TECHNICAL WHITE PAPER

Enterprise Distributed Ledger Technology

Version 2.0.0 | ULedger Engineering Team | 2026
www.uledger.io

ABSTRACT

ULedger is an enterprise-grade distributed ledger platform designed for immediate finality, post-quantum cryptographic security, and native multi-chain interoperability. It combines a Council-based Byzantine Fault Tolerant (BFT) consensus mechanism with a dual zero-knowledge proof stack — Groth16 for per-transaction validity and PLONK for block integrity — and a proprietary Cross-Merkleization protocol that anchors independent chains to one another without trusted intermediaries. First-class support for ML-DSA-87 (NIST FIPS 204, CRYSTALS-Dilithium Level 5) makes ULedger one of the few enterprise ledger protocols natively resistant to quantum adversaries today. A hierarchical wallet identity system, sandboxed WebAssembly smart contracts, and native fungible/non-fungible token standards complete a platform targeting regulated industries, multi-party enterprise networks, and next-generation agentic commerce applications.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 The Problem Space

Enterprise distributed ledger deployments face a converging set of challenges that existing protocols handle in isolation, but rarely together:

- Quantum threat: Classical elliptic-curve and RSA-based signatures will be broken by sufficiently powerful quantum computers. NIST completed its first post-quantum cryptography standards in 2024, yet most production DLT platforms have not integrated them natively.
- Cross-chain fragmentation: Multi-chain architectures require either trusted bridge operators or complex relay networks. Neither approach is suitable for high-assurance enterprise use cases.
- Transaction privacy versus auditability: ZK proofs can validate transaction correctness without revealing private inputs, but most enterprise chains still rely on plain-text ledger inspection for auditing.
- Identity and access control: Enterprise user lifecycle — onboarding departments, teams, and users with role-based permissions — does not map cleanly onto simple public-key/address models.
- Immediate settlement: Probabilistic finality is unsuitable for financial and legal instruments. BFT protocols achieve deterministic finality, but their design must carefully balance fault tolerance, liveness, and operational simplicity.

ULedger addresses all five problems through a unified protocol design, detailed in the sections that follow.

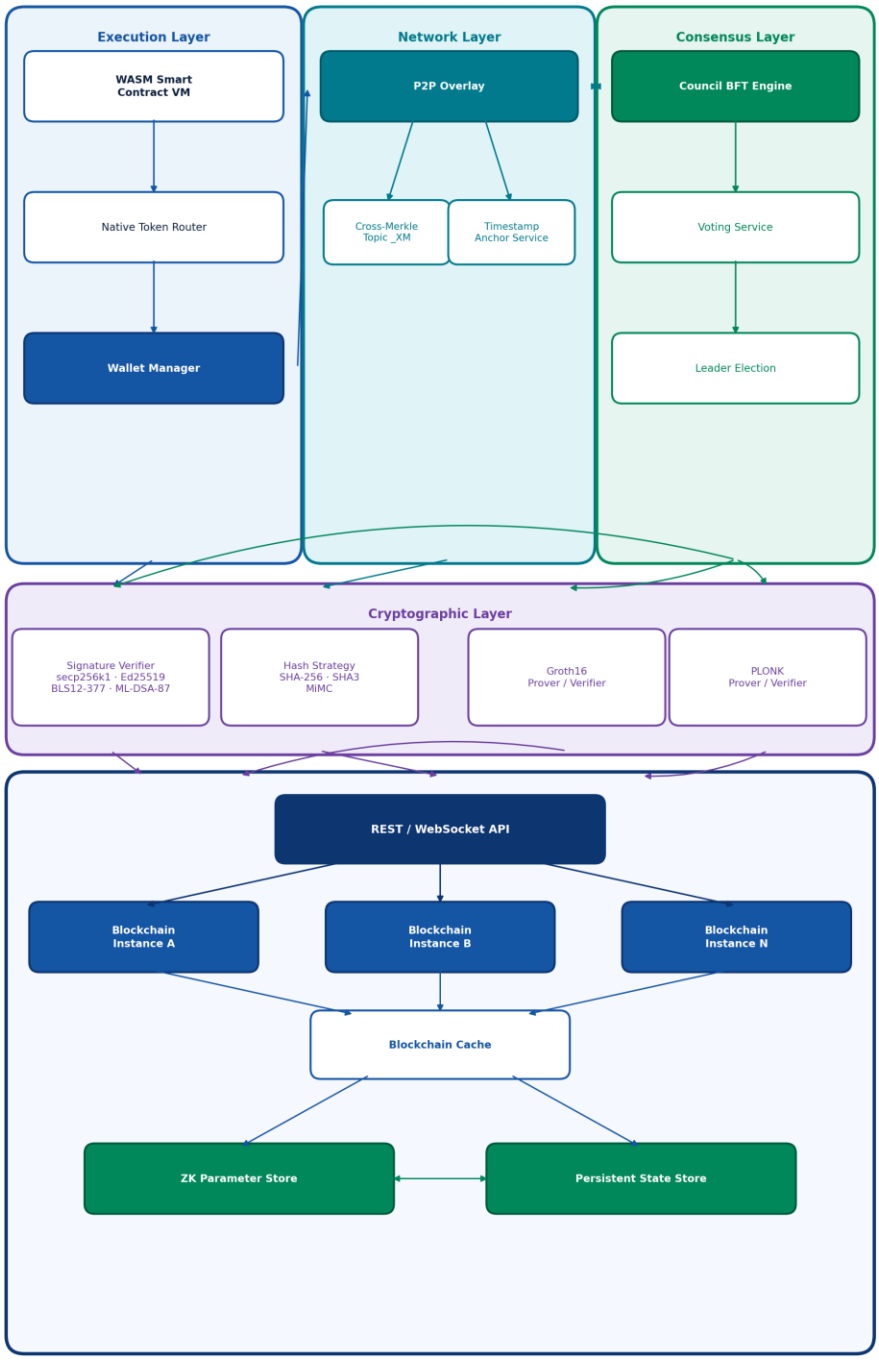
1.2 Design Principles

ULedger is built on five non-negotiable design principles:

- Security-first: Post-quantum cryptography and ZK proofs are not optional add-ons; they are core protocol primitives.
- Algorithm agility: No single signature or hash algorithm is hardcoded. Each chain deployment selects its algorithms from a supported set, enabling gradual migration as standards evolve.
- Composable interoperability: Cross-chain state references are embedded directly in block structure, not layered on top via external bridges.
- Enterprise identity: The wallet model mirrors organizational hierarchies natively.
- Deterministic finality: Every committed block is final. There are no forks, no reorgs, no probabilistic confirmation windows.

2. ARCHITECTURE OVERVIEW

ULedger is a multi-chain node that manages one or more independent blockchains simultaneously. Each blockchain is a self-contained state machine with its own transaction pool, consensus instance, cryptographic configuration, and smart contract environment. Cross-chain state is shared through the Cross-Merkleization protocol described in Section 7.



2.1 Per-Blockchain Configuration

Every blockchain instance is independently configured with:

- A key type for signing operations (secp256k1, ed25519, bls12377, mldsa87)
- A hashing strategy for block and transaction IDs (HASH_SHA256, HASH_SHA3_256, HASH_SHA3_512, HASH_MIMC_BN254, HASH_MIMC_BLS24315, HASH_MIMC_BW6_761)
- A consensus configuration (transaction threshold, time threshold, minimum council size, timeouts)
- A set of feature flags (enableXMOnBlockMint, enableTxProof, enableJoinCouncilOnStartup)
- Optional principal chain relationships for Cross-Merkleization topology

This algorithm agility means an existing deployment can migrate to post-quantum keys on new chains while legacy chains continue operating, with both anchored together via Cross-Merkleization.

2.2 Key Differentiators

Feature	Description
Cross-Merkleization	Proprietary multi-chain state anchoring using ZK-SNARK proofs, ensuring cryptographic consistency across distributed ULedger chains without trusted intermediaries.
Post-Quantum Ready	Native ML-DSA-87 (CRYSTALS-Dilithium) signatures protect deployments against current and future quantum computing threats. NIST FIPS 204 compliant.
Multi-Cryptographic Suite	Full support for secp256k1, ED25519, BLS12-377, and post-quantum algorithms for maximum deployment flexibility.
BFT Consensus	Modified Byzantine Fault Tolerant consensus with deterministic leader election delivers immediate, irreversible finality.
WASM Smart Contracts	WebAssembly-based execution with gas metering ensures predictable, resource-controlled on-chain logic.
Native Token Standards	Built-in fungible, non-fungible, and multi-token standards equivalent to ERC-20, ERC-721, and ERC-1155.
Hierarchical Wallets	Parent-child wallet relationships with cascading access control model real-world enterprise permission structures.

3. COUNCIL BFT CONSENSUS

3.1 Overview

ULedger employs a Council Leader Proposal consensus algorithm: a modified BFT protocol with deterministic leader election, cryptographically signed votes, and a three-phase commit sequence. Council members are known, authenticated participants — the model targets consortium and private networks where identity is established out-of-band, providing performance and determinism that permissionless Sybil-resistant designs sacrifice.

3.2 Council Membership

A council is the set of nodes authorised to vote on and finalise blocks for a given blockchain. Membership is established by broadcasting a signed JoinCouncil message containing the node's instance identity and peer address, verified against the chain's registered key. The minimum viable council size is 3 nodes, providing tolerance for 1 Byzantine member.

