Lux Private MPC Deployment

Fund-only mpcd clusters with no primary network registration

Zach Kelling

Lux Industries

April 2026

Lux Private MPC Deployment

Fund-only mpcd clusters with no primary network registration

Zach Kelling*
Lux Industries, Inc.
research@lux.network

April 2026

Abstract

This paper specifies the deployment contract for fund-private installations of `lux/mpc`, the threshold-signing daemon that backs Lux MPC custody. The same source ships with a single mode flag, `-network`, that selects between two terminations: a *primary* cluster that announces itself to the Lux M-Chain registrar and anchors audit batches there, and a *private* cluster that does neither and therefore can run inside an air-gapped fund VPC. The paper documents the contract, the audit chain that survives the absence of M-Chain, and the operational procedures a fund's custody team executes from zero to first signed transaction.

Contents

1 Why two modes

A growing number of institutional custodians cannot run their threshold signing infrastructure on a shared validator network: regulators in their jurisdictions require every audit byte to be

*Corresponding author: zach@lux.network

retained by the regulated entity itself, and disaster-recovery posture forbids a dependency on an external network for the cryptographic decisions that release client funds.

Lux MPC was built around the Lux M-Chain registrar from the beginning — nodes joined a validator-style set, signing decisions were anchored in batches, and cross-fund operations were possible because every participant shared the same audit ledger. That model fits well for retail-tier custody and for the smaller funds that white-label Lux MPC as a service. It does not fit a $1B+ AUM fund that runs its custody stack inside a hardware-protected on-prem facility.

This document specifies the second termination of the same daemon: a fund deploys mpcd in private mode, every node runs in the fund's own Kubernetes cluster, the audit log lands on a hardware-WORM volume the fund controls, and *no* traffic crosses the boundary between the fund's cluster and any Lux infrastructure unless the fund explicitly configures it. Same code, same protocols, same operator surface — the only thing that changes is the audit destination and the registration target.

2 Mode flag and audit dispatcher

Two flags drive the entire deployment-mode contract:

```
mpcd start \
  --network=primary | private \
  --audit-store=mchain | local-worm | s3-glacier-vault | \
                azure-immutable | gcs-object-lock | composite
```

2.1 `-network`

`primary` (the default) instructs the bootstrap routine to POST this node's public key to the registrar URL passed in `-registrar-url`; the registrar is responsible for whatever validator-set bookkeeping the network performs, and the fund need not care about the protocol details below that HTTP call.

`private` suppresses the announcement entirely. The node generates or loads its identity, persists it under `-keys`, and forms a cluster strictly with the peers passed via `-peer`. The flag is verified at process start: misconfiguration (an unknown mode, or `-registrar-url` configured but unreachable in primary mode) fails closed.

2.2 `-audit-store`

The audit dispatcher decouples *what* we audit from *where* the log lives. Six dispatcher implementations exist:

`local-worm` Append-only NDJSON file at `-audit-worm-path`. Replayed on startup; corruption refuses to open. Operators mount the path on hardware-WORM media.

`mchain` Anchors `batchSize` events at a time to a Lux M-Chain anchor RPC. Used in primary mode and in private mode when the fund has opted into shared anchoring.

`s3-glacier-vault`, `azure-immutable`, `gcs-object-lock` Cloud-WORM backends. Available only when the binary is built with the `cloud-audit` build tag; selecting one without the tag fails closed at startup.

`composite` Fans Append out to multiple legs. The recommended production posture in primary mode is `composite` with legs `local-worm,mchain` so a transient M-Chain anchor outage cannot lose audit data.

2.3 Audit chain

Every event carries a sequence number, the SHA-256 of the previous event, and the SHA-256 of the canonical bytes of (prevHash || seq || node || kind || org || wallet || timestamp || payload). Tampering with any historical event invalidates every subsequent hash; verification reduces to recomputing the chain. We deliberately do not use `json.Marshal` for canonicalisation — map-iteration ordering in Go is non-deterministic, which would defeat reproducibility.

