XENORRA: A Post Quantum Identity Network with AI Assisted Verification

Xenorra Protocol Research Group

***Abstract*—Large scale quantum computers running Shor's algorithm would break the elliptic-curve signatures that authenticate almost every deployed blockchain. Replacing those signatures is not a drop-in change: the NIST standardized lattice scheme ML-DSA-65 produces 3,309-byte signatures and 1,952-byte public keys, roughly fifty and sixty times larger than Ed25519. In a ledger that stores every signature forever, this moves the binding constraint from computation to bandwidth and storage. This paper presents XENORRA, a proof of stake layer-1 protocol designed for that reality from genesis. We review the quantum threat, the FIPS 203/204/205 standards, and the architectures of Bitcoin, Ethereum, Solana, Avalanche, Polkadot, Cardano, Qubic, and recent client side validation designs, then justify each architectural choice against its alternatives. XENORRA uses ML-DSA-65 for transactions, votes, and node identity, a hash-based SLH-DSA recovery key so that account control survives a break of lattice assumptions, SHA3-256 for all consensus hashing, and a Tendermint-family BFT protocol with deterministic proposer rotation, which removes the need for a post-quantum VRF. We show that the absence of practical post-quantum signature aggregation erases the communication advantage of linear (HotStuff-style) protocols, and that segregating signature witnesses from transaction bodies, with pruning after finality, reduces long-term history growth by about 95%. The paper gives a normative specification of encodings, validity rules, and state transitions; security arguments that separate proven properties from assumption-based reductions and from claims that still need measurement; a Rust workspace architecture with vetted dependencies; and a benchmarking methodology. We report no measured performance results. Analytical estimates suggest that a 100-validator network can carry several hundred transfers per second with 2-second blocks on commodity links, with signature bytes setting the limit.**

***Index Terms*—Post-quantum cryptography, blockchain, ML-DSA, SLH-DSA, Byzantine fault tolerance, Tendermint, proof of stake, cryptographic agility, Rust, client-side validation.**

I. INTRODUCTION

ALMOST every public blockchain authenticates value transfers with ECDSA, Schnorr, or EdDSA over elliptic curves. All three rest on the hardness of the discrete logarithm problem, which Shor's algorithm solves in polynomial time on a sufficiently large fault-tolerant quantum computer [1]. No such machine exists today, and estimates of when one might exist vary widely. The direction of travel is clear. Resource estimates for breaking 2048-bit RSA fell below one million noisy qubits in 2025 [2], and a 2026 estimate puts a 256-bit elliptic-curve key within reach of fewer than half a million physical qubits running for minutes [3]. Standards bodies have stopped treating the question as speculative. NIST published its first three post-quantum standards in August 2024 [4]–[6]. Its final transition guidance, issued in October 2026, deprecates quantum-vulnerable signature and key-establishment algorithms after 2030 and disallows them after 2035 [7].

For a blockchain, the problem has an unusual shape. A bank can rotate the keys behind a TLS endpoint overnight. A ledger cannot rotate the keys of users who have lost interest, lost their wallets, or died; it cannot re-sign history; and its consensus security depends on validator signatures whose forgery would let an attacker rewrite the past. Moving to post-quantum signatures costs far more on a ledger than in a TLS handshake, because every node stores and re-verifies every signature. Ilie *et al.* [8] put it directly: once signatures are post-quantum, authorization data comes to dominate the on-chain footprint.

This paper asks what a layer-1 protocol looks like when it is designed for post-quantum signatures from the first block, under the constraint that every component must be implementable with today's standardized primitives and auditable Rust libraries. We call the result XENORRA. The work started from the Xenorra Whitepaper v2.0, which proposed a post-quantum identity network but deferred post-quantum consensus to a later phase. Our analysis led us to reverse that decision, for reasons set out in Section IV: a chain whose validators sign with elliptic curves is exposed to retroactive history rewriting the moment those curves fall, regardless of how well its user accounts are protected.

A. Contributions

The paper makes the following contributions.

1) A source-checked review of the quantum threat to ledgers and of the post-quantum signature, key-encapsulation, and hash standards, assessed against the specific roles keys play in a blockchain: transaction authentication, consensus voting, block proposal, network identity, and wallet recovery (Sections IV and V).
2) A comparative analysis of eight production or research blockchain architectures, identifying which of their mechanisms survive the move to large signatures and which do not (Section VI).
3) An argument, with explicit byte counts, that without practical post-quantum signature aggregation the asymptotic communication advantage of linear BFT protocols disappears, which changes the usual ranking of consensus families (Section XI).
4) The XENORRA architecture, in which every major decision is recorded with its alternatives, justification,

Protocol specification draft, revision 1.0, November 2026. This document supersedes the cryptographic and consensus sections of the Xenorra Whitepaper v2.0 (draft of 25 October 2026). All performance figures are analytical estimates unless explicitly marked otherwise; no benchmark results are reported.

security and performance implications, implementation cost, and known limitations (Sections VIII–XVI and Appendix E).
5) A normative specification of encodings, domain separation, transaction and block validity, and the state-transition function, written so that independent Rust teams can produce interoperable implementations (Appendix A).
6) A formal security model that separates mathematically proven properties from reductions to hardness assumptions and from properties that need empirical validation (Sections XVII and XVIII, Appendix C).
7) A Rust workspace design, dependency assessment, and reproducible benchmarking methodology. No benchmark results are claimed (Sections XIX–XXI).

B. What this paper does not claim

We do not claim that XENORRA is "quantum-proof." Its security reduces to the hardness of lattice problems (Module-LWE and Module-SIS) and to properties of SHA-3, and either could be weakened by future cryptanalysis, classical or quantum. We use the term *post-quantum-resistant* throughout. We make no claim to a novel consensus protocol. XENORRA adapts Tendermint [9], a protocol with a decade of deployment and independent formal analysis, and the novelty lies in how it is parameterized for large signatures. The two external references supplied with the project brief, Qubic and Pearl Research, serve as starting points for investigation. Neither is evidence for any design decision here, and Section III explains what each does and does not establish.

C. Organization

Section II motivates the work and Section III surveys related research. Sections IV–VI cover the threat model, cryptographic foundations, and existing systems. Section VII states design objectives, and Sections VIII–XVI present the protocol layer by layer. Sections XVII and XVIII analyze security; Sections XIX–XXII cover implementation, performance, and scaling. Section XXIII lists open problems, Section XXIV gives the roadmap, and Section XXV concludes. The appendices hold the normative specification, Rust interfaces, proofs, the useful-computation analysis, the decision register, roadmap detail, and protocol constants.

II. RESEARCH MOTIVATION

Three observations motivated this project. Each one points to a gap in current designs that swapping signature algorithms does not close.

A. Ledgers have long security horizons

Mosca framed the migration problem as an inequality: if x is the time data must stay secure, y the time needed to migrate, and z the time until a cryptographically relevant quantum computer (CRQC) exists, then trouble begins when $x + y > z$ [10]. For confidentiality, the "harvest now, decrypt later" attack makes x the lifetime of the secret. Signatures behave differently. An attacker cannot forge a signature retroactively in a way that changes a past, already-verified action, but in a blockchain "past" is defined by consensus. If validator keys can be forged, an attacker can construct an alternative history that a newly joining node cannot distinguish from the real one (Section IV). For user accounts, any public key that is visible on chain becomes a target the moment a CRQC runs, regardless of when the key was published. For a ledger, x is its whole intended lifetime.

B. Size, not speed, is the binding constraint

Post-quantum signatures are fast enough. On modern x86 processors ML-DSA verification takes tens of microseconds per signature (Section V). They are large. A single-input transfer authenticated with ML-DSA-65 occupies about 3.4 KB in our encoding, against about 0.2 KB for an Ed25519 equivalent (Table XVIII). At 1,000 transfers per second, signature bytes alone amount to about 286 GB per day. Ilie *et al.* reach the same conclusion from the privacy side: in their client-side validation design, removing signatures and proofs from the chain reduces a single-input transaction to three 32-byte hashes [8]. Any credible post-quantum ledger must decide what happens to signature bytes after they have been checked.

C. Several common building blocks have no standardized post-quantum replacement

Many proof-of-stake protocols depend on primitives that Shor's algorithm breaks and for which no standardized post-quantum substitute exists. Verifiable random functions (VRFs) used for leader election in Ouroboros Praos [11] and Polkadot's BABE are built on elliptic curves. BLS signature aggregation, which makes Ethereum's attestation load manageable [12], relies on pairings. Verifiable delay functions over groups of unknown order become insecure once order-finding is cheap. Research schemes exist for lattice-based aggregation [13] and hash-based multi-signatures [14], but none is standardized, and their sizes and costs are still moving. A design for deployment today has to work without them and leave room to adopt them later.

D. Research questions

These observations lead to four questions that structure the rest of the paper.

RQ1 Which NIST-standardized primitives should fill each cryptographic role in a ledger, and what hedges are worth their cost? (Sections V, IX)

RQ2 Which consensus family remains efficient when every vote carries a 3.3 KB signature that cannot be aggregated? (Section XI)

RQ3 How should a ledger store, propagate, and eventually discard post-quantum authorization data without weakening verifiability? (Sections XIII, XV)

RQ4 Can useful computation, as promoted by projects such as Qubic and Pearl, be incorporated without compromising determinism or security? (Section III, Appendix D)

III. Background and Related Work

This section surveys the four bodies of work that most directly shaped XENORRA: client-side validation, post-quantum migration proposals for existing chains, useful-computation consensus, and performance engineering for cryptographic workloads. It closes with the application context inherited from the Xenorra identity whitepaper.

A. Client-side validation and Compact Shielded CSV

Client-side validation (CSV) separates the roles a blockchain plays: ordering and publishing commitments on one side, validating transaction semantics on the other. In a CSV system the chain records only enough to prevent double spends, and each recipient checks the history of the coins it receives. Shielded CSV [15] combined this with recursive zero-knowledge proofs so that a coin carries a constant-size proof of its entire history, and used half-aggregated Schnorr signatures to reduce on-chain data to roughly 64 bytes per transaction. Its authors noted that the post-quantum version of the scheme was unclear, because no post-quantum signature with Schnorr-style aggregation is known.

Compact Shielded CSV [8] removes signatures from the chain altogether. A block is a list of *blobs*, each of the form $\beta = (t, [(n_1, dg_1), \ldots, (n_k, dg_k)])$, where t commits to a transaction, n_i is the nullifier of a spent coin (a construction inherited from Zcash), and $dg_i = \mathsf{H}(n_i, sk, t)$ is a *degriefer* that binds the nullifier to the owner's secret key and to this particular transaction. The consensus rules check no proofs and no signatures. Every blob is committed together with the list of earlier blobs that share one of its nullifiers, and off-chain proofs must show each such earlier conflicting blob to be invalid. A griefer who copies a nullifier from the mempool cannot compute the right degriefer without sk, so the owner can later prove the griefer's blob invalid. The resulting on-chain footprint is $2N + 1$ hashes for an N-input transaction, independent of the size of the post-quantum proof system used off chain. The authors formalize the protocol in Lean 4 and prove no double spending, no inflation beyond the accumulated subsidy, and spendability (an honest owner's coin is always either spent by that owner or spendable now).

We examined this design closely as a candidate base layer for XENORRA and decided against it for the first release, for four reasons that its authors largely acknowledge. Every user must generate recursive post-quantum proofs, and hash-based recursive proof systems are still expensive to produce on consumer hardware. The paper identifies *proof of delivery* as an unsolved problem: a recipient can falsely claim never to have received a coin proof. The nullifier index that miners maintain cannot be pruned, since any future blob may conflict with any past one. And the model has no native accounts, which the identity modules of XENORRA need. We do adopt its central observation: authorization data should not be a permanent part of global state once it has been checked, and we apply it in a transparent setting through witness segregation and pruning (Section XV). A CSV-style shielded payment module remains on our research track (Section XXIII).

B. Post-quantum migration of existing chains

Bitcoin's most developed proposal, BIP-360, began as a "pay to quantum-resistant hash" output type and has since been narrowed to Pay-to-Merkle-Root, which removes Taproot's key-path spend so that outputs no longer expose a public key directly; post-quantum signature opcodes themselves have been deferred to a later proposal [16]. The question of coins whose public keys are already exposed remains open. Ethereum researchers have proposed replacing BLS attestation aggregation with hash-based multi-signatures derived from XMSS-style constructions, aggregated with succinct proofs [14]. Both efforts show how hard retrofitting is: existing chains must preserve compatibility with billions of dollars held under classical keys, and they cannot assume users will migrate. A new chain does not carry this legacy burden, and that is the main reason we argue for post-quantum consensus from genesis.

C. Useful computation as consensus

The idea of making proof-of-work useful is old. Primecoin searched for prime chains, and Ball *et al.* formalized proofs of useful work for fine-grained problems such as orthogonal vectors [17]. Later work built a provably secure consensus protocol around useful combinatorial optimization, derived proofs of useful work from arbitrary matrix multiplication, and analyzed the economics of such schemes; the last two come from researchers associated with Pearl. Two difficulties recur. The useful work must be verifiable much more cheaply than it is produced, and the reward for doing it must not depend on information the miner can manipulate.

a) Qubic: Qubic runs a network of 676 "computors," with a quorum of 451 (two thirds) required for decisions, and directs mining effort toward training neural networks in a project called Aigarth. Its July 2026 article "Anna Reached 0" reports that a small trinary neural network, about 150 KB in size, reached zero errors on a task of adding two numbers, across seven versions of increasing difficulty, without pretrained weights or examples of the rule. The article is unusually candid about what this means. It describes the result as a test of interpolation, states that extrapolation is "next on the list rather than done," and calls the claim of a path to general intelligence "a thesis … not a result." Independent verification depends on code that has not yet been released. We classify the zero-error result on addition within the training range as a reported experimental result without independent replication, and the claims about generalization, trading, and general intelligence as project objectives.

Two further points bear on consensus design. We found no peer-reviewed analysis of Qubic's own consensus, and its documentation describes an "arbitrator" role that publishes the computor list each epoch, which is a point of centralization. And the August 2025 episode in which Qubic redirected its miners to Monero gives an empirical lesson about outsourced mining. Qubic publicly claimed majority hashrate; a measurement study accepted at AFT 2026 estimated its average share at 23.4%, with only brief hourly excursions near 50%, found deeper reorganizations consistent with selfish mining,

and estimated that Qubic earned about 4% less than honest mining would have [18]. Whatever one concludes about the intent, the episode shows that a large pool of rented, mobile computation can be pointed at another chain's consensus, a risk for any security model that depends on computation.

b) Pearl: Pearl Research Labs sells LLM inference and is developing the Pearl Protocol, a proof-of-useful-work layer-1 in which mining is derived from AI computation. It is a competing protocol project, and we read its material with that in mind. Its research page lists a CUDA grouped-GEMM kernel for mixture-of-experts models with self-reported speedups, an announcement of a GPU BLAKE3 implementation described only in a social-media post, the matrix-multiplication proof-of-useful-work paper mentioned above, and *Hawkeye*, a study of reproducing GPU-level non-determinism accepted at MLSys 2026. The last of these bears directly on our design: it shows that bit-exact reproducibility of GPU computation takes explicit engineering effort, because floating-point reductions and kernel scheduling vary between runs and devices. Consensus-critical computation must be bit-exact on every honest node, so we keep GPUs out of the consensus path (Section XXII-A).

Appendix D evaluates whether useful computation could be offered as an optional, non-consensus XENORRA service. It can, provided it is treated like an oracle (verified by redundancy, bonding, and challenge) and never feeds back into block validity or leader selection.

D. Performance engineering for cryptographic workloads

BLAKE3 [19] uses a Merkle-tree internal structure, which allows SIMD and multi-threaded hashing of large inputs and gives it several times the single-core throughput of SHA-256 or SHA3-256. In a ledger, hashing is rarely the bottleneck. Inputs are small (headers, transaction bodies, Merkle nodes), so the tree mode yields little, and under post-quantum signatures the dominant CPU cost is signature verification (Section XX). The more transferable lessons concern parallel transaction validation (Block-STM [20], Solana's declared-access scheduling [21]), zero-copy decoding, and careful storage-engine choice. Section XXII-A assesses each technique for XENORRA.

E. Identity application context

The Xenorra Whitepaper v2.0 described a network for decentralized identifiers (DIDs) and verifiable credentials, with staked verifiers who check identity claims against external sources using a commit-reveal protocol, and an advisory AI layer whose outputs are signed attestations. This paper keeps that application scope. The DID registry, credential registry, and verifier market are native modules of the execution layer (Section XIV). It revises the underlying protocol in three places, each justified later. Consensus signatures become post-quantum at genesis. VRF-based selection is replaced, because no standardized post-quantum VRF exists. Hybrid classical/post-quantum signatures become optional, with a hash-based recovery key serving as the main hedge against lattice cryptanalysis.

IV. QUANTUM COMPUTING THREAT MODEL

This section states what a quantum adversary can and cannot do to a ledger, so that the mitigations in later sections can be checked against specific capabilities.

A. Shor's algorithm and public-key cryptography

Shor's algorithm solves integer factorization and discrete logarithms in abelian groups in time polynomial in the bit length [1]. It breaks RSA, finite-field Diffie–Hellman, and every elliptic-curve scheme used by mainstream blockchains: ECDSA over secp256k1 (Bitcoin, Ethereum), Ed25519 (Solana, Cardano, Polkadot, Cosmos), Schnorr/BIP-340, BLS12-381 aggregate signatures (Ethereum consensus), and ECVRF. The algorithm recovers the private key from the public key alone, without any message–signature pairs.

Concrete resource estimates have fallen steadily. For RSA-2048, Gidney's 2025 estimate requires under one million noisy qubits running for less than a week [2], about a twentieth of the qubit count estimated in 2019. In March 2026, Babbush *et al.* estimated that 256-bit ECDLP on secp256k1 needs fewer than 1,200 logical qubits and 90 million Toffoli gates (or 1,450 logical qubits and 70 million Toffoli gates), which they translate into fewer than 500,000 physical superconducting qubits at a 10^{-3} error rate, with a run time measured in minutes [3]. The authors attest to their circuits with a zero-knowledge proof and have not published them, so the estimate cannot yet be independently reproduced in full. For RSA-2048, Gidney and Ekerå estimated twenty million noisy physical qubits and eight hours in 2019, and Gidney reduced this to under one million noisy qubits running for less than a week in 2025 [2]. These are estimates under stated assumptions about physical error rates and surface-code overheads. They do not predict when such hardware will exist. Elliptic-curve keys are generally expected to fall before RSA-2048 of comparable classical strength, because the group operations act on much smaller numbers.

B. Grover's algorithm and hash functions

Grover's algorithm finds a preimage in an unstructured space of size N with about $\frac{\pi}{4}\sqrt{N}$ oracle queries [22]. Taken at face value, this halves the security level of an n-bit hash against preimage search. The practical effect is smaller, because Grover iterations are inherently sequential. If an attacker is limited to circuits of depth D (in oracle calls), splitting the search over P machines requires each to search N/P elements in at most D iterations, so

$$\sqrt{N/P} \le D \implies P \ge \frac{N}{D^2}, \qquad P \cdot D \ge \frac{N}{D}. \tag{1}$$

Total work grows as the depth budget shrinks. For this reason NIST defines its post-quantum security categories by comparison with the cost of attacking AES and SHA-2/SHA-3 under a maximum circuit depth [5].

For collisions, the Brassard–Høyer–Tapp algorithm uses $O(2^{n/3})$ queries, but it needs $2^{n/3}$ qubits of quantum-accessible memory. Bernstein showed that once hardware cost is counted, it does not beat classical parallel collision

search [23]. A 256-bit hash such as SHA3-256 retains about 128 bits of collision resistance and at least 128 bits of preimage resistance against known quantum attacks. Symmetric primitives with 256-bit outputs or keys are not the weak point.

C. Attack surfaces specific to ledgers

Table I maps blockchain primitives to quantum attacks and their consequences. Four surfaces deserve comment.

Exposed account keys. Any public key visible on chain can be attacked offline at the attacker's leisure. In Bitcoin, public keys are exposed by pay-to-public-key outputs, address reuse, and every spend. Hashing the key into the address delays exposure only until the first spend.

Mempool interception. When an owner spends from a hash-protected address, the public key appears in the pending transaction. An attacker who can derive the private key within one block interval can broadcast a conflicting transaction with a higher fee. This requires a fast CRQC and is the hardest attack to rule out by key hygiene alone. The minutes-scale run times in recent estimates [3] mean it can no longer be dismissed as requiring implausibly fast hardware.

Validator keys and history rewriting. In proof-of-stake systems, a joining node decides which chain is canonical by checking validator signatures. An attacker who can forge the signatures of a past validator set can produce an alternative history, valid in every respect, from any point after that set was active. Classical long-range attacks already require weak-subjectivity checkpoints [24]; a CRQC removes the need to buy or steal old keys, because the keys can be computed from public data. A chain whose consensus signatures are classical is exposed to this attack for its entire history, even if its user accounts are post-quantum.

Transport and identity. Peer-to-peer traffic on a public chain is mostly public, so harvest-now-decrypt-later attacks have limited value, though they do expose metadata such as which node first relayed a transaction. Active impersonation is more serious: an attacker who can forge node identities can eclipse a victim [25].

D. Adversary classes

Since the timeline is uncertain, we analyze XENORRA against four nested adversary classes instead of a single point estimate.

