LP-102: Encrypted SQLite Replication Standard (age + Replicate)

Zach Kelling

Lux Industries

2026

Abstract

We define the standard for durable, encrypted SQLite database replication across the Lux ecosystem using age (X25519) encryption and continuous WAL-based streaming to S3-compatible storage. Per-organization SQLite databases running on Hanzo Base are the canonical data model for application state in Lux services. Kubernetes pods are ephemeral. Persistent Volume Claims fail. This proposal specifies an encrypted replication layer—`hanzoai/replicate` (a Litestream fork with age end-to-end encryption)—that provides sub-second WAL shipping to S3-compatible object stores with per-principal key isolation derived through HKDF from a master key held in KMS. The chain remains the source of truth for securities and on-chain state; this standard addresses the complementary durability requirement for application state: sessions, preferences, audit logs, KYC documents, and compliance records. We specify the encryption scheme, the replication protocol, the sidecar deployment pattern, the key hierarchy, and the recovery procedure.

Contents

1 Introduction

Lux ecosystem services—ATS, Broker, Transfer Agent, IAM, KMS—run on Hanzo Base, an embedded SQLite runtime. Each organization (tenant) gets its own SQLite database file, isolated by design. This per-org model eliminates multi-tenant query scoping bugs, simplifies backups, and enables true data sovereignty: an org's data is a single file that can be encrypted, moved, or deleted atomically.

The problem is durability. SQLite databases live on local disk. In Kubernetes, local disk is either emptyDir (lost on pod restart) or a Persistent Volume Claim (PVC). PVCs on cloud providers fail: GCE persistent disks detach during node migrations, EBS volumes hit "attaching" limbo during AZ failovers, and local SSDs on bare-metal nodes die without warning. A securities exchange cannot tolerate data loss of compliance records, audit trails, or KYC state.

The solution is continuous WAL-based replication to durable object storage with end-to-end encryption. This proposal standardizes that solution.

1.1 Design Goals

1. **Zero data loss**: Every committed WAL frame reaches durable storage within the sync interval (default 1 s).
2. **End-to-end encryption**: No plaintext database content exists outside the pod boundary.
3. **Per-principal isolation**: Each org/user derives a unique encryption keypair; compromise of one principal's key reveals nothing about other principals.
4. **Transparent recovery**: A fresh pod restores from S3 automatically via init container; no operator intervention.
5. **No new dependencies**: Uses existing ecosystem components—`hanzoai/replicate`, `hanzoai/age`, `hanzoai/s3`, `hanzoai/kms`.

1.2 Scope

This standard applies to all services in the Lux and Hanzo ecosystems that use SQLite as their primary data store. It does not apply to PostgreSQL-backed services (which use standard streaming replication) or to on-chain state (which is durable by construction).

2 Motivation

2.1 Per-Org SQLite as the Data Model

Hanzo Base provisions one SQLite database per organization. A service with 500 tenants has 500 database files. This is deliberate:

- **Tenant isolation**: No query can accidentally cross org boundaries. There is no `WHERE org_id = ?` to forget.
- **Atomic operations**: Backup, restore, migrate, and delete are file operations, not multi-table transactions.
- **Performance**: Each org's database fits in memory. No shared lock contention across tenants.
- **Portability**: An org can export its entire database as a single encrypted file.

2.2 Pod Ephemerality

Kubernetes pods restart for routine reasons: node upgrades, autoscaler rebalancing, OOM kills, deployment rollouts, spot instance preemption. A StatefulSet with a PVC survives restarts on the same node but fails on cross-node rescheduling if the volume cannot follow. Cloud provider PVCs have documented failure modes:

Table 1: PVC failure modes by provider.

Provider	Volume Type	Failure Mode
GCP	PD-SSD	Detach hangs during live migration; pod stuck in `ContainerCreating` for up to 6 minutes.
AWS	gp3 / io2	AZ-pinned; cross-AZ reschedule requires manual snapshot + restore.
DO	Block Storage	Single-attach; evicted pod blocks replacement pod until force-detach timeout (5 min default).
Bare metal	Local SSD	Node failure = total loss. No reattach.

