# SPEC-2 — Lucid RDP GUI Distributables

## Background

Lucid RDP is a Tor-only, blockchain-anchored remote desktop platform targeting Raspberry Pi 5 (Ubuntu Server) where every session is issued a single-use ID, recorded locally as encrypted chunks, and anchored to an on‑system data chain; monetary flows use TRON + USDT via an isolated Tron-Node service. The MVP ships with immutable on-chain contracts and a minimal Admin UI on the Pi for provisioning, proofs export, ledger mode selection, and key rotation.

This document scopes and plans the *distributable client surfaces*—delivered separately from the server/appliance images—to connect to the existing platform via .onion endpoints. We define three GUIs:

1. **User GUI** — for end users to initiate/join sessions, apply client‑enforced controls (trust‑nothing), and view/export proofs.
2. **Admin GUI** — for operators administering a Pi appliance (bootstrap, provisioning, session manifests, payouts/withdrawals, keys, backups, diagnostics).
3. **Node GUI** — for node workers to monitor PoOT/relay/storage duties, wallet/resource usage, and payout batches.

### Source document synthesis (why these GUIs exist)

* **Tor-only + .onion** access for all GUIs/APIs — mandates client apps that speak over Tor.
* **On‑System Data Chain** (manifests, encrypted chunk roots) + **MongoDB collections** — determines what the GUIs view/export.
* **TRON payout routers (PR0/PRKYC) & wallets** — drives Admin/Node payout and withdrawal flows.
* **Client‑controlled session policy & privacy shield** — shapes the User GUI’s session controls and enforcement.
* **Observer role & read‑only endpoints** — informs a safe, restricted viewer mode in User/Admin.

### GUI deliverables at a glance (separate from server images)

* Each GUI is a **standalone distributable** (desktop app first; web‑only is acceptable when served via Tor from the Pi/node) that authenticates using local-first accounts (magic link + TOTP), talks to token‑scoped APIs, and bundles a Tor client where required. Server‑side contracts, payouts, and data‑chain behavior remain unchanged.

## Requirements (Draft)

We will refine these per GUI in the next step. This draft simply orients the scope for review.

### Cross-GUI

* Must: Tor‑only network paths; bundle Tor (no clearnet fallback). All HTTP(S)/gRPC traffic forced through SOCKS5 (127.0.0.1:9150) with .onion host allow‑list.
* Must: Desktop targets: Windows/macOS/Linux.
* Must: Implemented with **Python + Tkinter/ttk** (“a window to the application”).
* Must: Local config encrypted-at-rest; secrets never leave client.
* Should: Local-first identity (magic link + TOTP) with token‑scoped API access.
* Should: Offline‑tolerant UI with queued actions; telemetry‑off by default.
* Won’t (MVP): Mobile clients; local blockchain ops; non‑Tor egress.

### User GUI (initial cut)

* Session connect/join; client policy enforcement (clipboard, file, devices, privacy shield), re‑entry flows; proofs viewer/export; lightweight notifications.

### Admin GUI (initial cut)

* Bootstrap/provisioning wizard, session manifests, anchor/payout visibility, router toggle (PR0/PRKYC), key rotation, backups, diagnostics, OTA check.

### Node GUI (initial cut)

* WorkCredits/PoOT metrics, relay/storage availability, leader slots, energy/TRX budgeting, payout batches/receipts; watchdog & alerts.

## Method

### Architecture (per GUI)

* **Stack:** Python 3.11+, Tkinter/ttk, Requests + PySocks (SOCKS5), optional grpcio over SOCKS, cryptography (Fernet/XChaCha20), orjson, pydantic for models, appdirs for cross‑platform paths.
* **Tor integration:** Ship platform‑specific tor binary + torrc (SOCKSPort=9150, ControlPort=9151 CookieAuth=1, IsolateSOCKSAuth=1). Use stem to health‑check; start Tor as child process in a sandboxed runtime dir; hard‑fail if not reachable; .onion allow‑list enforcement at client layer.
* **Process model:** Single‑process UI with worker thread pool for network I/O; subprocess for Tor. Graceful shutdown ensures Tor termination.
* **Security:**
  + Enforce .onion host allow‑list and scheme (https or h2 over Tor).
  + Pin backend pubkeys (TOFU file updated by Admin).
  + Config vault: OS keyring when available, else passphrase‑derived key via Argon2id.
  + Crash‑safe logs (PII‑redacted) with user opt‑in.