3.3 Leader Election

The block proposer (leader) for each height and round is selected deterministically:

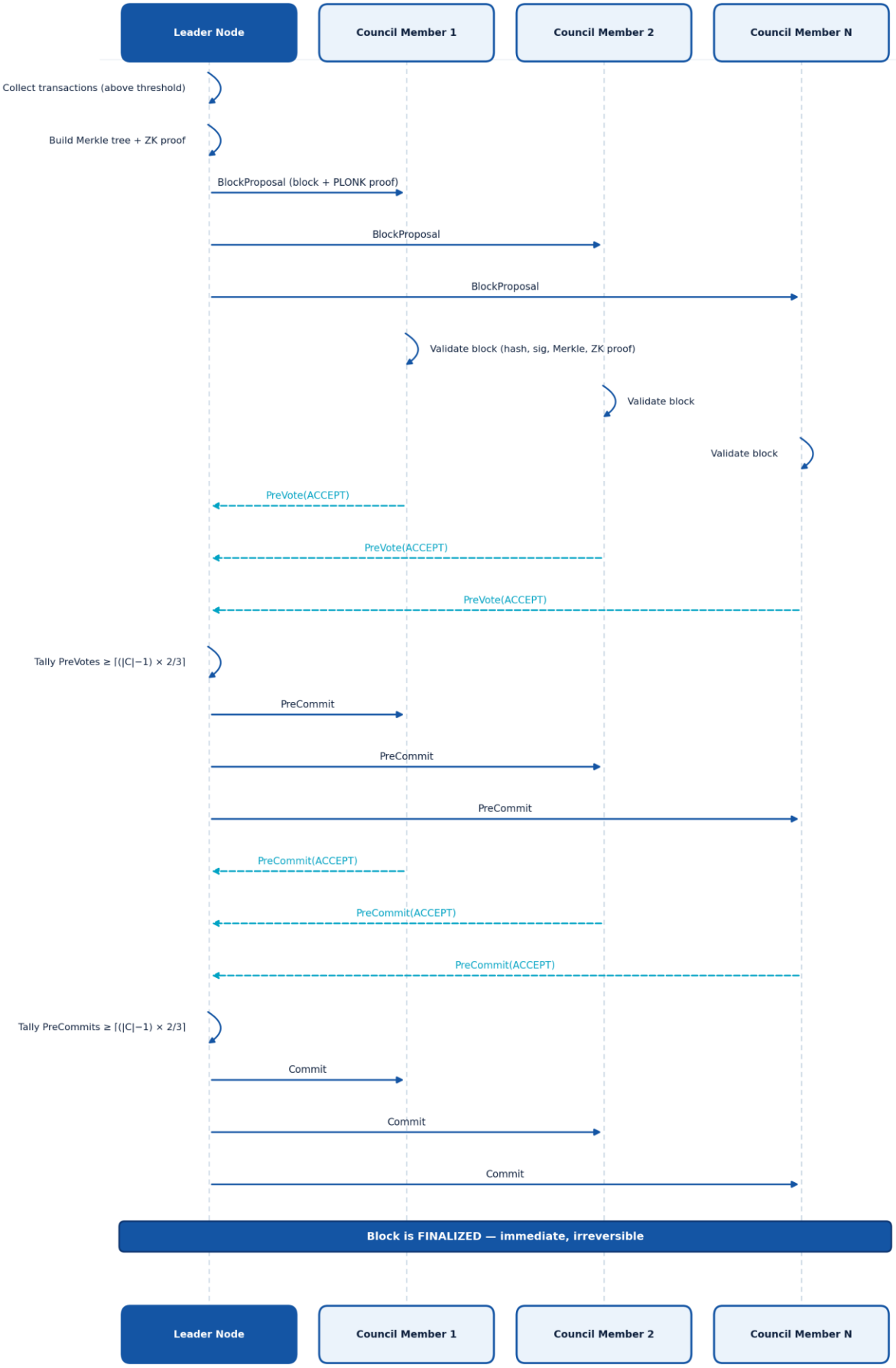
```
seed  = prevBlockHash ‖ height (8 bytes, big-endian) ‖ round (8 bytes, big-endian)
hash  = SHA3-512(seed)
index = uint64(hash[0:8]) mod |council|
```

This function is:

- Unpredictable in advance — requires knowledge of the previous block hash, which is not known until finalization.
- Deterministic given inputs — every council member independently computes the same leader.
- Round-aware — if the current leader fails to produce a block within the configured timeout, round increments and a new leader is elected from the same height.

3.4 Three-Phase Voting Protocol

Block finalization proceeds through three phases. Each vote carries: BlockHash, BlockHeight, Round, Stage, CouncilMemberAddress, PeerId, Vote (boolean accept/reject), and a cryptographic Signature over the vote body.


Leader Node
Council Member 1
Council Member 2
Council Member N
Collect transactions (above threshold)
Build Merkle tree + ZK proof
BlockProposal (block + PLONK proof)
BlockProposal
BlockProposal
Validate block (hash, sig, Merkle, ZK proof)
Validate block
Validate block
PreVote(ACCEPT)
PreVote(ACCEPT)
PreVote(ACCEPT)
Tally PreVotes ≥ ⌈(|C|−1) × 2/3⌉
PreCommit
PreCommit
PreCommit
PreCommit(ACCEPT)
PreCommit(ACCEPT)
PreCommit(ACCEPT)
Tally PreCommits ≥ ⌈(|C|−1) × 2/3⌉
Commit
Commit
Commit
Block is FINALIZED — immediate, irreversible
Leader Node
Council Member 1
Council Member 2
Council Member N

3.5 Quorum Rule

A phase advances when the number of agreeing votes meets:

```
required = ceil ((|council| − 1) × 2/3)
```

This formulation ensures that even if the proposing leader node is counted separately, a strict supermajority of the remaining council must agree, providing fault tolerance of floor((|council| − 1) / 3) Byzantine members.

3.6 PreVote Validation

Before casting a PreVote, each council member independently validates:

1. Correct block height continuity (Height == currentHeight + 1)
2. Block contains at least one transaction
3. Block hash integrity — recomputed from block data and compared against the proposed hash
4. Leader signature over the block hash
5. Cross-reference PLONK proofs for any embedded cross-chain references
6. Transaction structure and ValidUntilHeight window
7. PLONK zero-knowledge proof for block header correctness

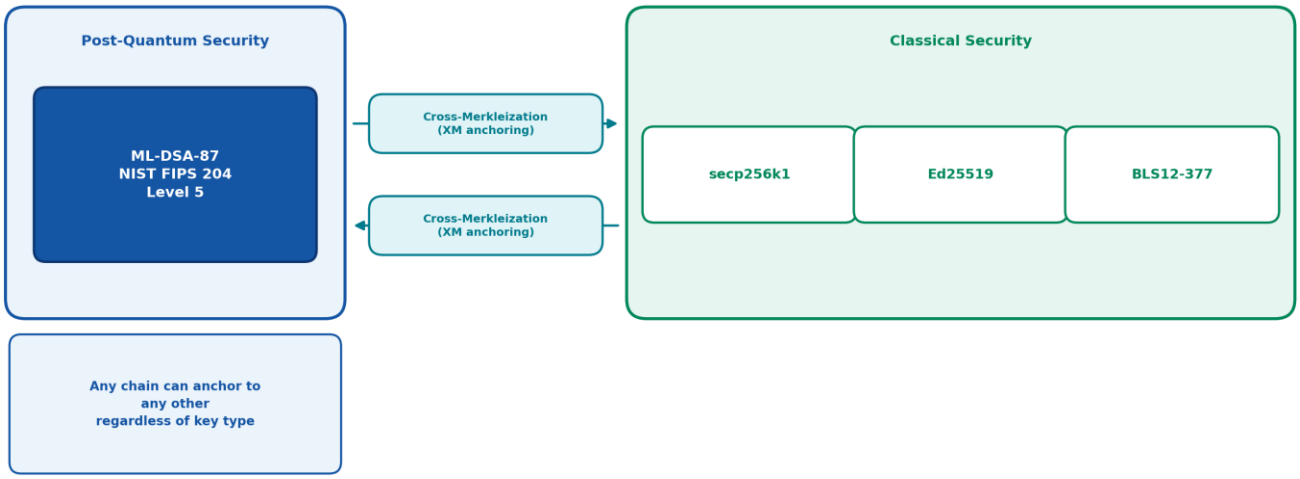
3.7 Failure Recovery

Failure Scenario	Response
Leader timeout (no proposal)	Round increments → new leader elected for same height
Malicious proposal	PreVote rejected by honest majority → timeout → new round
Network partition	Consensus pauses until ⅔ majority is available again
Double-vote (equivocation)	Detected and flagged; duplicate votes discarded

3. COUNCIL BFT CONSENSUS

4.1 Block Structure

Each block carries both protocol metadata and cryptographic commitments across its full content:



4.2 Transaction Structure

Transactions have a clear separation between sender-provided inputs and node-assigned outputs:

Inputs (sender-provided)

