Zaxon Working Group
Request for Discussion and Implementation Record

ZDS 0003
Security Assessment

Zaxonlite Security and Trust Plan

STATE
committed

CREATED
2026-07-21

LAST UPDATED
2026-07-21

INTENDED STATUS
Remediated

AUTHORS
paxos-zig project

DISCUSSION

Repository security assessment, trust model, and remediation record for Zaxonlite

LABELS
security, zaxonlite

Status of This Memo

This document is an internal Zaxon discussion record authored in Typst and tracked in git. It intentionally follows RFC-style structure so the design scope, rationale, trade-offs, and operational constraints remain explicit. Documents that still use the placeholder number XXXXX are provisional drafts. Numbered ZDS documents are part of the project record.

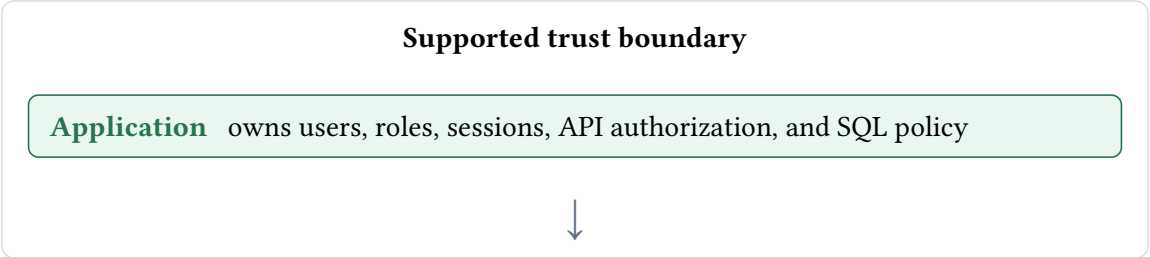
Contents

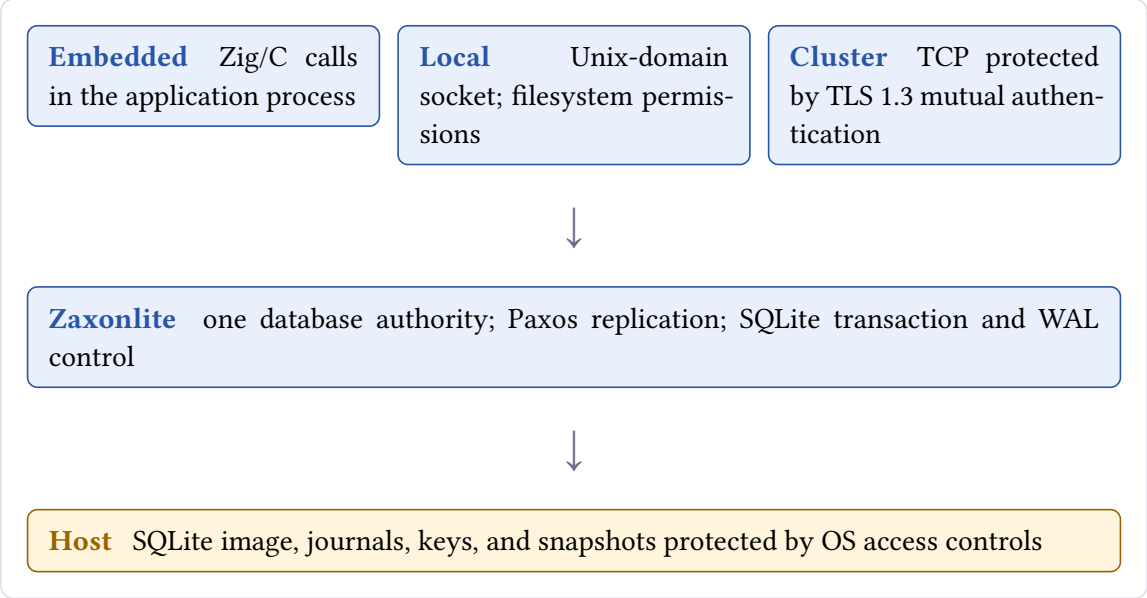
1. Executive decision

Zaxonlite is an embedded distributed SQLite database, not a database server for unrelated users, tenants, or database roles. The application is the security principal. It authenticates its users, applies its own roles and permissions, chooses which SQL to issue, and owns the process or local socket through which it reaches Zaxonlite. Zaxonlite supplies replication, durability, and a narrow transport boundary between configured nodes.

This changes the earlier analysis in important ways:

1. Internal role-based access control, separate reader/writer/admin identities, and a database user catalogue are not product requirements.
2. Local service should use a Unix-domain socket protected by owner/group/mode rather than a loopback TCP socket. Production TCP uses mutual TLS. The CLI's explicit `--dev-psk` exception is confined to numeric loopback for a one-machine quickstart and does not claim confidentiality or node identity.
3. Cluster enrollment should be a simple, one-time token exchange followed by per-node mTLS. It should resemble Incus enrollment: one explicitly configured issuer signs a bounded CSR with the operator's CA key, without introducing dynamic membership, database users, or a general identity-policy system.
4. SQL containment is not a hostile-tenant sandbox. A narrow SQLite authorizer is still needed to preserve Zaxonlite's outer transaction, WAL-capture settings, and reserved metadata when a trusted application has a bug or SQL-injection flaw. Loadable extensions are already compiled out of the bundled SQLite.
5. Assurance should resemble SQLite: deterministic local tests, crash campaigns, fuzzing, fault injection, and long-running soak tests. CI integration is intentionally deferred until the project adopts a CI system.





Security status
Production TCP is fail-closed mTLS. PSK-only TCP requires explicit `--dev-psk`, an owner-only provider, and numeric loopback for the listener and every peer; plaintext TCP remains solely failpoint-gated. Automated enrollment is intentionally small and static-membership-aware: an existing mTLS principal requests a bundle for one configured node, that node creates its key and CSR locally, and the issuer consumes the token durably before signing. Operators still distribute the node-ID revocation file out of band.

1.1. Post-implementation review · 21 July 2026

Finding	State	Reviewed implementation
SEC-001	Closed for production	TLS 1.3 mutual authentication is mandatory for production TCP; peer CN is bound to configured node ID and clients reject a server certificate that is not a canonical node principal. The Zig and C cluster facades accept the same provider-file TLS identity. The CLI-only `--dev-psk` exception is numeric-loopback-only and explicitly disclaims confidentiality and unique node identity.
SEC-002	Closed	`ZXP1` retains the already-chosen stop sign. Snapshot activation requires matching reports from a voter read quorum plus exact manifest and image hashes; no file signatures or second consensus phase.
SEC-003	Closed	Global and per-peer admission caps, handshake and idle deadlines, shutdown cancellation, and immediate node-ID revocation teardown bound established storage sockets. The raw TLS-passthrough gateway separately caps concurrent connections at 128.
SEC-004	Bounded	Frames, sender queues, held envelopes, explicit transaction input, and snapshot/backup transfers have checked limits.

Finding	State	Reviewed implementation
		Larger aggregate stress campaigns remain release evidence rather than a new scheduler.
SEC-005	Closed	`sqlite3_set_authorizer` protects transaction ownership, attached databases, `__zaxon_*`, and capture-changing pragmas; the capture contract is rechecked and loadable extensions are compiled out.
SEC-006	Closed pragmatically	Remote SQL is limited to 1 MiB, 10,000 rows, 16 MiB of copied result data, and 10 million SQLite VM instructions by default. Results remain bounded materialized JSON rather than adding a streaming query protocol. Embedded callers remain unlimited by default.
SEC-007	Closed	Single-node local service uses an owner-only 0600 Unix-domain socket. Production TCP uses mTLS; the disclosed `--dev-psk` exception is numeric-loopback-only and intended for local development.
SEC-008	Closed pragmatically	`enrollment.zig` persists only a domain-separated token hash, binds the bundle to database, issuer, and configured node ID, enforces a finite expiry, and consumes it through a synced `pending` to `used` rename before signing. The joiner generates a P-256 key and signed CSR locally; the issuer discards CSR extensions and returns a one-year client/server TLS certificate. Replay, expiry, identity, permissions, and end-to-end mTLS tests pass.
SEC-009	Substantially closed	Data roots are narrowed to 0700; PSK and private-key providers must be regular non-symlink 0600 files; atomic replacement and directory durability remain centralized.
SEC-010	Closed	The C ABI remains small, exposes provider-path TLS for the cluster facade, and checks counts, lengths, partial TLS configuration, conversions, initialized outputs, and ownership without adding safeclib.
SEC-011	Closed pragmatically	PSK-only redirects resolve only to caller seeds. Under mTLS an advertised address may leave the seed list, but the new connection must present the cluster-CA certificate `zaxon-node-<advertised-id>` before the request is replayed; redirect depth remains bounded.
SEC-013	Disclosed profile	The base format is plaintext at rest. Direct WAL/page I/O is not advertised as SQLCipher/VFS composable; platform disk encryption is the supported practical mitigation.

