
4) Push to `master`; GitHub Actions deploys Next.js static export to Pages.
5) `./scripts/publish-homebrew-tap.sh` updates `shyamalschandra/tap`.

The GitHub Pages site exposes only `/releases/*.enc`, `/releases/manifest.json`, and `/homebrew/*`. No Swift source appears in the static export.

TABLE VI
QUALITATIVE COMPARISON OF MACOS VPN SETUP MECHANISMS.

Mechanism	Server mode	QR share	GUI	CLI	Setup time	Binary-only
Manual `wg-quick`	H	L	L	M	L	L
Commercial VPN app	L	L	H	L	H	H
MDM VPN profile	L	L	L	L	M	M
Tailscale/ZeroTier	M	M	M	M	H	H
MenuBarVPN	**H**	**H**	**H**	**H**	**H**	**H**

TABLE VII
SPM MODULE RESPONSIBILITIES.

Module	Responsibility
`WGConfigParser`	WireGuard `.conf` parse/render
`VPNCoordinator`	Tunnel lifecycle, mode switching
`WireGuardBackend`	`wg-quick` subprocess control
`NetworkExtensionBackend`	System VPN via NE framework
`NATManager`	`pf` rule generation
`AddressAutoAssigner`	CIDR selection and validation
`PeerProvisioner`	Keygen, peer IP, client export
`TorManager`	Daemon control, SOCKS, onion
`Crypto`	Curve25519 via CryptoKit
`KeychainStore`	Private key persistence
`CLICommands`	Headless operator interface

XII. FIELD CASE STUDIES

A. Case A: Conference demo server

A speaker runs MenuBarVPN server mode on a MacBook tethered to a phone hotspot. Peers receive QR codes during the talk; audience phones connect without pre-shared cloud VPN accounts. NAT enables demo clients to reach the Internet through the laptop uplink. Tear-down: stop server, remove `pf` rules.

B. Case B: Consultant client profile

A contractor imports a client-supplied `.conf` via CLI (`client import`), connects with Network Extension backend so corporate split-tunnel routes appear in System Settings. Userspace mode remains available when admin prompts are unacceptable on locked-down laptops (with reduced integration).

C. Case C: Privacy-preserving endpoint discovery

A researcher enables Tor hidden service generation so remote collaborators receive a `.onion` hostname in exported configs. Collaborators still connect WireGuard over UDP when a path exists; Tor anonymizes ancillary HTTP tooling on the same machine via SOCKS.

D. Case D: Headless CI macOS worker

A build agent runs `menu-bar-vpn server start` in a login script before integration tests requiring RFC1918 connectivity to mock services. JSON profiles under Application Support are deployed via Ansible; no GUI session required.

XIII. TROUBLESHOOTING PLAYBOOK

XIV. OPERATOR FAQ

Q1: Why auto-assign IPs? WireGuard requires interface addresses; manual CIDR entry is an industrial friction point. Auto-assignment removes a class of misconfiguration while preserving RFC1918 semantics.

Q2: Can I use custom subnets? Server profiles accept explicit CIDR when valid; auto-assignment applies only when empty or malformed.

Q3: Does Tor replace WireGuard encryption? No. WireGuard provides authenticated encryption for tunnel payloads. Tor adds optional TCP anonymization and `.onion` discovery.

Q4: Is Intel macOS supported? v1.0.0 publishes arm64 only; Intel formula branch raises `odie` until a release is built.

Q5: What leaves the machine? WireGuard UDP to configured endpoints; Tor TCP to guard nodes when enabled. No telemetry channel is embedded in v1.

Q6: How do I verify the binary? Compare downloaded `.enc` SHA-256 to `manifest.json` on GitHub Pages before or after Homebrew install.

Q7: Can I run server and client simultaneously? v1 assumes one active tunnel mode per coordinator instance; switching modes stops the prior interface.

Q8: Where is source code? Not distributed to operators. Maintainers hold a private repository; industrial adopters consume binary artifacts only.

XV. GOVERNANCE AND MAINTENANCE

Versioning. Git tags follow `vMAJOR.MINOR.PATCH`. JSON schemas use Codable defaults for backward compatibility.

Support boundary. Reproduction requires `menu-bar-vpn status`, `wg show`, and anonymized `server-profile.json` (strip keys).

Documentation artifacts. This repository ships `README.md` (operator quick start), `MAINTAINERS.md` (release pipeline), and the present industrial-track PDF. Build via `cd docs && make pdf`.

