

Research Plan: Professional README for Open-Source Go Tor Client Library

Contents

Purpose	2
Overview	3
Questions	4
Synthesis	6
Not Investigating	7

Instructions for implementing agents: - Work through questions in order within each phase - Check off each question (- [x]) as you complete it - Use /research-item [question + context from this plan] to research each question - If a question needs splitting, break it up before researching - If you discover new questions, add them with checkboxes - After all questions are complete, perform the synthesis step

Purpose

Inform the creation of a minimal, high-impact README.md for `tor-go` — a pure Go Tor v3 client library with zero external dependencies (beyond golang.org/x/crypto). The README needs to stand out on GitHub, communicate the project's unique value proposition instantly, and be useful to developers considering adoption. Audience: the project maintainer (cvsouth) who will write the README.

Overview

We're investigating three focused areas: (1) what the best Go library READMEs actually do structurally, (2) what badges and visual elements create professionalism without clutter, and (3) how to present code examples that are minimal but compelling for a networking/crypto library. We start concrete (real examples) before abstracting patterns.

Questions

- **Q1: What do the best-rated open-source Go library READMEs look like structurally?**
 - **Purpose:** Establish concrete patterns from successful projects rather than relying on generic advice
 - **Depth:** STANDARD
 - **Dependencies:** None
 - **Key angles:**
 - Examine READMEs of popular Go libraries with similar profiles (networking, crypto, security): wireguard-go, age (filippo.io), mkcert, gost, go-ethereum, syncthing, caddy
 - What sections do they include and in what order?
 - How long are they (line count, word count)?
 - What's the ratio of prose to code?
 - How do they handle the “zero dependencies” or “pure Go” selling point?
 - What do they deliberately omit?
 - **Not seeking:** Generic “how to write a README” blog advice; we want real patterns from real successful projects
- **Q2: What GitHub badges and visual elements signal professionalism for Go libraries without creating clutter?**
 - **Purpose:** Identify which badges actually add value vs which are noise, specifically for a Go library
 - **Depth:** STANDARD
 - **Dependencies:** None
 - **Key angles:**
 - Which badges do top Go libraries actually use? (Go Report Card, pkg.go.dev reference, license, CI status)
 - What's the sweet spot for badge count? (diminishing returns)
 - Do any successful Go READMEs use a logo or banner image?
 - How do projects handle the “no external dependencies” claim visually?
 - shields.io vs other badge sources for Go
 - **Not seeking:** Exhaustive badge catalog; we want the minimal effective set
- **Q3: How should a security/networking Go library present code examples that are both minimal and compelling?**
 - **Purpose:** The code example is the most critical section — it determines whether a developer keeps reading or leaves
 - **Depth:** STANDARD
 - **Dependencies:** Q1
 - **Key angles:**
 - How do wireguard-go, age, and similar crypto/networking libraries present their API in the README?
 - What's the ideal length for a “quick start” code example in a Go library README?
 - Should the example show the library API, the CLI tool, or both?

- How to handle examples that inherently require network access (Tor consensus, circuit building)?
 - How do projects balance “looks simple” with “actually works if you copy-paste”?
- **Not seeking:** Full tutorial content; just the README-level quick start pattern
- **Q4: What README anti-patterns should be specifically avoided for this type of project?**
 - **Purpose:** Knowing what NOT to do is as important as knowing what to do — especially for security-related projects where trust matters
 - **Depth:** STANDARD
 - **Dependencies:** Q1, Q2
 - **Key angles:**
 - Common README mistakes that make security/crypto libraries look amateur
 - Over-documentation vs under-documentation — where’s the line?
 - How to handle “this is not production-ready” disclaimers without killing adoption
 - What makes developers trust or distrust a security-related Go library from its README alone?
 - How to present limitations honestly without being off-putting
 - **Not seeking:** General software documentation anti-patterns unrelated to READMEs

Synthesis

Combine findings into a **concrete README template/blueprint** specific to tor-go: - Recommended section order with rationale - Specific badge recommendations (with markdown) - Code example strategy (what to show, what to omit) - Tone and trust-building guidance for a security library - A “do / don’t” checklist - Word count target

The deliverable should be actionable enough that the maintainer can write the README directly from it.

Not Investigating

- **Full documentation strategy** (godoc, wiki, tutorials) — we're only focused on README.md
- **Community management** (CONTRIBUTING.md, CODE_OF_CONDUCT) — separate concern
- **CI/CD setup** — we only care about badge display, not pipeline configuration
- **Marketing/promotion** beyond the README itself — no blog posts, social media strategy
- **Comparison with non-Go projects** — patterns from Rust/Python/JS READMEs may not transfer well