Pravelle — Tutorial & User Guide

A live ZK dark pool on BNB Smart Chain Testnet: add ERC-20 test funds, place hidden orders, and withdraw without revealing which private note you spent. Guided mode talks directly to DarkPool and the real 16-statement Venus verifier. Start with the quick path below; operator, proof and infrastructure detail lives in Operations & Proof System.

Download PDF | Chinese tester guide PDF | Open the app →

Quickstart for normal users

1. Choose a path

Use the demo anywhere, or open the hosted live app with the tester token supplied by the operator.

2. Connect wallet

Switch to BNB Smart Chain Testnet (chain 97) and sign the app-scoped account request.

3. Add funds

Enter an ERC-20 and base-unit amount, then approve and confirm in your wallet.

4. Use privately

Place or cancel a real hidden order; withdraw once your note has current ASP paths.

Demo: no wallet or prover is needed; use the walletless demo. **Live testing:** the hosted app uses the pinned Pravelle HTTPS services. Open the operator-provided URL containing your individual prover token once; the page stores it for that origin and removes it from the address bar. Local development can still use the reviewed v1.0.1 tester tunnel.

CONTENTS

1. Quickstart for normal users
2. The 60-second mental model
3. Tutorial A — Add private funds
4. Tutorial B — Place a private order
5. Tutorial C — Settle a matched trade
6. The compliance layer (ASP)
7. Tutorial D — Withdraw to a wallet
8. Troubleshooting
9. Where to go next

Operators and builders: architecture, infrastructure and the proof system moved to Operations & Proof System.

---

1. The 60-second mental model

Pravelle replaces *accounts with visible balances* with **notes**: a note is a hidden record `commitment = Poseidon(owner_key, asset, value)` stored as a leaf in an on-chain Merkle tree. Inside the pool, the chain sees commitments rather than owners and balances. The deposit and withdrawal boundaries are still public: token and amount are visible when funds enter, and token, amount, and destination are visible when funds leave.

Everything you do is a **statement** proven in zero knowledge and verified on-chain:

You want to…	Statement	What stays hidden
Put funds into the pool	`deposit`	the private-note owner (depositor, token, and amount are public at the door)
Move value privately	`transfer`	amounts, sender, recipient
Withdraw to a wallet	`transfer` (Withdraw slot)	the source note
Post a hidden order	`place`	price & size (only a commitment is booked)
Retract an order	`cancel`	which order
Settle a matched trade	`half-match` ×2 → `settleMatch2`	both parties' notes, terms, change
Swap assets	`swap`	the leg amounts
Batch-auction settle	`batch`	per-order fills
Committed-wallet variants	`*-wallet` (8)	the whole wallet's contents

There are **16 finalized verifier slots**. Each accepted action uses a real Venus PLONK proof checked by the on-chain `VenusDarkPoolVerifier`. There is **no mock path** in the live app. The legacy `half-match-wallet` HTTP route remains disabled until its witness is migrated to the same key-safe authorization model used by the active Guided routes.

Two on-chain Merkle trees hold the state:

- **note pool** (`poolRoot`) — every note commitment.
- **order book** (`bookRoot`) — every resting order commitment.

Plus two **compliance roots** (Privacy-Pools style), maintained by an **ASP** (Association Set Provider):

- **association root** — an advisory set the operator publishes. It is *not* enforced: spends do not prove inclusion, and a deposit is spendable straight away.
- **blocklist root** — a sanctions IMT. Every spend proves *exclusion* (non-membership).

---

2. Tutorial A — Add private funds (Guided DarkPool deposit)

A deposit creates a private note from public funds. You need pTST or pUSD on BSC testnet, a little test BNB for gas, an individual prover token, and a reachable configured prover.

In the browser: Guided mode in `/app.html` now submits directly to the live DarkPool. `/deposit.html` remains a standalone recovery-file tool for builders; it is not required for the Guided flow.

1. Open the hosted live app (or a local copy with the tester tunnel), connect MetaMask, and switch to BNB Smart Chain Testnet.
2. Sign the structured Pravelle account request. The signature combines with a device-local secret to unlock the encrypted vault; it is not the spending seed. Approve it only on the trusted Pravelle site.
3. Create a recovery passphrase, then download the encrypted recovery file before funding. Recovery requires the file, passphrase, and connected wallet; keep the file and passphrase separately.
4. Select **Add funds**, paste the ERC-20 address, and enter an integer amount in token base units. For a 6-decimal token, one whole token is `1000000` base units.
5. The page builds a public deposit witness and sends it to the configured prover. The validated dual-4090 path takes about **73 seconds** before queue and browser overhead.
6. If needed, approve the ERC-20 spend, then confirm the DarkPool deposit in the wallet. After the receipt confirms, Guided mode records the note metadata and updates the private balance.

> The recovery JSON contains encrypted key material but no device secret or passphrase. A wallet signature alone cannot recreate the account. Nondeterministic wallets can recover with the passphrase and refresh the local device wrap. Keep the recovery file and passphrase separate.