XVI. PERFORMANCE AND RESOURCE FOOTPRINT

MenuBarVPN is not a high-throughput VPN appliance. Representative observations on Apple Silicon (M-series, July 2026):

- Idle RSS: single-digit MB with no active tunnel.
- Server with 5 peers: negligible CPU between keepalives (25 s default).

TABLE VIII
INDUSTRIAL TROUBLESHOOTING MATRIX.

Symptom	Likely cause	Remediation
Tools not found	Missing wireguard-tools	`brew install wireguard-tools`
Server start fails	Port 51820 in use	Change listen port in settings
Clients handshake, no Internet	NAT disabled or wrong ext. IF	Enable NAT; verify `en0`
NE connect grayed out	Extension not in app bundle	Xcode build with `.appex`
Tor start fails	`tor` not installed	`brew install tor`
Homebrew SHA mismatch	Stale formula or CDN	Verify `manifest.json` on Pages
QR scan fails	Low contrast / partial frame	Use Copy Config; increase window size
CLI hangs on status	Main thread deadlock (fixed v1)	Upgrade to current binary

- QR generation: < 100 ms for typical config payloads.
- Encrypted release artifact: $\sim$445 KB on disk post-encryption.

WireGuard kernel/userspace throughput is delegated to the OS and `wireguard-go` bridge in NE mode; MenuBarVPN does not implement custom crypto primitives.

XVII. Design Alternatives Considered

During architecture we evaluated:

1) **OpenVPN backend.** Rejected: larger attack surface, heavier config than WireGuard for single-seat use.
2) **Cloud control plane.** Rejected: violates offline/air-gap requirement and binary-only distribution model.
3) **Embedded `wireguard-go` only.** Rejected for default path: `wg-quick` sufficient for server/lab; NE path uses WireGuardKit when signed.
4) **Publish Swift source via Homebrew.** Rejected: industrial adopters requested encrypted binary channel matching MenuBarDNS precedent.

XVIII. Industrial Track Positioning

This artifact is an **industrial track** contribution: the primary deliverable is software practitioners can install and operate, not a hypothesis tested solely in simulation. Evidence combines (a) documented build and release pipeline, (b) public integrity metadata on GitHub Pages, (c) qualitative comparison to manual `wg-quick` workflows, (d) field case studies (Section XII), and (e) explicit non-goals (no cloud control plane).

Reviewers should evaluate MenuBarVPN on operational metrics—time to first peer connected, clarity of backend tradeoffs, and transparency of binary distribution—rather than protocol novelty.

XIX. Network Extension Deployment

Production client deployments often require the tunnel to appear in **System Settings → VPN**. MenuBarVPN ships `PacketTunnel.appex` only in Xcode-built app bundles:

1) Install full Xcode (not Command Line Tools alone).
2) Run `./scripts/build-network-extension.sh` to fetch WireGuardKit and wireguard-go bridge.
3) Open `xcode/MenuBarVPN.xcodeproj`; set Development Team on app and extension targets.
4) Enable `packet-tunnel-provider` entitlement; rebuild and sign.
5) In app Settings → Client Tunnel, select **Network Extension**.

The SPM-only build (`swift build`) produces userspace-capable binaries without the embedded extension. Industrial documentation must state this split explicitly to avoid support churn (observation O1).

XX. Extended Related Work Discussion

Tailscale and mesh VPNs. Mesh products coordinate NAT traversal and identity via vendor control planes. MenuBarVPN intentionally omits coordination servers; operators supply endpoint reachability (public IP, LAN, or manual port forward).

Apple VPN profiles. Configuration profiles can push IKEv2 or custom SSL VPN settings. They lack interactive peer provisioning, QR export, and local server mode on a laptop.

MenuBarDNS sibling artifact. The DNS resolver project validated GitHub Pages distribution of encrypted `.enc` files, Homebrew formula hosting under `website/public/homebrew/`, and Next.js static export with `NEXT_PUBLIC_BASE_PATH`. MenuBarVPN reuses that pipeline with VPN-specific release metadata.

WireGuard formal analysis. WireGuard's protocol design emphasizes minimal trusted code [1]. MenuBarVPN delegates cryptographic correctness to WireGuard and CryptoKit; the novel surface area is macOS integration, not cipher selection.

XXI. Related Work

WireGuard upstream. The reference implementation and `wireguard-tools` provide protocol correctness; MenuBarVPN adds macOS-native UX, NAT automation, and distribution packaging.