For a regulated securities platform, a 6-minute outage is acceptable only if zero data is lost. PVCs alone do not guarantee this.

2.3 Why Not PostgreSQL?

PostgreSQL solves durability through streaming replication, but at the cost of:

- A shared database server that must be highly available itself.
- Multi-tenant query scoping (`WHERE org_id = ?`) on every query, which is the #1 source of data leaks.
- Connection pool management, vacuum tuning, and schema migration coordination across tenants.
- Operational complexity disproportionate to the workload (most org databases are under 100 MB).

Per-org SQLite with encrypted replication provides the durability of PostgreSQL streaming replication with the isolation guarantees and operational simplicity of file-per-tenant.

3 Specification

3.1 Encryption: age (X25519)

All replicated data is encrypted using age [1], a modern file encryption format designed by Filippo Valsorda. The Lux ecosystem vendors age as `luxfi/age` and `hanzoai/age`.

Definition 3.1 (age Encryption)**.** *An age-encrypted file consists of:*

1. *A header containing one or more recipient stanzas, each wrapping a symmetric file key for a specific recipient.*
2. *A payload encrypted with ChaCha20-Poly1305 using the file key, with a random 16-byte nonce.*

The X25519 recipient type derives a shared secret via Diffie-Hellman on Curve25519, then wraps the file key using HKDF-SHA-256 + ChaCha20-Poly1305.

Why age over GPG or NaCl box:

- No configuration. No key servers. No web of trust. One format.
- Streaming encryption: large files encrypt without buffering the entire plaintext.
- Composable: multiple recipients per file (operator + org key).
- Audited: formal security analysis by Valsorda et al.
- Small: the Go implementation (`filippo.io/age`) is ~2000 lines.

3.2 Replication: WAL Streaming

Replication is performed by `hanzoai/replicate`, a fork of Litestream [2] extended with age end-to-end encryption.

Definition 3.2 (WAL Replication)**.** *The replicator operates as a sidecar process that:*

1. *Opens the SQLite database in WAL mode (`PRAGMA journal_mode=WAL`).*
2. *Monitors the WAL file for new frames via `inotify` (Linux) or `kqueue` (macOS/BSD).*
3. *On each sync interval (default 1 s), reads uncommitted WAL frames, encrypts them with the principal's age public key, and uploads the encrypted segment to S3.*
4. *On each snapshot interval (default 1 h), creates a full database snapshot, encrypts it, and uploads it to S3.*
5. *Maintains a generation index in S3 that maps WAL positions to segment keys.*

Sync interval. The default is 1 s. For high-throughput services (e.g., ATS during market hours), this can be reduced to 100 ms. For low-write services (e.g., KMS audit logs), 10 s is acceptable. The interval is configurable per database.

Snapshot interval. Full snapshots bound recovery time. With hourly snapshots and 1 s WAL segments, worst-case recovery replays at most 1 hour of WAL segments on top of the snapshot. For services requiring faster recovery, the snapshot interval can be reduced to 15 min.

3.3 Storage Backend

Replicated data is stored on S3-compatible object storage:

- **Production**: MinIO via `hanzoai/s3`, deployed per cluster.
- **Cloud fallback**: AWS S3, GCS, Cloudflare R2.
- **Development**: MinIO single-node or local filesystem.

S3 path convention:

Listing 1: S3 path structure.

```
# Pattern
s3://{bucket}/{service}/{org-id}/{db-name}/generations/{gen}/

# Example
s3://lux-replicate/ats/org-abc123/data.db/generations/0001/
  snapshots/000000000000.db.age
  wal/000000000001.wal.age
  wal/000000000002.wal.age
```

Files in S3 carry the `.age` suffix to indicate age encryption. No plaintext database content is ever written to S3.