$\mathcal{A}_0$ **(classical)** Probabilistic polynomial-time, controls the network subject to partial synchrony and corrupts less than one third of stake.

$\mathcal{A}_1$ **(slow CRQC)** $\mathcal{A}_0$ plus the ability to run Shor's algorithm with latency of hours to days per key. Breaks any long-exposed elliptic-curve key.

$\mathcal{A}_2$ **(fast CRQC)** $\mathcal{A}_1$ with latency below one block interval. Enables mempool interception.

$\mathcal{A}_3$ **(fast CRQC with cryptanalytic advance)** $\mathcal{A}_2$ plus an unexpected classical or quantum weakening of one structured post-quantum assumption, for example a significant improvement in lattice reduction.

XENORRA is designed to remain safe and live against $\mathcal{A}_2$ under the assumptions of Section XVII. Against $\mathcal{A}_3$ we aim for a narrower guarantee: if lattice assumptions fail, account owners can still regain control through hash-based recovery keys, and the network can halt and migrate without accepting forged history (Section IX-H). We make no claim of security against an adversary who breaks SHA-3, since that would undermine both the hash-based hedge and every commitment in the system.

V. POST-QUANTUM CRYPTOGRAPHIC FOUNDATIONS

A. Three kinds of primitive

The project brief asks us to keep three kinds of primitive apart, for good reason. A *digital signature scheme* lets anyone holding a public key check that a message was approved by the holder of the matching private key. Signatures provide authentication and non-repudiation, which is what a ledger needs for transactions, votes, and identities. A *key-encapsulation mechanism* (KEM) lets two parties agree on a shared secret, which then keys a symmetric cipher. A KEM provides confidentiality but authenticates nobody: ML-KEM [4] can protect a peer-to-peer session from eavesdroppers, but cannot prove who is on the other end. A *hash function* provides commitments, Merkle trees, and identifiers, and is the building block of hash-based signatures. In XENORRA, ML-DSA and SLH-DSA are the signatures, ML-KEM appears only inside the transport handshake, and SHA-3 is the hash.

B. Families of post-quantum assumptions

Lattice-based schemes rest on problems such as Learning With Errors (LWE) and the Short Integer Solution problem (SIS), here in their module variants (MLWE, MSIS). They offer the best balance of size and speed among standardized options. NIST standardized ML-DSA (signatures, from CRYSTALS-Dilithium [26]) and ML-KEM (key encapsulation, from CRYSTALS-Kyber). FN-DSA (from Falcon, based on NTRU lattices) has smaller signatures but had not been published even as a draft standard by November 2026 [27].

Hash-based signatures need only a secure hash function, which makes them the most conservative choice. Stateful schemes, XMSS and LMS, have small signatures but require the signer never to reuse a one-time key, which NIST considers suitable only for tightly controlled environments such as firmware signing. The stateless SLH-DSA [6], derived from SPHINCS+, removes the state at the cost of signatures between 7.8 and 50 KB.

Code-based cryptography has produced KEMs, including HQC, which NIST selected in 2025 as a backup to ML-KEM, but no standardized signature scheme with practical sizes. *Multivariate* signatures can be very small, but Rainbow, a NIST finalist, was broken by Beullens with a key-recovery attack that ran over a weekend on a laptop [28]. *Isogeny-based* cryptography produced SIKE, which Castryck and Decru broke with a classical attack [29], and SQIsign, whose signatures are very compact but whose signing is slow. Two well-studied finalists were broken by classical cryptanalysis late in the competition. This is our main reason for preferring

TABLE I
QUANTUM ATTACK SURFACES IN A TYPICAL PROOF-OF-STAKE LEDGER AND THE CORRESPONDING XENORRA DESIGN RESPONSE.

Primitive	Typical instantiation	Quantum attack	Consequence if broken	XENORRA response
Transaction signatures	ECDSA, Ed25519, Schnorr	Shor (key recovery)	Theft from any account whose public key is known; mempool interception	ML-DSA-65 (FIPS 204); key hidden behind address hash until first spend; SLH-DSA recovery key
Consensus votes and proposals	Ed25519, BLS12-381	Shor	Equivocation without attribution; forged commit certificates; history rewriting	ML-DSA-65 for all votes from genesis; no aggregation dependency
Leader election randomness	ECVRF, BLS threshold	Shor	Predictable or biasable leaders	Deterministic weighted round-robin; no VRF
Aggregate signatures	BLS	Shor	Forged quorum certificates	Not used; certificate is a list of ML-DSA signatures
Zero-knowledge proofs	Groth16, KZG	Shor (pairings, discrete log)	Soundness failure: proofs of false statements	None in base protocol; hash-based proof systems only, on research track
Peer transport	X25519 + Ed25519 (Noise, TLS)	Shor	Harvest-now-decrypt-later; peer impersonation, eclipse	Hybrid X25519+ML-KEM-768 key exchange; ML-DSA peer authentication bound to channel
Hash functions	SHA-256, Keccak, BLAKE2	Grover, BHT	Margin reduced; 256-bit outputs remain adequate	SHA3-256 with domain separation
Wallet derivation	BIP-32 (secp256k1)	Shor on xpub	All child keys exposed from one public key	Hardened-only KMAC-based derivation of ML-DSA seeds

standardized schemes and for adding a hash-based hedge to the lattice primary.

C. ML-DSA

ML-DSA works over the ring $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$ with $q = 2^{23} - 2^{13} + 1 = 8{,}380{,}417$. The public key contains a seed for a matrix $\mathbf{A} \in R_q^{k\times\ell}$ and a vector $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$, where $\mathbf{s}_1, \mathbf{s}_2$ are short secret vectors. Signing follows the Fiat–Shamir with aborts paradigm. The signer samples a masking vector $\mathbf{y}$, computes a commitment from $\mathbf{w} = \mathbf{A}\mathbf{y}$, derives a sparse challenge polynomial c by hashing the commitment with the message representative μ, and sets

$$\mathbf{z} = \mathbf{y} + c\,\mathbf{s}_1. \tag{2}$$

It rejects and restarts if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or if certain low-order bits would leak information about the secret, so that the distribution of $\mathbf{z}$ is independent of $\mathbf{s}_1$. The verifier recomputes the commitment from $\mathbf{A}\mathbf{z} - c\,\mathbf{t}$ with the help of a short hint, and checks the challenge. The message representative is $\mu = H(H(pk) \,\|\, M')$, where M' prefixes the message with a domain byte and the context string, so every signature is bound to the public key and the context [5]. Security reduces to MLWE and a variant of MSIS (SelfTargetMSIS) in the quantum random-oracle model [30]. The reduction is not tight, and concrete parameters are chosen from the best known lattice attacks.

Signing is hedged by default: a 32-byte random value from the signer's generator is mixed into the derivation of $\mathbf{y}$, which protects against the fault attacks that are effective on purely deterministic lattice signatures [31]. Key generation can be run from a single 32-byte seed ξ, which is what makes deterministic wallet derivation possible (Section IX-F). FIPS 204 defines a further variant, HashML-DSA, a pre-hash variant; XENORRA uses pure ML-DSA over a 32-byte transaction digest instead, which keeps a single signing mode per key, as the standard recommends.

D. SLH-DSA

SLH-DSA signs with a few-time signature scheme (FORS) at a leaf selected pseudorandomly from the message, and authenticates that leaf through a hypertree of one-time-signature (WOTS+) Merkle trees up to a root in the public key [6]. Because leaf selection is pseudorandom and the tree is very large, a signer can sign many messages without keeping state, at the cost of large signatures. Its security depends only on properties of the underlying hash function (here SHAKE256). The "s" parameter sets optimize for size and the "f" sets for signing speed.

E. Sizes and costs

Table II collects sizes from the standards and published cycle counts. The ML-DSA cycle counts come from the round-3 Dilithium specification [32], which differs from final ML-DSA in small details (signature sizes for the higher parameter sets changed slightly), so the timings approximate FIPS 204 code without measuring it. We found no published, peer-reviewed cycle counts for final FIPS 204 implementations on comparable hardware; measuring them is the first task in the benchmarking plan (Section XXI).

ML-DSA-65 verification at roughly 180,000 cycles takes about 60 µs on a 3 GHz core, so a 16-core machine can verify on the order of a quarter of a million signatures per second. Verification does not limit throughput. SLH-DSA-128s signing, by comparison, takes about 645 million cycles in the AVX2 SHA-2 implementation measured by its designers [33], around a fifth of a second, which is acceptable for a recovery key but not for routine transactions. The SHAKE variant is slower still.

F. Suitability by role

Table III rates each standardized signature scheme against the roles keys play in a ledger. The deciding factors differ

TABLE II
SIZES (BYTES) AND PUBLISHED PERFORMANCE OF STANDARDIZED AND CANDIDATE SIGNATURE SCHEMES. CYCLE COUNTS ARE MEDIANS FROM THE CITED SUBMISSION DOCUMENTS ON THE HARDWARE STATED THERE; THEY ARE PROXIES FOR, NOT MEASUREMENTS OF, FIPS-FINAL IMPLEMENTATIONS.

Scheme	NIST cat.	$\|pk\|$	$\|sk\|$	$\|\sigma\|$	Sign (cycles)	Verify (cycles)	Source of timings
Ed25519 (reference)	–	32	32	64	–	–	classical baseline (sizes only)
ML-DSA-44	2	1,312	2,560	2,420	259,172	118,412	Dilithium2, AVX2, Skylake [32]
ML-DSA-65	3	1,952	4,032	3,309	428,587	179,424	Dilithium3, AVX2, Skylake [32]
ML-DSA-87	5	2,592	4,896	4,627	538,986	279,936	Dilithium5, AVX2, Skylake [32]
SLH-DSA-128s	1	32	64	7,856	644,740,090	861,478	SPHINCS+-SHA2-128s-simple, AVX2 [33]
SLH-DSA-128f	1	32	64	17,088	33,651,546	2,150,290	SPHINCS+-SHA2-128f-simple, AVX2 [33]
SLH-DSA-192s	3	48	96	16,224	–	–	sizes only [6]
SLH-DSA-256s	5	64	128	29,792	–	–	sizes only [6]
FN-DSA-512 (Falcon)	1	897	–	666	–	–	not yet standardized [27]
ML-KEM-768 (KEM)	3	ek 1,184, ct 1,088		–	–	–	key encapsulation, not a signature [4]

by role. For transactions, size dominates because every node stores and relays every signature. For votes, the cost is size times n, and signing must finish within a round. For recovery, conservatism dominates and size is secondary. For network identity, the signature is exchanged once per connection, so size and speed are minor concerns.

G. Hybrid schemes

A hybrid signature combines a post-quantum and a classical signature so that forging requires breaking both. It is valuable during a transition, when an unexpected weakness in a young post-quantum scheme is the main risk and the classical scheme is still secure. The IETF has standardized composite ML-DSA for X.509 for this purpose [34]. The protection lasts only until a CRQC exists. After that, the classical half adds bytes and nothing else. A hybrid key exchange, by contrast, keeps its value longer: X25519MLKEM768 [35] protects session secrets against harvest-now-decrypt-later collection by a future CRQC (through ML-KEM) and against an unexpected break of ML-KEM (through X25519) during the years before such a computer exists, at a cost of about 1.1 KB per handshake. XENORRA uses hybrid key exchange in its transport by default and offers hybrid signatures only as an optional suite (Section IX).

H. Implementation hazards

The algorithms are standardized, but implementations are new, and the hazards are specific. Signing uses rejection sampling, so the number of iterations varies, and implementations must not leak secret-dependent timing in the rejected iterations. Deterministic signing is vulnerable to fault injection [31]. Verification, the operation consensus depends on, has had bugs too. In 2026 the RustSec advisory database recorded cases in which a Rust ML-DSA library accepted malformed signatures, including an optimized code path that accepted a crafted invalid signature, and a timing side-channel in another library caused by hardware division on secret-derived values [36]. For a ledger, a verification discrepancy between two implementations is a consensus failure: nodes running different libraries would disagree about block validity and split the network. Section XIX turns this into concrete requirements: a pinned primary implementation, differential testing against a second one, and negative test vectors for malformed signatures.

VI. COMPARATIVE ANALYSIS OF EXISTING BLOCKCHAINS

We reviewed eight systems to find out which architectural ideas survive the move to large signatures. Table IV summarizes the comparison and Table V compares consensus families. The discussion below is organized by design dimension, since the question is which mechanism to adopt for each layer.

A. State model: accounts or UTXOs

Bitcoin [37] and Cardano use unspent transaction outputs (UTXOs). Each input references a previous output and carries its own authorization, so an N-input transaction needs N signatures unless the same key controls all inputs and the scheme allows key reuse. Under ML-DSA-65 every extra signature costs 3.3 KB, which makes coin consolidation and multi-input payments expensive. UTXOs do have real advantages: transactions declare exactly what they consume, which simplifies parallel validation, and the model maps cleanly onto client-side validation [8].

Ethereum, Solana [21], Polkadot [39], and Avalanche's C-chain use accounts. A transaction carries one signature regardless of how much value it moves, and replay protection comes from a per-account nonce. Identity records such as DID documents are naturally account-shaped. In exchange, parallel execution needs either declared access sets (Solana) or optimistic concurrency control (Block-STM [20]).

For XENORRA the signature cost settles the question: one 3.3 KB signature per transaction instead of per input is a large saving, and the identity modules need accounts anyway. We adopt an account model with declared access sets (Section XIV).

B. Execution model

Ethereum executes general smart contracts in the EVM, sequentially, with gas metering. Solana executes BPF programs in parallel over transactions that declare their accounts

TABLE III
SUITABILITY OF SIGNATURE SCHEMES BY LEDGER ROLE. ++ PREFERRED, + ACCEPTABLE, – POOR FIT, × UNSUITABLE. ML-KEM IS EXCLUDED BECAUSE IT DOES NOT AUTHENTICATE.

Role	ML-DSA-44	ML-DSA-65	ML-DSA-87	SLH-DSA-s	XMSS/LMS	Deciding consideration
Transaction authentication	+	++	+	–	×	Size per transaction; long-lived keys favour margin; state reuse risk excludes stateful schemes
Consensus votes	+	++	–	×	–	$n \cdot \lvert\sigma\rvert$ per round; signing within the round deadline
Block proposals	+	++	+	–	–	One per round; size less critical
Validator operator / governance	–	+	++	+	+	Rarely used, high value; stateful acceptable in HSMs only
Account recovery	–	–	–	++	–	Independent hardness assumption is decisive
Network identity	+	++	+	–	×	One signature per connection
Wallet seed derivation	++ (32-byte seed ξ)			+ (three seeds)	×	Deterministic key generation from a seed

TABLE IV
ARCHITECTURAL COMPARISON OF REVIEWED SYSTEMS. "PQ STATUS" REFERS TO THE DEPLOYED PROTOCOL, NOT RESEARCH PROPOSALS.

System	State	Consensus / Sybil resistance	Finality	User signatures	PQ status	Relevant lesson for XENORRA
Bitcoin [37]	UTXO	Nakamoto PoW	Probabilistic	ECDSA, Schnorr (secp256k1)	Vulnerable; BIP-360 proposed [16]	Compact block relay; signature-free consensus layer; legacy keys make migration hard
Ethereum [12]	Account	Gasper PoS (LMD-GHOST + FFG)	Checkpoint, minutes	ECDSA; BLS for consensus	Vulnerable; research on hash-based multisig [14]	Aggregation is central to its validator scaling and has no standardized PQ substitute
Solana [21]	Account	PoS with PoH and Tower BFT	Seconds (optimistic)	Ed25519	Vulnerable	Declared account access enables parallel execution
Avalanche [38]	UTXO and account	Snow metastable sampling	Probabilistic, sub-second	secp256k1	Vulnerable	Repeated sampling multiplies signed messages per decision
Polkadot [39]	Account	NPoS, BABE + GRANDPA	Finality gadget	sr25519, Ed25519	Vulnerable	Upgradeable runtime; VRF-based slots need a PQ VRF
Cardano [11]	Extended UTXO	Ouroboros Praos	Probabilistic	Ed25519	Vulnerable	Formal methods culture; VRF dependency
Qubic	Account	Quorum of 451/676 computors, uPoW	Per tick (quorum)	Not assessed	Not assessed	No peer-reviewed consensus analysis; arbitrator role; uPoW claims unverified
Compact Shielded CSV [8]	Nullifiers (UTXO)	Agnostic (data availability only)	Inherits	None on chain (degriefer)	Designed for PQ	Keep authorization data off the permanent ledger

in advance. Polkadot runs its runtime as WebAssembly so that the state-transition function itself can be upgraded on chain. Bitcoin and Cardano use restricted scripting languages that are easier to analyze. General smart contracts carry significant cost in specification size, audit burden, and attack surface; the history of EVM exploits makes that cost concrete. A post-quantum chain with a narrow initial purpose does not need them at launch. Section XIV evaluates this trade-off in detail.

C. Block propagation

Bitcoin relays compact blocks containing short transaction identifiers, so peers rebuild blocks from their mempools. Solana's Turbine splits blocks into erasure-coded shreds sent along a tree. Tendermint-based chains split proposals into fixed-size parts authenticated by a Merkle tree [9]. Compact relay is worth far more when transactions are large: a 3.4 KB transaction that has already reached a validator's mempool should not be sent again inside the block. XENORRA uses compact proposals (Section X).

D. State commitments

Ethereum commits to state with a hexary Merkle–Patricia trie over Keccak-256, whose proofs are large (several kilobytes) because each level carries up to sixteen sibling hashes. Diem and Aptos use the Jellyfish Merkle tree, a sparse binary Merkle tree optimized for storage on log-structured key–value stores [40], whose proofs carry one sibling per level and can be compressed by omitting empty subtrees. Bitcoin commits only to the transactions in each block, and light clients cannot query UTXO state. For XENORRA we choose a binary sparse Merkle tree in the Jellyfish style, because compact proofs serve light clients and the identity use case, where relying parties need to check a credential's revocation status against a header.

E. Finality and fork choice

Proof-of-work chains and longest-chain proof-of-stake protocols such as Ouroboros [11] offer probabilistic finality: a block becomes harder to revert as it is buried. Ethereum's Gasper [12] combines a fork-choice rule (LMD-GHOST) for

liveness with a finality gadget (Casper FFG) that finalizes checkpoints every few minutes. Polkadot follows the same split with BABE for block production and GRANDPA for finality. Tendermint [9] and HotStuff [41] give deterministic finality per block: once a block is committed it cannot be reverted unless at least one third of stake signs conflicting messages.

The split designs keep producing blocks during periods of asynchrony, which Tendermint does not. They pay for it with an unpredictable leader schedule to resist targeted denial of service, typically from a VRF. They need a separate fork-choice rule with its own attack literature, and finality votes that become expensive without aggregation. Deterministic finality simplifies light clients and relying parties: an identity attestation is either committed or it is not.

F. Light-client verification

Bitcoin light clients follow the heaviest header chain. FlyClient reduces this to a logarithmic sample using Merkle mountain range commitments. BFT chains support a different style: a light client that trusts a validator set can verify a later header by checking that more than one third of the trusted voting power signed it, skipping intermediate headers ("skipping verification" in the Tendermint light-client protocol [42]). Under post-quantum signatures each such check costs on the order of $2n/3$ ML-DSA verifications and about 220 KB of signature data for $n = 100$ (Section XX). This is acceptable when done once per validator-set change, but too much to do per block. XENORRA changes validator sets only at epoch boundaries and retains full commit certificates for epoch-boundary blocks (Section XV).

G. Consensus families

Table V compares consensus families on the criteria set out in the project brief. Nakamoto-style proof of work is the only family whose consensus layer needs no signatures at all, so signature size does not affect it; its energy cost, probabilistic finality, and the rented-hashrate risk illustrated by the Monero episode [18] rule it out for XENORRA. The families that scale furthest in validator count (Ethereum's Gasper, HotStuff with threshold signatures) do so by aggregating votes, and no standardized post-quantum scheme supports this. Certified DAG protocols such as Narwhal and Bullshark [43] attach a quorum certificate to every vertex, which multiplies post-quantum signature traffic, while uncertified DAGs such as Mysticeti [44] avoid this but are much younger. We return to the quantitative argument in Section XI-A.

H. What XENORRA adopts and what it redesigns

From this survey we adopt: the account model with nonces (Ethereum); declared access sets for parallel execution (Solana); compact block relay (Bitcoin); a binary sparse Merkle state tree (Diem/Aptos); Tendermint consensus with deterministic finality and skipping light clients (Cosmos); and the separation of authorization data from ordering data (client-side validation). We redesign: leader election, to avoid a VRF; commit certificates and their retention, because they cannot be aggregated; transaction encoding, to segregate witnesses; and key management, to support recovery under a different hardness assumption. We reject, for the initial release: general-purpose smart contracts, proof-of-work and useful-work consensus, and any dependence on pairings or elliptic-curve VRFs.

VII. XENORRA Design Objectives

The objectives below are written so that each can be checked. Where we give numbers they are targets for the first public testnet, to be confirmed or revised by the benchmarks of Section XXI; they are not measured results.

O1 Post-quantum authentication from genesis. Every signature that the protocol verifies (transactions, proposals, votes, evidence, peer handshakes, governance actions) uses a NIST-standardized post-quantum scheme. No safety-critical path depends on elliptic curves, pairings, or groups of unknown order.

O2 Cryptographic agility. Every key and signature carries an algorithm identifier. Adding, deprecating, or retiring a scheme is a governed parameter change with defined transition rules. The data model does not change.