What happens under the hood (the same as the CLI flow):

```
seed    = decryptVault(recoveryFile, recoveryPassphrase) // browser only
account = deriveAccount(seed)
out     = deposit_witness(account.address, token, value)
proof   = prove("deposit", out.proverWitnessJson)       // no spending key; ~73s
validated
dp.deposit(token, value, out.commitment, proof)          // real finalized verifier
```

The contract pulls exactly `value` of the ERC-20 (balance-delta checked) and appends the commitment to `poolRoot`. Your funds are now a hidden note.

---

3. Tutorial B — Place a private order

An order commits to `(side, price, size, expiry, base, quote, salt)` but books only a *commitment* — the market sees an opaque leaf, not your price or size.

```
out   = place_witness(spendingKey, terms)          // master key used only in browser WASM
proof = prove("place", out.proverWitnessJson)      // operation authorization; ~83s validated
dp.placeOrder(proof, out.orderCommitment, now)     // verifier.verifyPlace(...) ; _checkNow(now)
```

`now` must be within `NOW_WINDOW` (five minutes by default) of the chain clock. The matcher expires pending settlements after four minutes and the prover reserves the final minute for wrapping and submission, so an older proof must be rebuilt. The order lands at the next free `bookRoot` leaf. Guided **Trade privately** does exactly this; Advanced mode exposes the same on-chain action with more proof details.

> The order is opaque on-chain, but the matcher receives plaintext side, price, size, token pair, expiry, and a stable public identity so it can cross orders. Treat the matcher as privacy-trusted and keep it loopback/self-hosted unless that disclosure is acceptable.

---

4. Tutorial C — Settle a matched trade (the split-proof flow)

This is the heart of the dark pool. Two crossing orders settle **atomically** with each party proving only its own half — neither reveals its note to the other, and the matcher can't misroute.

The pieces a settlement binds

A `settleMatch2(proofSell, proofBuy, joint)` call spends **two orders + two backing notes** and mints **four settlement notes**. Its 23-field `joint` ties everything to *current* on-chain roots:

```
joint = { bookRoot, poolRoot, associationRoot, blocklistRoot, now,
          baseToken, quoteToken, pExec, fill, sellerMpk, buyerMpk,
          sellOrderNf, buyOrderNf, sellNoteNf, buyNoteNf,            // nullifiers (single-spend)
          outBaseToBuyer, outQuoteToSeller,                          // the two main legs
          outBaseChangeToSeller, outQuoteChangeToBuyer,              // each side's change
          buyerProceedsSalt, sellerProceedsSalt,                     // DH-derived note salts
          sellerChangeSalt, buyerChangeSalt }
```

The end-to-end flow (what each side runs)

1. **Back each order with a note.** The seller needs a *base* note in the pool ≥ its size; the buyer a *quote* note ≥ `fill × pExec`. (Deposit them as in Tutorial A.)
2. **Derive shared salts** for the four output notes via viewing-key Diffie–Hellman (`dh_salts`) — both sides arrive at the same set; the relay (public keys only) cannot.
3. **Contribute** (`half_match_contribution`) → each side's nullifiers + output commitments. The matcher assembles these into the joint (or, if you control both sides, build it directly).
4. **Build the witness** (`half_match_witness`) from the joint + your secrets + **all four Merkle paths**: the note's pool path, its **association** inclusion path, its **blocklist** exclusion proof, and the order's book path. `half_match_witness` runs `half_verify` locally, so a wrong joint or path is caught **before** GPU proving starts.
5. **Prove** each half (`statement: "half-match"`). The validated dual-4090 plus CPU-wrapper path is about **101 seconds per half** before queue/network overhead.
6. **Submit** `settleMatch2(proofSell, proofBuy, joint)`. The contract re-checks all roots, runs `verifyHalfMatch` on each side against the *same* joint, spends the four nullifiers, and mints the four outputs.

For an equal-size cross, Guided mode now runs steps 1–5 automatically in each party's signed-in browser. The party that completes the second half sees the wallet confirmation for step 6; the other browser detects the confirmed on-chain settlement and reconciles its local notes. Keep both browser tabs, matcher tunnels, and prover tunnels running throughout the four-minute match window.

> **Tip from the field:** `half_verify` is your friend. Assemble + validate **both** witnesses (instant) before proving anything; only spend prover time once both return `ok: true` .

Automatic settlement still fails closed unless both parties are online and each has a correctly denominated backing note with current pool/book paths and ASP-published association/blocklist paths. Placing a Guided order does not manufacture those prerequisites.