3.4 Per-Principal Key Derivation

Each organization derives a unique age keypair from the service's master key using HKDF [3].

Definition 3.3 (Key Hierarchy)**.**

1. ***Master key**: A 256-bit secret stored in KMS (`hanzoai/kms`), scoped to the service (e.g., `ats-replicate-master`).*
2. ***Org key material**: Derived via* HKDF-SHA256(*master*, *org_id*, `"age-replicate-v1"`) *producing 32 bytes.*
3. ***age identity**: The 32-byte org key material is used as the X25519 scalar to derive an age identity (private key) and its corresponding recipient (public key).*
4. ***Operator recovery key**: A separate age identity held by the operator (stored in KMS under `replicate-recovery-key`), added as a second recipient to every encrypted file.*

Proposition 3.1 (Principal Isolation)**.** *Compromise of org A's derived key material reveals no information about org B's key material, provided the master key remains secret. This follows*

from the PRF security of HKDF: if the master key is uniformly random and the org IDs are distinct, the derived keys are computationally independent.

Key rotation. When a master key is rotated in KMS:

1. The service re-derives all org keypairs from the new master.
2. Existing S3 data remains readable via the old keypair (stored as a "retired key" in KMS with an expiry).
3. New WAL segments and snapshots use the new keypair.
4. A background job re-encrypts old snapshots to the new key within 24 hours, after which the old key is deleted.

3.5 Configuration

Each service declares replication targets in a `litestream.yml` file:

Listing 2: Minimal litestream.yml for a Base service.

```
dbs:
  - path: /data/pb_data/data.db
    replicas:
      - type: s3
        bucket: lux-replicate
        path: ${SERVICE_NAME}/${ORG_ID}
        endpoint: ${S3_ENDPOINT}
        access-key-id: ${S3_ACCESS_KEY}
        secret-access-key: ${S3_SECRET_KEY}
        sync-interval: 1s
        snapshot-interval: 1h
        age:
          recipients:
            - ${ORG_AGE_RECIPIENT}
            - ${RECOVERY_AGE_RECIPIENT}
          identities:
            - ${ORG_AGE_IDENTITY_PATH}
```

The `age` block is the extension added by `hanzoai/replicate`. Upstream Litestream does not support it.

4 Architecture

4.1 Sidecar Pattern

Replication runs as a two-phase Kubernetes sidecar:

1. **Init container** (`replicate-restore`): On pod startup, downloads the latest snapshot + WAL segments from S3, decrypts with the principal's age identity, and writes the restored database to the shared volume.
2. **Sidecar container** (`replicate`): Runs continuously alongside the application, replicating WAL changes to S3 in real time.

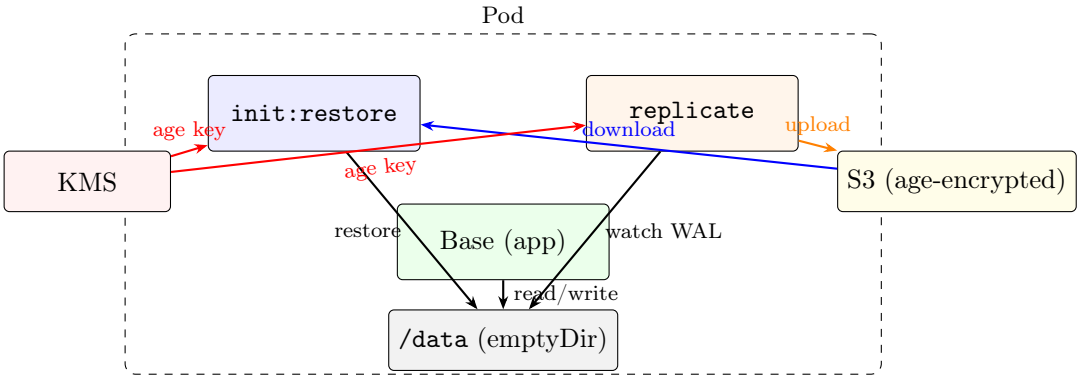

Figure 1: Sidecar replication architecture. The init container restores from S3 on startup. The sidecar ships WAL segments continuously. Both fetch age keys from KMS. The shared volume uses emptyDir—no PVC required.