O3 Hedged account control. An account's owner can regain control even if the primary signature scheme is broken, through a recovery key based on a different hardness assumption.

O4 Deterministic finality within seconds. A transaction included in a committed block is final. Target: median time to finality below 6 s on a geographically distributed network of 100 validators.

O5 Bounded node cost. Target: a validator in a 100-validator network operates within a sustained 50 Mbit/s symmetric link, 16 CPU cores, 64 GB of memory, and 2 TB of NVMe storage per year of operation at target load. A pruned full node needs substantially less storage.

O6 Decentralization. Target: at least 100 independent validators at mainnet and a protocol that remains usable at 200, with stake-concentration limits enforced by protocol parameters.

O7 Implementability and auditability. The normative specification is complete enough for independent implementation. All cryptographic dependencies are standardized and have maintained Rust implementations. The base protocol fits in a specification that one review team can audit.

O8 Minimal on-chain personal data. Identity data stays off chain; the chain stores commitments, revocation status, and public keys only (inherited from the Xenorra Whitepaper v2.0).

O9 Sustainable incentives. Validator rewards are funded by a declining issuance schedule and by fees, with the issuance curve and fee split set by modelling.

A. Non-goals

Some properties are out of scope for the first release. We do not aim for the highest transactions-per-second figure:

TABLE V
QUALITATIVE COMPARISON OF CONSENSUS FAMILIES. "PQ OVERHEAD" IS THE GROWTH IN SIGNED-MESSAGE BYTES PER DECISION WHEN SIGNATURES ARE 3.3 KB AND CANNOT BE AGGREGATED; n IS THE NUMBER OF VALIDATORS.

Family	Security assumption	Fault tolerance	Finality	Energy	PQ overhead per decision	Assessment for XENORRA
Proof of Work [37]	Honest majority of hashrate; synchrony	$< 1/2$ hashrate (less with selfish mining)	Probabilistic	Very high	None in consensus	Rejected: energy, slow finality, rentable hashrate
Longest-chain PoS [11]	Honest stake majority; synchrony; VRF	$< 1/2$ stake	Probabilistic	Low	One signature per block; needs PQ VRF	Rejected: VRF dependency, probabilistic finality
Delegated PoS	Small elected set; voter diligence	$< 1/3$ of delegates	Fast	Low	Low (n small)	Rejected as a goal: concentrates power; small n is an outcome to avoid
PBFT	Partial synchrony	$< 1/3$	Deterministic	Low	$O(n^2)$ messages	Superseded by rotating-leader variants
Tendermint [9]	Partial synchrony	$< 1/3$ stake	Deterministic, per block	Low	$\approx 2n$ votes per node per height	**Selected**: mature, accountable, simple; no aggregation needed
HotStuff family [41]	Partial synchrony	$< 1/3$	Deterministic	Low	Linear only with threshold/aggregate signatures; otherwise QC of $\approx 2n/3$ signatures to each node	Viable; advantage lost without aggregation
Certified DAG [43]	Partial synchrony / async	$< 1/3$	Deterministic	Low	Certificate of $2f+1$ signatures per vertex, n vertices per round	Rejected for v1: highest PQ signature load
Uncertified DAG [44]	Partial synchrony	$< 1/3$	Deterministic, low latency	Low	One signature per vertex	Research track for v2
Avalanche [38]	Subsampled voting; synchrony	Parameterized	Probabilistic	Low	k signed queries per round, many rounds	Rejected: probabilistic, many signed messages
Useful PoW [17]	As PoW plus verifiable useful work	As PoW	Probabilistic	High	As PoW	Rejected for consensus; optional service only (App. D)

given the signature sizes in Section V, throughput claims in the tens of thousands per second would require either very large blocks or off-chain authorization, and we prefer a smaller, defensible number. General-purpose smart contracts are deferred (Section XIV). Payment privacy beyond pseudonymity is deferred to a shielded-pool research track. Useful computation is excluded from consensus.

B. How decisions are recorded

Every major design choice in Sections IX–XVI is recorded with seven attributes: the selected technology, the alternatives considered, the technical justification, security implications, performance implications, implementation complexity, and known limitations. The body of the paper discusses the reasoning; Appendix E collects all decisions in one register for review. A decision that cannot be settled from available evidence is marked as an open question (Section XXIII) instead of being settled silently.

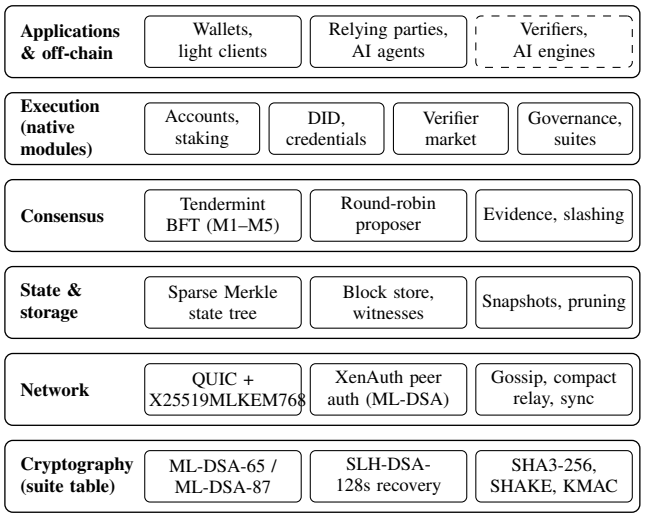


Fig. 1. XENORRA layers. Each layer depends only on the layers beneath it (arrow), and the cryptographic layer is reached only through the suite table. Dashed: off-chain participants, which interact with the chain solely by submitting signed transactions and attestations.

VIII. PROPOSED SYSTEM ARCHITECTURE

XENORRA is organized as five protocol layers plus an off-chain tier (Fig. 1). Each layer has a narrow interface to its neighbours, and the same boundaries become crate boundaries in the Rust implementation (Section XIX).

a) Cryptographic layer: Signature schemes, hashing, key derivation, and the algorithm registry (the "suite table") that maps identifiers to schemes and records their lifecycle status. All other layers call this layer through a single verification interface and never name an algorithm directly.

b) Network layer: Peer discovery, authenticated encrypted transport, gossip of transactions and consensus messages, compact block relay, and state synchronization. Peers are identified by the hash of an ML-DSA public key.

c) Consensus layer: A Tendermint-family BFT protocol over a stake-weighted validator set that changes only at epoch boundaries. It orders blocks, produces commit certificates, and collects evidence of equivocation. It treats block contents as opaque apart from a validity predicate supplied by the execution layer.

d) Execution layer: A deterministic state-transition function over native modules: accounts and transfers, staking, governance, the suite registry, and the identity modules (DID registry, credential registry, verifier market). There is no general-purpose virtual machine in the first release.

e) State and storage layer: An authenticated sparse Merkle tree over account and module state, an append-only block store with segregated witness storage, indices, snapshots, and pruning.

f) Off-chain tier: Wallets, verifiers, AI-assisted document checks, evidence storage, and external data sources. Nothing from this tier enters consensus except signed transactions and signed attestations, which are verified like any other transaction.

A. Node roles

A *validator* holds bonded stake and a consensus key, participates in voting, and runs behind one or more *sentry* nodes that shield its network address. A *full node* validates every block and keeps current state and recent history; by default it prunes witnesses and commit certificates older than the retention window (Section XV). An *archive node* keeps everything. A *light client* keeps a trusted validator set and verifies headers, epoch transitions, and state proofs. These roles differ only in policy. All run the same consensus and validation code.

B. Data flow

A wallet builds a transaction body, signs its digest with ML-DSA under a domain-specific context, and submits the body and witness to any node. The node checks format, fee, nonce, and signature, then gossips the transaction. The proposer of the next consensus round selects transactions from its mempool, executes them, and broadcasts a compact proposal: a header that commits to the post-execution state root, plus the identifiers of the included transactions. Validators rebuild the block from their mempools, fetch anything missing, re-execute it, and vote. When more than two thirds of voting power has precommitted, the block is final, and the commit certificate is carried in the next block. Section XII traces this lifecycle in full.

C. Why the layers are separated this way

Three separations do most of the work. Isolating algorithm choice in the suite table makes agility a data change instead of a code change in every crate. Treating block contents as opaque to consensus lets the consensus engine be tested as a pure state machine in deterministic simulation. Segregating witnesses from transaction bodies at the data-structure level makes pruning safe, because no consensus rule ever needs a pruned witness after finality.

TABLE VI
INITIAL SUITE TABLE. SIZES IN BYTES. STATUS AT GENESIS.

ID	Scheme	$\|pk\|$	$\|\sigma\|$	Status / permitted roles
`0x0001`	ML-DSA-44	1312	2420	Reserved
`0x0002`	ML-DSA-65	1952	3309	Active: all roles (default)
`0x0003`	ML-DSA-87	2592	4627	Active: operator, governance
`0x0101`	SLH-DSA-SHAKE-128s	32	7856	Active: recovery only
`0x0201`	ML-DSA-65 + Ed25519	1984	3373	Reserved (composite)
`0x0301`	FN-DSA-512	897	$\leq$666	Reserved (FIPS 206)

IX. CRYPTOGRAPHIC ARCHITECTURE

This section fixes the primitives, the roles they play, and the full signing and verification lifecycle. The normative encodings appear in Appendix A, and the constants and domain tags in Appendix G.

A. Signature schemes and the suite table

Every public key and signature in XENORRA is tagged with a 16-bit suite identifier. The suite table (Table VI) is part of consensus state, and each entry carries a lifecycle status.

a) Decision C1: ML-DSA-65 as the default signature scheme: We considered ML-DSA at all three parameter sets, SLH-DSA, FN-DSA (Falcon), and stateful hash-based schemes (XMSS/LMS). FN-DSA would cut transaction size by about three quarters (Table XVIII), which is a large saving. We still do not select it for launch. Its standard (FIPS 206) had not been published, even as a draft, when this specification was written [27]; its signing procedure requires double-precision floating-point Gaussian sampling, which is notoriously hard to implement in constant time; and Rust implementations are less mature than those for ML-DSA. We reserve a suite identifier for it and expect to revisit the decision once FIPS 206 is final and an audited constant-time implementation exists. Stateful hash-based schemes are unsuitable as general account keys because reusing a one-time key state, for example after restoring a wallet from backup, breaks security completely. SLH-DSA is conservative but its signatures are 7.8–50 KB (Section V), too large for routine use.

Among the ML-DSA parameter sets, ML-DSA-44 targets NIST category 2 and ML-DSA-65 category 3. ML-DSA-44 would save about 25% per transaction. We choose ML-DSA-65 because ledger keys are long-lived and cannot be recalled, and because lattice cryptanalysis is the area where we most want margin. ML-DSA-87 (category 5) is used for keys that control large aggregate value: validator operator keys, governance keys, and treasury keys. We use one scheme family for transactions, votes, and peer identity so that only one implementation sits on the consensus-critical path.

b) Decision C2: hash-based recovery keys: An account may register a recovery key under SLH-DSA-SHAKE-128s. Only a 32-byte commitment to the recovery public key is stored. The key is meant to be used rarely, perhaps never, so its 7,856-byte signature and slow signing (hundreds of milliseconds) are acceptable. Its security rests only on properties of the hash function [6], a different and more conservative assumption than the lattice problems behind ML-DSA. If a lattice break forces ML-DSA into emergency freeze (Section IX-H), recovery keys remain valid and owners can rotate to a new scheme. We chose the SHAKE instantiation so that the protocol depends on a single hash family (Keccak), which keeps the dependency set and the audit scope small; the SHA-2 variant would sign faster on processors with SHA extensions, but recovery is not a performance-sensitive path.

c) Decision C3: hybrid signatures are optional: Hybrid (composite) signatures that combine ML-DSA with a classical scheme such as Ed25519 protect against an unexpected weakness in the post-quantum scheme, but only until a CRQC arrives, after which the classical half adds nothing. For a new chain without legacy classical keys, the more useful hedge is one that stays valid in a post-quantum world, which is the role of the SLH-DSA recovery key. We reserve a composite suite for institutions whose policies require it, aligned with the IETF composite-signature work [34], but do not make it the default. This revises the Whitepaper v2.0 recommendation of hybrid signatures for high-value operations.

B. Hashing

a) Decision C4: SHA3-256 for all consensus hashing: All consensus-relevant hashes use SHA3-256 [45] with explicit domain separation:

$$\mathsf{H}_{\mathtt{tag}}(m) = \text{SHA3-256}\big(\mathsf{enc}_{u8}(|tag|) \| tag \| m\big), \qquad (3)$$

where tag is an ASCII string from the registry in Appendix G and m is a canonical encoding. Key derivation uses KMAC256 and extendable output uses SHAKE256 [46]. We considered SHA-256 and BLAKE3. BLAKE3 is several times faster [19], but in XENORRA hashing accounts for a small share of validation cost next to signature verification (Section XX), and BLAKE3 is not a NIST standard, a concern for the regulated institutions the identity modules target. SHA-256 is equally standardized and faster on CPUs with SHA extensions, but adopting it would add a second hash family alongside the Keccak permutation that ML-DSA and SLH-DSA-SHAKE already require. Using SHA-3 throughout means a single permutation to implement, test, and audit. The structures that carry a hash (block headers, the state tree, Merkle roots) include a version field, so a future change of hash function is a versioned migration.

C. Addresses and key records

An account address is the domain-separated hash of the suite identifier and the account's *initial* public key:

$$addr \;=\; \mathsf{H}_{\mathtt{XEN/addr}}\big(\mathsf{enc}_{u16}(suite) \,\|\, pk_0\big). \qquad (4)$$

The address keeps the full 256-bit output, because truncating it would reduce the cost of finding a second public key that maps to a victim's address. Addresses are displayed in Bech32m with human-readable prefix `xen` on mainnet and `txen` on test networks.

An account is created implicitly when it first receives funds. Its public key is not revealed until it first spends, which gives unused addresses the same protection that pay-to-public-key-hash gives in Bitcoin, though, as Section IV notes, this is not a defence against a fast CRQC once a key is in use. After the first spend the account stores a *key record*, which can take one of two forms:

- *Hashed*: the state stores only a 32-byte commitment $\mathsf{H}_{\mathtt{XEN/key}}(suite \| pk)$, and every transaction carries the full public key in its witness.
- *Registered*: the state stores the full public key, and transactions omit it.

a) Decision C5: two key-record modes: The choice trades bandwidth for state size. A registered ML-DSA-65 key adds 1,952 bytes of permanent state per account; a hashed key adds 1,956 bytes to every transaction but those bytes sit in the witness, which is pruned after finality. Permanent state is the more expensive resource, since every validator must hold it in fast storage indefinitely. Hashed mode is the default, and accounts that transact frequently (exchanges, verifiers, payment processors) can pay a refundable storage deposit to register their key. The protocol does not need to guess which accounts are active, because owners make the choice and bear its cost.

D. Transaction authentication lifecycle

Signing and verification proceed as in Algorithm 1. Three details matter for interoperability.

What is signed. The signer computes the transaction identifier $txid = \mathsf{H}_{\mathtt{XEN/tx}}(\mathsf{enc}(tx))$ over the canonical encoding of the transaction body, which includes the chain identifier, sender, nonce, fee parameters, expiry height, and call. The ML-DSA message is $txid$, and the FIPS 204 context string [5] is set to `"xenorra/tx/v1"`. Each signed object type uses its own context string (Appendix G), so a signature produced for one purpose cannot be replayed as another.

Hedged signing. Wallets and validators use the hedged variant of ML-DSA signing, which mixes fresh randomness with the private key. Hedged signing is more robust against fault attacks than the deterministic variant [5], [31]. Because the signature is randomized, the same transaction can carry many valid signatures; this is harmless because $txid$ does not cover the witness (Section XII-G).

Strict decoding. Verifiers reject a witness whose public key or signature length differs from the suite's fixed length, whose suite is not *Active* (or *Deprecated*, during the grace period) at the current height, or whose public key does not match the account's key record. For an account with no key record yet, the verifier checks Equation (4) against the sender address.

E. Validator and network authentication

A validator has three keys with different exposure. The *consensus key* (ML-DSA-65) signs proposals and votes; it is

Algorithm 1 Transaction signing (wallet) and verification (node)

Input: body tx, key pair (pk, sk) of suite s, account state A, context $c_{\text{tx}} = \texttt{"xenorra/tx/v1"}$

function SIGNTX(tx, sk, s)
  $txid \leftarrow \mathsf{H}_{\texttt{XEN/tx}}(\mathsf{enc}(tx))$
  $\sigma \leftarrow$ ML-DSA.SIGN($sk, txid, c_{\text{tx}}$) ▷ hedged
  $pkopt \leftarrow pk$ **if** $A.key$ is absent or hashed **else** $\perp$
  return $(tx, w{=}(s, pkopt, \sigma))$
function VERIFYTX(tx, w, A, h)
  if status$(w.s, h) \notin \{$ACTIVE, DEPRECATED$\}$ **then**
    return REJECT
  if $|w.\sigma| \neq \text{siglen}(w.s)$ **then**
    return REJECT
  if $A.key = \perp$ **then** ▷ first spend
    require $w.pkopt \neq \perp \wedge \mathsf{H}_{\texttt{XEN/addr}}(w.s \| w.pkopt) = tx.sender$
    $pk \leftarrow w.pkopt$
  else if $A.key$ is *Hashed*(c) **then**
    require $\mathsf{H}_{\texttt{XEN/key}}(w.s \| w.pkopt) = c$; $pk \leftarrow w.pkopt$
  else
    require $w.pkopt = \perp$; $pk \leftarrow A.key.pk$
  $txid \leftarrow \mathsf{H}_{\texttt{XEN/tx}}(\mathsf{enc}(tx))$
  return ML-DSA.VERIFY$(pk, txid, w.\sigma, c_{\text{tx}})$

online, held in a remote signer that refuses to sign two conflicting messages for the same height and round (Section XIX), and can be rotated at epoch boundaries. The *operator key* (ML-DSA-87) controls stake, commission, and key rotation, and is kept offline. A *node key* (ML-DSA-65) authenticates the validator's sentries and peers at the network layer. Separating these keys limits what an attacker gains by compromising the most exposed one.

At the transport layer, XENORRA uses TLS 1.3 over QUIC with the hybrid X25519MLKEM768 key exchange [35], which protects session confidentiality against harvest-now-decrypt-later collection while retaining classical security if ML-KEM were found weak. Peer authentication is done at the application layer: after the handshake each side signs the TLS exporter value together with both node identifiers and the chain identifier using its ML-DSA node key. Binding the signature to the exporter means that an attacker who could impersonate the classical transport identity (for example with a CRQC) still cannot relay the post-quantum authentication between two separate sessions. Section X gives the handshake.

F. Wallet key derivation and recovery

a) Decision C6: hardened-only, KMAC-based derivation: ML-DSA has no algebraic structure that allows public child-key derivation in the style of BIP-32, and we would not want it: public derivation from an extended public key is exactly the property a CRQC would exploit. XENORRA wallets use a 24-word BIP-39 mnemonic [47] to derive a 512-bit seed, then derive every key along a fully hardened path with KMAC256, in the spirit of SLIP-0010:

$$(k_0, c_0) = \text{KMAC}_{\text{m}}(seed), \tag{5}$$

$$(k_{i+1}, c_{i+1}) = \text{KMAC}_{\text{c}}\big(k_i,\, c_i \| \mathsf{enc}_{u32}(j_i)\big), \tag{6}$$

$$\xi = \text{KMAC}_{\text{k}}\big(k_\ell,\, \mathsf{enc}_{u16}(suite)\big), \tag{7}$$

where KMAC_{m}, KMAC_{c}, and KMAC_{k} are KMAC256 keyed by their first argument, with customization strings `xenorra-hd-v1`, `xenorra-hd-child`, and `xenorra-keygen` and outputs of 512, 512, and 256 bits. The ML-DSA key pair is then ML-DSA.KEYGEN_INTERNAL(ξ), as defined in FIPS 204. SLH-DSA recovery keys use the same derivation with a 384-bit output split into the three n-byte seeds that FIPS 205's internal key generation takes. The path has the form `m/7363'/coin'/account'/role'/index'`, where `role` distinguishes spending, recovery, consensus, node, and DID-assertion keys. Hardened-only derivation has one cost: watch-only wallets cannot generate new receiving addresses from public data alone. They must be handed a list of pre-derived addresses.

Recovery keys derived from the same mnemonic protect against a broken signature scheme and a lost device, but not against a stolen mnemonic. Wallets should offer a separately generated recovery mnemonic, or a guardian-held recovery key, for users who want theft protection as well.

G. Key rotation and recovery

An owner rotates keys with a `RotateKey` call signed by the current key, which replaces the key record immediately. Recovery uses three calls. `InitiateRecovery`, signed by the SLH-DSA recovery key, names a new key record and starts a delay of D_{rec} blocks. During the delay the current key can cancel the recovery with `CancelRecovery`. After the delay, `FinalizeRecovery` installs the new key record. The delay protects owners whose recovery key, but not primary key, has been stolen. If the primary suite has been frozen, cancellation is impossible, because signatures under a frozen suite are rejected. This is intended, since a frozen suite's signatures can no longer be trusted.