The unused `.woodpecker.yml` and `.github/workflows/docs.yml` files have been removed. The plan specifies a reproducible local release command and evidence bundle, but does not select or require a CI vendor.

2. Scope and product assumptions

2.1. Reviewed surface

Area	Files and questions
Consensus core	`src/protocol.zig`, `src/replicated_log.zig`, codecs, effect ordering, identity assumptions, and membership bounds.
SQL and WAL	`sqlite.zig`, `wal.zig`, `prepared.zig`, `node.zig`; transaction ownership, SQLite configuration, frame validation, apply, and rebuild behavior.
Durable storage	`journal.zig`, `payload_store.zig`, snapshot and identity code; checksums, allocation bounds, atomic installation, recovery, and garbage collection.
Network service	`wire.zig`, `transport_auth.zig`, `server.zig`, `client.zig`, `gateway.zig`, `embedded.zig`; framing, authentication, resource lifetime, redirects, and shutdown.
Public surfaces	`main.zig`, `capi.zig`, `include/zaxonlite.h`; configuration, secret loading, input bounds, ownership, and errors.
Assurance	Build manifests, fuzz/soak/crash/cluster tests, benchmark tools, repository automation, and the Zaxonlite book.

The review is code-driven, not a formal proof or a penetration test. Confirmed findings have a reachable code path. Hardening findings identify a missing control within the supported deployment model. Root and Zaxonlite unit, format, crash, and book builds passed during the original review.

2.2. In-scope trust model

1. One application owner controls each database and is allowed to read and write all of it. The application implements end-user authentication and policy.
2. Cluster nodes are provisioned by the same operator on a trusted network. Non-Byzantine failures, accidental misconfiguration, stale credentials, and unauthenticated network connections are in scope.
3. Local callers are processes authorized by Unix socket ownership, group, and mode. Direct remote end-user access is not the normal production topology.
4. The base profile relies on the OS or device for encryption at rest and protects database files, journals, key material, and snapshots with filesystem access controls. File permissions, path validation, least-privilege service accounts, backups, and an honest disclosure of plaintext-at-rest behavior remain Zaxonlite operational requirements.
5. Application bugs and injection can submit unexpected SQL through the normal Zig, C, embedded, or network API. Zaxonlite must preserve its replication invariants even though it does not implement application policy.
6. Malformed, oversized, fragmented, slow, or replayed network messages and interrupted snapshot transfers are in scope.

2.3. Explicitly out of scope

1. Database-native users, row-level permissions, tenant isolation, and internal reader/writer/admin RBAC.
2. A Byzantine quorum, protection after root or live host takeover, hostile code in the application process, and confidentiality from the host operator.

3. PostgreSQL-style workload governance, mandatory query deadlines for embedded calls, a general SQL sandbox, and security-event/SIEM infrastructure.
4. Dynamic membership policy. Membership remains operator-configured; enrollment proves possession of an approved one-time token and binds a certificate to that configured node identity.

Host-compromise boundary
A party that takes over the host can read or replace SQLite files and key material and is outside this design. That does not eliminate API-reachable correctness risks: an application can accidentally submit transaction control, reserved-table writes, or file-attachment SQL without host takeover. The appropriate response is a small invariant guard, not a multi-user sandbox.

Physical theft and offline media
The current vendored SQLite, direct WAL-file reader, offline page applier, snapshots, and backups are plaintext. Therefore the base release does not protect an edge device after physical theft or offline disk cloning. Use device/filesystem encryption for that deployment. SQLCipher or an encrypted VFS is not presently a drop-in option because Zaxonlite performs direct I/O outside SQLite's VFS; an edge-encryption profile needs separate engineering and must not be advertised until the P3.4 compatibility gate passes.

3. Disposition of the external critique

Criticism	Disposition	Code-audited conclusion
Physical snapshots need signatures or normalized hashes	Narrowed	Normal rollover already decides the exact physical manifest hash in a Paxos stop sign, and Zaxonlite replicates page frames rather than SQL. The valid gap is transferring proof of that existing decision. Fix with retained stop-sign proof plus mTLS read-quorum confirmation, not a new consensus phase or file signatures.
Dangerous pragmas	Partly accepted	`wal_autocheckpoint` can replace Zaxonlite's WAL hook and must be denied. Transaction escape and reserved metadata also remain real. SQLite refuses `journal_mode` and `synchronous` changes inside the current outer transaction, but denying them makes the invariant robust to future changes.
SQL extension-loading RCE	Not a current finding	The bundled amalgamation is compiled with `SQLITE_OMIT_LOAD_EXTENSION`, and Zaxonlite does not register the shell's

Criticism	Disposition	Code-audited conclusion
		file helper functions. Preserve this as a tested build invariant.
Certificate eviction	Accepted with scope correction	Certificate validity must be intersected with the active registry and a persisted denylist, with live-socket closure. The initial product has static membership, so consensus-driven online eviction cannot be claimed yet.
At-rest encryption	Valid product-profile gap	The base threat model did not promise stolen-media protection, but edge users need an explicit answer. Current direct WAL/page I/O is not generically SQLCipher/VFS-compatible. Disclose plaintext, recommend platform encryption, and gate any future encrypted profile on end-to-end compatibility tests.

4. Revised risk register

Severity reflects the product assumptions above. It does not assume hostile database tenants or a compromised host.

ID	Priority	Risk	Required phase
SEC-001	Critical	Shared PSK does not authenticate a unique node and provides no transport confidentiality.	P0.2 + P1.1
SEC-002	Critical	Cross-epoch transfer validates snapshot bytes but does not carry and confirm the already Paxos-decided checkpoint stop sign before authoritative state is replaced.	P1.4
SEC-003	High	Connection threads and network waits are not adequately bounded.	P2.1
SEC-004	High	Large frame, payload, backup, and snapshot declarations can consume excessive memory, disk, or time.	P2.2
SEC-005	High	SQL issued through a normal API can escape Zaxonlite's outer transaction, replace its WAL hook through a pragma, or mutate reserved replication metadata.	P1.3
SEC-006	Medium	Remote queries have no configurable result or execution budget while materializing results.	P2.3

ID	Priority	Risk	Required phase
SEC-007	Medium	Loopback TCP is a weak local authorization boundary and the current listener has no Unix-domain-socket mode.	P0.2
SEC-008	Medium	The proposed enrollment token needs single-use, expiry, identity-binding, and atomic consumption semantics.	P1.1
SEC-009	Medium	File creation modes, path roots, symlink handling, and secret-file validation require a single documented policy.	P3.1
SEC-010	Medium	The C ABI validates some lengths only after converting pointers to slices or starting allocations.	P3.2
SEC-011	Medium	Client redirects need authenticated target identity; non-mTLS clients must remain confined to configured seeds.	P1.1
SEC-012	Deferred	Release checks are manual until the project adopts an actual CI system.	P0.3 + P4.1
SEC-013	Profile gap	The base build stores plaintext and direct WAL/page I/O is not proven compatible with SQLCipher or an encrypted VFS.	P3.4, only before claiming edge theft protection

4.1. Code evidence map

The map below records the original audit observation for traceability. The post-implementation table in the executive section is authoritative for the current tree; remediation evidence is summarized there and in the acceptance matrix.