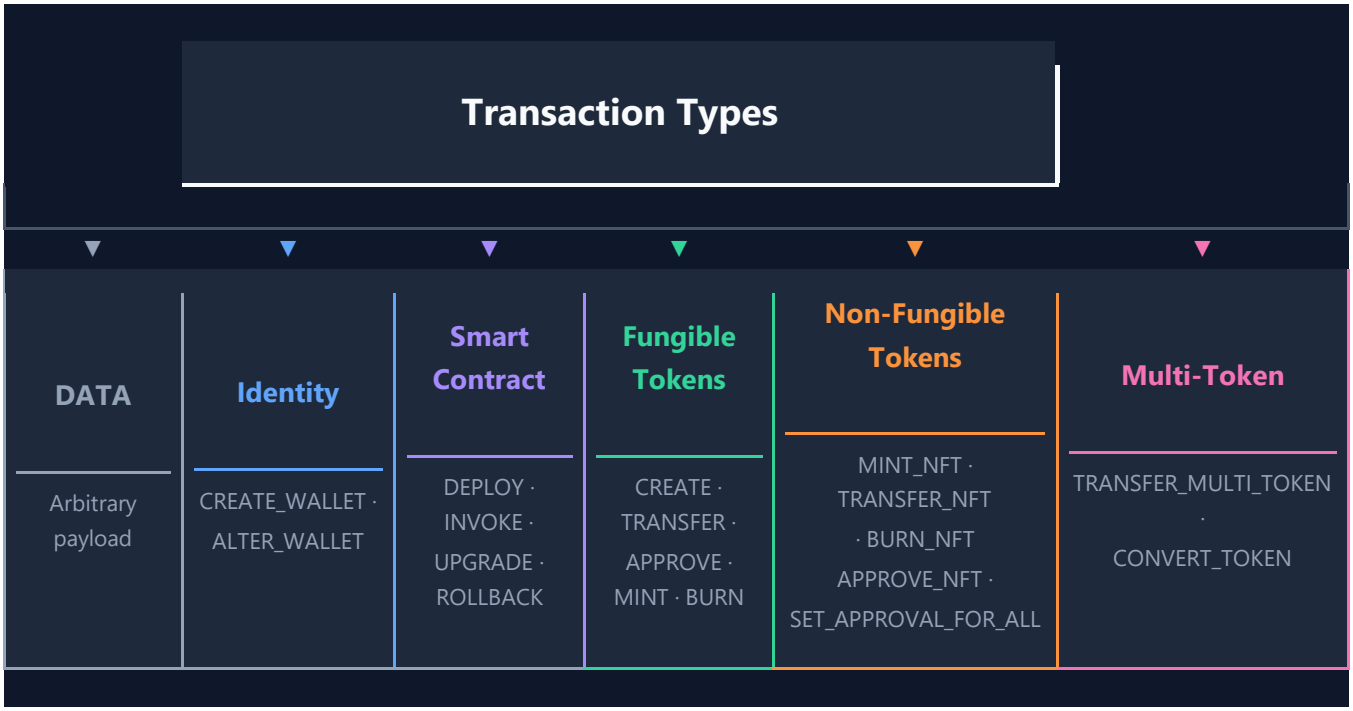
Field	Description
BlockchainId	Target chain identifier
From	Sender wallet address
To	Recipient wallet address
Payload	Operation data
PayloadType	Transaction type enum
SenderSignature	Signature over commitment
SenderTimestamp	Claimed send time
Suggestor	Optional delegating address
KeyType	Algorithm hint — overridden by the blockchain's configured key type on ingestion

Outputs (node-assigned)

Field	Description
TransactionId	Hash of canonical body including timestamps
BlockHeight	Height where transaction was included
Timestamp	Network-verified timestamp
Status	Execution result
GasUsed	Execution cost (smart contracts)
Proof	Groth16 ZK proof of validity

4.3 Transaction Types

ULedger defines a typed transaction taxonomy covering all protocol operations:



4.4 Transaction Commitment and Signing

To accommodate ZK circuit field-size constraints (specifically, the hard limit of the BN254 scalar field), four string fields in the signing commitment are each split into two 16-byte halves via SHA-256: BlockchainId, From, To, and Suggestor. The SHA-256 digest of each field (32 bytes) is partitioned into a high half (bytes 0–15) and a low half (bytes 16–31). The payload is Merkleized at a fixed depth of 6 levels (64 leaves, power-of-two,

zero-padded) before being committed, enabling bounded in-circuit Merkle path verification regardless of payload size.

5. CRYPTOGRAPHIC FOUNDATION

5.1 Signature Algorithms

ULedger supports four signature schemes as first-class citizens, selectable per blockchain deployment:

Algorithm	Type	Security Level	Key Size	Sig Size	Best For
secp256k1 (ECDSA)	Classical ECC	128-bit	33 bytes	64–72 bytes	General-purpose, broad ecosystem compatibility
Ed25519 (EdDSA)	Classical ECC	128-bit	32 bytes	64 bytes	High-throughput, IoT, deterministic signing
BLS12-377	Pairing-based ECC	128-bit	48 bytes	96 bytes	Signature aggregation, threshold, and multi-sig
ML-DSA-87	Post-Quantum (Lattice)	256-bit (NIST Level 5)	~2,592 bytes	~4,627 bytes	Quantum-resistant deployments

ML-DSA-87: Post-Quantum Security

ML-DSA-87 (CRYSTALS-Dilithium, standardized as NIST FIPS 204 in 2024) is the highest security variant of the Module Lattice-based Digital Signature Algorithm. It provides:

- Security Level 5 — equivalent to 256-bit classical security, the same level as AES-256.
- Quantum resistance — based on the hardness of Module Learning with Errors (MLWE), a problem for which no efficient quantum algorithm is known.
- Standard compliance — the first and only post-quantum signature standard finalized by NIST. Choosing ML-DSA-87 today means no migration risk when quantum computers capable of breaking elliptic curves become viable.

Deployments concerned about long-term data confidentiality — financial records, legal instruments, healthcare data — should run on mldsa87-configured chains. ULedger's algorithm agility allows mixed networks: a PQC-native chain can anchor its state to classical chains via Cross-Merkleization, enabling incremental migration.

5.2 Hash Strategies

Hash functions are configurable per chain. ULedger ships two families:

General-Purpose Hashing (block IDs, transaction IDs, Merkle display roots)

Config Value	Algorithm	Output	Use
HASH_SHA256	SHA-256	256 bits	Default; widest compatibility
HASH_SHA3_256	SHA3-256	256 bits	NIST Keccak variant
HASH_SHA3_512	SHA3-512	512 bits	Higher collision resistance

ZK-Friendly Hashing (in-circuit Merkle trees)

Config Value	Algorithm	Curve Field	Primary Use
HASH_MIMC_BN254	MiMC	BN-254	Transaction commitment hashing (secp256k1 Groth16 circuits)
HASH_MIMC_BLS24315	MiMC	BLS24-315	Block Merkle tree construction (PLONK circuits)
HASH_MIMC_BW6_761	MiMC	BW6-761	General ZK-friendly hashing; default for BLS12-377 transaction flows

PERFORMANCE NOTE

MiMC is approximately 100× more constraint-efficient inside a ZK circuit than SHA-256. Using MiMC natively for in-circuit operations keeps proof generation times and circuit sizes practical, while classical SHA-3 family hashes remain available for display and storage.

5.3 Leader Election Hash

Block leader election uses SHA3-512 specifically for its 512-bit output, providing a wide, uniform distribution over uint64 modulo council size with negligible bias.

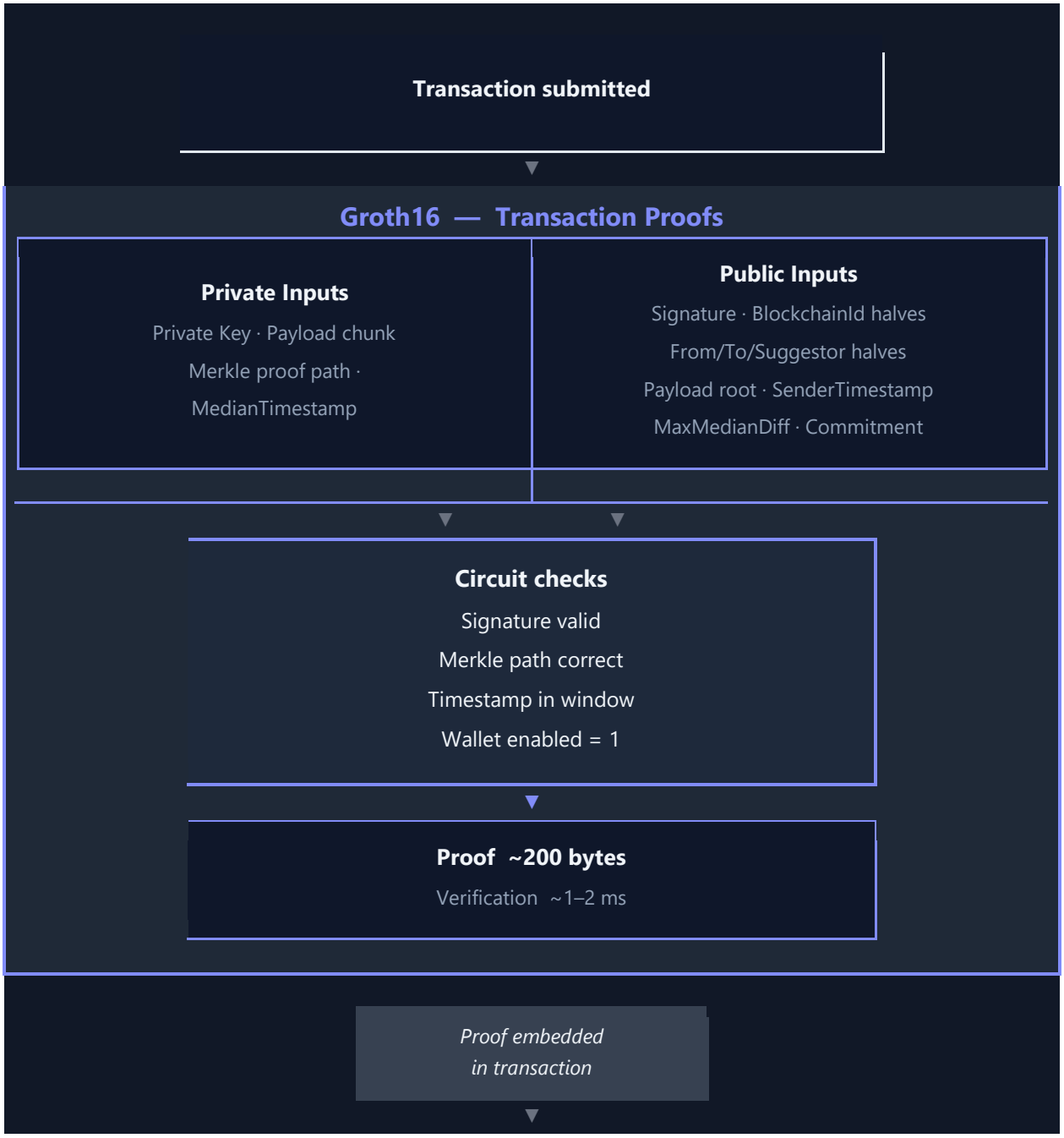
5.4 Key Derivation

Node and wallet keys can be derived deterministically from a single 256-bit master seed plus a salt using a standard key derivation function. This means a single backup recovers all key material across all supported algorithms — a critical operational requirement for enterprise key management.