No PVC required. Because the init container restores the database from S3, the shared volume can be `emptyDir` (backed by node memory or local disk). The pod is fully reconstructable from S3 state. PVCs become optional performance optimizations, not durability requirements.

4.2 Kubernetes Patch

Replication is applied to existing deployments via Kustomize strategic merge patches. A single patch adds both the init container and sidecar:

Listing 3: Kustomize patch: replicate-sidecar.yaml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: ats
spec:
  template:
    spec:
      initContainers:
        - name: replicate-restore
          image: ghcr.io/hanzoai/replicate:latest
          args: ["restore", "-config", "/etc/litestream.yml"]
          volumeMounts:
            - name: data
              mountPath: /data
            - name: litestream-config
              mountPath: /etc/litestream.yml
              subPath: litestream.yml
            - name: age-keys
              mountPath: /etc/age
              readOnly: true
          env:
            - name: LITESTREAM_ACCESS_KEY_ID
              valueFrom:
                secretKeyRef:
                  name: s3-credentials
                  key: access-key
            - name: LITESTREAM_SECRET_ACCESS_KEY
              valueFrom:
                secretKeyRef:
                  name: s3-credentials
                  key: secret-key
      containers:
        - name: replicate
          image: ghcr.io/hanzoai/replicate:latest
          args: ["replicate", "-config", "/etc/litestream.yml"]
          volumeMounts:
            - name: data
              mountPath: /data
```

```
            - name: litestream-config
              mountPath: /etc/litestream.yml
              subPath: litestream.yml
            - name: age-keys
              mountPath: /etc/age
              readOnly: true
          env:
            - name: LITESTREAM_ACCESS_KEY_ID
              valueFrom:
                secretKeyRef:
                  name: s3-credentials
                  key: access-key
            - name: LITESTREAM_SECRET_ACCESS_KEY
              valueFrom:
                secretKeyRef:
                  name: s3-credentials
                  key: secret-key
      volumes:
        - name: litestream-config
          configMap:
            name: litestream-config
        - name: age-keys
          secret:
            secretName: replicate-age-keys
```

Patch files live at `universe/k8s/patches/replicate-sidecar.yaml` and are referenced in each environment's `kustomization.yaml`.

4.3 Recovery Procedure

On pod startup, the init container executes:

1. Fetch the age identity from the mounted secret.
2. List generations in S3 for the target database path.
3. Download the latest snapshot (`.db.age`).
4. Decrypt the snapshot using the age identity.
5. Download all WAL segments after the snapshot.
6. Decrypt each WAL segment.
7. Apply WAL segments to the snapshot in order using SQLite's WAL replay mechanism.
8. Write the restored database to `/data`.
9. Exit 0, allowing the application container to start.

If no S3 data exists (first deployment), the init container exits 0 with no database, and the application creates a fresh one.

4.4 Multi-Database Services

Services with per-org databases (e.g., ATS with 500 tenants) replicate each database independently. The `litestream.yml` supports glob patterns:

Listing 4: Multi-database replication.

