Stegstr Contest — Evaluator Instructions

Version 1.0 · 2026-09-01 · Contest owner: David Brunk

1. What you are doing

The Stegstr contest received 119 entries. After triage and outreach, **18 finalists** made the final leaderboard at **stegstr.com/leaderboard/** — every entry that supplied a runnable deliverable. Your job is to actually use these versions and submit an **overall evaluation of each version you test** — one considered verdict on the whole version, with evidence. The winner is picked from these evaluations, not from claims. This is a one-time judging pass, not ongoing QA: entrants are not fixing things anymore, and later code changes are out of scope.

You are evaluating **applications**, not proposals. Every claim an entrant makes (test counts, robustness numbers, features) is unverified until you or the gauntlet confirms it.

2. What the gauntlet is (and why some versions have scores)

The **gauntlet** is our automated test harness. For entries that expose a command line, we:

1. Build the entrant's own code in a locked-down container (no network, capped resources).
2. Embed a known 43-byte secret message into a standard cover image using their encoder.
3. Run a **control check**: decode their own untouched output. If a codec can't recover its own uncompressed image, it is flagged rather than given a misleading 0%.
4. Push the stego image through **five lossy compression profiles** (light → heavy, plus a resize profile) that simulate what image-sharing pipelines do to pictures.
5. Decode each compressed image with their decoder. **Survival** = the exact secret came back.
6. Measure **invisibility** (PSNR/SSIM of stego vs. cover — higher PSNR = less visible change).

The test is **blind**: the encoder is not told which compression will hit it. The same cover, secret, and profiles are used for every entry. Survival percentages on the leaderboard are real, reproducible measurements of the entrant's own code — never self-reported numbers.

Why some versions have no score: the gauntlet needs a scriptable command line. Eight finalists are desktop or mobile apps (Tauri/Android) whose engines have no headless interface — these show a "manual test" chip. **Their steganography must be judged by you, by hand** (Section 5E). Two entries are "flagged": their code runs but a defect prevented fair automated scoring; the flag reason is shown on the board.

A gauntlet score is one dimension. A 100% survival entry with an unusable app, or one that leaks keys, should not win. That's what your evaluation adds.

3. Getting each version

- **Public entries**: the leaderboard row links straight to the GitHub repository. Clone and build per its README.
- **Evaluate the pinned snapshot.** The field is frozen: Appendix A lists the exact commit evaluated for every entry (also shown on each leaderboard row). After cloning, run `git checkout <hash>` so you judge the same code as everyone else. Anything an entrant pushed after the pin does not count.
- **Private entries** (#103 source zip, #117 source archive, #70 Android repo): request access from David and you'll receive the archive or an invite. Do not redistribute.
- Some entries also have live demo links on the board — demos are fine for a first look, but judge the real build; the contest is for a **locally runnable** app.

Safety — read this. This is contest code from strangers. Treat it as untrusted:

- Run builds inside a virtual machine, container, or at minimum a dedicated user account.

– Never enter a real Nostr private key (nsec), real wallet, or personal credentials. Generate throwaway identities inside each app.
– Don't test with images you'd mind leaking.
– If an app makes unexpected network connections, that is itself a **finding — report it**.

4. Version quick reference

Entry	Handle	What it is	Gauntlet	Access	Claimed extras to verify
#101	@rohaanehsaan2	Python CLI	100% · PSNR 50.1	Public	Lean CLI
#83	@riyadh034	Python	100% · PSNR 49.4	Public	CLI + HTTP API + MCP server; 106 tests claimed
#103	@bhakiya06	Node.js	100% · PSNR 36.3	**Private — ask David**	npm test suite
#93	@SamnangUY	Rust CLI	40% · PSNR 40.8	Public	Headless CLI, built-in survival simulator, open-mode encryption
#96	@dvyoga14	Tauri fork + Node CLI	20%	Public	Cross-platform builds, CLI/MCP claimed
#107	@juano26	Python GUI	20% · PSNR 39.2	Public	Guided installer, GUI, web panel
#87	@jayantbhakar	Python, dual engine	20% · PSNR 37.4	Public	Neural engine (large weights via Git LFS), FastAPI agent tool server, NIP-44 crypto, media-server upload
#63	@JAIMEBAJAJR	Python	0% · PSNR 69.6	Public	Near-invisible but zero survival; claims PDF/GIF/video carriers
#108	@wordsmithakif	Tauri desktop	manual	Public	MCP server, mac/win/linux releases, calibrate command, 9 upstream bug fixes
#79	@shwetadinkar	Tauri desktop	manual	Public	MCP server, 349 tests claimed, encrypted attachments to 50 MB, pointer mode
#118	@adamo61	Tauri desktop (Rust/TS)	manual	Public	Security focus: security review doc, path-traversal/NUL tests, nsec redaction
#117	@Adnan777	Tauri desktop	manual	**Private — ask David**	Own channel simulator, e2e tests
#99	@tanyat29	Tauri desktop	manual	Public	QIM + Reed–Solomon engine (engine ≈ #9)
#9	@muradabbaszade	Tauri desktop	manual	Public	Signal-only payload, multi-relay reconnect, offline queue
#81	@AFAQCEO	Tauri desktop	manual	Public	Full app build; final update was CI/docs only
#70	@DavidNMC	Android app	manual	**Private — ask David**	Native Android implementation
#82	@nayyariftikhar80	Python	flagged	Public	Didn't round-trip its own output in-harness — check by hand
#59	@SakibKaiser	Node/TS	flagged	Public	CLI reported success but wrote no output file — check by hand

5. What to evaluate

Work through these dimensions for each version you take on. You do not need to do every version — depth beats breadth. Tell David which ones you're covering.