---

5. The compliance layer (ASP) — required for settlement

Every spent note must prove it is **not on the sanctions blocklist**. The contract holds only the two roots; the trees' leaves are published off-chain by the ASP. The owner sets the roots:

```
publishAssociationRoot(uint256 r)   // onlyOwner — ADVISORY, not enforced on spends
publishBlocklistRoot(uint256 r)     // onlyOwner — the sanctions IMT root, enforced
```

To make a note settle-able, the ASP:

1. Still builds an **association tree** and gives each note an inclusion path, because `associationRoot` remains a proof public input. The pool no longer checks it against the published root, so this step does not gate spending.
2. Builds the **blocklist IMT**; `blocklist_exclusion({leaves, target})` returns the `excl{Lo,Hi,PathElements,PathIndices}` non-membership proof + the root.
3. Publishes both roots on-chain.

On BSC testnet, Guided mode polls the active ASP and DarkPool roots every 15 seconds. A new deposit is **spendable straight away** — there is no operator approval batch to wait for. The trade button is disabled only if the backing note is on the active blocklist. Testers never copy commitments or configure roots.

`settleMatch2` requires `joint.blocklistRoot == currentBlocklistRoot`. It does *not* constrain `joint.associationRoot`, so a sanctioned note cannot clear, but an un-approved one can.

---

6. Tutorial D — Withdraw to a wallet

A withdrawal spends one or more private notes and pays a real wallet address. Before Guided mode can build it, the notes must be discoverable in the current DarkPool and the ASP source must provide current association-inclusion and blocklist-exclusion paths.

1. Select **Withdraw**, enter the same ERC-20 address, an integer base-unit amount, and an optional destination (your connected wallet is the default).
2. The browser selects notes and builds the transfer witness locally. It sends only the key-safe prover witness to the configured prover; the validated transfer path is about **121 seconds** before queue/network overhead.
3. Confirm the on-chain withdrawal. The destination, token, and amount are public, while the input note linkage remains hidden and its nullifier prevents a second spend.

```
out   = transfer_witness(spendingKey, withdrawal) // browser builds exact-operation auth
proof = prove("transfer", out.proverWitnessJson)  // no master key crosses the boundary
dp.withdraw(proof, roots, nullifiers, token, recipient, amount)
```

7. Troubleshooting (lessons from the field)

Symptom	Cause / fix
"Verify on-chain state" errors with an `eth_getLogs` message	The view RPC rejects wide `getLogs`. Set `localStorage['pravelle:indexerRpc']` to an archive RPC (or use the wired default).
`ERR_CONNECTION_REFUSED` for `:8799/health`	The local-development browser has no working loopback forward. Restart the tester tunnel. Hosted production does not use this loopback address.
"prover route requires X-Prover-Token"	Open the operator-provided token URL once on the same web origin, then reload. Each tester has a separate token; do not publish it.
`libomp.so.5: cannot open shared object file`	The prover service was launched without its CPU-wrapper runtime. Install `libomp5`, keep the pinned wrapper library directory in `LD_LIBRARY_PATH`, and restart the service.
`MPI_ERRORS_ARE_FATAL` / `NULL communicator`	An obsolete GPU PLONK wrapper is running. Use the current dual-GPU raw prover plus MPI-free CPU `export-plonk-proof` wrapper; retrying the GPU wrapper is not a fix.
`StaleNow` on placeOrder / settleMatch2	The proof's `now` aged past `NOW_WINDOW` (five minutes by default). Rebuild the witness with a fresh `now` and re-prove; raise `nowWindow` only as an explicit measured-latency override.
`UnknownRoot` / `UnknownAssociationRoot` on withdraw or settle	The witness roots must equal the *current* on-chain pool/book/association/blocklist roots. Publish the ASP roots first and rebuild paths after state changes.
`BadProof` on settle	A half-match proof doesn't verify against the joint. Re-check the side, the VK build, and that `half_verify` passed locally before proving.
`half-match-wallet is disabled`	Expected on the HTTP prover. Its verifier slot is finalized, but the route remains off until the witness no longer needs a master spending key.
`claim stark proof: No such file` (prover service)	Statement-name hyphen vs. underscore mismatch for the proof file (e.g. `half-match`). The prover service normalizes hyphens to underscores for the STARK output.

Symptom	Cause / fix
SSH dropped via the gateway (fake-IP `198.18.x`)	A TUN/VPN is rewriting DNS. Pin the gateway's real IP in the SSH `ProxyCommand` (`HostKeyAlias` to keep host-key verification).
`nonce too low` on rapid txs	Public RPC load-balancing serves stale nonces. Retry, or pace transactions.

8. Where to go next

> **Testnet status:** use test assets only. The current release has extensive internal adversarial review and full prover/verifier validation, but it has not completed an independent third-party audit or a production-mainnet launch.

- Dual-4090 validation — full timings, repeatability, and offline SDK wrapper replay.
- Wallet and orders spec — the committed-wallet + order model in depth.
- Settlement hardening — the split-proof settlement security properties.
- On-chain SNARK generation — how the PLONK verifier + VKs are generated.
- Matcher operations — running the relay.
- Security audit — findings and fixes.

Download this guide as PDF