### GUI Component Breakdown

#### User GUI (Tkinter)

* **Windows:**
  + *Connect* (session ID / QR), *Policy* (toggles: clipboard, file, devices, privacy shield), *Proofs* (list, detail, export), *Settings* (Tor status, identity, keys), *About/Logs*.
* **State/View Models:** SessionModel, PolicyModel, ProofModel, IdentityModel.
* **Flows:** Connect → Enforce policy → Join → Live indicators → End → Proof receipt → Export.

#### Admin GUI (Tkinter)

* **Windows:**
  + *Bootstrap Wizard* (QR/device pairing), *Manifests* (sessions, anchors), *Payouts* (routers PR0/PRKYC, batches), *Keys* (rotate, backup), *Backups/Restore*, *Diagnostics* (Tor, latency, storage), *OTA*.
* **State/View Models:** ApplianceModel, PayoutRouterModel, KeyringModel, BackupModel.
* **Flows:** Pair → Provision → Monitor → Keys/Payouts → Backup → OTA.

#### Node GUI (Tkinter)

* **Windows:**
  + *Overview* (slots, uptime), *WorkCredits/PoOT*, *Relay/Storage health*, *Wallet* (TRX energy, balances), *Batches* (payout history), *Alerts*.
* **State/View Models:** NodeModel, MetricsModel, WalletModel, BatchModel.
* **Flows:** Authenticate → Monitor → Export reports → Acknowledge alerts.

### Shared Core Module (gui\_core)

* **Networking:** TorHttpClient, TorGrpcChannel enforcing .onion allow‑list.
* **Domain Models:** Sessions, Manifests, Proofs, Payouts.
* **UX Kit:** Tkinter theming, toasts, QR scan dialog, file pickers, pagination, virtual list.
* **Telemetry:** optional, anonymized; disabled by default.

### PlantUML — High‑level GUI runtime

@startuml  
actor User  
participant "User GUI" as UG  
participant "Tor (child)" as TOR  
participant "API over .onion" as API  
  
User -> UG: Launch app  
UG -> TOR: spawn tor + health check  
UG -> UG: Load encrypted config  
UG -> API: Auth + session lookup (via TOR)  
UG <- API: Session metadata  
User -> UG: Set client policy  
UG -> API: Join/Control session (policy enforced client-side)  
UG <- API: Proof receipts  
UG -> User: Export proofs  
UG -> TOR: graceful shutdown  
@enduml

### Core Parameters Module (Start)

**Purpose:** A shared module that defines, validates, persists, negotiates, and enforces per‑session connection settings (mouse, keyboard, file, and system usage). It also drives the *Terms of Connection* acceptance flow and generates a signed *Consent Receipt*.

#### Parameter Catalog (atomic capabilities)

* **Input**
  + mouse.enabled (bool)
  + mouse.rate\_limit\_hz (int, 1–240)
  + keyboard.enabled (bool)
  + keyboard.blocklist (list of keysyms, e.g., ['F12','PrintScreen','Ctrl+Alt+Del'])
  + keyboard.allowlist\_mode (bool) + keyboard.allowlist (optional)
* **Clipboard**
  + clipboard.host\_to\_remote (bool)
  + clipboard.remote\_to\_host (bool)
  + clipboard.max\_bytes (int)
* **File Transfer**
  + file.upload\_enabled (host→remote)
  + file.download\_enabled (remote→host)
  + file.max\_size\_mb (int)
  + file.allowed\_dirs (list of host paths; sandbox root required)
  + file.allowed\_extensions (list of globs)
* **System Usage**
  + system.screenshare\_allowed (bool)
  + system.screenshots\_allowed (bool)
  + system.audio\_in / system.audio\_out (bool)
  + system.camera (bool)
  + system.printing (bool)
  + system.shell\_channels (bool)

**Defaults (MVP):** mouse.enabled=true, keyboard.enabled=true, clipboard/file/system **all disabled** unless explicitly enabled by the user per session.