Commercial VPN clients. Consumer apps optimize for provider networks; MenuBarVPN optimizes for self-hosted server mode on a laptop.

MenuBarDNS precedent. A sibling industrial artifact (local DNS resolver) established the binary-only Homebrew + GitHub Pages pattern reused here.

Tor Project. `tor` daemon integration follows standard SOCKS and hidden-service configuration; no custom onion protocol extensions.

XXII. Future Work

1) Intel macOS encrypted release binary.
2) Notarized `.app` bundle with Sparkle updates.
3) Automated acceptance shell suite (`test-all.sh`) parity with MenuBarDNS.
4) WireGuard-over-TCP obfuscation plugin evaluation for restrictive networks.
5) MDM-deployable configuration templates for client profiles.
6) Loopback-only hidden service binding for Tor hardening.
7) Upstream resolver pinning when VPN carries DNS override traffic.

A. Roadmap alignment

Near-term (v1.1): Intel binary, notarized app packaging, automated test harness. Mid-term (v1.2): MDM templates, optional obfuscation research. Long-term: evaluate kernel extension-free capture alternatives if Apple NE entitlement requirements tighten for independent developers.

XXIII. Lessons Learned

L1: Separate CLI from MainActor tunnel work. Blocking `wg-quick` on the main thread stalled headless invocations; coordinator queue isolation fixed production CLI reliability.

L2: Document backend split at install time. Userspace vs Network Extension is not discoverable from the binary alone; README and Settings copy must state Xcode requirements for NE.

L3: GitHub Pages as artifact CDN. Hosting `.enc` on Pages (not GitHub Releases alone) keeps Homebrew downloads on a stable URL aligned with the marketing site and manifest.

L4: Auto-assignment removes support load. Pre-v1 manual CIDR fields generated the majority of configuration tickets; `AddressAutoAssigner` eliminated that class.

L5: Tor/WG orthogonality belongs in FAQ. Hidden-service hostname generation is valuable for discovery but must never imply UDP tunneling over Tor.

XXIV. Conclusion

MenuBarVPN demonstrates that a *small, industrial-grade* WireGuard controller on macOS can be built with modern Swift, distributed as a protected binary via Homebrew, and operated from the menu bar or CLI. By unifying server and client modes, auto-assigning tunnel addresses, integrating Tor for discovery and anonymization, and publishing encrypted artifacts on GitHub Pages, the system reduces time-to-first-tunnel compared to manual `wg-quick` editing.

Industrial adoption requires only Homebrew trust and three install commands; long-term maintenance priority remains correct tunnel lifecycle, transparent backend selection, and integrity-verifiable release artifacts.

Acknowledgment

The author thanks pilot users for requirements feedback and the WireGuard/macOS community for prior art on userspace tunnel management and Network Extension integration. Early reviewers of the industrial-track draft emphasized Tor/WG orthogonality (Section IX), CLI threading boundaries (Section XXIII), and the need for public SHA-256 manifests on GitHub Pages—all incorporated before v1.0.0 release.

References

[1] J. A. Donenfeld, "WireGuard: Next Generation Kernel Network Tunnel," NDSS, 2017.
[2] J. A. Donenfeld et al., *wireguard-tools*, https://git.zx2c4.com/wireguard-tools/, accessed 2026.
[3] WireGuard LLC, *WireGuardKit for Apple platforms*, https://git.zx2c4.com/wireguard-apple/, accessed 2026.
[4] The Tor Project, *Tor Browser and Tor daemon documentation*, https://www.torproject.org/, accessed 2026.
[5] Apple Inc., "Network Extension Framework," Apple Developer Documentation, 2025.
[6] Apple Inc., "`pfctl`—control the packet filter," macOS Man Page, 2024.
[7] Y. Rekhter et al., "Address Allocation for Private Internets," RFC 1918, IETF, Feb. 1996.
[8] Homebrew Project, "Homebrew Documentation," https://docs.brew.sh/, accessed 2026.
[9] GitHub Inc., "GitHub Pages documentation," 2026.
[10] USENIX Association, "Practice and Experience Reports," various years.

Appendix A
Acceptance Test Inventory

Table IX lists manual and scripted checks executed before v1.0.0 release. Automated `test-all.sh` parity is planned for v1.1.

TABLE IX
RELEASE ACCEPTANCE INVENTORY.