6. ZERO-KNOWLEDGE PROOF SYSTEM

ULedger integrates two complementary ZK proof systems targeting different verification scopes and performance tradeoffs:

ZK Proof Pipeline




Block finalized
PLONK — Block Header Proofs
Private Inputs
TransactionCount ·
MerkleChunk
MerkleIndex
Public Inputs
Merkle root · Block index
Previous index ·
MerkleElements
Circuit checks
Index = PreviousIndex + 1
Tx count ≥ 1
Merkle path valid
Root correct
Proof ~500 bytes
Verification ~3–5 ms
Proof embedded in block header
Proof broadcast via _XM topic
Chain state
Cross-chain anchoring

6.1 Groth16: Per-Transaction Proofs

Groth16 proofs are generated for each transaction and embedded as a compact, self-contained validity certificate. They prove, without revealing private inputs:

- The sender possesses a private key matching a registered wallet address
- The signature over the transaction commitment is valid
- The payload's Merkle root matches the signed root
- The transaction timestamp falls within the network's median time window
- The sending wallet is enabled at proof time

Two circuit variants cover different signing algorithms:

- secp256k1 path: emulated in-circuit ECDSA verification over the BN-254 field
- BLS12-377 path: in-circuit pairing check (Miller loop + final exponentiation) over the BW6-761 field

When enableTxProof is active on a chain, external validators can verify a transaction's correctness using only the public inputs and the ~200-byte proof — no wallet state lookup, no signature re-verification, no payload exposure.

6.2 PLONK: Block Header Proofs

PLONK proofs are generated by the block proposer and cover the entire block header including its Merkle root. They prove:

- The block's Merkle root is correctly computed from its inputs (transaction IDs, previous Merkle root, cross-chain references)
- The block index is exactly one greater than the previous block index (no height gaps or skips)
- At least one transaction is included
- The Merkle path for each leaf is valid

PLONK requires a separate proving key, verifying key, and structured reference string (SRS) per Merkle tree depth. Since ULedger supports depths 2 through 16, there are 15 independent circuit instances, each compiled and set up independently. This is a one-time offline process; during runtime each depth's keys are loaded into memory and selected by the block's actual Merkle tree depth. Groth16 similarly uses per-circuit trusted setups — one per signing algorithm variant.

6.3 Proof-Enabled External Validation

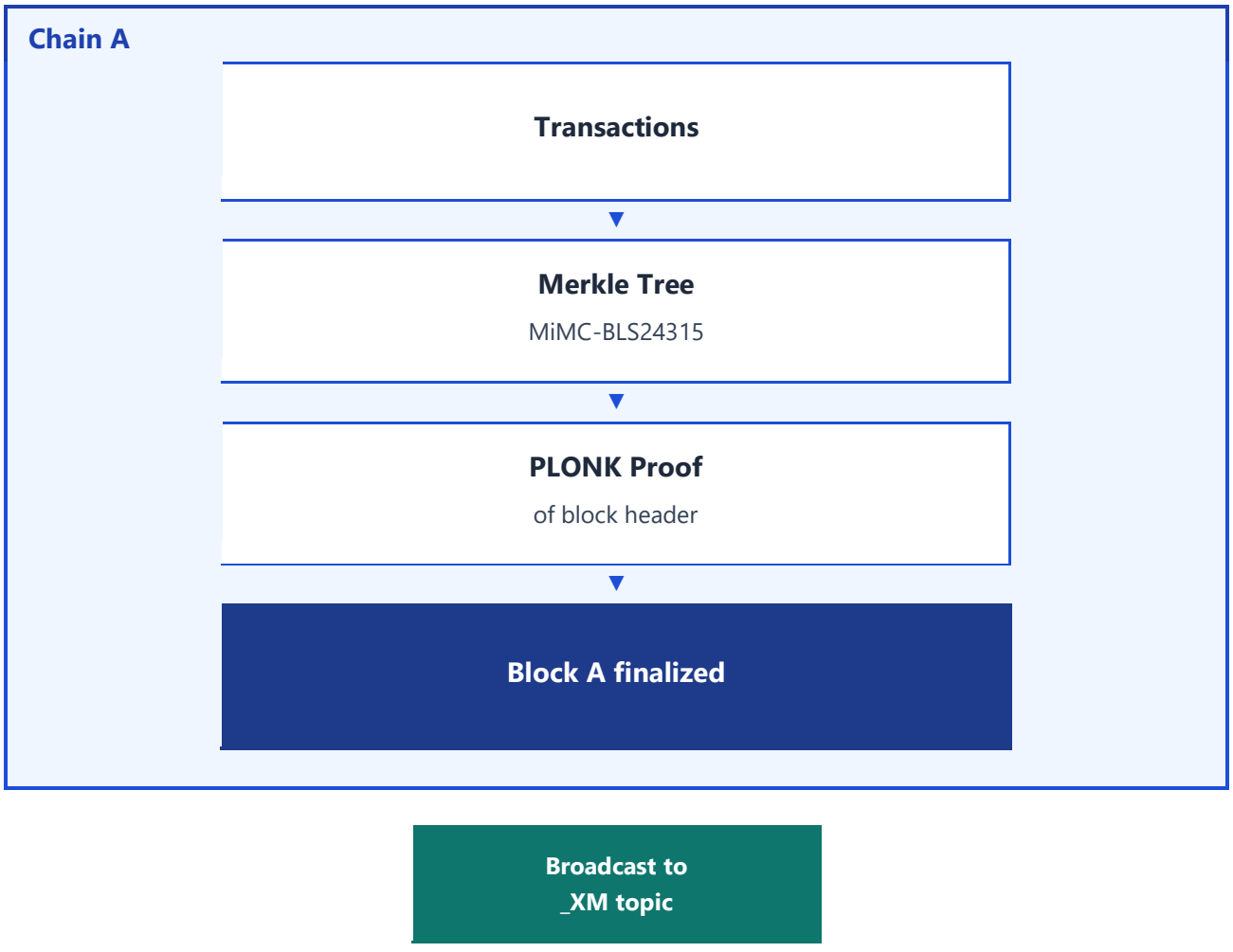
The combination of Groth16 transaction proofs and PLONK block proofs enables a powerful external validation model: a lightweight client can verify an entire block's correctness — including the validity of every included transaction — by checking two compact proofs without downloading or replaying any chain state. This is foundational for light clients, cross-chain verifiers, and auditing systems.

7. CROSS-MERKLEIZATION

Cross-Merkleization (XM) is ULedger's proprietary protocol for trustless, cryptographically-verified state anchoring between independent blockchain instances. It eliminates the need for bridge contracts, relay operators, or external oracle services.

7.1 Core Concept

When a block is finalized on Chain A, its PLONK-proven block header includes the Merkle root committing to all its transactions. It is broadcast to all other chains inside the network. Each receiving chain embeds that foreign Merkle root directly into its own next block's Merkle tree. The foreign chain's state becomes a cryptographic commitment in the receiving chain's immutable record.



7.2 Merkle Tree Construction

The block-level Merkle tree is the centerpiece of XM. Its inputs, in order:

1. Transaction IDs — hex-encoded, decoded to bytes, one per transaction
2. Previous Merkle root — the Merkle root of the prior block, linking chain history
3. Cross-references — each foreign chain's Merkle root, sorted by BlockchainId for determinism

These inputs are concatenated as a byte stream, chunked into 16-byte segments, and padded to the next power of two. The tree is constructed using MiMC hashing over the BLS24-315 field, producing both:

- A ProofMerkleRoot — the raw field element root used as a public input in the PLONK circuit
- A MerkleRoot — the ProofMerkleRoot re-hashed with the chain's configured strategy, used as the canonical block Merkle root for external consumption

Tree depth ranges from 2 to 16 levels depending on the number of inputs. Separate PLONK proving and verification keys exist per depth, enabling constant-time per-depth verification.

7.3 Cross-Reference Validation

Before a cross-reference is included in a block, the receiving chain independently verifies the PLONK proof attached to the CrossReferenceMessage. This verification:

- Confirms the Merkle root is correctly derived from the foreign block's data
- Confirms the block index progression is valid
- Requires no trust in the broadcasting node

Only after successful PLONK verification is the foreign Merkle root incorporated into the receiving block's tree.