Finding	Primary code	Observed implementation point
SEC-001	`transport_auth.zig`; `server.zig` connection handshake	One shared secret derives challenge/response and frame HMACs; the claimed hello identity is not bound to a per-node key.
SEC-002	`node.zig: prepareCheckpoint`, `buildFollowerSnapshot`, `completeRollover`, and `installSnapshot`; snapshot handlers in `server.zig`	Normal rollover already decides `zx1 <name> <manifest-sha256>` as a Paxos stop sign and followers require the same physical manifest. The transfer path verifies only the received manifest/image and omits that decided proof.
SEC-003	`server.zig: run`, `noteHandlerStarted`	Each accepted connection gets a detached handler thread. Active handlers are tracked for shutdown, but admission has no maximum or I/O deadline.
SEC-004	`wire.zig: max_frame_bytes`, `readFrameBody`; `server.zig: max_snapshot_bytes`; `client.zig: backupTo`	A frame can allocate up to 64 MiB and snapshot/backup declarations accept up to 1 TiB, without a small-database aggregate budget.

Finding	Primary code	Observed implementation point
SEC-005	`node.zig: writeRequest`, `openLiveDatabase`; `sqlite.zig: Db.exec`; `zaxonlite/build.zig`	Zaxonlite executes `begin immediate`, then application SQL, then its own metadata writes and `commit`; no authorizer separates those scopes. `wal_autocheckpoint` can replace the installed WAL hook. Loadable extensions are not reachable because SQLite is built with `SQLITE_OMIT_LOAD_EXTENSION`.
SEC-006	`node.zig: query`; JSON result paths in `server.zig` and `capi.zig`	Rows and cells are duplicated/materialized with no caller-configured execution, row, or byte budget.
SEC-007	listener construction in `server.zig`; address validation in CLI/configuration paths	Only IP/TCP listeners exist; no Unix-domain socket path or filesystem permission policy is implemented.
SEC-008	`enrollment.zig`; enrollment frames in `wire.zig`; issuer paths in `server.zig`; `enroll-token` and `enroll` in `main.zig`	The baseline had no enrollment module. Protocol v6 now carries one bounded CSR exchange over server-authenticated TLS. Token issuance itself requires an existing mTLS client and a server explicitly configured with the 0600 CA private-key provider.
SEC-009	path operations throughout `node.zig`, `journal.zig`, `payload_store.zig`, and configuration secret loading	Durability helpers exist, but one enforced policy for data roots, links, modes, ownership, and key files does not.
SEC-010	`capi.zig: zaxonlite_cluster_open`, prepared-value conversion, and exported handle functions	For example, `member_count` drives an allocation and slice of caller memory before it is reduced to product membership limits.
SEC-011	`client.zig: callClusterWithSecret`	A host and port from a `not_leader` response become the next endpoint; they are not resolved through an authenticated configured-member identity.
SEC-013	`wal.zig: readCommittedFrames`, `applyPayload`; `node.zig: rebuildMaterializedImage`, snapshot/backup paths	Zaxonlite parses the standard WAL directly and writes page images outside SQLite. A VFS that encrypts those bytes cannot transparently cover these paths; the bundled SQLite is not SQLCipher.

The first release boundary is therefore smaller than the earlier report: per-node transport identity and eviction, confirmation of the already-decided checkpoint during transfer, bounded network resources, and preservation of replication invariants. An internal authorization server, RBAC policy language, telemetry pipeline, generalized untrusted-SQL sandbox, and a second snapshot consensus protocol are intentionally absent.

5. Baseline focused findings (pre-remediation)

The findings below preserve the original audit wording and exit criteria. They are not a second current-status report; the post-implementation review and acceptance matrix are authoritative for the tree reviewed on 21 July 2026.

SEC-001 Shared PSK is not node identity — Critical

Boundary: Cluster TCP transport.
Evidence: Protocol v4 computes an authentication value from a cluster-wide secret. Every holder has the same proof and the peer-supplied node identifier is not cryptographically bound to a unique key. Application traffic and consensus traffic are not confidential.
Impact: A leaked or stale cluster secret permits impersonation of a configured node, message observation, and connection attempts from any reachable host.
Remediation: Replace PSK authentication with TLS 1.3 mutual authentication. Issue one certificate per configured node after one-time enrollment. Bind certificate identity to cluster ID and node ID, validate it against configured membership, and remove plaintext protocol use in production.
Verification: Negative tests reject an unknown CA, expired certificate, wrong cluster, wrong node ID, certificate/hello mismatch, client-only identity on a peer channel, and plaintext or downgraded connections. Packet capture contains no SQL or snapshot plaintext.

SEC-002 Snapshot transfer drops the existing checkpoint proof — Critical

Boundary: Consensus safety during snapshot recovery.
Evidence: Normal checkpointing already has the right consensus object. The leader checkpoints its physical SQLite image, hashes a manifest, and proposes `zx1 <name> <manifest-sha256>` as a stop sign. Paxos chooses that stop sign. Followers apply the same replicated page frames, build their own physical image, and require the identical manifest digest before rollover. However, `installSnapshot` accepts a newer transferred manifest and image after structural and digest checks without learning or confirming the decided stop sign that named that manifest.
Impact: A buggy, stale, misconfigured, or impersonated source can cause durable state replacement without evidence that the existing Paxos checkpoint was chosen. mTLS authenticates a source but does not make an arbitrary image authoritative.
Remediation: Keep physical snapshots and reuse the existing Paxos decision—do not add signatures over independently generated database files or a second consensus phase. Define `CheckpointProofV1` as the sealed configuration ID, stop-sign slot, exact decided stop-sign bytes (including next configuration and membership), manifest digest, applied slot, and chain hash. Retain this small proof beside each transferable snapshot. Before destructive installation, the lagging node obtains the same proof digest from a read quorum of the stop sign's voter configuration over mTLS, persists the agreed proof with the staged image, verifies the manifest and physical database hash against it, and only then activates the snapshot. If the quorum cannot confirm, automatic transfer stops and operator-assisted recovery is required. Keep the old journal until proof, image, identity, and new journal are durably installed.
Verification: Tests prove byte-identical physical snapshots after freelist reuse, schema churn, large BLOBs, page truncation, and supported vacuum/checkpoint cases. Conflicting proof replies, a single lying/stale source, wrong voter set, wrong stop slot, chain mismatch,

missing quorum, corruption, and every install crash retain the last valid state. No test requires snapshot-file signatures or a new Paxos phase.

Why physical snapshots are valid here

The usual "same SQL can produce different SQLite files" warning does not describe Zaxonlite's replication path. SQL executes once on the leader; Zaxonlite replicates the resulting WAL page frames and followers apply those page bytes at the same page numbers, including the final database size. The physical file is therefore the canonical state for a chosen frame history. The existing descriptor `chain` is also a rolling history hash and belongs in the checkpoint proof, but it does not replace verification of the physical image that recovery will install.

Initial proof scope

P1.4 covers checkpoint stop signs produced by `checkpoint()`, which carry the same static voter membership into the next epoch. Although the generic replicated-log layer has `reconfigure()`, Zaxonlite does not expose online voter-set changes. A snapshot that also changes voter membership is therefore rejected by the initial production transfer path until a separate reconfiguration/handover protocol defines which old/new quorum must attest it.

SEC-003 Unbounded connection and wait lifetime — High

Boundary: Availability of the local and cluster services.
Evidence: The server creates a thread per accepted connection and blocking reads can wait indefinitely. No global connection cap, per-peer cap, handshake deadline, idle deadline, or shutdown-aware admission policy is enforced.
Impact: Malformed or simply slow clients can consume threads and file descriptors until legitimate small-cluster traffic cannot make progress.
Remediation: Add fixed admission limits sized for a small configured cluster, handshake and idle deadlines, bounded work queues where appropriate, and prompt shutdown cancellation. Exempt only explicitly internal operations whose lifecycle is otherwise bounded.
Verification: Slowloris, partial-frame, excess-connection, repeated-handshake, and shutdown tests remain within documented thread and descriptor limits and recover without restart.