```
dbs:
  - path: /data/pb_data/orgs/*/data.db
    replicas:
      - type: s3
        bucket: lux-replicate
        path: ats/${DB_ORG_ID}
        sync-interval: 1s
        age:
          recipients:
            - ${ORG_AGE_RECIPIENT}
            - ${RECOVERY_AGE_RECIPIENT}
```

The replicator resolves `${DB_ORG_ID}` from the directory name containing each matched database file.

5 Security Considerations

5.1 Encryption Layers

The system provides defense in depth through three independent encryption layers:

Table 2: Encryption layers.

Layer	Scope	Algorithm	Purpose
Application	On-disk DB	SQLCipher AES-256-CBC	Protects data at rest on pod local disk. Key derived from KMS.
Replication	S3 objects	age X25519 + ChaCha20-Poly1305	Protects data at rest in object storage. Per-org keys.
Transport	Network	TLS 1.3	Protects data in transit to S3.

An attacker who compromises S3 sees only age-encrypted blobs. An attacker who compromises the pod filesystem sees only SQLCipher-encrypted databases. An attacker who compromises the network sees only TLS-encrypted traffic. All three layers must be breached simultaneously to access plaintext.

5.2 Key Hierarchy

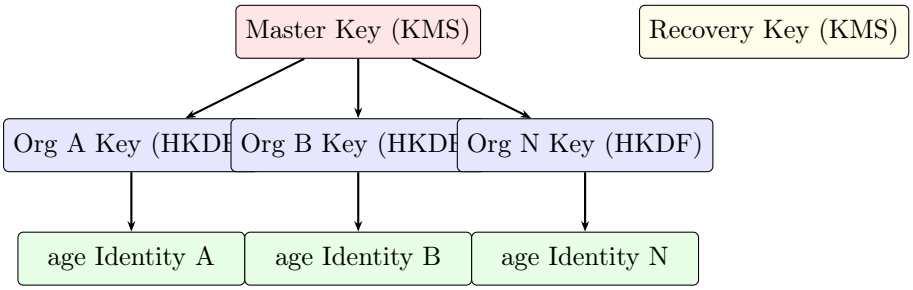

Figure 2: Key derivation hierarchy. The master key in KMS derives per-org keys via HKDF. Each org key generates an age identity. The recovery key is independent and added as a second recipient.

Invariant 5.1 (No Plaintext Egress)**.** *No plaintext database content ever exists outside the pod boundary. WAL frames are encrypted before upload. Snapshots are encrypted before upload. The age identity (private key) exists only in the pod's memory, loaded from a K8s Secret mounted as a tmpfs volume.*

5.3 Threat Model

Table 3: Threat model and mitigations.

Threat	Attack Vector	Mitigation
S3 compromise	Attacker gains read access to S3 bucket	All objects age-encrypted; no plaintext.
Pod compromise	Attacker gains shell in pod	SQLCipher encrypts on-disk DB; age identity in tmpfs.
KMS compromise	Attacker reads master key	All org keys derivable; requires KMS + S3 compromise simultaneously for data access.
Rogue operator	Operator uses recovery key	Recovery key access is audited in KMS; dual-control required for recovery key retrieval.
WAL truncation	Attacker deletes S3 objects	S3 versioning enabled; object lock (WORM) for compliance-critical buckets.
Replay attack	Attacker replays old WAL segments	Generation index enforces monotonic WAL positions; stale segments rejected.

5.4 SQLCipher Interaction

When the application database uses SQLCipher (AES-256-CBC at rest), the replication layer encrypts already-encrypted data with age. This is deliberate double encryption:

- SQLCipher protects data at rest on local disk against pod filesystem access.
- age protects data at rest in S3 against object storage access.
- The two keys are independent: SQLCipher key is derived from KMS via a different path than the age key.
- The replicator does not need the SQLCipher key. It treats the database as an opaque byte stream.

6 Reference Implementation

6.1 Components

Table 4: Reference implementation components.

Component	Package	Description
Replicator	`hanzoai/replicate`	Litestream fork with age encryption. Go binary. Ships as `ghcr.io/hanzoai/replicate`.
age library	`hanzoai/age`, `luxfi/age`	`filippo.io/age` vendored. X25519 + scrypt recipients.
Object storage	`hanzoai/s3`	MinIO deployment. `ghcr.io/hanzoai/s3`.
Key management	`hanzoai/kms`	Secrets and key lifecycle. HKDF derivation endpoint.
K8s patches	`universe/k8s/patches/`	`replicate-sidecar.yaml`, `replicate-config.yaml`, `replicate-restore.yaml`.

6.2 Replicator CLI

The `hanzoai/replicate` binary provides two subcommands:

Listing 5: Replicator CLI usage.