7.4 Security Properties of Cross-Merkleization

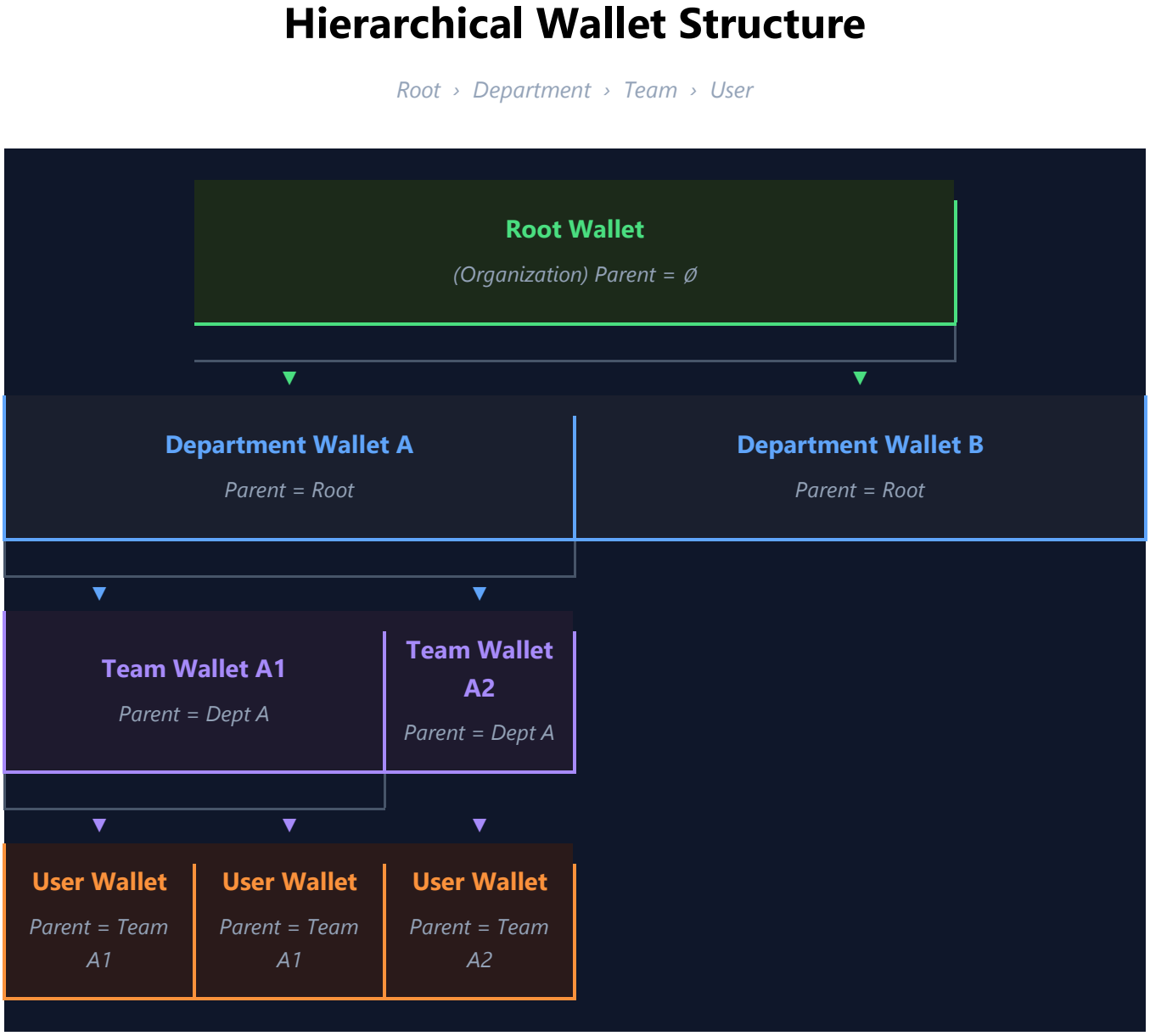
Property	Guarantee
Integrity	A cross-reference cannot be forged without a valid PLONK proof
Non-repudiation	Once a foreign root is in a finalised block, it is immutably anchored
Independence	No trust required in the foreign chain's operators
Efficiency	O(1) verification per cross-reference (~3–5 ms, ~500 bytes)
Composability	Any number of chains can mutually anchor, enabling mesh topologies

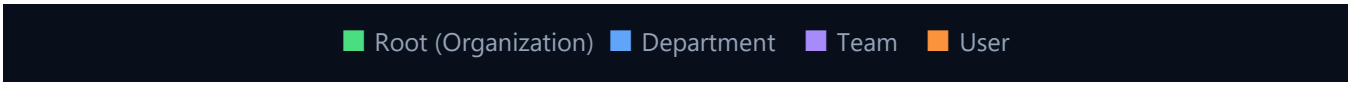
8. HIERARCHICAL WALLET SYSTEM

8.1 Motivation

Enterprise user management requires more than public key → address mappings. Organisations have departments, teams, and individual users. Permissions flow top-down. Offboarding must be atomic across an entire subtree. ULedger's hierarchical wallet model maps these structures directly onto the protocol layer.

8.2 Wallet Structure





Each wallet record contains:

Field	Description
address	SHA-256 of lowercase public key hex
publicKey	Wallet's public key
keyType	Signing algorithm for this wallet
parent	Parent wallet address (empty for root)
satelliteWallets	List of direct child addresses
authGroups	Named permission groups with CRUD capabilities
enabled	Active/disabled flag

8.3 Permission Model

Authorisation groups give named sets of CRUD-style permissions. Built-in groups:

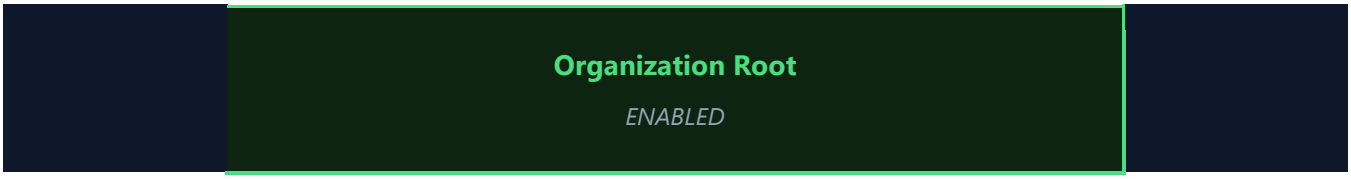
Group	Capabilities
admin	Full create, read, update, delete on all resources
wallet	Wallet management operations

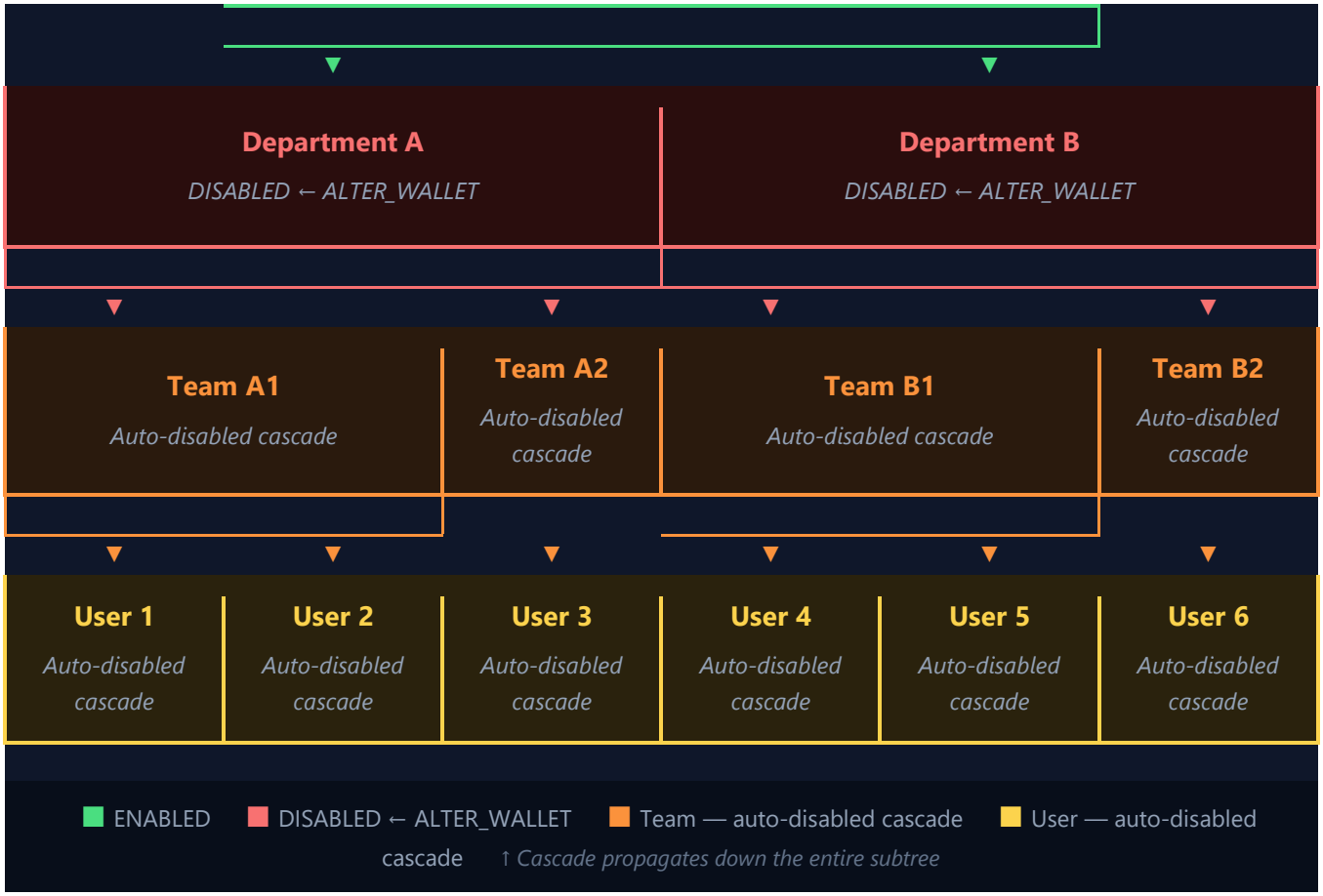
Custom groups can be defined per deployment. A wallet with the admin group possessing all four CRUD flags is considered an administrator.