SEC-004 Transfer declarations exceed a SQLite-shaped deployment — High

Boundary: Network parsing, memory use, disk use, and recovery time.
Evidence: Frame parsing permits large single allocations and snapshot/backup metadata can describe sizes far beyond the intended small embedded database profile. Structural checks do not provide an aggregate transfer budget.
Impact: A reachable peer can cause memory pressure, disk exhaustion, or very long processing even when every individual chunk is well formed.
Remediation: Set conservative configurable frame, message, query-result, snapshot, backup, and per-connection in-flight limits. Stream large objects to a staged file, check aggregate byte counts with overflow-safe arithmetic, reserve disk conservatively, and clean

up partial transfers.
Verification: Boundary and over-limit tests cover each field, aggregate budget, integer overflow, disk-full behavior, interrupted cleanup, and a small-cluster load run at the documented maximum.

SEC-005 Application SQL can violate replication invariants — High

Boundary: Correctness guard inside a trusted single-application database.
Evidence: `Node.writeRequest` owns an outer transaction and invokes `sqlite3_exec` for application SQL. SQLite statements such as `COMMIT`, `ROLLBACK`, `SAVEPOINT`, `ATTACH`, `DETACH`, or direct writes to `__zaxon_*` can be supplied through the normal API. `PRAGMA wal_autocheckpoint` can install SQLite's automatic-checkpoint hook and replace Zaxonlite's frame-counting WAL hook. This needs no direct file access or host takeover.
Impact: A trusted application bug, migration error, or upstream SQL injection can end the transaction Zaxonlite expects to capture, change private metadata, or create side effects not represented by the replicated WAL payload. This is a replicated-state correctness problem, not cross-tenant privilege escalation.
Remediation: Install a narrowly scoped SQLite authorizer for application statements. Deny transaction-control actions that conflict with the owned transaction, `ATTACH`/`DETACH`, and reads or writes of the reserved `__zaxon_*` namespace. For `SQLITE_PRAGMA`, deny writes to `wal_autocheckpoint`, `journal_mode`, `synchronous`, `page_size`, `locking_mode`, `auto_vacuum`, `writable_schema`, and `query_only`; also deny application-triggered `wal_checkpoint`. Allow read-only inspection where SQLite supplies no value and allow ordinary application pragmas such as `user_version` unless tests establish an invariant conflict. Use a separate internal scope for Zaxonlite's own statements. After each application batch, cheaply assert that WAL mode, page size, autocommit state, and the installed frame hook still match the capture contract before commit/payload extraction. Preserve normal SQLite DDL, triggers, functions, and migrations.
Verification: Tests submit each denied operation directly and from triggers/views where SQLite reports it to the authorizer; verify that `wal_autocheckpoint` cannot replace the hook; and prove the request fails atomically with identical replicas. A build-time assertion keeps `SQLITE_OMIT_LOAD_EXTENSION` enabled. The compiled SQL surface has no `load_extension`, `writefile`, or `readfile` function unless a future explicitly reviewed build registers one. A broad SQLite compatibility suite proves that supported application SQL remains usable.

Rebuttal: extension loading and two pragma examples
The audited build passes `-DSQLITE_OMIT_LOAD_EXTENSION` when compiling the SQLite amalgamation, and Zaxonlite does not register the SQLite shell's `writefile`/`readfile` helpers. Therefore SQL extension-loading RCE is not a current finding. Also, `writeRequest` has already opened `BEGIN IMMEDIATE`; SQLite refuses changing out of WAL mode or changing the synchronous level inside that transaction. These facts reduce the critic's examples, but they do not remove the authorizer requirement: `wal_autocheckpoint`, transaction escape, attachment, private metadata, and future connection changes still need an explicit invariant boundary.

SEC-006 Remote result and execution budgets are absent — Medium

Boundary: Availability, not database authorization.
Evidence: Queries can run for an unbounded time and materialize large results before returning them. The same policy should not be forced onto trusted embedded callers or legitimate long migrations.
Impact: A bad query or application-generated workload can consume CPU and memory and block useful work.
Remediation: Stream remote rows, expose optional cancellation/deadline through `sqlite3_progress_handler`, and set conservative safety-oriented `sqlite3_limit` values. Make remote result and execution budgets configurable and allow an explicit trusted deployment to disable a deadline. Embedded calls remain cancellable by their owning application and need no default wall-clock limit.
Verification: Remote tests cover cancellation, row/byte limit, deeply nested or oversized SQL rejection, long valid migration with the deadline disabled, and cleanup after interruption.

SEC-007 Loopback TCP is not the preferred local boundary — Medium

Boundary: Single-host service mode.
Evidence: The implementation currently exposes TCP. Binding to a loopback address limits reachability but does not express ownership, group access, or namespace policy as directly as a Unix-domain socket.
Impact: Other local processes can attempt access and operators can accidentally widen the bind address.
Remediation: Add Unix-domain-socket transport with configurable path, owner, group, and mode; reject an unsafe pre-existing path and remove the socket on orderly shutdown. Production TCP uses mTLS. If a local-development exception is retained, make it explicit, PSK-authenticated, and numeric-loopback-only.
Verification: Permission tests cover owner-only and group access, unsafe existing paths, symlink behavior, restart cleanup, and refusal to start plaintext TCP in a production configuration.

SEC-008 Enrollment token lifecycle must be explicit — Medium

Boundary: Initial node admission.
Evidence: The baseline had no one-time token implementation. A reusable bearer token would merely have replaced the former long-lived PSK with another shared secret.
Impact: A copied, replayed, or misdirected token could enroll the wrong node or remain useful after the intended join.
Remediation: Implemented in `enrollment.zig`: the opaque versioned bundle carries the issuer endpoint, full cluster CA certificate, database ID, issuer and target node IDs, 256-bit random secret, and expiry. The server confirms the target is a configured non-revoked member, verifies the CSR signature and exact node common name, atomically renames and syncs the token record to `used`, and only then signs. The token is never a traffic key.
Verification: Unit tests cover the exact expiry boundary, node/database identity binding, truncation, replay, and two concurrent redemptions with exactly one winner. The CLI inte-

gration issues a token through an existing mTLS client, enrolls a locally generated key/CSR, authenticates with the result, and rejects a copied-token replay. A crash after consumption may require a replacement token but cannot reuse the old token or create a second issuance.

SEC-009 Filesystem policy is distributed across call sites — Medium

Boundary: Protection of host-owned files before host compromise.
Evidence: Database images, journals, payloads, snapshots, socket paths, and key files are created and opened by several components. A single enforceable root, creation-mode, symlink, ownership, and secret-file policy is not yet documented and tested.
Impact: Operator mistakes or unsafe paths can expose data or redirect writes. This control does not claim to defend against root.
Remediation: Resolve all managed paths below an explicit data root, use restrictive creation modes, refuse unsafe secret permissions, avoid following unexpected links, fsync parent directories where atomic replacement requires it, and document backup/key separation.
Verification: A local matrix covers umask variation, path traversal, link swaps, wrong owner/mode, read-only and full disks, rename interruption, and recovery under an unprivileged service account.

SEC-010 C ABI needs earlier validation and clearer ownership — Medium

Boundary: Memory safety at the public C boundary.
Evidence: Some functions form Zig slices or begin work before all lengths, counts, and conversions are validated. The header does not yet make every handle, output initialization rule, and ownership transfer mechanically obvious.
Impact: A malformed caller can trigger oversized work, checked-build traps, or ambiguous cleanup even though the implementation language is Zig.
Remediation: Keep the C surface small. Validate pointer/count pairs before slicing or allocating, use checked integer conversions, initialize outputs on every path, cap counts with the same protocol limits, and document opaque-handle lifecycle and allocator ownership in the header.
Verification: Compile and execute a C smoke suite, fuzz every exported function with malformed pointer/count combinations that are valid to present, and run sanitizer-enabled boundary tests on supported toolchains.