H. Cryptographic agility and migration

Suites move through the lifecycle RESERVED → ACTIVE → DEPRECATED → RETIRED, with an emergency transition to FROZEN from any state. Status changes are governance actions with minimum notice periods (Appendix G):

1) *Planned migration.* A new suite becomes ACTIVE. After notice, the old suite becomes DEPRECATED: no new key records may use it, but existing keys still sign, and wallets prompt users to rotate. After a further period the old suite becomes RETIRED, and its signatures are rejected. Accounts that have not rotated must use their recovery key.
2) *Emergency freeze.* If credible evidence of a break emerges, a governance fast track or a supermajority of validators can freeze a suite. Signatures under it are

TABLE VII
CRYPTOGRAPHIC ROLES AND PRIMITIVES. KEM = KEY ENCAPSULATION; IT NEVER AUTHENTICATES.

Role	Primitive	Notes
Transaction auth.	ML-DSA-65	Hashed or registered key record
Account recovery	SLH-DSA-SHAKE-128s	Commitment only in state; delay D_{rec}
Proposals, votes	ML-DSA-65	Remote signer, double-sign guard
Validator operator	ML-DSA-87	Offline; controls stake and key rotation
Governance, treasury	ML-DSA-87	Multi-key threshold accounts
Node identity	ML-DSA-65	Bound to TLS exporter
Session keys	X25519 + ML-KEM-768	Hybrid KEM in TLS 1.3; confidentiality only
Hashing, Merkle trees	SHA3-256	Domain-separated, Eq. (3)
Key derivation	KMAC256	Hardened only, Eqs. (5)–(7)
Randomness beacon	SHA3-256 commit–reveal	No VRF; see Section XI-L

rejected from the activation height on. Recovery keys under a different suite remain usable.

Accounts that registered no recovery key and whose primary suite is frozen are stuck until a further mechanism exists. The most promising candidate is a hash-based zero-knowledge proof of knowledge of the derivation seed ξ behind the address. Because the address commits to pk_0, which is a deterministic function of ξ, such a proof would show ownership without relying on the broken scheme. We list this as an open problem (Section XXIII), and wallets register a recovery key by default to keep the stuck population small.

I. Summary of key roles

Table VII summarizes which primitive serves which role. The table enforces the distinction the project brief asks for: ML-KEM is a key-encapsulation mechanism and appears only in the transport; it is never used to authenticate anything.

X. NETWORK ARCHITECTURE

The network layer has to do two things that pull against each other. It must move large messages (3.4 KB transactions, 3.4 KB votes, proposals of several megabytes) with as little redundancy as possible, and it must resist eclipse and denial-of-service attacks, which usually call for redundancy. This section describes how XENORRA balances the two.

A. Transport and peer authentication

a) Decision N1: QUIC with hybrid key exchange, libp2p for discovery and gossip: Connections use QUIC, which provides multiplexed streams without head-of-line blocking, connection migration, and built-in address validation against amplification. TLS 1.3 inside QUIC negotiates the hybrid X25519MLKEM768 group [35]. The QUIC implementation is `quinn` with rustls. We use the libp2p stack [48] for its Kademlia-based peer discovery and its GossipSub v1.1 publish–subscribe protocol, which includes peer scoring designed to resist Sybil and eclipse attacks on the mesh [49]. The alternatives were a bespoke transport (more control, more code to audit) and a Noise-based handshake, for which a post-quantum variant has been analyzed but which is not yet the default in mainstream libraries.

Algorithm 2 XenAuth: post-quantum peer authentication over an established QUIC/TLS session

Input: session S, node key (pk_A, sk_A), chain id χ, context $c_{\mathrm{p}} =$ "`xenorra/p2p/v1`"

1: $e \leftarrow$ 32-byte TLS exporter of S (label `EXPORTER-xenorra-auth`)
2: send $\mathsf{Hello}_A = (\chi, ver_A, pk_A, nonce_A)$; receive Hello_B
3: **abort** if $\chi_B \neq \chi$ or versions incompatible
4: $T \leftarrow \mathsf{H}_{\texttt{XEN/auth}}(e \| \mathsf{Hello}_A \| \mathsf{Hello}_B)$, ordered by role
5: send $\sigma_A \leftarrow$ ML-DSA.SIGN$(sk_A, T, c_{\mathrm{p}})$
6: receive σ_B; **abort** unless ML-DSA.VERIFY$(pk_B, T, \sigma_B, c_{\mathrm{p}})$
7: $nid_B \leftarrow \mathsf{H}_{\texttt{XEN/node}}(suite \| pk_B)$; apply peer policy to nid_B

libp2p identifies peers by a hash of an Ed25519 or secp256k1 key, which is not post-quantum. XENORRA treats the libp2p peer identifier as a transport label only, and authenticates peers with an application-level handshake (Algorithm 2). The XENORRA node identifier is $nid = \mathsf{H}_{\texttt{XEN/node}}(suite \| pk_{\mathrm{node}})$. Because each side signs the TLS exporter of its own session, a man-in-the-middle who can forge classical transport identities ends up holding two sessions with different exporters, and cannot make either party accept the other's ML-DSA signature.

B. Discovery and topology

Full nodes discover peers through a Kademlia DHT seeded from bootstrap nodes run by several independent operators. Each node keeps a bucketed address book with separate "new" and "tried" tables, limits on the number of addresses and connections per IPv4 /16, IPv6 /32, and autonomous system, a small set of *anchor* peers persisted across restarts, and periodic short-lived *feeler* connections to test addresses. These countermeasures follow the analysis of eclipse attacks on Bitcoin [25]. A node keeps at least eight outbound connections of its own choosing, which an attacker cannot fill by connecting inbound.

Validators do not expose their addresses. Each validator connects only to its own sentry nodes, and the sentries take part in the public network. Validators may keep direct, mutually authenticated links to other validators (a *validator overlay*), which cuts consensus latency without exposing them to arbitrary peers.

C. Message propagation

a) Decision N2: verify before relay, announce before push for large messages: Every gossiped message is fully validated before it is forwarded, including its signature. This costs a verification per hop, but forwarding unverified messages would let an attacker turn the network into an amplifier for invalid 3.3 KB signatures. GossipSub's validation hook implements this, and peers that send invalid messages lose score and are eventually disconnected.

Votes and transactions travel on separate topics. For votes, nodes periodically exchange compact bitmaps of the votes they already hold for the current height and round, and send only the missing ones, a technique carried over from Tendermint's gossip layer [9]. For transactions, eager push goes to a small mesh (degree D) and the remaining peers receive 32-byte announcements and pull on demand, which is GossipSub's IHAVE/IWANT mechanism. With signed messages of several kilobytes, the bandwidth saved by pulling instead of pushing outweighs the extra round trip.

D. Block propagation

a) Decision N3: compact proposals: A proposal carries the block header, the proposer's signature, and for each transaction an 8-byte short identifier derived from its witness hash with a key taken from the header, as in BIP-152. Keying short identifiers per block prevents an attacker from precomputing collisions. Receivers rebuild the block from their mempools and request missing transactions by short identifier from the proposer or any peer. A 600-transaction proposal needs about 5 KB plus the header, instead of about 2 MB. If reconstruction fails or the short identifiers collide, the receiver falls back to fetching the full block in Merkle-authenticated parts.

E. Synchronization

A new node synchronizes in three stages. (1) *Checkpoint*: the operator supplies a trusted recent block hash, as required by weak subjectivity [24]; the client refuses checkpoints older than the trusting period. (2) *State sync*: the node downloads a state snapshot in chunks, verifies each chunk against the state root in the checkpoint header, and rebuilds the state tree. (3) *Block sync*: the node fetches blocks from the checkpoint forward, verifying each commit certificate, until it reaches the head and switches to live consensus. Archive nodes may instead replay from genesis.

Light clients follow validator-set changes from epoch to epoch and verify individual headers using the skipping procedure of Section XV-F.

F. Spam and denial-of-service resistance

Defences are layered so that expensive work happens last.

1) *Connection level*: QUIC retry tokens for address validation, connection limits per address prefix, and handshake rate limits. ML-DSA verification during XenAuth costs tens of microseconds, so the handshake itself is not an attractive amplification target.
2) *Message level*: token-bucket rate limits per peer and per message class, and maximum message sizes per class.
3) *Transaction admission*: stateless checks (encoding, sizes, suite status, chain identifier, expiry) and stateful checks (nonce window, fee cap, balance) come before signature verification. Each peer has a signature-verification budget per second, and failing verifications cost the peer score.
4) *Consensus messages*: votes are accepted only from the current validator set, for the current or next height, and at most one per (validator, height, round, type) is relayed. A second, conflicting vote is kept as evidence and not relayed further.

G. Partitions and recovery

Tendermint chooses safety over liveness. If no side of a partition holds more than two thirds of voting power, no side commits, and the chain stops until the partition heals. On reconnection, nodes catch up on rounds through the rule that moves a node to a higher round on seeing messages from more than one third of voting power for that round, re-request missing votes by bitmap, and continue. A node that has fallen behind by whole heights uses block sync. No reorganization is needed, because no conflicting blocks were committed. Section XI-M discusses the much rarer case in which conflicting commits do occur.

XI. Consensus Protocol

A. Choosing a protocol under large signatures

The usual argument for HotStuff over Tendermint is communication complexity. HotStuff's leader collects votes and forwards a single quorum certificate, so each phase costs $O(n)$ messages instead of Tendermint's $O(n^2)$ [41]. That argument assumes the certificate has constant size, which requires a threshold or aggregate signature. Without aggregation, a certificate is a list of $q = \lfloor 2n/3 \rfloor + 1$ signatures, and the count of messages stops being a good proxy for cost.

Let $s_\sigma = 3{,}309$ bytes be the ML-DSA-65 signature size, $s_v \approx 3{,}376$ bytes the size of an encoded vote, and n the number of validators of equal weight. In Tendermint each validator receives every other validator's prevote and precommit, so per height

$$B_{\text{in}}^{\text{TM}} \approx 2\, n\, s_v. \tag{8}$$

In a HotStuff-style protocol without aggregation, each replica sends one vote per phase to the leader and receives one certificate of q signatures, so replica ingress is about $q\, s_\sigma$ per phase, similar to Tendermint. The leader must send the certificate to all n replicas:

$$B_{\text{leader,out}}^{\text{HS}} \approx n\, q\, s_\sigma \quad \text{per phase.} \tag{9}$$

For $n = 100$ this is about 22 MB per phase at the leader. Spreading the certificate by gossip removes the bottleneck but brings total traffic back to the $\Theta(n^2 s_\sigma)$ of Tendermint. In both cases the network carries on the order of n^2 signatures per decision. Certified DAG protocols are worse: every one of the n vertices in a round carries its own certificate, so each node receives about $n\, q\, s_\sigma$ bytes per round [43]. Uncertified DAGs such as Mysticeti [44] need only one signature per vertex and are an attractive target for later work, but they are much younger and have less independent analysis than Tendermint.

a) Decision K1: Tendermint: Since linear protocols lose their main advantage without aggregation, we choose on other grounds. Tendermint has been deployed for years in many networks, has independent correctness analyses and machine-checked specifications, has a maintained Rust implementation of the core state machine [50], and commits a block after three

communication steps in the good case. Its known weaknesses are that a failed proposer costs a timeout, that the leader schedule is predictable, and that it stops during long periods of asynchrony instead of continuing optimistically. We accept these in exchange for simplicity and maturity. Our modifications are listed in Section XI-F.

B. System model

The validator set $\mathcal{V} = \{1, \dots, n\}$ for an epoch is fixed, with voting powers $w_i > 0$ and $W = \sum_i w_i$. For a set $S \subseteq \mathcal{V}$ write $w(S) = \sum_{i \in S} w_i$. We write "$> 2/3$" for a set of validators whose power exceeds $2W/3$ (a *quorum*) and "$> 1/3$" for a set whose power exceeds $W/3$.

Assumption 1 (Fault threshold)**.** The Byzantine validators $B \subseteq \mathcal{V}$ of each epoch satisfy $w(B) < W/3$.

Assumption 2 (Partial synchrony [51])**.** There is an unknown global stabilization time GST and a known bound Δ such that every message sent between honest validators at time t is delivered by $\max(t, \mathrm{GST}) + \Delta$.

Partial synchrony is the standard way around the impossibility of deterministic consensus in a fully asynchronous network with even one faulty process, and tolerating Byzantine faults of less than one third is optimal in this model [51].

Assumption 3 (Unforgeability)**.** ML-DSA-65 is existentially unforgeable under chosen-message attack against quantum polynomial-time adversaries.

C. Validator selection and voting power

Validators register through the staking module with an operator key, a consensus key, a minimum self-bond, and a commission rate; token holders may delegate to them. At each epoch boundary the module ranks candidates by total bonded stake and selects up to $N_{\max}$ with stake above a minimum. Voting power is bonded stake capped per operator:

$$w_i = \min\Big(stake_i,\ c_{\max} \cdot \textstyle\sum_j stake_j\Big), \tag{10}$$

with the cap applied once, using the uncapped total, so that the computation is deterministic and non-iterative. The cap $c_{\max}$ limits how much power any one operator can hold. It cannot stop an entity from splitting stake across several operators; that is a Sybil problem with no protocol-level solution, and we rely on monitoring and on the identity requirements for institutional validators during the launch phase.

D. Proposer selection without a VRF

a) Decision K2: deterministic weighted round-robin: Each validator keeps a priority p_i, initially zero. Before each round, every priority is increased by the validator's power, the validator with the highest priority (lowest index on ties) is chosen, and its priority is reduced by W (Algorithm 3). Over any window, each validator proposes in proportion to its power. CometBFT uses the same proposer-priority algorithm [42]. Its weakness is predictability: an attacker knows the upcoming proposers and can try to knock them offline. Sentries, short propose timeouts, and the fact that a failed proposer costs only one round limit the damage. The alternatives were a post-quantum VRF, which is not standardized, and a hash-based beacon, which a proposer can bias by choosing whether to publish (Section XI-L). We judged a predictable but unbiasable schedule better than an unpredictable but biasable one.

Algorithm 3 Proposer selection for one round (deterministic)

Input: priorities $p_1..p_n$ (signed 64-bit), powers $w_1..w_n$, $W = \sum w_i$

```
for i = 1 to n do
    p_i ← p_i + w_i
j ← arg max_i p_i                          ▷ lowest index on ties
p_j ← p_j − W
p̄ ← ⌊Σ_i p_i / n⌋;  for i: p_i ← p_i − p̄    ▷ re-centre
if max_i p_i − min_i p_i > 2W then          ▷ bound spread
    d ← ⌈(max_i p_i − min_i p_i)/(2W)⌉;  for i: p_i ← ⌊p_i/d⌋
return j
```

E. The agreement protocol

XENORRA runs the Tendermint algorithm of Buchman, Kwon, and Milosevic [9] at each height h. Algorithm 4 restates it with stake-weighted thresholds. Each validator keeps a round r, a step in $\{\text{PROPOSE}, \text{PREVOTE}, \text{PRECOMMIT}\}$, a *locked* value and round (the value it has precommitted and must not abandon without justification), and a *valid* value and round (the most recent value it has seen a prevote quorum for, which it re-proposes if it becomes proposer). $valid(v)$ is the full block validity predicate of Section XI-G, including re-execution and every signature check.

F. Modifications for XENORRA

We change Tendermint in five ways. None alters its core locking logic, so the existing safety and liveness arguments carry over (Appendix C).

(M1) Post-quantum signed messages. Proposals and votes are signed with ML-DSA-65 under the context strings `"xenorra/proposal/v1"` and `"xenorra/vote/v1"`. The signed payload includes the chain identifier, height, round, message type, block identifier (or $\perp$), and a timestamp.

(M2) Justified re-proposals. A proposal with $validRound = vr \geq 0$ must carry the $> 2/3$ prevotes for its value at round vr, the proof-of-lock (POL) certificate. In standard Tendermint this justification can be gossiped separately. Making it mandatory costs a certificate (about 220 KB at $n = 100$) only in the rare rounds that re-propose a value, and it allows a violation of the locking rule (an "amnesia" attack) to be attributed to specific signers. Sheng *et al.* show that without such information Tendermint's forensic support is limited [52].

(M3) Immediate execution. The header of block h commits to the state root *after* executing block h. CometBFT commits to the state before execution. The proposer executes before proposing, and validators execute before prevoting. This gives light clients a state root for the block they have just verified,

Algorithm 4 Tendermint at height h, from the view of validator p (after [9]; thresholds are by voting power)

function StartRound(r')
  $r \leftarrow r'$; $step \leftarrow$ propose
  if Proposer$(h, r) = p$ **then**
    $v \leftarrow validValue$ **if** $validValue \neq \perp$ **else** BuildBlock()
    broadcast $\langle \text{Proposal}, h, r, v, validRound \rangle_{\sigma_p}$ ▷ with POL certificate if $validRound \geq 0$
  else
    schedule OnTimeoutPropose(h, r) after $\tau_{\text{prop}}(r)$
upon $\langle \text{Proposal}, h, r, v, -1 \rangle$ from Proposer(h, r) while $step =$ propose **do**
  broadcast $\langle \text{Prevote}, h, r, id(v) \rangle$ **if** $valid(v) \wedge (lockedRound = -1 \vee lockedValue = v)$ **else** $\langle \text{Prevote}, h, r, \perp \rangle$; $step \leftarrow$ prevote
upon $\langle \text{Proposal}, h, r, v, vr \rangle$ from Proposer(h, r) **and** $> 2/3$ $\langle \text{Prevote}, h, vr, id(v) \rangle$ while $step =$ propose $\wedge\ 0 \leq vr < r$ **do**
  broadcast $\langle \text{Prevote}, h, r, id(v) \rangle$ **if** $valid(v) \wedge (lockedRound \leq vr \vee lockedValue = v)$ **else** $\langle \text{Prevote}, h, r, \perp \rangle$; $step \leftarrow$ prevote
upon $> 2/3$ $\langle \text{Prevote}, h, r, * \rangle$ while $step =$ prevote, for the first time **do**
  schedule OnTimeoutPrevote(h, r) after $\tau_{\text{pv}}(r)$
upon $\langle \text{Proposal}, h, r, v, * \rangle$ from Proposer(h, r) **and** $> 2/3$ $\langle \text{Prevote}, h, r, id(v) \rangle$ while $valid(v) \wedge step \geq$ prevote, for the first time **do**
  if $step =$ prevote **then**
    $lockedValue \leftarrow v$; $lockedRound \leftarrow r$; broadcast $\langle \text{Precommit}, h, r, id(v) \rangle$; $step \leftarrow$ precommit
  $validValue \leftarrow v$; $validRound \leftarrow r$
upon $> 2/3$ $\langle \text{Prevote}, h, r, \perp \rangle$ while $step =$ prevote **do**
  broadcast $\langle \text{Precommit}, h, r, \perp \rangle$; $step \leftarrow$ precommit
upon $> 2/3$ $\langle \text{Precommit}, h, r, * \rangle$ for the first time **do**
  schedule OnTimeoutPrecommit(h, r) after $\tau_{\text{pc}}(r)$
upon $\langle \text{Proposal}, h, r', v, * \rangle$ from Proposer(h, r') **and** $> 2/3$ $\langle \text{Precommit}, h, r', id(v) \rangle$ while $decision[h] = \perp$ **do**
  if $valid(v)$ **then**
    $decision[h] \leftarrow v$; commit v with certificate; $h \leftarrow h + 1$; reset locked/valid state; StartRound(0)
upon $> 1/3$ messages $\langle *, h, r', * \rangle$ with $r' > r$ **do**
  StartRound(r')
function OnTimeoutPropose(h', r')
  if $h'{=}h \wedge r'{=}r \wedge step{=}$propose: broadcast $\langle \text{Prevote}, h, r, \perp \rangle$; $step \leftarrow$ prevote
function OnTimeoutPrevote(h', r')
  if $h'{=}h \wedge r'{=}r \wedge step{=}$prevote: broadcast $\langle \text{Precommit}, h, r, \perp \rangle$; $step \leftarrow$ precommit
function OnTimeoutPrecommit(h', r')
  if $h'{=}h \wedge r'{=}r$: StartRound$(r + 1)$

at the price of putting execution on the critical path of each round; since native-module execution is cheap compared with signature verification, which must happen before the prevote anyway, the extra latency is small.

(M4) Compact proposals. The proposal carries a header and short identifiers (Section X); $valid(v)$ is evaluated after reconstruction.

(M5) Epoch-bounded validator sets. The validator set changes only at epoch boundaries, every E blocks. The header of the last block in an epoch commits to the next validator set. This bounds the work of light clients and the retention of full commit certificates (Section XV).

G. Block validity

$valid(v)$ holds if and only if: the header is well formed and its version, chain identifier, height, and parent hash are correct; the timestamp equals the BFT time computed from the previous commit certificate (below); the previous commit certificate is valid for the parent; every transaction and witness is well formed and the transaction and witness roots match the header; every evidence item is valid and not previously included; executing the block on the parent state succeeds and yields exactly the state, receipt, and next-validator roots in the header; and the block respects size and weight limits. Appendix A gives the complete rule list.

H. Commit certificates, time, and finality

A commit certificate for block v at height h is the set of precommits for $id(v)$ in the deciding round, encoded as a signer bitmap plus, for each signer, its timestamp and signature. It is valid if the signers' power exceeds $2W/3$ and

TABLE VIII
EVIDENCE TYPES AND PENALTIES. FRACTIONS ARE GOVERNANCE PARAMETERS; VALUES ARE SUBJECT TO MODELLING.