8.4 Lifecycle Operations

Creating a child wallet: Requires the sender to have an admin or wallet (with create permission). The parent wallet must exist and be enabled. On creation, the child is added to the parent's satelliteWallets list atomically.

Disabling a wallet (offboarding): Disabling a wallet triggers a BFS cascade: all descendant wallets in the subtree are disabled atomically. A disabled organizational unit means every wallet under it is immediately inactive — no orphaned active children.




Department A
DISABLED ← ALTER_WALLET
Department B
DISABLED ← ALTER_WALLET
Team A1
Auto-disabled cascade
Team A2
Auto-disabled cascade
Team B1
Auto-disabled cascade
Team B2
Auto-disabled cascade
User 1
Auto-disabled cascade
User 2
Auto-disabled cascade
User 3
Auto-disabled cascade
User 4
Auto-disabled cascade
User 5
Auto-disabled cascade
User 6
Auto-disabled cascade
ENABLED
DISABLED ← ALTER_WALLET
Team — auto-disabled cascade
User — auto-disabled cascade
↑ Cascade propagates down the entire subtree

9. SMART CONTRACTS AND NATIVE TOKEN STANDARDS

9.1 WebAssembly Execution Environment

Smart contracts are deployed and executed as WebAssembly (WASM) modules within a sandboxed virtual machine. WASM is the same bytecode format used by modern web browsers and edge computing platforms, enabling contracts to be authored in any language with a WASM compilation target, including Rust, Go, C++, and AssemblyScript. The execution environment provides:

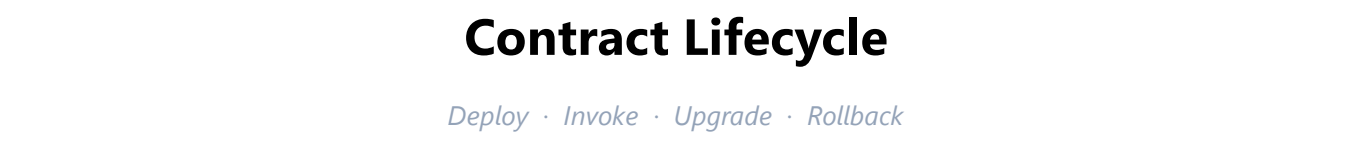
- Gas metering per host function call and per memory operation (growth and allocation), with differentiated costs for reads, writes, deletes, and key checks against storage
- Linear memory isolation — contracts cannot access memory outside their own heap
- Atomic state transactions — storage writes commit on success, roll back on failure
- Call stack depth limits (default: 1,000 frames) and a circuit breaker for repeated failure protection

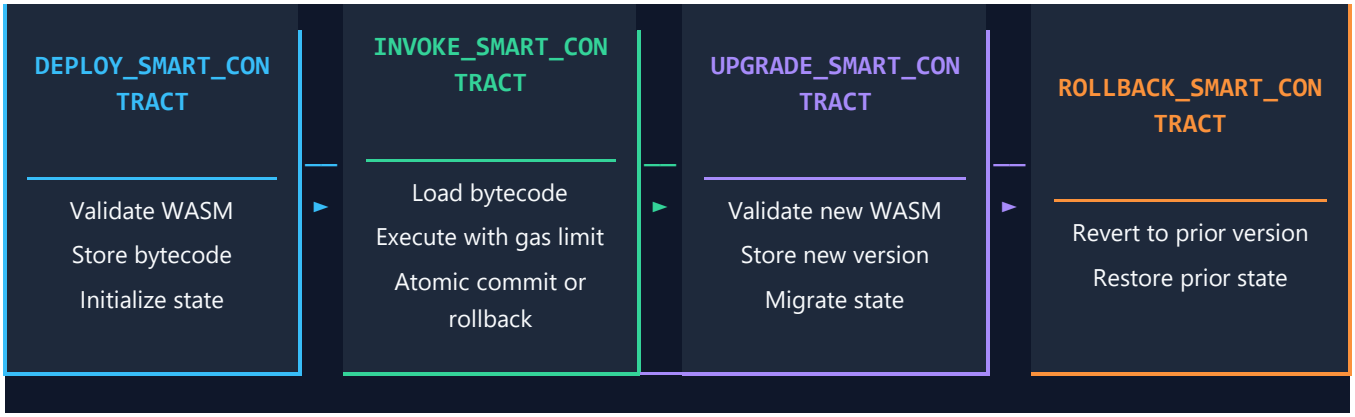
9.2 Host Functions

Contracts interact with the ledger through a defined host function interface:

Function	Description
get_storage(key)	Read from contract state
set_storage(key, value)	Write to contract state
delete_storage(key)	Remove from contract state
has_key(key)	Check key existence
get_caller()	Invoking wallet address
get_owner()	Contract owner address
get_block_height()	Current block height
emit_event(topic, data)	Emit a subscribable event
log(message)	Debug logging

9.3 Contract Lifecycle





9.4 Native Token Standards

ULedger implements token operations as native transaction types — not as smart contract logic — enabling optimised execution, native ZK proof coverage, and simpler integration:

Standard	Equivalent	Operations
Fungible Tokens	ERC-20	CREATE_TOKEN · TRANSFER_TOKEN · APPROVE_TOKEN · MINT_TOKEN · BURN_TOKEN
Non-Fungible Tokens	ERC-721	MINT_NFT · TRANSFER_NFT · BURN_NFT · APPROVE_NFT · SET_APPROVAL_FOR_ALL
Multi-Token Batch	ERC-1155	MINT_MULTI_TOKEN · TRANSFER_MULTI_TOKEN · CONVERT_TOKEN

Native token operations participate in the same ZK proof and consensus pipeline as all other transactions, providing equivalent finality and auditability guarantees.

10. NETWORK LAYER AND TIME ANCHORING

10.1 P2P Topology

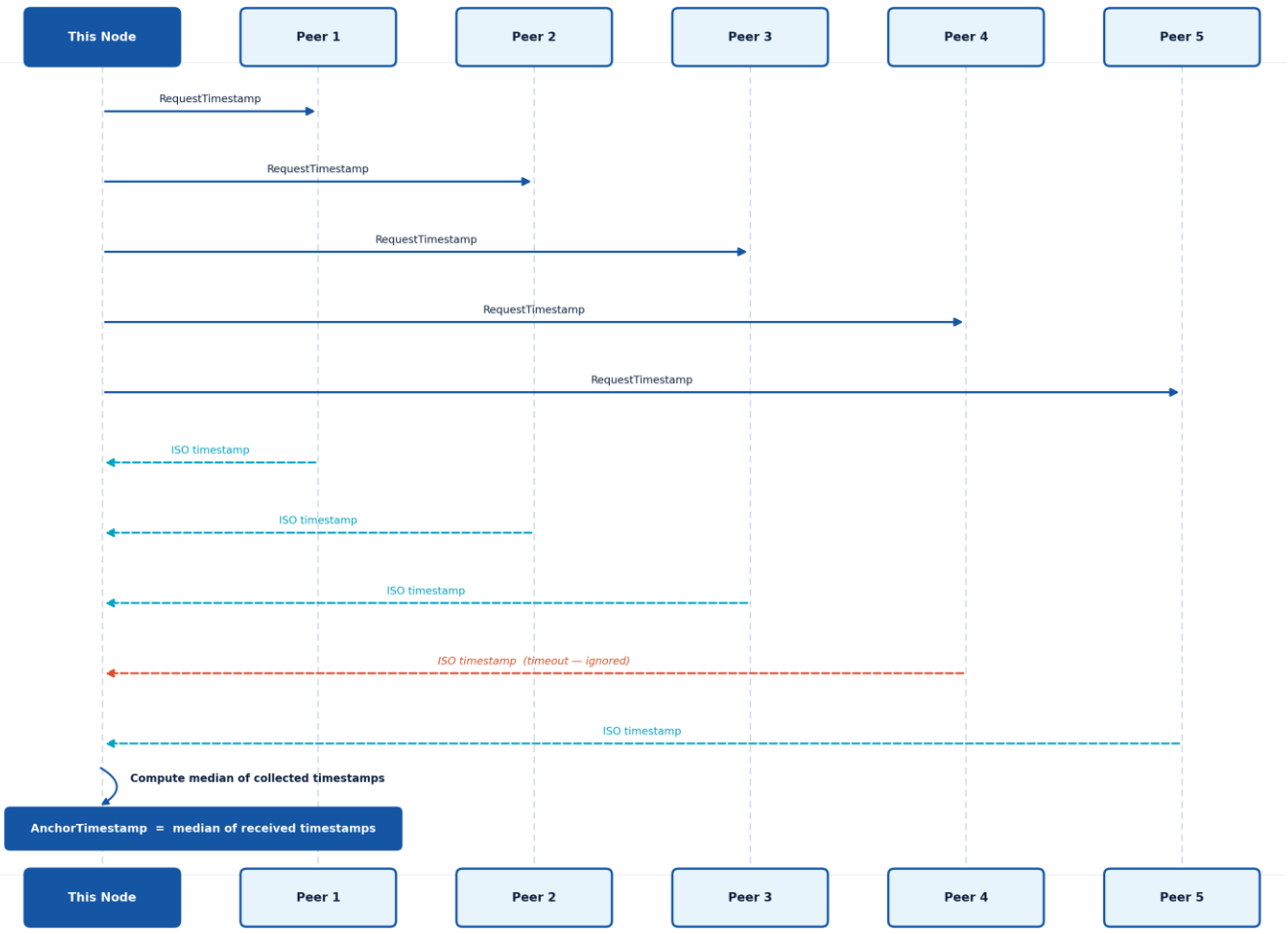
ULedger nodes form a structured peer-to-peer overlay with Kademlia DHT-based routing and publish-subscribe messaging. Each blockchain instance operates on its own topic namespace:

- {blockchainId} — main topic for transactions, block proposals, and votes
- {blockchainId}_XM — dedicated topic for Cross-Merkleization messages

Nodes discover peers through bootstrap nodes specified in configuration, then maintain connections through the DHT.