SEC-011 Redirects need authenticated node targets — Medium

Boundary: Client routing within a small static cluster.
Evidence: A redirect can influence where a client connects. With the present PSK, the endpoint is not bound to a unique certificate identity.
Impact: A stale, malformed, or manipulated redirect can send a client outside its expected cluster or cause persistent routing loops.
Remediation: After mTLS, accept an advertised address only from an authenticated seed, require the new target certificate to be exactly `zaxon-node-<advertised-id>` under the cluster CA, and cap redirect depth. Without mTLS, rotate only through caller-supplied endpoints.

Verification: Unknown address, wrong certificate, cross-cluster target, repeated member, and redirect-depth tests fail closed while a valid leader move succeeds.

6. Simple cluster identity and enrollment

6.1. Design goal

The design should feel like SQLite plus a small trusted cluster, not like a general identity platform. Operators configure the membership and trust a cluster CA. A one-time token authorizes exactly one configured node to obtain its first certificate. Normal traffic then uses ordinary mutual TLS.

One-time enrollment, then ordinary mTLS

Joining node		Existing member
Generate key pair and CSR	→	Create a short-lived, single-use token
Pin bundled CA and issuer node identity	←	Bundle carries CA, issuer endpoint, database and target IDs
Present token, intended node ID, and CSR	→	Check configured membership, expiry, and unused token
Atomically install key, certificate, and CA	←	Sync token as used, then sign node certificate
Reconnect with node certificate	⇄	TLS 1.3 mutual authentication for all later traffic

The token is bootstrap data, not a long-lived cluster credential. It should be easy to print or copy once and safe to discard immediately after enrollment. The node private key is generated on the joining node and never carried inside the token.

6.2. Token and certificate contract

Field/control	Required meaning
Version	Allows the token encoding to evolve and rejects unknown security-critical semantics.
Cluster/database ID	Prevents cross-cluster enrollment.
Issuer and intended node ID	Both are fixed in the bundle. The issuer accepts the target only when that ID is already present and non-revoked in its static role registry; roles remain consensus capabilities, not user authorization roles.
Issuer endpoint	Names the one node allowed to redeem this bundle. The certificate common name must equal the bundled issuer node ID.
Cluster CA certificate	Pins the TLS trust root before the bearer secret is revealed. It is copied into the installed identity only after the authenticated exchange succeeds.
Random secret	At least 256 bits from the operating-system CSPRNG.
Derived token ID and expiry	The domain-separated SHA-256 of the secret names the persisted record without storing the bearer value. Expiry defaults to ten minutes and is capped at 24 hours.

Field/control	Required meaning
Node certificate	Contains or maps unambiguously to cluster ID and node ID, uses an appropriate EKU, and has a bounded validity period.

Enrollment is allowed only through a dedicated bootstrap path. The member validates the token, configured identity, certificate request, and expiry; atomically records consumption; signs the node certificate; and returns the leaf certificate. The implemented order is deliberately fail-closed: `pending` is renamed to `used` and the token directory is synced before signing. A crash after that boundary may require the operator to issue a new token, but an ambiguous retry cannot create a second certificate from the old one. This avoids a reusable token or a more complicated certificate-result cache.

After enrollment, every node-credential TCP channel verifies the chain, cluster binding, certificate validity, revocation state supported by the release, configured node identity, and hello/certificate agreement. Certificate rotation uses an already authenticated node channel or a new operator-issued one-time token.

6.3. Eviction without an identity platform

A valid certificate is necessary but never sufficient for node transport admission. The effective rule is:

Transport admission
valid TLS chain and time **and** certificate cluster/node identity matches the hello **and** node ID is in the active runtime member registry **and** node ID is not locally revoked.

The initial product has operator-configured static membership. It does not yet expose the replicated log's general reconfiguration primitive as an online membership-management API. The practical first-release eviction procedure is therefore explicit:

1. add the node ID to a small persisted revocation file on every surviving member through the normal configuration-management channel;
2. reload that file atomically, reject new handshakes, and immediately close all active sockets whose authenticated identity matches the revoked node;
3. rotate any enrollment material the node could read, issue replacement certificates as needed, and follow the documented offline voter-replacement procedure if the voter set itself must change;
4. use short-lived node certificates so an omitted denylist update has a bounded lifetime, but never rely on certificate expiry as the only eviction control.

When Zaxonlite later exposes committed membership changes, applying a decided stop sign with a new member set must atomically replace the transport allowlist and close sockets for removed nodes. Consensus membership then overrides certificate validity. Until that product feature exists, the plan must not claim consensus-driven online eviction.

Scope of eviction
This prevents a stale or stolen node certificate from retaining network access. It does not make Paxos Byzantine-tolerant. If an attacker controls a live voter and its process, the operator must isolate it and restore a healthy voter configuration; tolerating malicious votes is outside this product.

Simplicity boundary
Do not add database users, an OAuth/OIDC service, policy evaluation, dynamic membership approval, or separate administrative roles. The certificate says "this is configured node N in cluster C." The application remains the one database authority.

7. Encryption at rest: an explicit deployment profile

Physical theft is a legitimate edge-product concern, but it is not solved by declaring generic SQLite-VFS compatibility. The current architecture reads the standard WAL directly, packages page bytes, applies them to an offline database file with positional writes, hashes physical snapshots, and produces plaintext logical backups. Those operations bypass SQLite's VFS.

The pragmatic base release does three things:

1. state plainly that database files, captured payloads, journals, snapshots, backups, and relevant metadata contain plaintext;
2. recommend platform full-disk or filesystem encryption for edge devices and require operators to protect keys separately from removable media where the platform permits it;
3. keep filesystem permissions and least privilege even when disk encryption is present, because disk encryption normally protects powered-off media rather than a running compromised process.

SQLCipher or a specialized encrypted VFS is a separate optional profile. Before claiming support, a design spike must choose where encryption occurs. One viable shape is encrypting page bodies before they enter Zaxonlite's replicated payload format while retaining authenticated routing metadata; another is teaching all direct-I/O paths to use the same pager codec. Either way, the profile needs a key-provider callback, per-database key identity, startup ordering, memory zeroization, rotation/rekey rules, encrypted snapshots/backups, and mixed-key failure handling. All nodes must apply the same cryptographic representation without putting raw keys into Paxos messages, tokens, logs, or manifests.

Release decision
At-rest encryption is not a gate for the server/cloud base profile under the agreed threat model. It becomes a hard gate before Zaxonlite advertises protection against stolen edge devices or offline disk cloning. Until the compatibility suite passes, documentation must say "use platform disk encryption," not "works with SQLCipher/VFS encryption."

8. C interface decision: keep it small

The review considered adding `rurban/safeclib`. Safeclib implements checked C library operations, including many C11 Annex K-style memory and string functions. Those functions are useful in C programs that otherwise perform substantial raw buffer manipulation. They do not validate that an arbitrary foreign pointer is readable, prove a pointer/count pair belongs to an object, fix an integer conversion, define handle lifetime, or make a late length check early.

Zaxonlite's C ABI is implemented in Zig, and the public C shim should contain almost no string or memory manipulation of its own. Adding safeclib would add a C dependency, build and platform work, runtime-constraint-handler semantics, and another API family without addressing the boundary's principal risks.

Question	Decision	Reason
Use safeclib inside the Zig implementation?	No.	Zig slices and checked operations already replace the C calls safeclib is designed to harden.
Expose safeclib types/functions in the public ABI?	No.	It would enlarge and couple the API without improving pointer provenance or handle safety.
Keep the C API?	Yes, deliberately small.	A thin ABI is valuable to C applications and easier to review, fuzz, and keep stable than a wrapper framework.
How is it hardened?	Explicit lengths and ownership.	Validate pointer/count relationships before slice creation, use checked casts, cap all work, initialize outputs, and test the compiled C surface.