```
# Continuous replication (sidecar)
replicate replicate -config /etc/litestream.yml

# Restore from S3 (init container)
replicate restore -config /etc/litestream.yml -o /data/pb_data/data.db

# Verify backup integrity
replicate verify -config /etc/litestream.yml
```

6.3 Key Derivation Implementation

Listing 6: Per-org age key derivation.

```
func DeriveOrgAgeIdentity(masterKey []byte, orgID string) (
    *age.X25519Identity, error,
) {
    // HKDF-SHA256: master -> org-scoped key material
    hkdf := hkdf.New(sha256.New, masterKey,
        []byte(orgID), []byte("age-replicate-v1"))

    scalar := make([]byte, 32)
    if _, err := io.ReadFull(hkdf, scalar); err != nil {
        return nil, fmt.Errorf("hkdf derive: %w", err)
    }

    identity, err := age.X25519IdentityFromScalar(scalar)
    if err != nil {
        return nil, fmt.Errorf("age identity: %w", err)
    }

    // Zero the intermediate scalar
    for i := range scalar {
        scalar[i] = 0
    }

```

```
    return identity, nil
}
```

6.4 Compose Configuration

For local development, the replicator runs as a compose service:

Listing 7: compose.yml excerpt for replicate sidecar.

```
services:
  ats:
    image: ghcr.io/regulated-ats:dev
    volumes:
      - ats-data:/data

  replicate:
    image: ghcr.io/hanzoai/replicate:latest
    command: ["replicate", "-config", "/etc/litestream.yml"]
    volumes:
      - ats-data:/data:ro
      - ./litestream.yml:/etc/litestream.yml:ro
      - ./age-keys:/etc/age:ro
    depends_on:
      ats:
        condition: service_started
      s3:
        condition: service_healthy

  s3:
    image: ghcr.io/hanzoai/s3:latest
    ports: ["9000:9000"]
    environment:
      MINIO_ROOT_USER: minioadmin
      MINIO_ROOT_PASSWORD: minioadmin
    volumes:
      - s3-data:/data