Offence	Proof	Penalty
Duplicate vote	Two signed votes, same (h, r, type), different ids	Correlated slash, jail, tombstone
Duplicate proposal	Two signed proposals, same (h, r)	As above
Lock violation	Precommit at r and prevote at $r' > r$ for another value without a POL certificate from a round in $[r, r')$	As above
Light-client attack	Signed header conflicting with the canonical chain	As above
Downtime	Missed more than a fraction θ of the last L blocks	Small slash, jail

every signature verifies. Block $h+1$ carries the certificate for block h.

Block time follows Tendermint's BFT time: the timestamp of block $h+1$ is the power-weighted median of the timestamps in the commit certificate for block h. With fewer than one third Byzantine power, this median lies between timestamps reported by honest validators, so faulty clocks alone cannot move it outside the honest range.

A committed block is final. Under Assumption 1 two different blocks cannot be committed at the same height (Theorem 1). XENORRA needs no fork-choice rule in normal operation. If a node ever observes two valid commit certificates for different blocks at the same height, which by Theorem 2 implies that validators holding at least one third of power misbehaved, it stops processing, records both certificates as evidence, and waits for operator intervention (Section XI-M).

I. Timeouts

Timeouts grow linearly with the round, $\tau_x(r) = \tau_x^0 + r\,\delta_x$ for $x \in \{\text{prop}, \text{pv}, \text{pc}\}$, so that after GST they eventually exceed the actual message delay and liveness follows. Initial values are tuning parameters for testnets (Appendix G). The propose timeout must cover proposal propagation, block reconstruction, execution, and verification of all transaction signatures in the block, so it scales with block size.

J. Evidence and penalties

Table VIII lists the misbehaviour XENORRA punishes. Equivocation and lock violation endanger safety and are slashed. Downtime endangers only liveness and is handled mainly by jailing, which removes the validator from the active set until it unjails, with a small penalty.

a) Decision K3: correlated slashing: For safety offences, the slashed fraction depends on how much stake was slashed for safety offences in a surrounding window:

$$\begin{aligned} \mathit{slash}_i &= \min(1,\, \kappa\, \rho_{\text{win}}) \cdot \mathit{stake}_i, \\ \rho_{\text{win}} &= \textstyle\sum_{j \in \mathcal{S}_{\text{win}}} \mathit{stake}_j \big/ \sum_j \mathit{stake}_j, \end{aligned} \tag{11}$$

where $\mathcal{S}_{\text{win}}$ is the set of validators slashed within the window and κ is a multiplier. With $\kappa = 3$, an isolated operator error (for example a misconfigured backup signer) costs little, while a coordinated attack by one third of stake loses the whole stake. Ethereum's consensus specification uses a penalty of the same shape. Delegators share their validator's penalties, which gives them a reason to choose operators carefully.

K. Epochs, unbonding, and weak subjectivity

Changes to stake take effect at epoch boundaries. Unbonding stake remains slashable for an unbonding period U. A light client or syncing node must start from a trusted header no older than a trusting period $T_{\text{trust}} < U$, so that any validator set it trusts is still slashable for anything it signs, the standard defence against long-range attacks based on old keys [24]. Because every consensus key in XENORRA is post-quantum from genesis, a CRQC does not make old keys any easier to obtain, so the defence keeps working against $\mathcal{A}_2$.

L. Randomness beacon

The identity modules need unpredictable randomness to assign verification tasks to verifiers. Consensus itself does not use it. XENORRA produces a beacon by commit–reveal among validators. In epoch e each validator includes a commitment $c_i = \mathsf{H}_{\texttt{XEN/rand}}(\rho_i)$ in a transaction; in epoch $e+1$ it reveals ρ_i. The beacon for epoch $e+2$ is $\mathsf{H}_{\texttt{XEN/beacon}}(\rho_{i_1} \| \dots \| \rho_{i_k})$ over all valid reveals in validator-index order. A validator that commits and does not reveal is penalized.

This construction is biasable: the last validators to reveal can see the outcome with and without their reveal and choose, at the price of the penalty. A coalition controlling m withheld reveals can choose among up to 2^m outcomes. Bias-resistant alternatives either need a VRF or threshold signatures (not standardized post-quantum) or verifiable delay functions over groups of unknown order (broken by Shor's algorithm). We use this beacon only where limited bias is tolerable. Verifier assignment combines it with stake caps and redundancy, so a biased draw can tilt but not decide a task, and we list bias-resistant post-quantum randomness as an open problem.

M. Recovery from safety failure

If conflicting blocks are committed, Assumption 1 has failed, and no protocol rule can decide which history is "correct." XENORRA makes the failure visible and attributable: the evidence identifies at least one third of stake by signature (Theorem 2). Recovery is then a social process, as in every BFT chain. Validators coordinate off chain to choose a branch, slash the culprits by a governance-approved state change, and restart. The protocol's job is to make that process quick and well informed. Nodes halt instead of following either branch, and the evidence they collect is complete.

XII. TRANSACTION LIFECYCLE

Fig. 2 follows a transfer from the wallet to final pruning. Each stage is described below, with the checks it performs and the state it touches.

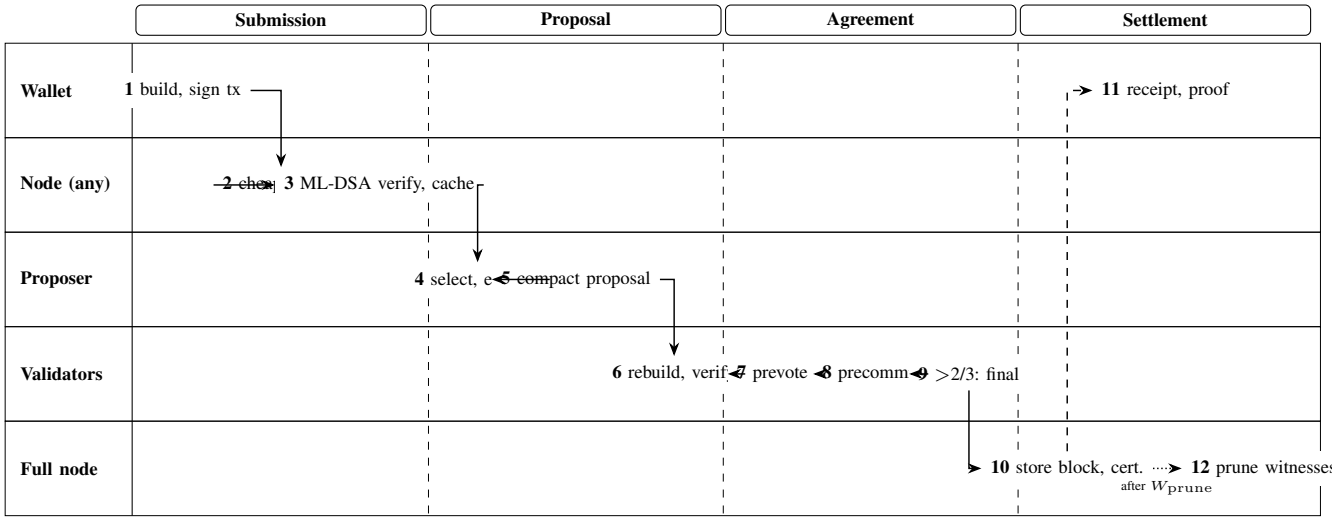


Fig. 2. Transaction lifecycle. Signatures are verified once at admission (3) and cached, so block validation (6) re-verifies only transactions a validator had not already seen. Prevotes and precommits (7, 8) are exchanged among all validators. After the retention window (12), full nodes drop witnesses and per-block certificates, keeping bodies, receipts, and the witness root.

A. Construction and signing

The wallet reads the sender's current nonce and key-record mode from any node (verifying the response against a state proof if it runs as a light client), builds the transaction body (Appendix A-C), and signs it as in Algorithm 1. The body fixes the chain identifier, nonce, maximum fee per weight unit, priority tip, weight limit, and an expiry height after which the transaction can never be included. The expiry bounds how long a signed but unbroadcast transaction remains a liability if the wallet device is later compromised.

B. Admission

A node receiving a transaction applies the admission pipeline of Section X: decode and size checks, suite status, chain identifier, expiry, nonce window (the nonce must lie between the account's current nonce and that nonce plus a small look-ahead), fee cap at or above the current base fee, and sufficient balance for the worst-case fee and transferred amount. Only then does the node verify the signature. A transaction that passes is added to the mempool keyed by $(sender, nonce)$, and its witness hash is cached as verified, so that block validation later does not repeat the signature check for transactions it has already seen.

The mempool orders transactions by tip per weight unit, subject to nonce order per sender. A transaction with the same sender and nonce replaces an earlier one only if it raises the tip by a minimum fraction. This prevents cheap churn, in which an attacker makes nodes verify a stream of 3.3 KB signatures by bumping fees by one unit.

C. Proposal and execution

When a validator becomes proposer, it takes transactions in priority order up to the block's byte and weight limits, executes them against the parent state, drops any that fail admission-level checks at execution time (for example because an earlier transaction in the block spent the balance), and computes the header roots. Transactions that fail during execution for reasons the sender controls, such as a call to a non-existent DID, are *included* with a failure receipt and pay their fee; otherwise a sender could get free verification work from every validator by submitting doomed transactions.

D. Validation and voting

Validators rebuild the block from the compact proposal, verify any signatures not already cached, re-execute, and compare every root. Signature verification is independent across transactions and runs on all cores (Section XIV-E). A validator prevotes for the block only if every check passes. The consensus steps of Algorithm 4 then run to commitment.

E. Finality, receipts, and notification

When a node sees a valid commit certificate for the block, every transaction in it is final. Wallets and relying parties can prove inclusion to a light client with the header, the commit certificate (or a chain of epoch certificates to a trusted set), a Merkle path to the transaction in the transaction root, and a Merkle path to its receipt. In the good case, finality takes three message delays after the proposal is received, plus execution time.

F. Pruning

After W_{prune} blocks (Appendix G), full nodes delete the block's witnesses and its commit certificate unless the block ends an epoch. The header, transaction bodies, and receipts remain. Pruned data is no longer needed for consensus: the witnesses were checked by more than two thirds of voting power before commitment, and the header's witness root still commits to them, so an archive node can later serve them to anyone who wants to re-verify.

TABLE IX
BLOCK HEADER FIELDS (CANONICAL ORDER). ALL HASHES ARE 32 BYTES.

Field	Type	Meaning
`version`	u16	Protocol version
`chain_id`	u64	Network identifier
`height`	u64	Block height
`epoch`	u64	Epoch number
`time_ms`	u64	BFT time (weighted median)
`parent`	hash	Hash of parent header
`last_commit`	hash	Hash of parent's commit certificate
`tx_root`	hash	Merkle root of *txid*s
`witness_root`	hash	Merkle root of *wtxid*s
`state_root`	hash	State root *after* this block
`receipts_root`	hash	Merkle root of receipts
`evidence_root`	hash	Merkle root of evidence
`validators`	hash	Current validator set
`next_validators`	hash	Set for next height (changes at epoch end)
`params`	hash	Consensus parameters and suite table
`proposer`	u16	Proposer index in validator set

G. Replay and malleability

Replay. A transaction is valid at most once on a given chain because its nonce must equal the account's current nonce, which then increments. It cannot be replayed on another XENORRA network because the chain identifier is part of the signed body. It cannot be replayed as a different message type because each type has its own ML-DSA context string. After its expiry height it cannot be included at all.

Malleability. ML-DSA's hedged signing produces a different valid signature each time, and anyone holding a valid signature might try to produce another. If the transaction identifier covered the signature, a relayer could change a transaction's identifier without invalidating it, the malleability problem that led Bitcoin to segregated witness. In XENORRA, $txid$ covers only the body and $wtxid = \mathsf{H}_{\texttt{XEN/wtx}}(txid \| \mathsf{enc}(w))$ identifies the witnessed form. Account state, receipts, and user-facing references use $txid$; mempool deduplication and the witness root use $wtxid$.

XIII. LEDGER AND STATE MANAGEMENT

A. Block structure

A block consists of a header, an ordered list of transaction bodies, a parallel list of witnesses, a list of evidence items, and the commit certificate for the parent block. Table IX lists the header fields. The header commits separately to bodies and to witnesses, which is what makes witness pruning safe: a pruned node can still prove that a transaction was included, and an archive node can prove that a given witness was the one checked.

The block hash is $\mathsf{H}_{\texttt{XEN/header}}(\mathsf{enc}(header))$. A header is 356 bytes in canonical encoding (computed from the field types in Table IX), small enough that a light client can store every epoch-boundary header for years.

B. Merkle lists

Lists of transaction identifiers, witness identifiers, receipts, and evidence are committed with a binary Merkle tree using distinct prefixes for leaves and interior nodes, following RFC 6962:

$$\begin{aligned} \mathrm{MTH}(\langle\rangle) &= \mathsf{H}_{\texttt{XEN/mt}}(\varepsilon), \\ \mathrm{MTH}(\langle d \rangle) &= \mathsf{H}_{\texttt{XEN/mt}}(\texttt{0x00} \| d), \\ \mathrm{MTH}(D) &= \mathsf{H}_{\texttt{XEN/mt}}\big(\texttt{0x01} \| \mathrm{MTH}(D_{<k}) \| \mathrm{MTH}(D_{\geq k})\big), \end{aligned} \tag{12}$$

where $|D| > 1$, k is the largest power of two smaller than $|D|$, and $D_{<k}$ and $D_{\geq k}$ are the first k elements and the rest. The leaf/node prefixes prevent an interior node from being reinterpreted as a leaf, the classic second-preimage weakness of naive Merkle trees. The split rule gives a unique tree shape for every list length, without the duplicated-last-element ambiguity that once affected Bitcoin's transaction tree.

C. State model

All state lives in one key–value map, partitioned into namespaces: accounts, validators, delegations, unbonding entries, the suite table, consensus parameters, governance proposals, the randomness beacon, and the identity-module tables (DID documents, credential anchors and revocation status, verifier registry, model registry). The authenticated key of an entry is

$$k = \mathsf{H}_{\texttt{XEN/skey}}(\mathsf{enc}_{u8}(ns) \,\|\, key), \tag{13}$$

which spreads entries uniformly over the tree, so that an attacker cannot deepen particular paths by choosing keys.

An account entry holds a nonce, a balance, a key record (absent, hashed, or registered; Section IX), an optional recovery record (suite, commitment, delay, and any pending recovery), and the storage deposit currently held. Balances are unsigned 128-bit integers in base units. Every arithmetic operation in the state-transition function is checked, and overflow makes the transaction fail.

D. Authenticated state tree

a) Decision L1: binary sparse Merkle tree: The state root commits to the map through a binary sparse Merkle tree of depth 256 in the style of the Jellyfish Merkle tree [40]. Leaves and interior nodes are hashed as

$$\begin{aligned} \mathrm{leaf}(k, v) &= \mathsf{H}_{\texttt{XEN/smt}}\big(\texttt{0x00} \| k \| \mathsf{H}_{\texttt{XEN/val}}(v)\big), \\ \mathrm{node}(\ell, r) &= \mathsf{H}_{\texttt{XEN/smt}}(\texttt{0x01} \| \ell \| r), \end{aligned} \tag{14}$$

with a fixed placeholder for empty subtrees and with subtrees that contain a single leaf collapsed to that leaf. A membership or non-membership proof carries one sibling hash per non-trivial level, about $\log_2 N$ hashes for N entries: roughly 27 hashes, or 860 bytes, for a hundred million entries. Non-membership proofs matter for the identity application, because "this credential has not been revoked" is a statement of absence.

We considered a hexary Merkle–Patricia trie (larger proofs, more complex specification), a Verkle tree (smaller proofs, but based on polynomial commitments over elliptic curves, which are not post-quantum), and no authenticated state (cheapest, but no light-client state queries). The binary tree's cost is more hashing per update, about 27 hash evaluations instead of about 7 for a hexary trie, which is negligible next to signature verification.

E. Versioning and history

Tree nodes are stored under keys that include the version (block height) at which they were written, as in the Jellyfish design. This lets a node serve proofs against any retained historical root without copying state, and lets a pruner delete nodes that became stale before the retention horizon. Full nodes keep state versions for the most recent W_{state} blocks and for epoch boundaries; archive nodes keep all versions.

F. Supply accounting

The state-transition function maintains the invariant

$$\begin{aligned}\textstyle\sum_a bal_a + \sum_v bonded_v + \sum_u unbond_u \\ + treasury + \textstyle\sum_a dep_a = S_0 + I_h - Burn_h,\end{aligned} \tag{15}$$

where S_0 is the genesis supply, I_h the cumulative issuance up to height h, and $Burn_h$ the cumulative burned base fees and slashed amounts. Appendix C proves that each native call preserves Equation (15). Implementations must check it as a debug assertion after every block and as a property-based test over random transaction sequences.

XIV. Execution Environment

A. Native modules instead of a virtual machine

a) Decision E1: no general-purpose smart contracts in the first release: The case for smart contracts is real: they let applications evolve without protocol upgrades and attract developers. Their costs are just as real. A deterministic VM, its gas schedule, and its host interface add more specification and audit surface than the rest of the protocol combined; every contract becomes a potential source of exploits that users experience as failures of the chain; and gas metering must be calibrated against worst-case behaviour that is hard to bound. The applications XENORRA targets first (transfers, staking, identity registration, credential anchoring, the verifier market) have well-understood state transitions that can be written as native modules, specified exactly, and tested exhaustively.

We ship a fixed set of native modules (Table X) and defer a VM. If one is added later, the candidates are WebAssembly with floating point disabled and fuel metering, a RISC-V interpreter, or a resource-oriented language such as Move. The decision criteria are: a deterministic execution specification independent of any one implementation, metering whose worst case can be bounded per weight unit, a mature Rust implementation, and an isolation boundary that keeps contract failures out of the consensus-critical native modules.

B. The state-transition function

Let S be the state at the parent block and B a block with transactions $tx_1, \ldots, tx_m$. The state-transition function passes the state through three stages:

$$\begin{aligned}T_0 &= \mathrm{BeginBlock}(S, B), \\ T_i &= \mathrm{ApplyTx}(T_{i-1}, tx_i), \quad i = 1, \ldots, m, \\ \mathrm{STF}(S, B) &= \mathrm{EndBlock}(T_m, B),\end{aligned} \tag{16}$$

and returns the new state together with a list of receipts. *BeginBlock* pays the previous block's rewards, applies included evidence (slashing and jailing), and, at an epoch's first block, processes matured unbonding entries. *ApplyTx* for transaction i checks the nonce and fee cap, charges the maximum fee, increments the nonce, and executes the call in a nested state overlay. If the call succeeds, the overlay is merged; if it fails, the overlay is discarded but the fee and nonce changes stand. The unused part of the fee is then refunded. *EndBlock* records downtime, and at the last block of an epoch computes the next validator set, settles the randomness beacon, and applies governance-scheduled parameter changes. Appendix A-G gives each call's preconditions and effects.

TABLE X
NATIVE CALLS IN THE FIRST RELEASE.

Module	Calls
Bank	`Transfer`
Account	`RegisterKey`, `UnregisterKey`, `RotateKey`, `SetRecovery`, `InitiateRecovery`, `CancelRecovery`, `FinalizeRecovery`
Staking	`CreateValidator`, `EditValidator`, `Delegate`, `Undelegate`, `Redelegate`, `Unjail`, `RotateConsensusKey`, `BeaconCommit`, `BeaconReveal`
Governance	`Propose`, `Vote`, `Execute`
Identity	`DidCreate`, `DidUpdate`, `DidDeactivate`, `CredAnchor`, `CredRevoke`
Verification	`VerifierRegister`, `VerifyRequest`, `VerifyCommit`, `VerifyReveal`, `Challenge`, `ModelRegister`

C. Determinism rules

Every honest node must compute the same state root from the same block. The specification forbids floating-point arithmetic, iteration over unordered collections, reading local time (only the header's BFT time is available), and unbounded loops. All arithmetic is on fixed-width integers with explicit overflow handling, and any rounding is specified (always toward zero unless stated). Collections that are iterated, such as the validator set or pending unbonding entries, are kept in canonical order by key. These rules are enforced in the Rust implementation by keeping the execution crate free of `std::collections::HashMap` iteration, floating-point types, and system-clock access, checked by lints in continuous integration (Section XIX).

D. Resource accounting

Each transaction is charged in *weight* units, which approximate worst-case validation cost:

$$\begin{aligned}weight(tx) = \alpha_{\text{byte}}\,|tx|_{\text{bytes}} + \alpha_{\text{sig}}(s) \\ + \alpha_{\text{r}}\,n_{\text{read}} + \alpha_{\text{w}}\,n_{\text{write}} \\ + \alpha_{\text{new}}\,b_{\text{new}} + \alpha_{\text{call}}(c),\end{aligned} \tag{17}$$

where $|tx|_{\text{bytes}}$ includes the witness, $\alpha_{\text{sig}}(s)$ is the verification cost of suite s, n_{read} and n_{write} count state accesses, b_{new} counts newly allocated state bytes, and $\alpha_{\text{call}}(c)$ is a fixed cost per call type. Native calls touch a statically known set of keys, so n_{read} and n_{write} are known before execution and the weight can be computed during admission. The coefficients

are set so that one weight unit corresponds to a fixed amount of reference-machine time, calibrated by the benchmarks of Section XXI. Until those benchmarks exist, the coefficients are unknown and must not be guessed in a deployed network.

New state carries a refundable *storage deposit* proportional to b_{new}, returned when the state is deleted. This applies to registered keys, DID documents, and credential anchors, and it is what makes the hashed/registered key choice of Section IX self-regulating.