#### Data Model (Pydantic)

from pydantic import BaseModel, Field  
from typing import List, Optional  
  
class InputParams(BaseModel):  
 mouse\_enabled: bool = Field(True)  
 mouse\_rate\_limit\_hz: int = Field(60, ge=1, le=240)  
 kb\_enabled: bool = Field(True)  
 kb\_blocklist: List[str] = Field(default\_factory=lambda: ['PrintScreen'])  
 kb\_allowlist\_mode: bool = False  
 kb\_allowlist: Optional[List[str]] = None  
  
class ClipboardParams(BaseModel):  
 host\_to\_remote: bool = False  
 remote\_to\_host: bool = False  
 max\_bytes: int = 65536  
  
class FileParams(BaseModel):  
 upload\_enabled: bool = False  
 download\_enabled: bool = False  
 max\_size\_mb: int = 25  
 allowed\_dirs: List[str] = Field(default\_factory=list)  
 allowed\_extensions: List[str] = Field(default\_factory=list)  
  
class SystemParams(BaseModel):  
 screenshare\_allowed: bool = False  
 screenshots\_allowed: bool = False  
 audio\_in: bool = False  
 audio\_out: bool = True  
 camera: bool = False  
 printing: bool = False  
 shell\_channels: bool = False  
  
class ConnectionParams(BaseModel):  
 version: int = 1  
 input: InputParams = InputParams()  
 clipboard: ClipboardParams = ClipboardParams()  
 file: FileParams = FileParams()  
 system: SystemParams = SystemParams()  
  
 def policy\_hash(self) -> str:  
 import orjson, hashlib  
 blob = orjson.dumps(self.dict(by\_alias=True, exclude\_none=True))  
 return hashlib.sha256(blob).hexdigest()

#### Consent & Terms

* **Terms of Connection** (ToC) text versioned and locally cached; a SHA‑256 hash (terms\_hash) is bound to the policy\_hash for each session.
* **Consent Receipt** signed by the connecting user using their local key. Sent with the JoinSession request and recorded in session manifest.

class ConsentReceipt(BaseModel):  
 user\_id: str  
 session\_id: str  
 policy\_hash: str  
 terms\_hash: str  
 accepted\_at\_iso: str  
 signature\_b64: str # ed25519 over user\_id|session\_id|policy\_hash|terms\_hash|accepted\_at

#### Enforcement Surfaces (client‑side)

* **UI gating:** Tkinter widgets for dis/allow toggles; disabled controls gray out associated features.
* **Event filters:**
  + Mouse: throttle motion events to mouse\_rate\_limit\_hz; drop events when mouse\_enabled=false.
  + Keyboard: blocklist/allowlist filtering of key events before they’re sent to the remote channel.
  + Clipboard: intercept Tk clipboard handlers; enforce directions and size caps.
  + File: file pickers rooted to allowed sandbox; size/type checks pre‑transfer.
  + System: disable screenshare/screenshot triggers; audio/camera channels never opened when false.

#### UI — Connection Settings (Tkinter)

* **Tabs:** *Input • Clipboard • Files • System*.
* **Profiles:** Strict, Standard, Custom (persisted locally; never auto‑enable high‑risk items).
* **Preflight:** summary panel shows computed policy\_hash, ToC version, and diff from last used profile.

#### PlantUML — Terms & Join Flow

@startuml  
actor User  
participant GUI  
participant Tor  
participant API as Backend  
  
User -> GUI: Configure Connection Params  
GUI -> GUI: Compute policy\_hash  
GUI -> GUI: Display Terms of Connection (vX)  
User -> GUI: Accept terms (checkbox + confirm)  
GUI -> GUI: Build ConsentReceipt (sign)  
GUI -> Tor: ensure tor up  
GUI -> Backend: JoinSession(params, consent) via .onion  
Backend --> GUI: OK + session token  
GUI -> User: Start session window (enforcement active)  
@enduml

## Implementation

### Production Workflow for Distributable Images (Cross‑platform)

**Repo Layout**

/gu i-core/ # shared python package (models, tor client, widgets)  
/gui-user/  
/gui-admin/  
/gui-node/  
/build/ # pyinstaller specs, torrc templates, signing scripts  
/ci/ # pipelines (GitHub Actions or GitLab CI)

**Versioning**: Semantic versioning MAJOR.MINOR.PATCH; embed git SHA and Tor version into --version output.

**Build Toolchain**: Python 3.11, uv or poetry for locking; **PyInstaller** for freezing; optional **Nuitka** for hardened builds.

**Tor Vendor Step** 1. Fetch platform‑specific Tor (Windows x64, macOS universal, Linux x64/arm64) from official mirrors. 2. Verify signatures (GPG) against Tor Project keys; fail build on mismatch. 3. Copy into vendor/tor/<os>/ with minimal torrc template.