10.2 Distributed Median Timestamp

Transaction replay protection requires a reliable shared notion of current time across distributed nodes. ULedger uses a distributed median timestamp protocol:



Key Properties:

- Queries a configurable set of peers with a 5-second timeout per request
- Computes the median (not mean) to resist outlier manipulation
- A transaction is valid only if anchorTimestamp − senderTimestamp ≤ 5 seconds (enforced via MAX_MEDIAN_DIFF = 5s)

This validity window prevents replay attacks while tolerating reasonable network clock drift. Since a single compromised or clock-skewed peer cannot move the median more than a bounded amount, this provides Byzantine-tolerant time anchoring without a dedicated time oracle.

10.3 Council Join Protocol

Nodes announce council membership with a signed JoinCouncil message:

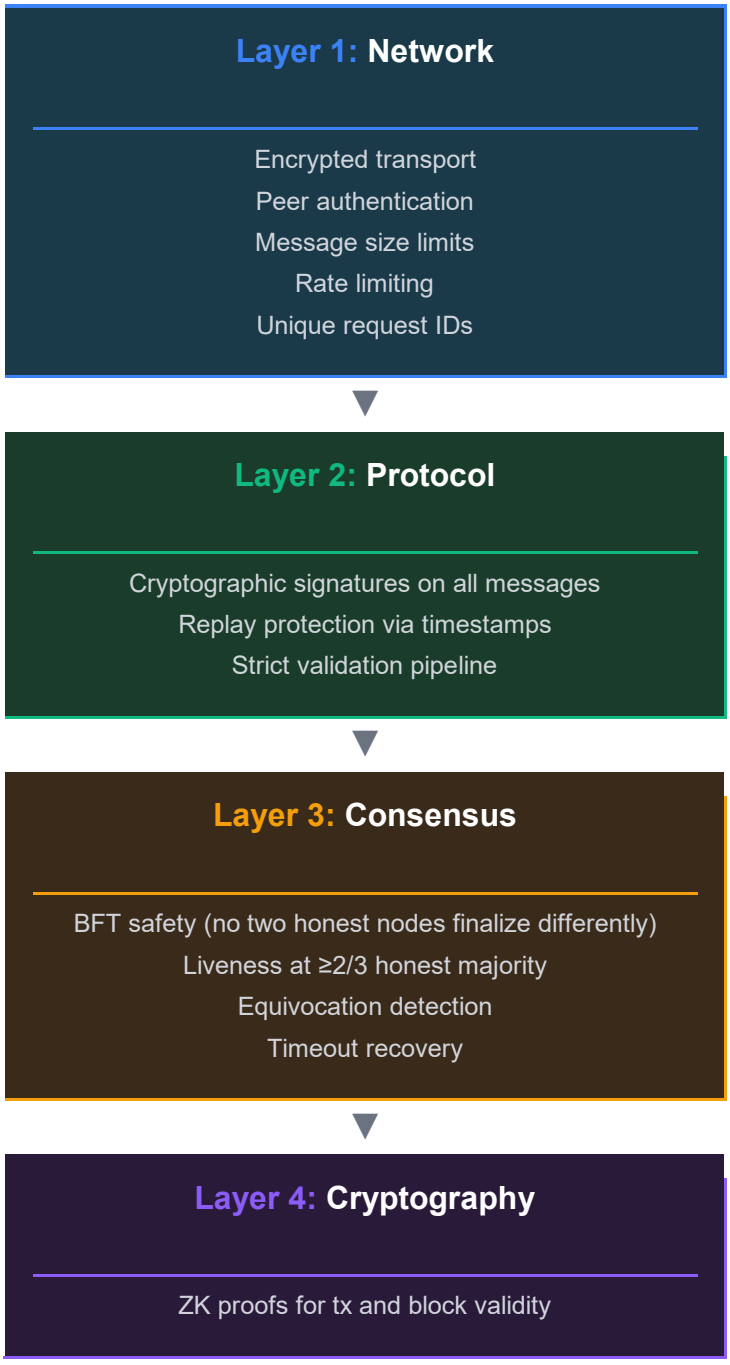
```
JoinCouncilData = { instanceId, peerId }
Signature = Sign(JoinCouncilData, nodePrivateKey)
```

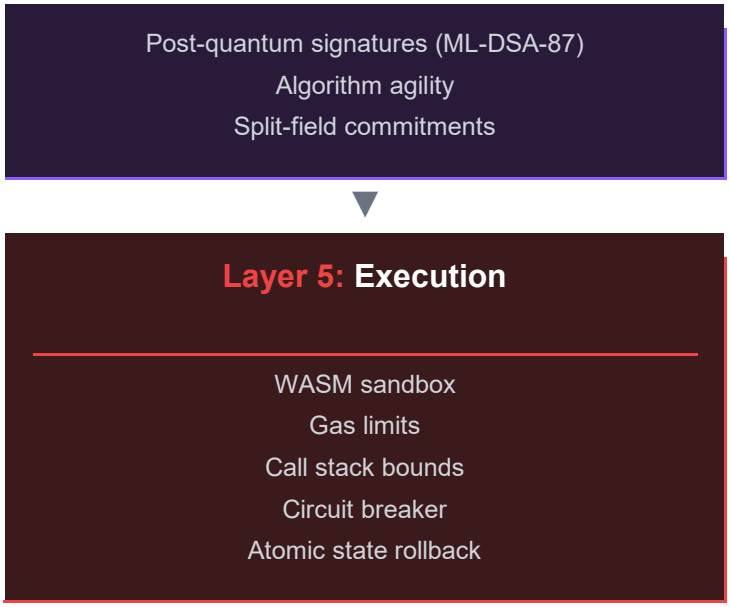
The receiving council verifies the signature against the node's registered key before admitting the member. Principal blockchains may restrict join broadcasts to enforce permissioned topologies.

11. SECURITY PROPERTIES

11.1 Defense-in-Depth Layers

Five-Layer Defense Stack





11.2 Consensus Safety Guarantees

Property	Guarantee
Safety	No two honest council members will finalize different blocks at the same height. Requires fewer than n/3 Byzantine members.
Liveness	The protocol makes progress as long as ≥⅔ of council members are honest and reachable.
Immediate finality	Once a block reaches the Commit phase, it is irreversible. No reorganisations, no probabilistic confirmation depth.
No long-range attacks	Immediate finality eliminates the class of long-range reorg attacks that affect chains with probabilistic finality.

11.3 Transaction Validity Pipeline

Every transaction must pass a sequential validation pipeline:

8. Structure — correct format, required fields present, valid transaction type
9. Signature — valid signature over the canonical commitment using the declared KeyType
10. Timestamp — anchorTimestamp − senderTimestamp ≤ MAX_MEDIAN_DIFF (5 seconds)
11. Wallet state — sender wallet exists, is enabled, key type matches
12. Duplicate check — no pending or confirmed transaction with the same ID
13. ZK proof (if enableTxProof) — Groth16 proof verifies all the above in zero-knowledge

11.4 Post-Quantum Threat Model

Classical signature schemes (secp256k1, Ed25519, BLS12-377) are vulnerable to Shor's algorithm on a sufficiently powerful quantum computer. Timelines are uncertain, but most security analysts project cryptographically relevant quantum computers within 10–15 years.

ULedger's mitigation strategy:

- ML-DSA-87 available today — new chains can deploy with quantum-resistant signatures immediately
- Algorithm agility — existing chains can migrate to PQC chains with Cross-Merkleization anchoring old state
- Hash functions — SHA3-256/512 and MiMC are quantum-resistant under Grover's algorithm (halved security, still ≥128 bits at SHA3-256)

> Chains storing data with long-term sensitivity — healthcare, legal, financial — should deploy on mldsa87 chains immediately to protect against harvest-now/decrypt-later attacks.

12. PERFORMANCE SUMMARY

12.1 Consensus Parameters

Parameter	Value
Block Finality	Immediate — no probabilistic confirmation required
Consensus Threshold	66.67% (⅔ of council nodes)
Max Transactions per Block	200
Voting Phases	3 (PreVote, PreCommit, Commit)
Timestamp Validity Window	5 seconds (MAX_MEDIAN_DIFF)
Minimum Council Size	3 nodes
Fault Tolerance	Up to ⅓ of nodes Byzantine or offline
Supported Key Types	4 (secp256k1, ED25519, BLS12-377, ML-DSA-87)
ZK Proof Systems	2 (Groth16, PLONK)

Block time and raw throughput figures are dependent on network topology, hardware specifications, and deployment configuration. ULedger's architecture supports horizontal scaling through additional council nodes and sharding strategies for high-volume deployments. Detailed benchmarking data is available on request.