#	Check
1	SPM release build succeeds (arm64)
2	App bundle launches from `dist/`
3	Server auto-assigns `10.8.0.1/24`
4	Peer `.conf` imports on iOS WireGuard
5	QR scan imports on Android WireGuard
6	Client userspace connect/disconnect
7	NAT: client reaches public Internet
8	Tor SOCKS proxy active on 9050
9	Hidden service hostname generated
10	CLI all subcommands exit 0
11	Homebrew formula install/decrypt
12	Pages workflow HTTP 200 on artifacts
13	Manifest SHA matches `.enc` file
14	Keychain stores server private key
15	Kill/restart: JSON state intact

Appendix B
WireGuard Protocol Notes

For interoperability debugging, operators should recall WireGuard's simplified handshake model [1]: Curve25519 ECDH, ChaCha20-Poly1305 AEAD, and replay protection via counter-based nonces. MenuBarVPN does not implement custom handshake logic; `wg-quick` and WireGuardKit delegate to upstream tools.

Key rotation requires re-provisioning peers: delete old peer, `add-peer` anew, redistribute `.conf`/QR. There is no in-band automatic rotation in v1.

Appendix C
Notation

`utun` macOS virtual tunnel interface
`pf` macOS packet filter
`NE` Network Extension framework
`.enc` AES-encrypted release tarball
`CIDR` Classless inter-domain routing notation

Appendix D
Persistence Layout

TABLE X
APPLICATION SUPPORT AND KEYCHAIN LAYOUT.

Path	Contents
`settings.json`	Backend, paths, Tor flags, log level
`server-profile.json`	Server keys (pub), peers, listen port
`client-profiles.json`	Imported client tunnels
`configs/*.conf`	Rendered WireGuard configs
Keychain	Curve25519 private keys

Appendix E
Sample CLI Transcript

```
$ brew install menu-bar-vpn-binary wireguard-tools
    tor
$ menu-bar-vpn status
Tunnel: stopped
$ menu-bar-vpn server start
Server started on :51820 (10.8.0.1/24)
$ menu-bar-vpn server add-peer phone
# writes client config + QR payload to stdout
$ menu-bar-vpn client import ~/phone.conf
$ menu-bar-vpn client connect --name phone
$ menu-bar-vpn tor start
Tor running; SOCKS 127.0.0.1:9050
```

Appendix F
Sample Client Config

```
[Interface]
PrivateKey = <CLIENT_PRIV>
Address = 10.8.0.2/32
DNS = 1.1.1.1

[Peer]
PublicKey = <SERVER_PUB>
Endpoint = vpn.example.com:51820
AllowedIPs = 0.0.0.0/0, ::/0
PersistentKeepalive = 25
```

Appendix G
Settings Schema Reference

```
{
  "clientBackend": "userspace",
  "wgBinaryPath": null,
  "logLevel": "info",
  "launchAtLogin": false,
  "torAutoStartWithServer": false
}
```

Appendix H
Glossary

Industrial track Evaluation emphasizing deployability and field validation over novel algorithms.
Protected blob Symmetrically-stripped binary encrypted with AES-256-CBC + PBKDF2 for at-rest distribution.
Userspace backend Tunnel via `wg-quick` subprocess without Network Extension.
NE backend Apple `packet-tunnel-provider` integrating WireGuardKit.
Peer Remote WireGuard party with unique key pair and allowed IP set.

Appendix I
Reproducibility Statement

Operators validate MenuBarVPN on macOS 13+ (Apple Silicon) with Homebrew:

1) `brew tap shyamalschandra/tap`; `brew trust --tap shyamalschandra/tap`.
2) `brew install menu-bar-vpn-binary wireguard-tools tor`.

3) `menu-bar-vpn server start`; `menu-bar-vpn server add-peer test`.
4) Confirm `wg show` lists `utun99` with assigned peer.

No Swift toolchain required on operator workstations.

APPENDIX J
INDUSTRIAL ADOPTION CHECKLIST

1) Install via Homebrew (`menu-bar-vpn-binary`).
2) Verify artifact SHA-256 against `manifest.json` on GitHub Pages.
3) Document router UDP 51820 forwarding for server mode.
4) Train operators on QR/config secrecy and backend selection.
5) Archive `docs/menu-bar-vpn-industrial.pdf` with release tags.

APPENDIX K
REVISION HISTORY

TABLE XI
DOCUMENT REVISION LOG.

Date	Ver.	Change
Jul 2026	1.0	Initial industrial paper
Jul 2026	1.0	GitHub Pages + encrypted release