E. Parallel execution

Two kinds of work in block validation can run in parallel, and they differ greatly in difficulty.

Signature verification needs only the transaction, its witness, and the account's key record as of the parent state. Within one block, a transaction that changes its own sender's key (`RotateKey`, `FinalizeRecovery`) can affect only later transactions from the same sender, whose nonces are ordered after it, so the rule is simple: verify each transaction against the key record in effect at its position, computed by a fast sequential pass over key-changing calls. All verifications then run on a thread pool. Since verification dominates CPU time (Section XX), this captures most of the available speedup.

Execution can be parallelized as well, because each native call declares the keys it reads and writes. The scheduler builds a conflict graph from these declarations and runs non-conflicting transactions concurrently in waves, producing a result identical to sequential execution in block order. Fee and reward accumulators, which every transaction touches, are handled as per-transaction deltas merged deterministically at the end of the block. For the first release we specify *sequential* execution semantics and implement execution sequentially with parallel signature verification. Parallel execution is an optimization that must be differentially tested against the sequential reference, as Block-STM was against its baseline [20]. It never changes the specification.

F. Identity and verification modules

The identity modules implement the application layer of the Xenorra Whitepaper v2.0 within these rules. A DID document stores a controller account, a list of verification methods (each an ML-DSA or SLH-DSA public key or key commitment with a suite identifier), and service endpoints; no personal data is stored. A credential anchor stores a salted commitment to a credential, its schema identifier, its issuer DID, and a revocation flag. Relying parties check revocation with a non-membership or membership proof against a recent header.

The verification market runs the request, allotment, commit, reveal, aggregation, and challenge protocol described in the whitepaper. Each step is a transaction: verifiers commit to a hash of their result in one window and reveal it in the next, the module aggregates revealed results against a quorum rule, and a bonded challenge window follows finalization. Task allotment draws from the randomness beacon of Section XI-L, weighted by a capped selection score. AI-assisted checks produce attestations, each a signed record naming a registered model hash, a commitment to the input, a structured result, and a confidence class, that the verifier includes in its reveal. Model output never enters consensus directly. The chain checks the verifier's signature and the protocol's aggregation rule, and the verifier remains accountable and slashable for its result.

TABLE XI
DEFAULT RETENTION BY NODE ROLE. W_{prune} AND W_{state} MUST EXCEED THE EVIDENCE WINDOW; VALUES ARE IN APPENDIX G.

Data	Full node	Archive node
Headers	All	All
Tx bodies, receipts	All	All
Witnesses	Last W_{prune} blocks	All
Commit certificates	Last W_{prune} blocks and every epoch boundary	All
State versions	Last W_{state} and epoch boundaries	All
Evidence	Until expiry	All

XV. Storage Architecture

A. Storage engine

a) Decision S1: RocksDB behind a narrow key–value interface: The state tree writes new nodes under version-prefixed keys and rarely updates in place, which suits a log-structured merge tree. RocksDB is the most widely deployed LSM engine in blockchain nodes and is used, directly or through wrappers, by Sui, Polkadot SDK, and parts of reth. Its costs are a C++ dependency and write amplification during compaction. We considered MDBX, a memory-mapped B+-tree used as reth's main database, which offers excellent read latency but allows a single writer and handles a write-heavy versioned tree less naturally. Two pure-Rust engines with stable on-disk formats, redb and fjall, avoid the C++ toolchain but have shorter production histories. We rejected sled because it has had no stable release since 2021. All storage access goes through a small trait (Appendix B), so the engine can be replaced without touching consensus or execution code, and light or embedded nodes can use a pure-Rust engine.

B. Data layout

The node stores data in separate column families: headers by height and by hash, transaction bodies, witnesses, receipts, commit certificates, evidence, state-tree nodes, a stale-node index for pruning, a transaction index ($txid \mapsto$ height and position), and node metadata. Witnesses and commit certificates sit in their own column families, keyed by height, so that pruning is a range deletion. An optional per-account transaction index for explorers and wallets is disabled by default, because it grows with history and is not needed for validation.

C. Retention and pruning

Each class of data has its own retention rule (Table XI). Consensus never needs data older than the evidence window, apart from the epoch-boundary data that light clients and syncing nodes use to cross validator-set changes.

Pruning witnesses is safe for consensus because no validity rule refers to a witness after its block is committed: the block's

validity was decided when more than two thirds of voting power prevoted for it, and every honest validator in that set verified every witness. It does change what a newcomer can check for itself. A node that syncs from a checkpoint trusts that the history before the checkpoint was validated, which it already does under weak subjectivity. A node that wants to re-verify every signature since genesis must obtain witnesses from an archive node and check them against the witness roots in the headers. The witness roots guarantee that it cannot be given altered witnesses, but nothing in the protocol guarantees that some archive node will keep serving them. Section XXIII lists incentives for archival storage as an open problem.

D. Snapshots and state synchronization

At configured epoch boundaries, nodes produce a state snapshot divided into chunks, each covering a contiguous range of state keys and accompanied by Merkle range proofs against the state root of that epoch-boundary header. A syncing node can verify each chunk independently as it arrives and reject a bad chunk from one peer without discarding the rest, instead of discovering corruption only after rebuilding the whole tree. Once all chunks are verified and inserted, the node checks that the rebuilt root equals the header's state root.

E. Data availability

In a BFT chain, data availability follows from the voting rule: an honest validator prevotes only for a block it has fully received and validated, so a committed block is held by honest validators with more than one third of voting power. Under Assumption 1 committed data is available at commitment time. XENORRA does not use data-availability sampling. Sampling is needed when light clients must protect themselves against a dishonest majority of block producers, which is outside our threat model for the first release.

F. Light clients

A light client stores a trusted header $\hat{H}$ (initially a checkpoint), the validator set $\hat{V}$ it commits to, and the time it was trusted. To verify a target header H_t it proceeds as follows.

1) If H_t is in the same epoch as $\hat{H}$, the client checks that the commit certificate for H_t carries valid signatures from members of $\hat{V}$ with more than two thirds of $\hat{V}$'s power.
2) Otherwise it walks epoch-boundary headers. Each boundary header is verified against the current trusted set as in step 1, and its `next_validators` hash commits to the following set, which the client then adopts.
3) As an optimization, the client may skip epochs. It accepts a later header directly if signers holding more than one third of the *trusted* set's power signed it and the trusted header is within the trusting period. If any honest validator from the trusted set signed, the header is genuine; the trusting period ensures that those validators are still slashable if they lied. If the overlap is insufficient, the client bisects.

Each epoch crossing costs about q ML-DSA verifications and $q \cdot s_\sigma$ bytes of signatures, roughly 220 KB at $n = 100$. With epochs of about one day this is negligible for a phone that synchronizes occasionally, whereas checking a full certificate for every block would cost about 9.6 GB per day at a 2-second block interval (Section XX). This asymmetry is why validator sets change only at epoch boundaries. Once the header is trusted, state queries are answered with sparse-Merkle proofs against its state root.

XVI. Economic and Incentive Model

The economic layer has to fund validator security, price scarce resources, and give stakeholders a say in protocol change. We specify mechanisms and the relationships between parameters. We do not fix supply figures, inflation rates, or reward percentages: those depend on market conditions and on measurements that do not yet exist, and choosing them now would be guesswork presented as design. Each parameter below is listed with the evidence needed to set it (Table XII).

A. Token utility

The native token is used for: bonded stake that secures consensus and is slashable; fees for transactions and verification requests; refundable storage deposits for registered keys, DID documents, and credential anchors; verifier collateral in the identity market; and voting in governance. Each use creates demand linked to actual network activity. None depends on the token being used as a speculative asset, although we cannot prevent that.

B. Security budget

Under correlated slashing (Equation (11)) with $\kappa = 3$, an attack that breaks safety requires validators with at least one third of stake to sign provably conflicting messages, and costs them their entire bonded stake. If S_b tokens are bonded at price P, the minimum cost of a safety attack is

$$C_{\text{safety}} \geq \tfrac{1}{3}\, S_b\, P. \tag{18}$$

Liveness is cheaper to attack: withholding votes by more than one third of stake halts the chain without signing anything provably wrong, and costs only downtime penalties and forgone rewards. Equation (18) gives a lower bound only. It ignores bribery markets, stake held by exchanges on behalf of users, and the possibility that an attacker profits from a crash in token price, all of which change the calculus. The equation does show that security scales with the value bonded, which depends on the staking yield the protocol offers.

C. Issuance

a) Decision M1: issuance as a function of the staking ratio: Let $s = S_b/S$ be the fraction of supply bonded. Annual issuance, as a fraction of supply, follows a curve that rises to a target ratio s^* and declines beyond it:

$$I(s) = \begin{cases} I_{\min} + \Delta I\, \dfrac{s}{s^*}, & s \leq s^*, \\ I_{\min} + \Delta I\, 2^{-(s-s^*)/d}, & s > s^*, \end{cases} \tag{19}$$

with $\Delta I = I_{\max} - I_{\min}$, so that the staking yield is $y(s) = I(s)/s$. Below the target, the yield is high, which attracts stake. Above it, issuance falls off, which discourages locking more supply than security needs. Polkadot's staking design uses a curve of this general shape [39], and it has the useful property that the equilibrium staking ratio responds to the parameters in a predictable way. The values of $I_{\min}$, $I_{\max}$, s^*, and d must come from modelling: s^* from the bonded value required for an acceptable C_{safety}, and $I_{\max}$ from the yield needed to compensate validators' operating costs and capital costs at that ratio. Issuance is expected to decline over time as fees grow, through a governance-controlled schedule for $I_{\max}$ with a published lower bound.

D. Transaction fees

a) Decision M2: a weight-based base fee with burning: Each block has a target weight W^* and a maximum of $2W^*$. The base fee per weight unit adjusts after each block in the manner of EIP-1559 [53]:

$$b_{h+1} = b_h \left(1 + \frac{1}{8} \cdot \frac{W_h - W^*}{W^*}\right), \tag{20}$$

where W_h is the weight used by block h. Transactions pay $weight \cdot (b_h + tip)$ up to their declared maximum. The base-fee portion is burned (or, for a governance-set fraction ϕ, paid to the treasury), and the tip goes to the proposer. Roughgarden's analysis shows that this mechanism makes bidding the base fee plus a small tip a good strategy for users when blocks are not full, and that burning the base fee removes the proposer's incentive to manipulate it [54]. Burning links supply to usage, partly offsetting issuance.

A single weight dimension under-prices some resources and over-prices others. Bytes, signature verifications, and state writes have different marginal costs, and multidimensional fee markets price them separately. We start with one dimension, calibrated by Equation (17), because it is simpler to implement and reason about, and we treat a move to separate byte and compute dimensions as a likely upgrade once testnet data shows how far the single price diverges from true costs.

E. Validator rewards and costs

A validator i with stake share π_i earns, per year,

$$R_i = \pi_i\big(I(s)\,S + \bar{F}_{\text{tip}}\big) - L_i, \tag{21}$$

where $\bar{F}_{\text{tip}}$ is total tips and L_i its penalties, and it keeps a commission on the share attributable to delegators. Running a validator costs hardware, bandwidth, and operations, roughly independent of stake. Post-quantum signatures push the bandwidth term up: at $n = 100$ and 2-second blocks, vote traffic alone is about 2.7 Mbit/s before gossip redundancy (Section XX). The smallest stake at which a validator breaks even rises with this fixed cost, and that in turn limits how many validators the network can sustain. We treat $N_{\max}$, the maximum active set size, as an economic as well as a technical parameter, set from measured operating costs.

F. Storage deposits

New state requires a deposit of δ_{byte} tokens per byte, refunded when the state is deleted. Deposits make state growth cost something for as long as the state exists. The deposit rate should be set so that the cost of holding the projected state on a validator's storage is covered by the opportunity cost of the deposited tokens.

G. Verifier market

Verification fees are paid by the requester and split between the verifiers whose results were accepted, the validators, and the treasury. Verifiers post collateral that caps the value of tasks they may take and is slashed for results overturned on challenge. The split, collateral ratios, and challenge bonds are application parameters, set separately from the consensus parameters above and initially conservative.

H. Governance

On-chain governance controls parameters, suite lifecycle changes, treasury spending, and protocol upgrades. Proposals require a deposit. Voting power is bonded stake, and delegators inherit their validator's vote unless they vote themselves. A proposal passes with a quorum of participation and a supermajority of votes cast, then waits through a time lock before taking effect, so that participants who disagree can exit. Every parameter has hard bounds written into the protocol, which governance cannot exceed without an upgrade. This limits the damage from a captured or careless vote.

Two paths are faster. A protocol upgrade is coordinated by a version field and an activation height, and nodes that have not upgraded halt at that height instead of continuing on an incompatible rule set. An emergency suite freeze (Section IX-H) can be activated by validators holding more than two thirds of power signing a freeze message, with a short delay, because waiting out a normal vote while a signature scheme is being actively broken would be worse.

Stake-weighted governance has known weaknesses: low turnout, concentration of power in large holders and exchanges, and vote buying. Hard parameter bounds, time locks, and delegator override reduce these risks but do not remove them, and we list governance design as an area that needs more work (Section XXIII).

XVII. Formal Security Model

This section states the security properties XENORRA aims for, the assumptions under which each holds, and the kind of evidence behind it. Proofs and proof sketches are in Appendix C. We are explicit about the status of each claim, because "secure" covers very different things: some properties follow from counting arguments once assumptions are granted, some rest on the conjectured hardness of a computational problem, and some can only be supported by measurement.

TABLE XII
ECONOMIC PARAMETERS. NO VALUES ARE PROPOSED WHERE THE EVIDENCE TO SET THEM DOES NOT YET EXIST; THE RIGHT-HAND COLUMN STATES WHAT IS REQUIRED.

Parameter	Role	Evidence required to set it
S_0 (genesis supply)	Unit of account; denominator for ratios	Arbitrary in itself; distribution and vesting require legal review (Whitepaper v2.0 §10.6)
$I_{\min}, I_{\max}, s^*, d$	Issuance curve, Eq. (19)	Target C_{safety}; validator cost survey from testnet; sensitivity analysis of equilibrium staking ratio
W^* (target weight)	Block capacity	Benchmarks: weight at which the reference validator finishes validation within the propose timeout with margin
α coefficients	Weight formula, Eq. (17)	Micro-benchmarks of each resource on reference hardware (Section XXI)
ϕ (treasury share of base fee)	Funds audits and development	Treasury budget model; governance choice
δ_{byte} (storage deposit)	Prices permanent state	Cost of validator storage per byte-year; target state size
κ, downtime penalty, jail period	Slashing, Eq. (11)	$\kappa = 3$ follows from the one-third safety threshold; others from testnet downtime distribution
$N_{\max}$, $c_{\max}$	Active set size; per-operator cap	Bandwidth benchmarks at increasing n; validator break-even analysis; concentration targets
U, T_{trust}	Unbonding and trusting periods	Constraint $T_{\text{trust}} < U$ with margin for evidence propagation; user tolerance for lock-up
Governance quorum, threshold, time lock	Upgrade and parameter control	Turnout data from testnet governance; comparable networks

A. Parties and adversary

The system consists of validators, full nodes, light clients, and account holders. The adversary is a quantum polynomial-time algorithm that, beyond the capabilities of the class it belongs to ($\mathcal{A}_0$–$\mathcal{A}_3$, Section IV):

- controls message scheduling subject to Assumption 2: it can delay, reorder, and duplicate messages, but after GST must deliver messages between honest validators within Δ;
- can create arbitrarily many network identities and full nodes;
- corrupts validators adaptively, subject to Assumption 1 in every epoch, and learns their keys and state on corruption;
- can submit arbitrary transactions and observe all public data, including the mempool.

We do not model side channels on honest machines, compromise of a majority of software supply chains, or failure of the hash function; Section XVIII treats the first two as engineering risks.

B. Cryptographic assumptions

Assumption 4 (Hash)**.** SHA3-256, with the domain separation of Equation (3), is collision resistant and second-preimage resistant against quantum polynomial-time adversaries, and may be modelled as a quantum-accessible random oracle where proofs require it.

Assumption 5 (Recovery signatures)**.** SLH-DSA-SHAKE-128s is existentially unforgeable under chosen-message attack against quantum polynomial-time adversaries.

Assumption 3 (ML-DSA unforgeability) reduces, in the quantum random-oracle model and non-tightly, to the hardness of MLWE and SelfTargetMSIS [26], [30]. Assumption 5 reduces to properties of the hash function alone, which is why it is the more conservative of the two. ML-KEM's IND-CCA security protects transport confidentiality only. No safety or liveness property of XENORRA depends on it.

C. Properties

Definition 1 (Agreement)**.** No two honest nodes commit different blocks at the same height.

Definition 2 (Validity)**.** Every committed block satisfies $valid$ with respect to its parent.

Definition 3 (Liveness)**.** After GST, every transaction that remains valid and is held by honest validators is eventually committed, provided its fee exceeds the base fee.

Definition 4 (Accountable safety)**.** If Agreement is violated, then from the messages observed by honest nodes one can efficiently compute a set of validators F with $w(F) \geq W/3$ (in the epoch concerned), together with a proof, checkable by anyone holding the validator set, that every member of F violated the protocol.

Definition 5 (Authorization)**.** No state transition debits or modifies an account unless the account's current key holder signed the corresponding transaction (or, for recovery, the recovery key holder signed and the delay elapsed).

D. Results

Theorem 1 (Agreement)**.** *Under Assumptions 1 and 3,* XENORRA *satisfies Agreement, except with probability negligible in the security parameter.*

Theorem 2 (Accountable safety)**.** *Under Assumption 3 alone (without Assumption 1), and provided honest nodes retain the consensus messages they receive within the evidence window,* XENORRA *with modification (M2) satisfies Accountable safety, except with negligible probability. In one case (a fresh proposal committed in a later round with no surviving prevote certificate), culpability rests on an accusation that every honest validator could refute and the guilty cannot, instead of on self-contained evidence (Appendix C).*

Theorem 3 (Liveness)**.** *Under Assumptions 1, 2, and 3, with timeouts that grow without bound in the round number, every height is eventually committed after GST, and a valid*

transaction held by all honest validators is committed within a bounded number of heights after GST once an honest proposer includes it.

Theorem 4 (Authorization)**.** *Under Assumptions 3, 4, and 5, Authorization holds for every committed block, except with negligible probability.*

Lemma 1 (No replay)**.** *A signed transaction changes state at most once on a given chain and never on a chain with a different identifier.*

Proposition 1 (Supply conservation)**.** *Every native call and every block-level transition preserves Equation* (15)*.*

Lemma 2 (Recoverability under a frozen primary suite)**.** *If an account has a registered recovery key under a suite that is not frozen, and the account's primary suite is frozen, then the recovery key holder can install a new key record within D_{rec} blocks plus the inclusion delay, and no party without the recovery key can prevent or redirect this, except with negligible probability under Assumption 5.*

E. Status of each claim

Table XIII classifies each property by the kind of evidence that supports it. The distinction guides audit planning. Mathematically proven properties need their proofs checked and their implementations tested against the specification; reduction-based properties need parameter choices reviewed against current cryptanalysis; empirical properties need the experiments of Section XXI.

F. Failure conditions

Operators will need to know what happens when each assumption fails.

Fault threshold exceeded ($w(B) \geq W/3$). Agreement may fail. Theorem 2 guarantees that culprits holding at least one third of stake are identified, nodes halt on detecting conflicting commits, and recovery is social (Section XI-M). If $w(B) \geq 2W/3$ the adversary can commit arbitrary valid-looking blocks, and only weak-subjectivity checkpoints held by users protect them.

Partial synchrony violated for long periods. Safety holds; the chain stops committing until the network stabilizes.

ML-DSA broken (adversary $\mathcal{A}_3$ with a lattice advance). Forged votes become possible, so Agreement and Authorization fail for anyone using ML-DSA keys. The response is an emergency freeze (Section IX-H): validators that still control their nodes halt on the freeze, accounts recover via SLH-DSA keys, and the network migrates validators to a replacement suite. Blocks committed before the break remain trustworthy for nodes that hold checkpoints from that period. A newcomer who must decide between histories after the break needs an out-of-band checkpoint. No signature-based system can avoid this dependency.

SHA-3 broken. Every commitment, address, and the recovery scheme fail together. XENORRA makes no claims in this case.

XVIII. Threat Analysis

Tables XIV and XV enumerate the threats listed in the project brief, together with others that emerged during design, and give for each the attack, the mitigation, and the residual risk. This section discusses the entries that need more than a table row.

A. Quantum-enabled adversaries

Against $\mathcal{A}_1$ and $\mathcal{A}_2$, every signature in XENORRA is ML-DSA or SLH-DSA, so Shor's algorithm has nothing to attack. Grover's algorithm reduces the margin on SHA3-256 preimages to about 128 bits, which is adequate (Section IV). Mempool interception, which threatens Bitcoin and Ethereum users once their keys are exposed, does not arise, because exposing an ML-DSA public key gives a quantum adversary no advantage we know of. The residual quantum risk is $\mathcal{A}_3$: a quantum (or classical) algorithm that significantly weakens Module-LWE or Module-SIS. The response is the emergency freeze and SLH-DSA recovery described in Sections IX-H and XVII. Two of the original NIST finalists fell to classical attacks [28], [29], and we treat this residual risk as real.