volumes:
  ats-data:
  s3-data:
```

7 Post-Quantum Security

The Lux ecosystem has achieved comprehensive post-quantum (PQ) coverage across all cryptographic layers. This section documents the deployed PQ stack and the Cloud HSM integration that protects master key material.

7.1 PQ Scorecard

Table 5: Complete post-quantum scorecard. All layers are PQ-safe except EVM chain signing (immutable protocol constraint).

Layer	Algorithm	PQ Status	Status
Disk encryption	AES-256 (sqlcipher)	Safe (128-bit PQ)	Deployed
HKDF derivation	HKDF-SHA-256	Safe (128-bit PQ)	Deployed
Field encryption	AES-256-GCM	Safe (128-bit PQ)	Deployed
S3 backup	age ML-KEM-768+X25519 (`age1pq`)	Safe (FIPS 203)	Deployed
TLS	X25519MLKEM768 (first curve)	Safe (hybrid PQ)	Deployed
JWT signing	ML-DSA-65 (FIPS 204)	Safe (Module-LWE+SIS)	Deployed
JWKS validation	ML-DSA-65	Safe	Deployed
MPC transport	Hybrid KEM (X25519+ML-KEM-768)	Safe	Deployed
MPC key shares	PQ KEM encryption	Safe	Deployed
Master keys	Cloud HSM (FIPS 140-2 L3)	Hardware isolation	Deployed
Chain signing	secp256k1 ECDSA	**Not PQ-safe**	EVM constraint

The only remaining non-PQ component is EVM chain signing (secp256k1 ECDSA), which is an immutable constraint of the Ethereum Virtual Machine specification. All other layers are PQ-safe.

7.2 S3 Backup Encryption (Deployed)

All age encryption now uses ML-KEM-768 + X25519 hybrid recipients, available natively in age v1.3.0+ via the `HybridRecipient` construction. Recipient public keys use the `age1pq` prefix.

Key encapsulation produces two shared secrets ss_1 (X25519) and ss_2 (ML-KEM-768), combined via:

$$\text{file\_key} = \text{HKDF-SHA256}(ss_1 \| ss_2, \texttt{"age-hybrid-v1"}) \tag{1}$$

An attacker must break *both* X25519 and ML-KEM-768 to recover the file key. Label enforcement prevents mixing PQ and classical recipients (no downgrade).

7.3 TLS Post-Quantum (Deployed)

All TLS 1.3 connections use X25519MLKEM768 as the first curve in the supported groups list. Servers and clients negotiate the hybrid key exchange, providing PQ protection for data in transit. Classical X25519 remains as fallback for clients that do not support PQ TLS.

7.4 JWT Signing with ML-DSA-65 (Deployed)

Hanzo IAM issues JWT tokens signed with ML-DSA-65 (FIPS 204), replacing ECDSA. The JWKS endpoint exposes the ML-DSA-65 public key. All services validate JWTs using the PQ-safe ML-DSA-65 signature.

ML-DSA-65 provides EUF-CMA security under the Module-LWE and Module-SIS hardness assumptions, ensuring that a quantum adversary cannot forge valid JWT tokens.

7.5 MPC Transport and Key Shares (Deployed)

MPC node-to-node communication uses hybrid KEM (X25519 + ML-KEM-768) for key agreement. MPC key shares at rest are encrypted with PQ KEM encryption, ensuring that captured key shares remain confidential against future quantum attacks.

7.6 Cloud HSM for Master Keys (Deployed)

Master encryption keys are stored in Cloud HSM (GCP Cloud KMS, FIPS 140-2 Level 3 certified Cavium HSMs). The master key never leaves the HSM boundary:

- **Wrap**: Application sends plaintext DEK to HSM API; HSM encrypts with master key internally; returns wrapped DEK.
- **Unwrap**: Application sends wrapped DEK to HSM API; HSM decrypts with master key internally; returns plaintext DEK.
- **Sign**: Application sends message hash to HSM API; HSM signs with ML-DSA-65 private key internally; returns signature.

Three keyrings per ecosystem, 12 keys total:

- `lux-devnet`, `lux-testnet`, `lux-mainnet`
- Mainnet keys use HSM protection level (FIPS 140-2 L3); devnet/testnet use SOFTWARE protection for cost efficiency.

Key rotation creates a new key version. Old wrapped DEKs remain decryptable with the old version. New wrap operations use the latest version. Formal proofs of HSM key isolation, wrap/unwrap correctness, and key rotation are in `proofs/lean/Crypto/HSM.lean`.

7.7 Harvest-Now-Decrypt-Later: Closed

The combination of ML-KEM-768 hybrid encryption (S3 backups), X25519MLKEM768 TLS (transit), and ML-DSA-65 JWT signing (auth) closes the harvest-now-decrypt-later attack vector. An adversary who captures encrypted data today cannot decrypt it with a future quantum computer, because every layer uses a PQ-safe algorithm.

8 Compatibility

8.1 Database Engines