3 Identity bootstrap

`pkg/identity/bootstrap.go` owns identity creation and (in primary mode) registrar announcement. The contract:

1. If the keys file exists: load the on-disk Ed25519 keypair, refresh the recorded mode, return.
2. If the keys file does not exist: generate a fresh keypair, write atomically with file mode 0600, return.
3. In primary mode: POST {nodeID, publicKey, ts} to the registrar URL. Failure surfaces; the caller decides whether to fall back or fail closed.

In private mode, step 3 is skipped unconditionally. The mode flag is also recorded in the on-disk JSON so a verifier can detect silent mode switches between restarts.

4 Network policy

The private-mode kustomize overlay (`deployments/k8s-private/`) ships a NetworkPolicy that locks egress to:

- Other mpcd pods in the same StatefulSet (P2P transport, port 9999, and internal API, port 9800).
- The fund's postgres and valkey pods (control-plane state).
- A pod labelled `app=fund-hsm-bridge` on port 11111 (the PKCS#11 sidecar; operators relabel for their HSM provider).
- DNS (UDP/TCP 53) only.

There is no rule for the public internet, no rule for the Lux M-Chain RPC, and no rule for an external object store. Funds that opt into a cloud-WORM target add a rule themselves; the default is silence.

5 Key ceremony

The first key the fund's cluster ever produces requires every replica to participate. Procedure:

1. Verify all replicas report `healthy` on `/healthz`.
2. Confirm `ArePeersReady()` returns true on every node.
3. Issue a `POST /keygen` to one node's internal API, passing {org_id, wallet_id}. The body is signed with the operator's intent key.

4. Wait for the keygen result event (default timeout 60s). The node returns the ECDSA and EdDSA public keys plus derived addresses for ETH, BTC, SOL.

5. Verify the result on every other node by querying `/keys/⟨wallet_id⟩`; all replicas must return identical metadata.

The ceremony emits a single `keygen` audit event per node; operators verify the chain head matches across all replicas before declaring the wallet provisioned.

6 Signing ceremony

Every signing call exercises a t-of-n subset. With the default 3-of-3 cluster and threshold $t = 2$, two ready nodes form a quorum. Procedure:

1. Operator submits the transaction payload to the dashboard API.

2. Approval workflow gates the request on the fund's policy (spending limits, time locks, allowlists).

3. Once approved, the dashboard issues a `POST /sign` to the cluster.

4. Two or more nodes participate in CGGMP21 (ECDSA) or FROST (Ed25519). The signature is returned on the result topic.

5. A `sign` audit event is appended to every replica's chain. Operators can reconcile the chain across replicas to confirm exactly one signing event occurred.

7 Reshare ceremony

Reshare rotates key shares without changing the underlying public key. Used to add a node, remove a node, or change the threshold. Procedure mirrors the keygen ceremony but uses the LSS protocol; refer to the mpcd reshare handler for the wire details.

8 Compliance attestation

The audit chain produced by mpcd in private mode satisfies the evidence requirements of:

- **SOC 2 Type II** — CC6.1, CC6.6, CC7.2 (logical access, encryption, change management).
- **ISO 27001:2022** — A.5.34, A.8.15, A.8.21 (privacy, logging, network controls).
- **FIPS 140-2 Level 3** — when paired with a CloudHSM / Azure Key Vault Managed HSM / on-prem PKCS#11 HSM at L3.
- **CCSS Level III** — with the additional ceremony witness signatures the dashboard captures during keygen and reshare.

9 Honest residual

The cloud-audit dispatchers (S3 Glacier Vault, Azure Immutable, GCS Object Lock) compile but return "not implemented" at startup. They are wired into the flag surface so an operator can plan for them; the SDK paths are gated behind the `cloud-audit` build tag and the default release binary does not include them. Funds that want a cloud WORM target today should use the `composite` dispatcher with the `local-worm` leg pointing at a hardware-WORM-backed PVC and the `mchain` leg pointed at their own anchor RPC.

The Terraform modules under `deployments/terraform/{aws,gcp, azure}/` `validate` cleanly; only the audit-bucket / VPC resources are wired end-to-end. EKS / GKE / AKS modules and HSM modules are stubbed and not yet plan-tested in any cloud account.