**PyInstaller Freezing** - One .spec per GUI per OS; include: - Data files: tor, torrc, icons, LICENSES. - Hidden imports: pydantic, grpc, pkg\_resources as needed. - --noupx for determinism; --clean; --noconfirm. - Post‑freeze smoke test: launch app headless, ensure Tor starts and API .onion DNS fails closed when Tor absent.

**Installers** - **Windows:** build MSI (WiX) or EXE (NSIS). Include service‑free run; per‑user install default. - **macOS:** .app inside signed .dmg; hardened runtime; notarize with Apple Developer ID; staple ticket. - **Linux:** AppImage (primary) + tar.gz; optional .deb/.rpm via fpm.

**Code Signing & Notarization** - Windows: Sign .exe/.msi with EV/OV cert; timestamp (RFC3161). - macOS: codesign --deep --options runtime for .app and .dmg; notarize; staple. - Linux: produce SHA256 sums and optional minisign signatures.

**Reproducibility & Supply Chain** - Pin all dependencies; vendor wheels for offline builds. - Generate SBOM (CycloneDX) and attach to release. - Capture build manifest: tool versions, Tor version, GPG key IDs.

**Updates** - In‑app update check (Tor‑fetched JSON manifest + signatures). Download updates over Tor only. No auto‑apply on macOS unless notarized.

**Runtime Hardening** - Network egress self‑test: refuse to operate without Tor SOCKS; verify .onion allow‑list. - Certificate/public‑key pinning for backend TLS where used. - Sandboxed temp dirs; lock down file dialogs to user profile by default.

**Packaging Scripts (outline)** - build/scripts/fetch\_tor.py – download & verify Tor. - build/scripts/freeze.py – run PyInstaller for each GUI/OS. - build/scripts/sign\_win.ps1 – SignTool. - build/scripts/sign\_mac.sh – codesign + notarize. - build/scripts/make\_installers.\* – WiX/NSIS, dmgbuild, appimagetool.

**CI Pipeline (per tag)** 1. Lint/type‑check (ruff, mypy). 2. Unit/UI smoke tests (pytest + headless Tk). 3. Build core + three GUIs with PyInstaller. 4. Sign/notarize artifacts. 5. Run integration tests: Tor spin‑up, onion reachability, fail‑closed checks. 6. Generate SBOM + checksums. 7. Publish GitHub Release (assets per OS + manifest.json).

### PlantUML — Build & Release Pipeline

@startuml  
start  
:Push tag vX.Y.Z;  
:CI spawn matrix (Win/macOS/Linux);  
:Fetch & verify Tor;  
:Freeze GUIs with PyInstaller;  
:Run smoke tests;  
if (Platform == Windows) then (yes)  
 :Sign .exe/.msi;  
elseif (Platform == macOS) then (yes)  
 :Codesign .app + .dmg;  
 :Notarize & staple;  
else (Linux)  
 :Create AppImage + checksums;  
endif  
:Generate SBOM + release manifest;  
:Publish artifacts to release;  
stop  
@enduml

## Milestones

1. **M0 — Skeletons Ready (1 wk):** Repo layout, Tkinter shells for 3 GUIs, Tor bootstrap working.
2. **M1 — Networking Core (1–2 wks):** TorHttpClient/TorGrpcChannel, .onion allow‑list enforcement, health checks.
3. **M2 — Views & Models (2 wks):** Core windows and state models implemented for User/Admin/Node.
4. **M3 — Packaging Alpha (1 wk):** PyInstaller binaries on all OS; Tor bundled; unsigned artifacts.
5. **M4 — Installers & Signing (1–2 wks):** MSI/DMG/AppImage; codesign/notarize where available.
6. **M5 — Beta (1 wk):** End‑to‑end smoke tests over Tor; update mechanism; SBOM.
7. **M6 — Release (0.5 wk):** Tag, publish, checksums, docs.

## Gathering Results

* **Acceptance:** Each GUI can start Tor, refuse clearnet, and complete its primary flow against staging .onion endpoints.
* **Security checks:** Pinning verified; fail‑closed tests; config encryption verified.
* **Distribution:** Installers run on fresh VMs; hash and signature verification match.
* **UX:** Policy controls behave as specified; critical metrics visible; proofs export succeeds.

## Need Professional Help in Developing Your Architecture?

Please contact me at [sammuti.com](https://sammuti.com) :)