A. Operability & stability

– Does it install and launch by following only its own README?
– Smooth operation, no crashes, no hangs, no dead buttons or broken flows.
– Try wrong inputs on purpose (huge image, tiny image, empty message, weird characters, cancel mid-operation). Graceful errors or crashes?

– Different combinations of steps in different orders — order-dependent breakage was the #1 problem in earlier versions of this app.

B. User-to-user interaction

– Two instances (two machines, or two profiles): can two users actually find each other, exchange posts/messages, and see each other's content?

– Information transmits **correctly**: what user B decodes is exactly what user A sent — text intact, no truncation, no corruption, attachments intact.

– Feeds update; DMs arrive; nothing silently disappears.

C. Security, encryption & leaks

– Encryption between users succeeds: an encrypted message can be read by the intended recipient and **cannot** be read by a third instance/user without the key.

– **No leaks**: private keys never shown in logs, files, URLs, or the UI beyond where required; the hidden payload is not recoverable from the image without the key; no plaintext of an "encrypted" message sitting in temp files or localStorage.

– The decode must come **from the image pixels alone**. Decode on a fresh machine/instance that never saw the original — if it only decodes where it was encoded, that's a fake.

– Unexpected network traffic, analytics, or calls to odd hosts: report immediately.

D. Local mode and relay mode

– **Local**: with networking off, does the core embed/decode work fully offline?

– **Relay-connected**: connect to a Nostr relay — does publish/subscribe work? Does the app handle a dead or slow relay gracefully (timeouts, reconnects), or does it hang at "connecting"? (Known weak spot — several versions hang when a configured relay is down.)

E. Steganography quality (the heart of the contest)

– **Invisibility**: compare cover and stego image at 100% and zoomed. Visible noise, banding, blockiness, color shifts — anywhere? Distortion should be evenly spread, not concentrated in one region. Try smooth-gradient images (sky), busy photos, and text/screenshot images.

– **Survival through real transference**: send the stego image through channels you personally use that recompress or resize images (most messaging and social apps do; email and file sync usually don't), then decode the received copy on the other end.

· **Verify a real round trip happened**: if the received file is byte-identical (same size) to what you sent, the channel didn't recompress it and the test proves nothing. Send from a different device or have another person return it.

· Also try: export/re-save as JPEG at lower quality in an image editor, resize by a few percent, convert formats. Note exactly which transformations kill the payload.

– **Capacity**: how much data fits in a normal photo? Does the app fail clearly when the message is too big, or corrupt silently?

– For gauntlet-scored entries, your manual results should roughly agree with the board; if they don't, that's a valuable finding either way.

F. Extra features — describe what's built, then test whether it works

Entrants added features beyond the brief. For each version, list what it actually ships, then verify it does what it claims. Watch especially for:

– **Agentic/AI integration**: MCP servers, HTTP/OpenAPI tool servers meant for AI agents. Does the server start? Can a client (or an AI agent) actually call embed/decode through it?

- **Headless CLI**: real scriptable commands with sensible exit codes and output? (This also tells us whether an unscored entry could have been gauntlet-scored.)
- **Cryptocurrency payments**: none are verified by us. If a version claims zaps/wallet/payment integration, describe it and test it with throwaway amounts or test networks only — never real funds.
- Offline queues, attachments, calibration tools, built-in survival simulators, multi-relay handling, release binaries/installers for mac/win/linux — do they work as shipped?
- A feature that exists but doesn't work counts against the version in your evaluation, not for it.

6. How to submit your evaluation

Submit through the form linked at the top of the leaderboard ("Submit evidence"). Sign-in is required so every evaluation is attributed.

- **One submission = one overall evaluation of one version.** Cover the whole version: what works, what doesn't, how it compares on the dimensions in Section 5, and your bottom-line verdict on whether it should win. This is not a bug tracker — significant defects belong inside your overall write-up.
- State the **version** (entry #) and confirm you tested the **pinned commit** (Appendix A), plus your environment (OS + versions).
- Attach evidence for your key claims: screen recordings, screenshots, logs, and image files (cover + stego + received copy) where relevant. A verdict with evidence outweighs a verdict without it.
- Be fair: apply the same tests to every version you compare. Never state a result you didn't actually run.

Questions, access to private entries, or something ambiguous mid-test: contact David directly rather than guessing.

Appendix A — pinned snapshots (what you evaluate)

Git entries: `git checkout` this commit after cloning. Archive entries: the archive David provides is the pinned artifact (fingerprint shown).

Entry	Pinned snapshot
#101	`1f4ed55` (git)
#83	`ed64b7a` (git)
#103	zip, SHA-256 starts `d944dd5503befb40`
#93	`12a8ed2` (git)
#96	`87ee402` (git)
#107	`3a7a4cf` (git)
#87	`9eb5be2` (git)
#63	`8dbaaab` (git)
#108	`c6a5ff1` (git)
#79	`85980dc` (git)
#118	`ec144a6` (git)
#117	source archive, tree fingerprint `768c299a1ec191dc`
#99	`b76d324` (git)
#9	`574391b` (git)
#81	`7d3efec` (git)
#70	`9f0618d` (git)
#82	`1844519` (git)
#59	`8624d5c` (git)