B. Validator collusion and Byzantine behaviour

A coalition below one third of stake cannot break safety or liveness. Between one third and two thirds, it can halt the chain by withholding votes, and it can break safety only by signing conflicting messages, which correlated slashing makes maximally expensive and Theorem 2 makes attributable. Above two thirds, it controls the chain. This holds for every BFT protocol. The protocol-level defence is to make one third of stake expensive to acquire (the issuance curve), hard to concentrate (the per-operator cap), and costly to misuse (correlated slashing). The non-protocol defences are validator diversity requirements during the launch phase and public monitoring of stake concentration.

The predictable proposer schedule allows a targeted denial-of-service attack on the next proposer. Such an attack costs one round per targeted proposer. Sustaining it requires taking down proposers continuously, against sentry architectures that hide their addresses. We judge this acceptable compared with the alternative of a biasable random schedule, and we propose to measure it in adversarial testnet runs (Section XXI).

C. Sybil and eclipse attacks

Stake provides Sybil resistance for consensus. Network identities are free, and an open network cannot prevent one party from presenting many of them, so an attacker can run many nodes and try to surround a victim. XENORRA adopts the countermeasures shown to work for Bitcoin [25]: bucketed address tables with per-prefix limits, outbound connections chosen by the node, anchor connections, and feeler probes. On top of these it adds GossipSub peer scoring [49] and post-quantum authentication of peers. Light clients add a further safeguard by cross-checking headers with several independent full nodes; any conflicting signed header they receive is itself slashable evidence. None of these measures is provably sufficient, and eclipse resistance is listed as an empirical property in Table XIII.

TABLE XIII
STATUS OF SECURITY AND PERFORMANCE CLAIMS.

Claim	Status	Basis and what would invalidate it
Quorum intersection ($> 2/3$ quorums share honest power)	Proven (combinatorial)	Arithmetic on weights given Assumption 1; invalidated only if the threshold assumption fails
Agreement, Liveness (Theorems 1, 3)	Proven for Tendermint [9]; argued here for (M1)–(M5)	Locking argument plus unforgeability; invalidated by a flaw in our modifications or an implementation divergence from the specification
Accountable safety (Theorem 2)	Proof sketch	Relies on mandatory POL certificates (M2); needs independent review and model checking
Authorization, No replay	Reduction to EUF-CMA and hash properties	Invalidated by a practical attack on MLWE/MSIS (ML-DSA) or on SHA3
Recoverability under suite freeze	Reduction to SLH-DSA EUF-CMA	Invalidated by an attack on SHA3/SHAKE
Supply conservation	Proven per call (App. C)	Must be re-proven for every new call type; implementation checked by property tests
ML-DSA-65 achieves NIST category 3	Assumption, current cryptanalysis	Advances in lattice reduction or quantum algorithms for lattices
Eclipse and DoS resistance	Empirical	Peer-selection rules and rate limits must be tested adversarially; no proof
Throughput, latency, bandwidth	Analytical estimates only	Section XX; to be measured (Section XXI)
Beacon bias bound (2^m outcomes for m withheld reveals)	Proven (counting)	Bias exists by construction; acceptable only for uses that tolerate it
Economic security, Eq. (18)	Lower-bound argument	Ignores bribery, external hedging, custodial stake; requires market modelling

TABLE XIV
THREAT MATRIX (PART 1): CRYPTOGRAPHIC AND CONSENSUS THREATS. ASSURANCE: P = PROVEN UNDER STATED ASSUMPTIONS, R = REDUCTION TO A HARDNESS ASSUMPTION, E = EMPIRICAL, REQUIRES TESTING.

Threat	Attack	Mitigation	Residual risk	Assur.
Quantum key recovery ($\mathcal{A}_1$, $\mathcal{A}_2$)	Shor's algorithm on exposed public keys	No elliptic-curve or pairing primitive in any signed role; ML-DSA-65 everywhere	None known	R
Lattice cryptanalysis ($\mathcal{A}_3$)	Practical attack on MLWE/MSIS	Suite agility; emergency freeze; SLH-DSA recovery; migration plan	Stuck accounts without recovery keys; checkpoint dependence for newcomers	R
Classical cryptanalysis	Weakness in SHA3, ML-DSA parameters	Standard primitives; category-3 margin; versioned hash fields	SHA3 break is unmitigated	R
Validator collusion ($< 1/3$)	Equivocation, censorship	Quorum intersection; tips and rotation limit censorship to single rounds	Temporary censorship by consecutive faulty proposers	P
Validator collusion ($\geq 1/3$)	Halt or conflicting commits	Correlated slashing; accountable safety; halt on conflict	Chain halt; social recovery	P
Long-range attack	Re-sign history with old keys	Weak subjectivity, $T_{\text{trust}} < U$; PQ keys from genesis	Users without checkpoints	P
Beacon bias	Withhold reveals	Reveal penalties; use only where bias tolerable	Up to 2^m choices for m withholders	P
Proposer DoS	Attack predictable next proposer	Sentries; short timeouts; one round lost per success	Latency under sustained attack	E
Double spend	Conflicting txs in two histories	Agreement + nonces	As for collusion $\geq 1/3$	P
Replay	Re-submit signed tx	Nonce, chain id, context strings, expiry	None	P
Malleability	Alter witness to change id	$txid$ excludes witness; strict decoding	None	P

D. Long-range attacks

An attacker who obtains the keys of a past validator set, by buying them from former validators who have since withdrawn, can build an alternative history from that point. Weak subjectivity defeats this: nodes and light clients refuse to start from a checkpoint older than the trusting period, which is shorter than the unbonding period, so every validator set a node trusts is still slashable [24]. Our contribution here is negative but important: by using post-quantum consensus keys from genesis, XENORRA prevents a CRQC from turning every past validator key into a free key. In a chain whose early validators signed with elliptic curves, the attacker would not need to buy any keys.

E. Replay, double spending, and malleability

Lemma 1 covers replay. Double spending in an account model reduces to Agreement (no two committed histories) and nonce ordering (no two transactions with the same nonce in one history). Transaction identifier malleability is removed by excluding the witness from $txid$ (Section XII-G). One subtle case deserves mention: because ML-DSA signatures are randomized, a relayer can replace a valid witness with another valid witness for the same body only if it holds the private key. Without the key, it cannot produce a second valid signature, except by breaking strong unforgeability, which FIPS 204 targets. Even if it could, $txid$ and the state effects would be unchanged.

TABLE XV
THREAT MATRIX (PART 2): NETWORK, OPERATIONAL, AND IMPLEMENTATION THREATS.

Threat	Attack	Mitigation	Residual risk	Assur.
Sybil (network)	Many identities to dominate peer tables	Per-prefix and per-AS limits; outbound choice; peer scoring	Well-resourced AS-level attacker	E
Eclipse	Isolate a node or validator	Anchors, feelers, sentries, validator overlay, ML-DSA peer auth	Untested at scale	E
Network partition	Split validators	Safety preserved; round catch-up on heal	Halt during partition	P
Verification flood	Invalid signatures en masse	Ordered admission; per-peer budgets; verify-before-relay	CPU spikes within budget	E
Bandwidth flood	Large or duplicate messages	Size caps; rate limits; announce-then-pull	Link saturation by botnets	E
State bloat	Create many accounts/records	Storage deposits; existential deposit	Deposit level mis-set	E
Account key compromise	Theft	Rotation; recovery with delay; separate recovery mnemonic	Loss if both keys stolen	–
Consensus key compromise	Forced equivocation	Remote signer with double-sign guard; epoch rotation	Slashing of victim	–
Supply chain	Malicious dependency	`cargo-vet`, `cargo-deny`, pinned versions, reproducible builds	Compromise of a vetted maintainer	E
Implementation divergence	Libraries disagree on validity	Single pinned verifier; differential fuzzing; negative vectors	Undiscovered edge cases	E
Side channels	Timing or power leakage in signing	Constant-time implementations; HSMs for operator keys	Platform-specific leaks	E
Governance capture	Malicious parameter change	Hard bounds; time locks; delegator override	Slow-moving capture	–
Proposer ordering (MEV)	Reorder or insert transactions for profit	Out of scope for v1 native modules (no on-chain exchange)	Grows if contracts added	–

F. Denial of service

The main denial-of-service risks are verification floods (making nodes verify many invalid 3.3 KB signatures), bandwidth floods, and state bloat. The ordered admission pipeline (Section X) puts cheap checks before signature verification, per-peer verification budgets cap the work any one peer can impose, and the rule that only verified messages are relayed stops amplification. State bloat is priced by storage deposits. A residual risk is an attacker who pays valid fees to fill blocks; this is a pricing question, and the base fee rises exponentially under sustained demand (Equation (20)).

G. Key compromise

A stolen account key allows theft until the owner rotates it. The recovery delay D_{rec} is designed for the opposite case, a stolen recovery key with an intact primary key, in which the owner cancels the recovery. If both keys are stolen, as when they share a mnemonic, the account is lost; this is why wallets should offer a separate recovery mnemonic. A stolen validator consensus key allows the thief to equivocate and get the validator slashed. Remote signers with persistent double-sign protection and consensus-key rotation at epoch boundaries limit the exposure.

H. Supply-chain and implementation risk

We expect implementation defects to be more likely than cryptanalytic breaks of ML-DSA. The Rust post-quantum ecosystem is young: in 2026 alone, ML-DSA libraries received advisories for accepting invalid signatures and for timing leaks, and the widely used `pqcrypto` family was declared unmaintained after the upstream PQClean project was archived [36]. A consensus-critical verifier that accepts an invalid signature, or rejects a valid one that another implementation accepts, can split the network. Section XIX sets out the mitigations: a single pinned primary implementation in consensus, differential fuzzing against a second implementation, NIST and negative test vectors, dependency vetting with `cargo-vet`, and independent audits before mainnet.

XIX. RUST IMPLEMENTATION ARCHITECTURE

Rust's ownership model removes whole classes of memory-safety bugs that have affected node software written in C and C++, and its type system lets protocol invariants be expressed as types. Neither property makes an implementation correct. This section describes how the workspace is structured so that the specification maps cleanly onto code, which dependencies we recommend and why, and how the implementation will be tested.

A. Principles

1) *Deterministic cores, asynchronous shell.* The consensus state machine and the state-transition function are pure, synchronous libraries that perform no I/O and read no clock. They consume events and produce actions. Networking, storage, timers, and the signer live in an asynchronous shell built on Tokio. This "sans-I/O" structure lets the cores run unchanged under deterministic simulation.
2) *One canonical encoding.* Every byte string that is hashed or signed is produced by one encoding module, tested against the vectors in Appendix A. No other crate serializes consensus objects.
3) *Determinism enforced by tooling.* Consensus crates use `#![forbid(unsafe_code)]`, deny floating-point arithmetic via Clippy lints, use ordered collections for anything iterated, and have no access to system time.
4) *Algorithms behind the suite table.* No crate outside `xenorra-crypto` names a signature algorithm. Ev-

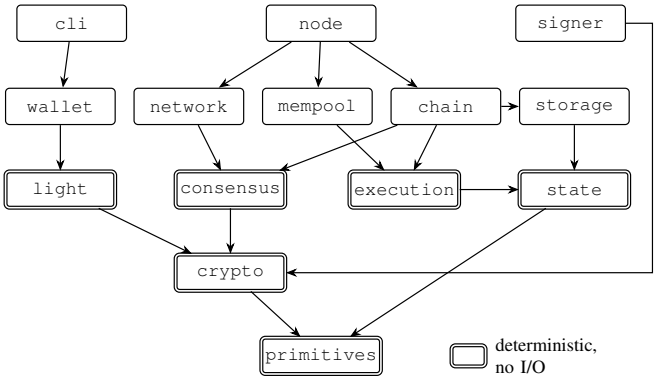


Fig. 3. Crate dependencies (arrows point to the crate depended upon; prefix `xenorra-` omitted). Double-bordered crates are deterministic and perform no I/O, so they compile unchanged into the deterministic simulator and the light client. Every crate also depends on `primitives`; only the direct edges that matter for layering are drawn.

erything else calls a single `verify(suite, pk, msg, ctx, sig)` interface.

B. Workspace

The project brief proposed eleven crates. After designing the protocol we made four changes. We merged `xenorra-core` into a `xenorra-primitives` crate that holds types, encoding, and hashing, because these must never diverge. We split the state tree out of storage, since the tree's hashing rules are consensus-critical and the storage engine is not. We renamed `xenorra-ledger` to `xenorra-chain`, which validates and applies blocks. And we added `xenorra-mempool`, `xenorra-light`, `xenorra-sim`, and `xenorra-vectors`. Table XVI lists the crates and Fig. 3 shows their dependencies. Appendix B gives the central types and traits.

C. Dependencies

Table XVII summarizes the dependency assessment. The survey behind it (November 2026) checked crates.io, the RustSec advisory database [36], repository activity, and published audits. Three findings drove the choices.

No pure-Rust post-quantum signature library is both stable and independently audited. RustCrypto's `ml-dsa` [55] reached 0.1.0 only in May 2026 and states that it has never been independently audited. It had a timing side-channel fixed in its release-candidate series. The `fips204`/`fips205` crates have stable APIs and self-reported constant-time testing but no third-party audit. Cryspen's `libcrux` applies formal verification to arithmetic, but its ML-DSA crate is pre-0.1 and received advisories in 2026 for accepting invalid signatures on paths the proofs did not cover. `aws-lc-rs` [56] has offered ML-DSA in its stable API since version 1.18.0 (August 2026), built on an optimized C and assembly implementation, and its FIPS 140-3 certificate covers ML-KEM but not yet ML-DSA.

Some popular options are unsuitable. The `pqcrypto` crates were declared unmaintained in RustSec after PQClean was archived. liboqs states that it is not recommended for production use. `bincode` is unmaintained, and Protocol Buffers encodings are not canonical, so neither may be used for bytes that are hashed or signed. Borsh has a written specification and canonical encoding; postcard has a documented stable format and is suitable for non-consensus messages.

Existing Rust blockchain frameworks offer no post-quantum signatures. Polkadot SDK supports Ed25519, sr25519, ECDSA, and experimental BLS; Commonware's cryptography crate offers Ed25519, BLS12-381, and secp256r1; reth targets Ethereum's secp256k1. Building on any of them would mean replacing their signature layers, which is most of what makes this design different. We reuse their engineering lessons (reth's storage layout, for example) without their code.

rust-libp2p supports no post-quantum peer identities. Since 2026 its TLS and QUIC transports negotiate the hybrid ML-KEM key exchange through rustls with the aws-lc-rs provider, but peer identifiers remain Ed25519, secp256k1, ECDSA, or RSA [48]. This confirms the need for the XenAuth layer of Section X. rustls must be at version 0.23.46 or later, which fixes a downgrade from hybrid to classical key exchange.

a) Decision R1: one pinned verifier per suite, plus differential testing: Consensus nodes verify ML-DSA with `aws-lc-rs` as the primary implementation. It is the most optimized option, it is maintained by a well-resourced team, and it sits on the path to FIPS validation. RustCrypto `ml-dsa` and `fips204` serve as independent implementations for differential fuzzing: every input the fuzzer generates is checked by all three, and any disagreement fails the build. For SLH-DSA, used only for recovery, `fips205` is primary and RustCrypto `slh-dsa` is the differential reference. SHA-3, SHAKE, and KMAC come from RustCrypto's `sha3` crate, cross-checked against `aws-lc-rs`. The conformance suite (Appendix A-I) includes NIST vectors and negative cases: wrong lengths, malformed hint encodings, repeated hint indices, out-of-range coefficients, and wrong context strings. Any independent implementation of XENORRA, in any language, must pass it. Funding an independent audit of the chosen ML-DSA and SLH-DSA verifiers is a Phase 7 deliverable (Section XXIV).

b) Decision R2: write the consensus engine, use Malachite as a reference: Malachite, a Rust implementation of Tendermint maintained by Circle and accompanied by a Quint specification [50], is the closest existing component. It is Apache-2.0 licensed but describes itself as alpha software that has not been externally audited. CometBFT [42] is mature but written in Go. We specify `xenorra-consensus` as our own sans-I/O state machine implementing Algorithm 4 with modifications (M1)–(M5), use Malachite's specification and test traces as an oracle for model-based testing, and revisit adopting Malachite directly at the Phase 5 gate, if its maturity and audit status have changed by then.

D. Validator signer

The signer is a separate process, optionally on separate hardware, that holds the consensus key. Before signing any proposal or vote it durably records the highest $(h, r, step)$ it has signed, with fsync, and it refuses to sign a conflicting message at or below that mark. It speaks a small authenticated protocol to the node. Keeping this logic separate means that a

TABLE XVI
PROPOSED RUST WORKSPACE. "DET." MARKS CRATES THAT MUST BE DETERMINISTIC AND FREE OF I/O.

Crate	Det.	Responsibility	Main external dependencies
`xenorra-primitives`	✓	Core types (hashes, addresses, amounts, headers, transactions, votes), canonical encoding, domain-separated hashing, Merkle lists	`sha3`
`xenorra-crypto`	✓	Suite table, signature and verification interface, ML-DSA and SLH-DSA backends, KMAC key derivation, signature cache	`aws-lc-rs`, `fips205`; `ml-dsa`, `fips204` (test only)
`xenorra-state`	✓	Sparse Merkle tree, proofs, versioned node keys	–
`xenorra-execution`	✓	State-transition function and native modules; weight accounting	–
`xenorra-consensus`	✓	Tendermint state machine, proposer selection, vote sets, commit certificates, evidence	–
`xenorra-storage`		Key–value trait, RocksDB backend, block and witness stores, pruning, snapshots	`rocksdb`
`xenorra-chain`		Block validation and application pipeline, sync orchestration	`rayon`
`xenorra-mempool`		Admission pipeline, prioritization, replacement rules	–
`xenorra-network`		libp2p integration, XenAuth, gossip topics, compact relay, rate limiting	`libp2p`, `quinn`, `rustls`
`xenorra-signer`		Remote validator signer with double-sign protection	`tokio`
`xenorra-light`	✓	Light-client verification of headers, epochs, and state proofs	–
`xenorra-wallet`		Mnemonic and HD derivation, transaction building and signing	`zeroize`
`xenorra-node`		Binary: wiring, configuration, RPC, metrics	`tokio`
`xenorra-cli`		Command-line client for users and operators	–
`xenorra-sim`		Deterministic multi-node simulation harness	`turmoil`
`xenorra-vectors`		Normative test vectors and the conformance runner	–

bug in the node, or a misconfigured failover that brings up two nodes for one validator, cannot make the validator equivocate, the most common cause of accidental slashing in deployed proof-of-stake networks.

E. Testing and verification strategy

Unit and property tests proptest checks codec round-trips, supply conservation (Equation (15)) over random transaction sequences, and Merkle proof soundness.

Fuzzing cargo-fuzz targets every decoder, the signature verifiers (differentially), the admission pipeline, and the state-transition function.

Differential execution Parallel execution is compared against sequential execution on recorded and generated blocks.

Deterministic simulation The consensus and chain crates run inside turmoil with simulated time, partitions, message loss, and Byzantine nodes, using seeded runs that can be replayed exactly.

Concurrency loom explores interleavings in the mempool, signature cache, and block pipeline.

Model checking Kani proves absence of panics and overflow in the codec and arithmetic helpers. A TLA+ or Quint model of the consensus modifications is checked for agreement and accountability, and traces from the model drive tests of the Rust state machine.

Benchmarks criterion for micro-benchmarks; the scenario harness of Section XXI for system tests.

Supply chain cargo-vet audits, cargo-deny licence and advisory checks, pinned lockfiles, and reproducible release builds.

XX. PERFORMANCE ANALYSIS

Every number in this section is an analytical estimate derived from encoding sizes and published cycle counts. None is a measurement of XENORRA software, which does not yet exist. We give the formulas so that the estimates can be recomputed when measurements replace the assumptions.

A. Transaction size

Table XVIII gives the encoded size of a single-input transfer under different signature schemes. The body (134 bytes) is the same in all cases. Under ML-DSA-65 the witness is 96% of the transaction with a registered key and 97.5% with a hashed key, and the transaction is about 17 times the size of its Ed25519 equivalent.

B. Throughput bounded by block size

With blocks of S_b bytes every Δ_b seconds and transactions of s_{tx} bytes, throughput cannot exceed

$$\lambda_{\max} = \frac{S_b}{s_{\mathrm{tx}}\,\Delta_b}. \tag{22}$$

Table XIX evaluates Equation (22) for registered-key ML-DSA-65 transfers. A 2 MiB block every 2 seconds gives about 300 transfers per second; 8 MiB every second gives about 2,400. These figures are upper bounds set by bytes alone. Whether a network of geographically distributed validators can propagate, validate, and vote on an 8 MiB block within one second is an empirical question, and we expect the answer to be no for $n = 100$ on commodity links. Our planning figure for the first testnet is the 2 MiB / 2 s configuration.

C. CPU

Using the AVX2 cycle count for the round-3 equivalent of ML-DSA-65 (179,424 cycles, Table II) at 3 GHz, one verification takes about 60 µs. A full 2 MiB block of 607 transfers needs about 36 ms of single-core verification time, or about 2.3 ms across 16 cores. Verifying the $2n$ votes of one height takes about 12 ms of single-core time at $n = 100$. Most transaction signatures are verified at admission and cached, so block validation repeats only the verifications for transactions a validator had not seen. Signing a proposal and two votes per height costs about 1.3 million cycles, under half a millisecond. CPU leaves ample headroom at the planned load, even if implementation overheads are several times the

TABLE XVII
DEPENDENCY ASSESSMENT (NOVEMBER 2026). VERSIONS ARE THE LATEST STABLE RELEASES AT THAT TIME; "AUDIT" REFERS TO INDEPENDENT THIRD-PARTY AUDITS FOUND DURING THE SURVEY.