The public header should define a compact rule set:

1. `null` is accepted only where explicitly stated; a nonzero length always requires a non-null pointer, and a null pointer always requires zero length.
2. Input buffers are borrowed only for the duration of the call. Output buffers name the function that frees them; opaque handles name their destroy function.
3. Lengths use one documented type at the ABI and are converted only after checking against Zig, SQLite, protocol, and allocation limits.
4. Every fallible function initializes output handles, pointers, and lengths to a safe empty value before doing other work.
5. Errors have stable numeric values and an optional bounded diagnostic string; no internal pointer or allocator ownership crosses the ABI.

9. Remediation plan

9.1. P0 — establish the intended product boundary

P0.1 · Application-owned authentication

Outcome: Document and enforce the supported topology: the embedding or socket-owning application is the single database principal. It authenticates end users, applies roles and SQL policy, and decides which operations reach Zaxonlite. Zaxonlite has no database user catalogue or internal RBAC. Update the book, examples, C header, and configuration reference to state this consistently.

Exit criterion: The book and public API documentation show embedded, local Unix-socket, and clustered-node profiles; no text promises per-user database authorization. A deployment checklist identifies the application and OS boundaries.

P0.2 · Safe transport defaults

Outcome: Implement Unix-domain sockets for local service and TLS 1.3 mutual authentication for production TCP. Keep the clearly labelled PSK-only development mode explicit and numeric-loopback-only, and make production configuration fail closed when plaintext

or shared-PSK transport is selected.
Exit criterion: Local-mode permission tests pass; TCP refuses unknown or mismatched certificates; packet capture exposes no application data; a production configuration cannot silently fall back to PSK or plaintext.

P0.3 · SQLite-style local assurance baseline

Outcome: Create one reproducible local release target that runs formatting, unit and integration tests, deterministic crash/fault campaigns, protocol and C-ABI fuzz smoke runs, and book compilation. Record seeds, versions, and artifacts. Treat longer crash, fuzz, and soak campaigns as scheduled/manual release evidence. Do not add a CI-vendor workflow in the initial release.
Exit criterion: A clean checkout can produce a versioned local evidence directory with commands, tool versions, seeds, pass/fail results, and compiled documentation. Rerunning a recorded failing seed reproduces the failure.

9.2. P1 — close integrity and identity gaps

P1.1 · One-time join and per-node mTLS

Outcome: Implement the enrollment and certificate contract in this plan. Operators preconfigure a member, create one short-lived token, and deliver it to the joining node. The joining node pins the cluster certificate, generates its own key and CSR, redeems the token once, and thereafter uses its per-node certificate. Peer admission requires configured identity. Client redirects may leave the seed list only under mTLS and only when the target certificate matches the advertised node ID; PSK-only clients remain seed-confined. Transport admission also requires the node to remain in the active registry and outside the persisted node-ID denylist. Reloading revocation state closes matching live sockets immediately.
Exit criterion: All enrollment replay and mismatch tests pass; any shared PSK is only an optional inner TLS layer and never the production identity boundary; every peer connection cryptographically binds cluster ID and configured node ID; certificate rotation succeeds without changing database authorization semantics; and revoking an enrolled node rejects new connections and closes existing ones before certificate expiry. The initial release documents operator-distributed revocation and offline voter replacement rather than claiming dynamic consensus eviction.

P1.2 · Confirm single-principal database semantics

Outcome: Do not implement database roles, user accounts, row policy, or separate reader/writer/admin credentials. Treat possession of the embedded handle or permission to access the Unix socket as full database authority. Avoid presenting the direct TCP SQL client as the default production ingress; the application remains in front of the database.
Exit criterion: Architecture, examples, and security documentation contain no multi-user promise; the Unix socket has explicit owner/group/mode controls; production examples put application authentication before Zaxonlite access.

P1.3 · Protect SQLite replication invariants

Outcome: Add a narrow application-SQL authorizer that denies conflicting transaction

control, database attachment/detachment, and access to reserved `__zaxon_*` objects. Give internal Zaxonlite statements a distinct execution scope. Deny writes to capture/durability pragmas—especially `wal_autocheckpoint`—and verify connection invariants before extracting a payload. Keep `SQLITE_OMIT_LOAD_EXTENSION` in the SQLite build. Audit migrations, triggers, pragmas, virtual tables, and common SQLite features so the guard does not become a generic SQL sandbox.
Exit criterion: Forbidden statements and indirect variants fail before side effects, the owned transaction and WAL hook remain intact, replicas converge, extension loading remains absent from the compiled SQL surface, and the documented SQLite compatibility suite—including long migrations and ordinary DDL—passes.

P1.4 · Carry the decided checkpoint proof through transfer

Outcome: Keep the current physical snapshot and Paxos-decided stop-sign design. Add `CheckpointProofV1`, retain it with transferable generations, and require a lagging node to obtain an identical proof digest from a read quorum of the named voter configuration over mTLS. Persist the agreed proof before validating and activating the staged manifest/image. Do not add per-file signatures, normalized SQLite hashing, or another consensus phase. The initial implementation accepts only checkpoint stop signs whose next voter set equals the current static voter set; membership-changing transfer remains unsupported.
Exit criterion: No stale, wrong-cluster, wrong-membership, conflicting, single-source, or unconfirmed snapshot can replace local state. A quorum-confirmed proof and matching physical image survive exhaustive injected crash points. Exact physical hashes converge across the documented page-layout stress suite.

9.3. P2 — bound network and SQLite resource use

P2.1 · Connection and time bounds

Outcome: Add small-cluster connection caps, per-peer caps, handshake deadlines, idle deadlines, bounded admission, and shutdown cancellation. Size defaults from configured members and documented operator connections, not from internet-scale assumptions.
Exit criterion: A slowloris and connection-flood campaign stays within the documented descriptors, threads, memory, and recovery time while healthy nodes retain quorum progress.

P2.2 · Frame and transfer budgets

Outcome: Define checked per-frame, per-message, per-result, per-snapshot, per-backup, per-connection, and aggregate in-flight limits. Stream large objects to staged storage and clean them on failure. Reject impossible declared sizes before allocation or disk mutation.
Exit criterion: Generated boundary tests cover every length/count field and aggregate; over-limit, overflow, disk-full, truncation, and interrupted-transfer cases fail without unbounded work or damage to the last committed database.

P2.3 · SQLite-compatible query controls

Outcome: Stream remote results. Add optional cancellation/deadline using SQLite's progress handler and conservative runtime limits for structural safety. Provide remote row/byte limits. Keep these settings configurable: a trusted embedded caller or operator-approved migration

can run without a default wall-clock deadline. Do not claim PostgreSQL-style workload isolation.
Exit criterion: Remote row/byte caps and cancellation are deterministic and clean up all state; structural SQLite limits prevent pathological inputs; the same suite proves that supported long transactions, schema migrations, and analytics complete when the operator disables the deadline within documented database and host constraints.

9.4. P3 — harden host and language boundaries

P3.1 · Filesystem and key policy

Outcome: Centralize data-root resolution, restrictive file and directory modes, secret permission checks, symlink/path rules, durable atomic replacement, staged-file cleanup, and backup/key separation. Run the service as a dedicated unprivileged account. Document that the base data, payload, snapshot, and backup formats are plaintext and recommend platform disk or filesystem encryption where powered-off media theft is in scope.
Exit criterion: The filesystem matrix passes under varied umasks, link races, wrong ownership/modes, traversal attempts, read-only and full disks, and crashes. Documentation explicitly states that host/root compromise and offline media confidentiality are not provided by the base build.

P3.2 · Small, reviewed C ABI

Outcome: Do not add safeclib. Apply early pointer/count validation, checked casts, shared work limits, output initialization, explicit ownership, stable errors, and opaque handle lifecycles to the thin Zig implementation and C header. Add a compiled C conformance and fuzz harness.
Exit criterion: Every exported function is listed in an ABI checklist and exercised from C; malformed boundary inputs fail predictably in debug, release-safe, and sanitizer-enabled supported builds; no allocator ownership is ambiguous.