12.2 ZK Proof Performance

Proof Type	System	Proof Size	Verification Time	Setup Type
Transaction proof	Groth16	~200 bytes	~1–2 ms	Per-circuit trusted setup
Block header proof	PLONK	~500 bytes	~3–5 ms	Universal SRS (MPC ceremony)
Cross-reference proof	PLONK	~500 bytes	~3–5 ms	Same universal SRS

12.3 Cryptographic Comparison

Algorithm	Quantum-Resistant	Sig Aggregation	Sig Size	Relative Speed
secp256k1	No	No	~72 bytes	Fast
Ed25519	No	No	64 bytes	Fastest

BLS12-377	No	Yes	96 bytes (sig) / 48 bytes (pubkey)	Moderate
ML-DSA-87	Yes (Level 5)	No	~4,627 bytes	Moderate

13. DEPLOYMENT MODELS

ULedger supports three primary deployment topologies, each suited to different organizational requirements:

13.1 Private Enterprise Network

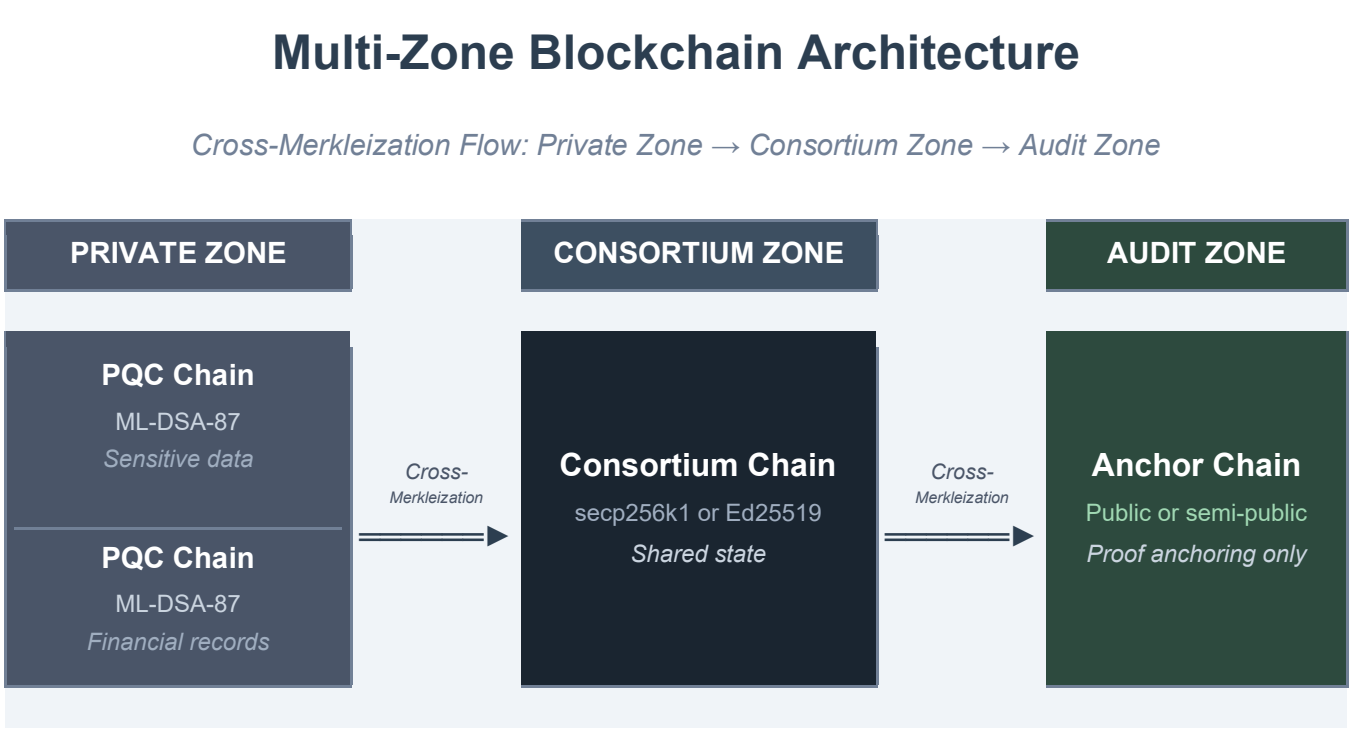
A single organization runs all council nodes. Wallets map to an internal organizational hierarchy. No external access. Ideal for internal audit trails, supply chain tracking, or inter-department settlement.

13.2 Consortium Network

Multiple organizations each operate one or more council nodes. Council membership is federated. Cross-Merkleization can anchor consortium chain state to external networks for independent auditability.

13.3 Hybrid Network (Recommended for Regulated Industries)

Private chains handle sensitive data with PQC (mldsa87) keys. Cross-Merkleization anchors those chains' Merkle roots into a more broadly accessible chain, providing external proof of state without exposing private chain data. This is the recommended model for regulated industries.



14. ENTERPRISE USE CASES

ULedger's architecture is applicable across any domain where data integrity, auditability, or cross-party verification is a requirement. ULedger's permissioned architecture, hierarchical access controls, and support for private chain deployments make it suitable for regulated industries where data sovereignty and compliance are non-negotiable.

Sector	Application
Financial Services	Real-time settlement, audit trails, regulatory reporting (FINRA CAT, SEC compliance)
Supply Chain	End-to-end provenance tracking across multiple organizations with trustless verification at each handoff
Healthcare	Secure interoperable patient record sharing with granular consent management and immutable audit logs
Government & Compliance	Tamper-evident record-keeping for regulatory filings, land registries, and identity management
Enterprise Data Integrity	Cross-department data anchoring ensuring consistency and preventing unauthorized modification
Agentic Commerce	Next-generation autonomous agent economic activity requiring trustless, verifiable micro-transactions

15. CONCLUSION

Enterprise data infrastructure is at an inflection point. The limitations of traditional databases like mutability, siloing, and the absence of cryptographic integrity guarantees are increasingly incompatible with the demands of regulated industries, multi-party workflows, and a threat landscape that now includes quantum computing.

ULedger provides a purpose-built response to these challenges. Its key contributions are:

1. **Native post-quantum security**: ML-DSA-87 (NIST FIPS 204) is a first-class signing algorithm, not a future roadmap item. Enterprises deploying ULedger today on PQC chains are protected against harvest-now/decrypt-later attacks.
2. **Cross-Merkleization**: a novel, proof-native multi-chain anchoring protocol that provides cryptographic state binding between chains without trusted intermediaries, bridge contracts, or relay operators.
3. **Dual ZK proof stack**: Groth16 for compact per-transaction validity certificates and PLONK for efficient block header integrity proofs, enabling lightweight external validation and auditability.
4. **Council BFT with immediate finality**: deterministic, irreversible block finalization with configurable council composition and ⅔ fault tolerance.
5. **Hierarchical wallets with CRUD permissions**: an identity model that mirrors enterprise organizational structures, with cascade disable for safe offboarding.
6. **Algorithm agility**: signature and hash algorithms are per-chain configuration, enabling incremental migration to post-quantum cryptography as organizational risk tolerance and standards evolve.
7. **WASM smart contracts + native token standards:** a full execution environment with native fungible, NFT, and multi-token operations at the protocol level.

Together these properties position ULedger as a platform for high-assurance enterprise applications, regulated financial instruments, supply chain integrity, inter-organizational settlement, and the emerging class of agentic commerce systems. Here, the combination of immediate finality, post-quantum resistance, and verifiable cross-chain state is not optional.

GLOSSARY OF TERMS

BFT (Byzantine Fault Tolerance): A property of a distributed system that enables it to continue operating correctly even if some nodes fail or act maliciously.

BLS12-377: A pairing-friendly elliptic curve used in ULedger for signature aggregation and ZK-circuit pairing checks.

Cross-Merkleization: ULedger's proprietary mechanism for cryptographically anchoring the state of one blockchain into another using Zero-Knowledge Proofs.

CRYSTALS-Dilithium: The underlying lattice-based signature algorithm behind ML-DSA-87. Standardized as NIST FIPS 204 in 2024.

ECDSA: Elliptic Curve Digital Signature Algorithm. A widely used classical digital signature scheme.

EdDSA: Edwards Curve Digital Signature Algorithm. A high-performance signature scheme based on twisted Edwards curves.

Gas Metering: A resource accounting mechanism that assigns a cost to each computational instruction executed in a smart contract, preventing resource exhaustion.

Groth16: A succinct ZK-SNARK proof system offering constant proof size and efficient verification. Used for per-transaction proofs in ULedger.

MiMC: A ZK-friendly hash function that is approximately 100× more constraint-efficient inside a ZK circuit than SHA-256.

ML-DSA-87: Module Lattice-based Digital Signature Algorithm (CRYSTALS-Dilithium Level 5). A NIST-standardized post-quantum digital signature algorithm offering 256-bit security.

Merkle Root: A single cryptographic hash that summarizes and commits to the entire contents of a dataset using a tree of hashes.

MLWE: Module Learning with Errors. The mathematical hardness problem underlying ML-DSA-87's post-quantum security.

PLONK: A universal ZK-SNARK proof system supporting updatable trusted setups. Used for block header proofs in ULedger.

secp256k1: The elliptic curve used by Bitcoin and Ethereum; provides broad ecosystem compatibility.

WASM (WebAssembly): A binary instruction format for a stack-based virtual machine, used as ULedger's smart contract execution environment.

XM: Abbreviation for Cross-Merkleization.

ZK-SNARK: Zero-Knowledge Succinct Non-Interactive Argument of Knowledge. A cryptographic proof that allows one party to prove knowledge of information without revealing it.