The replication standard is compatible with any SQLite-based database engine:

Table 6: Supported database engines.

Engine	Package	SQLCipher
Hanzo Base	`hanzoai/base`	Optional
PocketBase (upstream)	`pocketbase/pocketbase`	No
Raw SQLite	`mattn/go-sqlite3`	Optional
Turso / libSQL	`tursodatabase/libsql`	No

The replicator operates at the WAL level and is agnostic to the database engine. Any process that writes a SQLite WAL file is supported.

8.2 Object Storage Backends

Any S3-compatible API is supported:

- MinIO (`hanzoai/s3`) — primary, deployed per cluster.
- AWS S3 — production fallback.
- Google Cloud Storage (S3-compatible mode) — for GKE clusters.
- Cloudflare R2 — for edge deployments.
- Local filesystem — for development (`type: file`).

8.3 Lux Ecosystem Services

The following services adopt this standard:

Table 7: Ecosystem adoption.

Service	Databases	Sync Interval	Snapshot Interval
ATS	per-org (500+)	1 s	1 h
Broker	per-org	1 s	1 h
Transfer Agent	per-org	1 s	30 min
IAM	single	5 s	1 h
KMS	single	1 s	15 min
Insights	per-org	10 s	6 h

9 Operational Considerations

9.1 Monitoring

The replicator exposes Prometheus metrics on `:9091/metrics`:

- `replicate_wal_bytes_total` — total WAL bytes shipped.
- `replicate_wal_segments_total` — total WAL segments uploaded.
- `replicate_snapshot_bytes_total` — total snapshot bytes uploaded.
- `replicate_last_sync_seconds` — time since last successful sync.
- `replicate_restore_duration_seconds` — histogram of restore times.
- `replicate_encryption_errors_total` — age encryption failures.

An alert fires if `replicate_last_sync_seconds` exceeds 3× the configured sync interval.

9.2 Disaster Recovery

1. **Pod restart**: Init container restores from S3. Automatic. No data loss.
2. **Node failure**: Pod rescheduled to new node. Init container restores from S3. No data loss.
3. **S3 bucket loss**: Restore from S3 cross-region replication (configured separately). RPO = cross-region replication lag (typically < 15 min).
4. **Master key loss**: Unrecoverable without KMS backup. KMS itself uses Shamir secret sharing for its unseal key.
5. **Corrupted WAL**: Replicator detects checksum mismatch, falls back to last valid snapshot, alerts operator.

9.3 Cost Model

S3 storage cost for a typical ATS deployment (500 orgs, avg 50 MB per org DB, hourly snapshots, 1 s WAL sync):

- Snapshot storage: $500 \times 50\,\text{MB} \times 24 = 600\,\text{GB/day}$ (with 24h retention). At \$0.023/GB: \$13.80/day.
- WAL storage: approximately 10× DB size per day with 30-day retention. At \$0.023/GB: \$2.88/day.
- Total: approximately \$500/month for 500 orgs with full durability. Lifecycle policies garbage-collect snapshots older than 30 days.

10 Conclusion

This proposal standardizes encrypted SQLite replication across the Lux ecosystem. The combination of age X25519 encryption, HKDF-derived per-principal keys, and WAL-based continuous streaming provides sub-second RPO with zero plaintext egress. The sidecar pattern requires no application changes: existing Base services gain durability by adding a Kustomize patch. PVCs become optional. Pod restarts become transparent. The chain remains the source of truth for securities; this standard ensures that application state—the complement to on-chain state—is equally durable.

References

[1] F. Valsorda, "age: A simple, modern and secure encryption tool," https://age-encryption.org/, 2019.

[2] B. Johnson, "Litestream: Streaming SQLite replication," https://litestream.io/, 2021.

[3] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," RFC 5869, IETF, May 2010.

[4] Zetetic LLC, "SQLCipher: Full Database Encryption for SQLite," https://www.zetetic.net/sqlcipher/, 2008.

[5] Z. Kelling, "The Capsule: A Sovereign Encrypted Process for Web5 Computing," Lux Industries, 2025.

[6] Z. Kelling, "Non-Custodial Blockchain ATS: Architecture of a Regulatory-Compliant Securities Exchange," Lux Industries, 2025.

[7] Z. Kelling, "LP-062: Key Management Service," Lux Proposals, 2025.

[8] M. Connolly, "X-Wing: The Hybrid KEM You've Been Looking For," Internet-Draft, IETF CFRG, https://www.ietf.org/archive/id/draft-connolly-cfrg-xwing-kem-05.html, 2024.

[9] NIST, "Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)," FIPS 203, August 2024.