P3.3 · Basic operational diagnostics only

Outcome: Expose bounded counters and human-readable diagnostics needed to operate the supported model: enrollment outcomes, certificate expiry, connection-limit rejections, invalid frames, snapshot verification, recovery, and resource caps. Avoid a telemetry pipeline, security-event taxonomy, or SIEM contract.
Exit criterion: An operator can diagnose a failed join, impending certificate expiry, resource rejection, snapshot refusal, and recovery decision from local logs/status without secrets or SQL values being emitted.

P3.4 · Optional edge-encryption compatibility gate

Outcome: Do not treat SQLCipher or an encrypted VFS as plug-compatible. If an edge profile is product work, design one encryption boundary that covers direct WAL reads, replicated page payloads, offline page application, materialized images, snapshots, and backups. Define a key-provider API, key identity, startup/recovery ordering, memory lifetime, rotation, and mixed-key errors. Never replicate or log the raw key.
Exit criterion: This is not a base-release gate. Before an edge-theft-protected profile is

advertised, stolen-media inspection exposes no SQL/page plaintext; crash, snapshot transfer, rebuild, backup/restore, rekey, wrong-key, and rolling restart tests pass; and the documentation names the exact supported SQLCipher/VFS/encryption implementation rather than claiming generic composability.

9.5. P4 — release evidence and focused review

P4.1 · Deferred CI integration

Outcome: Keep the P0.3 release suite locally reproducible. When the project selects and operates a CI system, invoke the same targets there with pinned tools, least-privilege credentials, artifact retention, and branch/release policy. Do not recreate Woodpecker or GitHub Actions workflows now.
Exit criterion: Initial release: a reviewer reproduces the evidence locally from a clean checkout. Later CI phase: the adopted CI executes the same targets and the repository records the governing policy and provenance.

P4.2 · Documentation and configuration conformance

Outcome: Generate or test protocol constants, configuration defaults, error values, C declarations, and format/version claims where practical. Maintain a code-to-book checklist for security-relevant behavior.
Exit criterion: A conformance target fails when documented defaults, supported transports, limits, token fields, certificate identity, snapshot format, or ABI declarations diverge from implementation.

P4.3 · Focused adversarial review

Outcome: Commission a review scoped to this actual design: enrollment and mTLS identity binding, revocation/active-socket eviction, malformed/slow protocol input, transferred decided-checkpoint proof and crash installation, SQLite invariant/pragma guard, filesystem rules, and the small C ABI. Include at-rest cryptography only if the optional edge profile is claimed. Review consensus safety where those controls interact.
Exit criterion: All in-scope critical/high findings are fixed and retested. The report states the trusted-application, trusted-network, non-Byzantine, single-principal, and host-compromise assumptions; it does not grade Zaxonlite against generic multi-tenant database, RBAC, or distributed-PostgreSQL requirements.

10. SQLite-style assurance strategy

SQLite's useful lesson for this project is not a particular test count or CI provider. It is the combination of deterministic unit coverage, assertions and invariants, fault simulation, malformed-input testing, reproducible failures, cross-configuration testing, and long campaigns. Zaxonlite should apply that approach to the surfaces created by replication.

10.1. Test layers

Layer	Required coverage	Normal cadence
Deterministic unit	Codecs, checked arithmetic, token parsing, certificate identity and revocation	Every local release run.

Layer	Required coverage	Normal cadence
	mapping, authorizer/pragma decisions, checkpoint proof validation, manifest validation, C-ABI preconditions.	
State-machine model	Small Paxos histories, restart/replay, membership constraints, chosen-prefix and snapshot model comparisons.	Every local release run with fixed seeds.
Crash/fault injection	Write, fsync, rename, token consumption, certificate issuance, snapshot staging/install, journal truncation, disk full, partial I/O.	Smoke set on every release run; exhaustive campaign before a release candidate.
Protocol fuzzing	Every decoder and state transition, fragmented/slow input, length relationships, TLS/hello mismatch, redirect loops.	Bounded smoke run plus recorded longer campaign.
SQL compatibility	Representative SQLite DDL/DML, transactions, triggers, migrations, allowed and denied pragmas, functions, long operations, WAL-hook retention, compile-option assertions, and each authorizer denial.	Every local release run.
C ABI	C compilation, handle lifetime, pointer/count rules, checked conversions, output initialization, allocator ownership.	Every local release run; sanitizer campaign where supported.
Cluster soak	Three- and five-node loss, delay, restart, leader changes, certificate rotation/revocation, proof-confirmed snapshot recovery, exact physical image comparison, and resource caps.	Recorded pre-release campaign.
Optional edge encryption	Only for an advertised encrypted profile: stolen-media scan, wrong/mixed key, direct-WAL capture/apply, crash recovery, snapshot, backup/restore, rotation, and key leakage.	Required for that profile; excluded from the base release suite.
Book conformance	Constants, configuration, protocols, formats, security assumptions, examples, and successful Typst compilation.	Every local release run.

Each harness records the source revision, Zig/SQLite/Typst versions, configuration, platform, seed, duration, failpoint schedule, and result. A failure becomes a permanent deterministic regression case before closure.

11. Required security matrix

Boundary	Control	Required negative evidence	Gate
Application	Application-level authentication and authorization	Production examples never expose a direct unauthenticated	P0.1 / P1.2

Boundary	Control	Required negative evidence	Gate
		end-user path; documents make full database authority explicit.	
Local access	Unix-domain socket owner/group/mode	Unauthorized user, unsafe existing path, symlink, wrong mode, and stale socket fail safely.	P0.2
TCP access	TLS 1.3 mutual authentication	Unknown CA, wrong cluster/node, expired certificate, hello mismatch, plaintext, and downgrade fail.	P0.2 / P1.1
Enrollment	Pinned cluster CA/issuer and single-use token	Replay, expiry, copied token for a different member, concurrency, and issuance crash cannot create a second identity.	P1.1
Node eviction	Active registry plus node-ID revocation	A revoked but unexpired certificate cannot reconnect; existing matching sockets close on atomic reload. Static release procedures do not claim online consensus reconfiguration.	P1.1
SQL write path	Narrow SQLite invariant authorizer	Transaction escape, attachment, reserved namespace access, capture-changing pragmas, and WAL-hook replacement fail atomically; extension loading remains compiled out while supported SQLite SQL passes.	P1.3
Snapshot	Existing decided stop-sign proof plus staged physical install	Wrong cluster/slot/membership/chain, conflicting voter replies, missing read quorum, a bad single source, corruption, truncation, and every install crash retain valid state. No snapshot signatures or new consensus phase.	P1.4
Connections	Admission, thread, peer, and time bounds	Slow, fragmented, idle, and excess connections cannot exhaust the service or prevent healthy quorum progress.	P2.1
Transfers	Checked field and aggregate budgets	Overflow, over-limit, disk-full, premature EOF, and interrupted cleanup remain bounded.	P2.2

Boundary	Control	Required negative evidence	Gate
SQLite execution	Streaming, optional progress cancellation, runtime limits	Remote caps cancel and clean up; operator-approved long embedded or remote work completes with deadlines disabled.	P2.3
Filesystem	Root, ownership, mode, link, durability policy	Traversal, unsafe permissions, link swaps, full/read-only disk, and crash cannot expose secrets or replace last valid state.	P3.1
Data at rest	Base disclosure or explicitly supported encrypted profile	Base documentation never promises stolen-media protection. An edge profile passes direct-I/O, key, recovery, snapshot, backup, rekey, and media inspection tests before being advertised.	P3.1 / optional P3.4
C ABI	Early validation, checked casts, explicit ownership	Malformed valid-to-present boundary values fail consistently across supported build modes and C toolchains.	P3.2
Redirect	Seed-only under PSK; advertised node-ID pinning under mTLS	Unknown, wrong-certificate, malformed, and excessive-depth redirects fail closed; a valid single-seed mTLS redirect succeeds.	P1.1
Release evidence	Reproducible local target and recorded campaigns	A clean reviewer checkout reproduces the declared release result without a vendor-specific CI workflow.	P0.3 / P4.1

12. Release acceptance criteria

The initial network-capable production release is accepted only when all of the following are true:

1. The deployment documentation states that the application is the only database principal and owns all end-user authentication and authorization. No Zaxonlite RBAC or multi-tenant isolation is claimed.
2. The documented local-service profile uses a Unix-domain socket with tested owner/mode. Production TCP requires TLS 1.3 mutual authentication and has no silent PSK or plaintext fallback. The explicit development PSK mode requires an owner-only provider and numeric loopback for listener and every peer; its lack of confidentiality and unique node identity is prominent.
3. The configured issuer uses `--enrollment-ca-key` only on a TLS TCP listener. An existing mTLS principal creates a short-lived opaque bundle with `enroll-token`; `enroll` pins its bundled CA and issuer identity, generates the node key and CSR locally, redeems it once, atomically installs `node.key`, `node.crt`, and `ca.crt`, and removes the consumed bundle. Replay, expiry, wrong-node/wrong-cluster, malformed CSR, unsafe permissions, and ambiguous

post-consumption failures fail closed. The target must already exist in static membership; enrollment does not reconfigure Paxos.

4. A peer certificate identity, protocol hello, and configured membership entry must agree. An mTLS redirect target's certificate must match its advertised node ID; PSK-only redirects cannot leave the supplied seeds. Unknown and cross-cluster identities cannot open peer channels. The reloaded node-ID revocation file closes existing matching sockets and blocks new handshakes before certificate expiry. The static release documents offline voter replacement and does not claim online consensus membership management.
5. Application SQL cannot terminate Zaxonlite's outer transaction, attach or detach a database, access `__zaxon_*` state, or replace/change the WAL capture contract through a pragma. `SQLITE_OMIT_LOAD_EXTENSION` remains a tested build invariant. Supported SQLite migrations, DDL, triggers, safe pragmas, and long work continue to pass the compatibility suite.
6. No transferred snapshot replaces authoritative state unless a read quorum of the stop sign's configured voters confirms the same retained `CheckpointProofV1` for the existing Paxos-decided stop sign. The physical manifest and image must match that proof. This uses no snapshot-file signatures and no additional consensus phase. All staged-install crash points recover to either the old valid state or the fully verified new state.
7. Connection, thread, deadline, frame, message, result, transfer, disk, and aggregate budgets are documented, configurable where stated, checked before expensive work, and exercised at and beyond their boundaries.
8. Remote queries have configured SQL-text, row, copied-byte, and SQLite VM-step caps. Bounded materialization is accepted for this small SQLite-shaped API; a new streaming protocol is not a release gate. The exit criterion is compatibility with legitimate SQLite work when an authorized operator sets the VM budget to zero—not PostgreSQL-style workload governance.
9. Managed files and keys obey the documented root, permission, symlink, staging, fsync, and replacement rules under an unprivileged account. Base documentation states that files, payloads, snapshots, and backups are plaintext and does not claim survival of host/root compromise or offline media theft. Platform disk encryption is the supported practical mitigation.
10. If an edge-encryption profile is advertised, it names one exact supported encryption implementation and passes direct WAL/page I/O, snapshot, backup, crash recovery, wrong/mixed key, rekey, memory/key leakage, and stolen-media tests. Generic SQLCipher/VFS composability is never implied.
11. The public C ABI remains thin; every pointer/count pair is checked before use, conversions are checked, outputs are initialized, ownership is explicit, and the compiled C boundary suite passes. Safeclib is not a release dependency.
12. One clean local command produces the deterministic release evidence; longer crash, fuzz, and soak campaigns have recorded revisions, versions, seeds, and results. CI is not an initial-release acceptance criterion.
13. A focused adversarial review covers the defined design and has no unresolved in-scope critical or high findings. Findings based solely on database RBAC, Byzantine nodes, hostile co-resident application code, or post-root-compromise protection do not become product gates.

Embedded-only release distinction

An embedded-only build with no listener does not need Unix-socket, mTLS, enrollment, redirect, or slow-network gates for that artifact. It still needs the SQLite invariant guard, decided checkpoint proof if snapshot exchange is present, resource-safe inputs, filesystem/plaintext disclosure, C-ABI checks when exported, and the local assurance evidence.

13. Implementation order

The order minimizes rework: establish the real trust boundary first, build the transport identity used by redirects and enrollment, then carry the existing checkpoint decision through state transfer and bound all reachable work.

Order	Work package	Dependency and rationale
1	P0.1 product boundary + P0.3 harness	Freeze the supported topology and evidence format before changing public configuration and protocol.
2	P0.2 transport foundation	Implement Unix sockets and the TLS abstraction; make unsafe production configurations fail closed.
3	P1.1 enrollment, identity, revocation, redirects	Uses the TLS foundation and replaces the PSK with per-node identities; registry/denylist eviction closes stale credentials without adding dynamic membership.
4	P1.3 SQL invariant guard	Independent of enrollment and small enough to proceed alongside transport work; gate it with compatibility tests.
5	P1.4 transferred checkpoint proof	Reuses the already decided stop sign and an mTLS read-quorum confirmation; it adds neither file signatures nor a second consensus phase.
6	P2.1 and P2.2 network budgets	Apply identity-aware peer caps and consistent limits to the now-defined transport and transfer paths.
7	P2.3 SQLite-compatible query controls	Build streaming and optional cancellation on top of message/result budgets.
8	P3.1 filesystem/disclosure + P3.2 C ABI	Centralize host and language boundary rules, with much of this work able to run in parallel after constants are stable. P3.4 runs only if an encrypted edge profile is chosen.
9	P3.3 diagnostics + P4.2 conformance	Expose only the operational state needed to verify the controls and prevent book/configuration drift.
10	P4.3 focused review	Run after all in-scope release controls and evidence are present; remediate and replay the full local suite.
Later	P4.1 CI integration	Adopt only after the project chooses a CI system; call the same local targets rather than creating a second test path.

Critical network-release path

Product boundary → UDS + TLS → Join + identity + eviction → Checkpoint proof → Bounded service → Focused review

The SQL invariant guard and local assurance harness begin immediately after the boundary is frozen and need not wait for the network path. This preserves useful parallelism without expanding the architecture.

14. Reference basis

The design choices use primary project documentation:

1. Incus cluster joining uses a join token that conveys existing member addresses, a single-use secret, and the cluster certificate fingerprint, with configurable expiration. This is the model for Zaxonlite's deliberately smaller enrollment flow: Incus — Form a cluster.
2. Incus uses TLS client certificates for client/server authentication and recommends TLS 1.3. Zaxonlite borrows only this simple certificate pattern: Incus — Authentication.
3. Cowsql describes the same broad product shape—an embeddable C library that extends SQLite with replicated, fault-tolerant operation. It is a scope analogue, not the authority for the mTLS protocol: cowsql/cowsql.
4. Safeclib provides bounds-checking C library functions, but its own scope does not supply pointer provenance, ABI ownership, or Zig slice validation: rurban/safeclib.
5. SQLite exposes an authorizer at statement preparation, a progress handler for cancellable instruction intervals, and per-connection runtime limits. These enable the narrow invariant and optional availability controls in this plan: SQLite authorizer, progress handler, and runtime limits.
6. SQLite documents that `wal_autocheckpoint` replaces a previously registered WAL hook, which is why that pragma is a concrete capture-integrity finding: SQLite WAL hook.
7. SQLite documents that `SQLITE_OMIT_LOAD_EXTENSION` removes the extension loading mechanism. Zaxonlite already compiles its amalgamation with this option and must keep testing it: SQLite compile-time options.

15. Conclusion

The revised plan preserves Zaxonlite's intended character: a small SQLite-backed database replicated by Paxos, owned by one application, and deployed by one operator on a trusted network. It does not turn the project into a multi-user database or identity platform.

The security work is correspondingly concrete. Put authentication and policy in the application, use Unix-domain sockets locally, use one-time enrollment and per-node mTLS with immediate revocation on TCP, carry the existing Paxos-decided checkpoint proof through snapshot transfer, guard only the SQL operations and pragmas that can violate capture/replication, bound reachable resources, disclose the plaintext base profile, protect host files before host compromise, and keep the C ABI small. Prove those properties with a reproducible SQLite-style test program and a focused review of the system that is actually being built.