Need	Crate (version)	Licence	Maintenance / advisories	Audit	Decision
ML-DSA (primary)	`aws-lc-rs` 1.19.0	ISC/Apache-2.0	Active; ML-DSA stable since 1.18.0	FIPS 140-3 cert. covers ML-KEM, not ML-DSA	**Use** in consensus
ML-DSA (reference)	`ml-dsa` 0.1.2; `fips204` 0.5.0	MIT/Apache-2.0	Active; `ml-dsa` had a fixed timing advisory	None	**Use** for differential testing
ML-DSA	`libcrux-ml-dsa` 0.0.11	Apache-2.0	Pre-0.1; four 2026 advisories	Formal proofs of arithmetic only	Revisit later
SLH-DSA	`fips205` 0.5.0; `slh-dsa` 0.2.0-rc.5	MIT/Apache-2.0	Active	None	**Use** `fips205`; RustCrypto as reference
PQ (various)	`pqcrypto`, `oqs`	MIT/Apache-2.0	Unmaintained / not for production	–	**Reject**
Hashing	`sha3` 0.12.0	MIT/Apache-2.0	Active	–	**Use**
Async runtime	`tokio` 1.54.1	MIT	Active; widely deployed	–	**Use**
P2P	`libp2p` 0.57.1	MIT	Active; no PQ identities	–	**Use** with XenAuth
QUIC / TLS	`quinn` 0.11.12; `rustls` 0.23.46	MIT/Apache-2.0/ISC	Active; several 2026 DoS advisories in `quinn-proto`	rustls: 2020 audit	**Use**; pin and track advisories
Storage	`rocksdb`	Apache-2.0	Slow upstream releases; active fork	–	**Use** behind trait
Storage (alt.)	`redb` 4.5.0; `fjall` 3.1.13	MIT/Apache-2.0	Active; stable formats	–	Light/embedded nodes
Storage	`sled`	MIT/Apache-2.0	No stable release since 2021	–	**Reject**
Encoding	`borsh` 1.8.1	MIT/Apache-2.0	Active; written specification	–	**Use** (spec subset)
Encoding	`bincode`; `prost`	MIT; Apache-2.0	Unmaintained; non-canonical	–	**Reject** for consensus bytes
Testing	`proptest`, `cargo-fuzz`, `turmoil`, `loom`, `kani`, `criterion`	MIT/Apache-2.0	Active	–	**Use**
Consensus reference	Malachite 0.8.2	Apache-2.0	Active; alpha	Not externally audited	Reference and test oracle

TABLE XVIII
ENCODED SIZE (BYTES) OF A SINGLE-INPUT TRANSFER. BODY: 134 BYTES. "REGISTERED": KEY STORED IN STATE; "HASHED": PUBLIC KEY CARRIED IN THE WITNESS.

Scheme	Registered	Hashed	Ratio to Ed25519
Ed25519 (reference)	205	241	1.0
ML-DSA-44	2,561	3,877	12.5
ML-DSA-65	**3,450**	**5,406**	**16.8**
ML-DSA-87	4,768	7,364	23.3
FN-DSA-512	807	1,708	3.9
SLH-DSA-128s	7,997	8,033	39.0

reference figures. Classical chains with heavy smart-contract execution show the opposite pattern, with execution as their usual bottleneck.

TABLE XIX
UPPER BOUND ON TRANSFERS PER SECOND FROM EQ. (22) (REGISTERED ML-DSA-65 KEY, 3,450-BYTE TRANSACTIONS). HASHED-KEY FIGURES ARE ABOUT 36% LOWER.

Block size	$\Delta_b = 1\,\mathrm{s}$	$\Delta_b = 2\,\mathrm{s}$	$\Delta_b = 4\,\mathrm{s}$
1 MiB	303	152	76
2 MiB	607	304	152
4 MiB	1,215	608	304
8 MiB	2,431	1,216	608

D. Bandwidth

A validator's sustained ingress at transaction rate λ is approximately

$$B_{\mathrm{in}} \approx (1+\epsilon)\,\lambda\, s_{\mathrm{tx}} + \frac{2\,n\,s_v}{\Delta_b} + \frac{|CC| + |prop|}{\Delta_b}, \tag{23}$$

where ϵ is the duplicate-delivery overhead of gossip (small with announce-then-pull), $|CC|$ the commit certificate carried

TABLE XX
ESTIMATED VALIDATOR INGRESS (MBIT/S), $n = 100$, $\Delta_b = 2\,\text{s}$, BEFORE GOSSIP DUPLICATION.

λ (tx/s)	Transactions	Votes	Certificates	Total
100	2.8	2.7	0.9	6.4
300	8.3	2.7	0.9	11.9
600	16.6	2.7	0.9	20.2
1,000	27.6	2.7	0.9	31.2

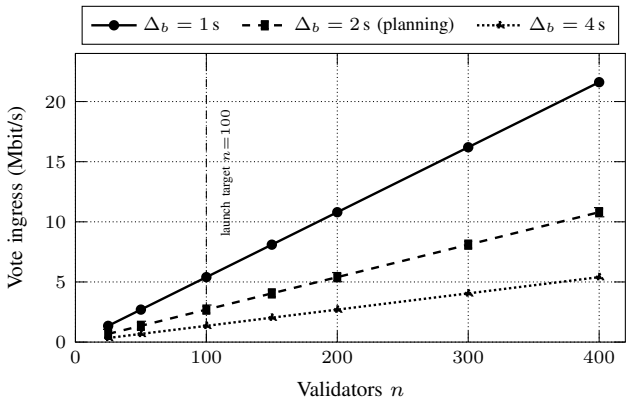


Fig. 4. Per-validator ingress from prevotes and precommits alone, $2ns_v/\Delta_b$ with $s_v = 3{,}376$ bytes (Eq. (8)). Analytical estimate; gossip duplication, transactions, and certificates are excluded (see Table XX for totals).

in each block, and $|prop|$ the compact proposal (a few kilobytes). Table XX evaluates the terms for $n = 100$ and $\Delta_b = 2\,\text{s}$ with $\epsilon = 0$. Egress depends on the node's position in the gossip mesh, and for a well-connected relay node can be several times its ingress. Fig. 4 shows how vote traffic alone scales with the validator count.

At the planning load of 300 transfers per second, ingress is about 12 Mbit/s before duplication, comfortably inside the 50 Mbit/s target of objective O5 even with a duplication factor of two to three. Vote traffic grows linearly in n per validator ($\Theta(n^2)$ across the network) and reaches about 11 Mbit/s at $n = 400$ with 2-second blocks. That remains feasible, but it explains why a few hundred validators is a practical ceiling for this design without aggregation.

E. Storage

Table XXI compares an archive node, which keeps everything including a commit certificate per block, with a pruned full node, which keeps headers, bodies, and 57-byte receipts and drops witnesses and per-block certificates after the retention window. Pruning reduces history growth by about 95% at every load level, because the retained body and receipt are about 5.5% of a transaction's bytes. State growth is additional and depends on the number of accounts, registered keys, and identity records, priced by storage deposits.

F. Latency

In the good case, a transaction submitted just before a proposal is committed after: propagation of the compact proposal and reconstruction ($\approx \delta$), validation and execution, a prevote round (δ), a precommit round (δ), and certificate assembly. With a one-way delay δ of 50–100 ms between well-connected regions and validation in tens of milliseconds, commit latency after the proposal is a few hundred milliseconds, so the 2-second block interval dominates expected time to finality. That interval is set mostly by the budget for propagating and validating megabyte-scale blocks. Objective O4 (median finality below 6 s) has a wide margin in the good case. The relevant risk is the tail: each failed proposer adds a propose timeout, and benchmarks must report p95 and p99 latency under faults (Section XXI).

TABLE XXI
HISTORY GROWTH (GB/DAY AND TB/YEAR), $n = 100$, $\Delta_b = 2\,\text{s}$. ARCHIVE INCLUDES PER-BLOCK COMMIT CERTIFICATES (9.6 GB/DAY).

	Archive node		Pruned full node	
λ	GB/day	TB/yr	GB/day	TB/yr
100	39.4	14.4	1.7	0.6
300	99.0	36.1	5.0	1.8
1,000	307.7	112.3	16.5	6.0

G. Summary

Under post-quantum signatures, XENORRA's throughput is limited by bytes: block size, gossip bandwidth, and history storage. Signature verification is not the limit. This finding guided the design choices with the largest effect: witness segregation and pruning, epoch-only certificate retention, compact relay, hashed-key mode as the default, and the decision not to chase transaction rates above what the network can carry.

XXI. BENCHMARKING METHODOLOGY

The estimates of Section XX must be replaced by measurements before any parameter in Table XII is fixed or any performance figure is published. This section specifies how. The methodology is designed so that a third party can reproduce every result from published artefacts.

A. Principles

1) *Three kinds of number.* Every published figure is labelled as an analytical estimate, a measured result, or a target. Targets and estimates never appear in a results table.
2) *Reproducibility.* Each run records the source commit, build flags, configuration, genesis file, random seeds, hardware model, CPU microcode, kernel version, and network emulation settings. Scripts, raw logs, and processed data are published with the results.
3) *Distributions, not averages.* Latencies are reported as p50, p95, and p99 with confidence intervals over at least five repetitions; throughput as a steady-state mean with its variance.
4) *Open-loop load.* Load generators submit transactions on a schedule (Poisson arrivals at a configured rate) independent of the system's responses, avoiding the coordinated-omission error in which a slow system appears faster because the generator waits for it.

5) *Steady state.* Measurements start after a warm-up of at least five minutes and cover at least ten minutes of steady state, with storage pre-populated to a realistic state size.

B. Metrics

Table XXII defines each metric precisely, since loosely defined metrics such as "TPS" are a common source of incomparable claims.

C. Reference hardware

Weights (Equation (17)) and timeouts are calibrated against a *reference validator*. We propose an initial definition to be confirmed in Phase 4: an x86-64 server with AVX2, 16 physical cores, 64 GB of memory, NVMe storage with at least 100k random-read IOPS, and a 1 Gbit/s link, plus an ARM64 variant of comparable class. Light-client benchmarks use a mid-range smartphone.

D. Scenarios

Table XXIII lists the benchmark scenarios. Each targets a specific question, and together they cover the cases the project brief lists.

E. Transaction mixes

Three mixes are used. *M-reg*: transfers from accounts with registered keys, the best case for size. *M-hash*: transfers from accounts in hashed mode, 30% of them first spends, the realistic case for a young network. *M-id*: a mix of transfers (60%), DID updates (15%), credential anchors and revocations (15%), and verification-market calls (10%). Each mix has an adversarial variant in which 5% of submitted transactions carry invalid signatures.

F. Comparison with estimates

Each measured result is compared with the corresponding estimate in Section XX. A deviation larger than a factor of two in either direction triggers investigation before publication, since it means either the model or the implementation is wrong. Measured ML-DSA cycle counts from S1 replace the round-3 figures in all estimates.

G. Independent verification

Before mainnet, at least one external team should reproduce scenarios S1, S3, and S5 from the published artefacts, and the results should be published alongside our own. Performance claims in any external communication must cite a specific scenario, configuration, and dataset.

XXII. Scalability Considerations

A. Performance-engineering techniques

The project brief asked which high-performance computing techniques are relevant to Xenorra and which would add complexity without benefit. Table XXIV gives our assessment. A technique belongs in the consensus path only if its output is bit-for-bit reproducible on every honest node and its benefit addresses a measured or well-estimated bottleneck.

B. Scaling the validator set

Vote traffic grows linearly per validator in n (Figure 4). Three directions could lift the ceiling of a few hundred validators. *Signature aggregation*: lattice-based synchronized multi-signatures, aggregation of Falcon signatures with LaBRADOR [13], and hash-based multi-signatures aggregated with succinct proofs [14] all reduce certificate size, but none is standardized, and some depend on synchronized key epochs that change the key-management model. *Uncertified DAG consensus* [44] needs one signature per vertex instead of per vote and could reduce vote traffic, at the cost of a younger protocol. *Committee sampling* has a subset of validators vote on each block, which reduces traffic but requires careful analysis of how the safety threshold degrades with committee size and of how committees are selected without a biasable beacon. All three are research items. The suite table and versioned certificate format leave room for an aggregate-signature suite without changing the rest of the protocol.

C. Scaling throughput

Within the current design, throughput scales with block size until propagation and validation exhaust the timeout budget, and S5 will locate that point. Beyond it, two approaches that keep signatures out of the replicated data look most promising. The first is *succinct witness proofs*: the proposer, or a prover network, produces a hash-based succinct proof that every witness in a block verifies, and validators check the proof instead of the signatures. Bandwidth then falls to bodies plus one proof. The difficulty is proving ML-DSA verification efficiently inside a hash-based proof system, which is an active research problem. The second is a *client-side-validated payment layer* in the style of Compact Shielded CSV [8], as a native module, which would cost $2N + 1$ hashes per payment on chain. That depends on solving the open problems that paper identifies, chiefly proof of delivery and prover cost. Payment channels and rollups settling on Xenorra are possible too, and would inherit its post-quantum security only if their own proof systems and signatures are post-quantum.

D. Scaling state

State grows with accounts, registered keys, and identity records. Storage deposits make growth cost something for as long as the state exists. If deposits prove insufficient, state expiry, in which entries untouched for a long period are moved out of active state and can be restored with a proof, is a well-studied option. It would complicate the light-client and identity use cases, which depend on non-membership proofs, so we do not adopt it now.

XXIII. Limitations and Open Research Problems

A. Limitations of this work

This paper specifies a protocol and analyzes it; it does not report an implementation or measurements. Our performance figures rest on encoding sizes, which are exact, and on cycle counts for the round-3 predecessor of ML-DSA, which are approximate. The security arguments for our modifications to

TABLE XXII
BENCHMARK METRICS AND THEIR DEFINITIONS.

Metric	Definition	Reported as
Throughput	Transactions committed (in blocks with valid commit certificates) per second of wall-clock time in the steady-state window, by transaction mix	Mean, std. dev.; saturation curve
Time to finality	From submission at a client to observation of the commit certificate by that client	p50/p95/p99
Block interval	Difference of consecutive BFT timestamps; also wall-clock interval at a reference node	p50/p95/p99
Rounds per height	Number of consensus rounds needed to commit	Histogram
Signature verification throughput	Verifications per second per core, per suite, for valid and for invalid inputs	Mean; scaling with cores
Block validation time	From complete block reconstruction to prevote decision, split into verification, execution, and hashing	p50/p99
Bandwidth	Ingress and egress per node, per message class (tx, vote, proposal, sync), and duplication factor	Mean and peak Mbit/s
Storage growth	Bytes per block by column family, for archive and pruned nodes	Bytes/block; GB/day
Memory	Resident set size, steady and peak; cache hit rates	MB
CPU	Utilization per core and per thread pool	%
Sync time	Time for state sync from checkpoint, and for block sync per 10,000 blocks	Minutes

TABLE XXIII
BENCHMARK SCENARIOS.

ID	Scenario	Setup	Question answered
S1	Micro-benchmarks	criterion on reference hardware: ML-DSA-44/65/87 and SLH-DSA keygen, sign, verify (valid and invalid inputs) for each library; SHA3 and KMAC; codec; SMT insert/proof	True primitive costs; calibration of α coefficients; library choice
S2	Single node	Execution and storage with synthetic blocks of increasing size against a state of 10^6–10^8 accounts; signature cache on and off	Validation time per block; storage write amplification; maximum sustainable weight
S3	Local multi-node	4, 16, 64 nodes on one cluster; `tc netem` latency 20–200 ms, jitter, 0.1–1% loss	Consensus latency and rounds per height under controlled delay
S4	Geo-distributed	$n = 50, 100, 150$ validators across at least five regions on three continents, sentries included	Realistic finality latency and bandwidth; check of Eq. (23)
S5	High load	Open-loop load increased to saturation for each mix; block size 1–8 MiB, Δ_b 1–4 s	Saturation throughput; latency–throughput curve; mempool behaviour
S6	Adversarial	Up to f Byzantine validators (silent, equivocating, slow proposer); invalid-signature floods; partition and heal; eclipse attempts on a full node; targeted proposer DoS	Liveness and latency under faults; evidence production; admission budgets; recovery time
S7	Validator scaling	n from 25 to 400 at fixed load in S4 topology	Growth of vote traffic and latency with n; practical $N_{\max}$
S8	Sync and light clients	State sync from checkpoint at several state sizes; light-client epoch verification on a smartphone	Time to join; light-client cost per epoch

Tendermint are proof sketches that need independent review and machine checking. The economic model specifies mechanisms and leaves the parameters open. The identity application layer is described at the level of the Xenorra Whitepaper v2.0 and needs its own detailed specification.

B. Open problems

We list the questions this work could not settle from available evidence. Each is a candidate for focused research before or after mainnet, as indicated.

OP1 Post-quantum signature aggregation. No standardized post-quantum scheme supports compact aggregation of independent signatures. A suitable scheme would raise the validator ceiling and shrink certificates. *Monitor; prototype candidates on testnet.*

OP2 Bias-resistant post-quantum randomness. Our beacon admits bias of up to 2^m choices by m withholders. Lattice-based threshold or VRF constructions, or hash-based alternatives with acceptable cost, would remove it. *Pre-mainnet for verifier allotment if bias proves material.*

OP3 Recovery without a registered recovery key. Accounts whose primary suite is frozen and that registered no recovery key are stuck. A hash-based zero-knowledge proof of knowledge of the key-generation seed behind an address (Section IX-H) is the most promising route. *Research.*

OP4 Archival incentives. Nothing compels anyone to keep pruned witnesses. Whether this causes harm, and how to pay archive nodes if it does, is unresolved. *Post-mainnet.*

OP5 FN-DSA adoption. FN-DSA would cut transaction size by about three quarters. It awaits a published FIPS 206 and a constant-time, audited Rust implementation. *Revisit when both exist.*

OP6 Audited Rust post-quantum libraries. No pure-Rust ML-DSA or SLH-DSA implementation had an independent audit as of November 2026. *Pre-mainnet: fund an audit (Phase 7).*

OP7 Fee dimensions. Whether a single weight dimension prices bytes and computation well enough, or a multidimensional market is needed. *Decide from testnet data.*

TABLE XXIV
ASSESSMENT OF PERFORMANCE-ENGINEERING TECHNIQUES.

Technique	Relevance to XENORRA	Decision
BLAKE3 / parallel hashing	Faster than SHA3, but hashing is a small share of validation cost and inputs are small, so tree-mode parallelism yields little [19]. Not a NIST standard.	Not adopted for consensus; one hash family (Keccak)
SIMD	ML-DSA (NTT, sampling) and Keccak benefit substantially from AVX2/NEON code paths	Adopted through the crypto backend
GPU acceleration	Could batch signature verification, but GPU results are hard to make bit-reproducible; adds driver supply-chain risk	Excluded from the consensus path; permitted for off-chain workloads (proving, analytics)
Parallel transaction validation	Signature verification is stateless and dominates CPU	Adopted (thread pool); execution parallelism later, with differential testing
Memory-efficient structures	Vote sets as bitmaps plus indexed signatures; arena allocation of tree nodes; signature cache keyed by $wtxid$	Adopted
Zero-copy networking and decoding	Avoids copying 3.3 KB signatures through the pipeline	Adopted where it does not complicate the codec (reference-counted buffers)
Asynchronous execution	Suits I/O; CPU-bound verification on async tasks would starve the reactor	Tokio for I/O; dedicated pool for CPU work
Efficient serialization	Fixed-layout, length-prefixed encoding; signatures read in place	Adopted (Appendix A)
Database optimization	Column families, bloom filters, range deletes for pruning, separate devices for WAL and data	Adopted; tuned by S2 benchmarks

OP8 Economic parameters. All parameters in Table XII. *Pre-mainnet modelling and review.*

OP9 Governance. Stake-weighted voting has known failure modes. Alternatives such as delegated councils, conviction voting, or bicameral designs need evaluation. *Pre-mainnet.*

OP10 Shielded payments. A client-side-validated, post-quantum payment module along the lines of [8] needs solutions to proof of delivery, unbounded nullifier-index growth, and prover cost on consumer devices. *Research.*

OP11 Smart contracts. Whether to add a virtual machine, and which, against the criteria in Section XIV. *Post-mainnet.*

OP12 Proposer predictability. How much latency targeted denial of service against known proposers costs on a live network. *Measure in S6.*

OP13 Formal verification. Machine-checked proofs of accountable safety for (M2), and of refinement between the specification and the Rust state machine. *Ongoing from Phase 2.*

OP14 Useful computation. Whether a verifiable-computation service (Appendix D) attracts real demand and can resist fraudulent results economically. *Research; never in consensus.*

OP15 Regulatory classification. Token classification and the compliance status of the identity modules depend on jurisdiction and final design, and need legal review, as the Whitepaper v2.0 notes. *Pre-launch.*

XXIV. DEVELOPMENT ROADMAP

Development proceeds in ten phases, each ending at a gate that must be passed before the next phase begins (Table XXV). Appendix F gives deliverables, dependencies, risks, and acceptance criteria for each phase. We do not attach calendar dates. Dates depend on funding and team size, and dates published before Phase 2 is complete would be guesses.

The order follows the principle that runs through this paper: research first, then specification, then implementation, then optimization. The formal specification (Phase 2) must be approved before any consensus-critical code is merged, and the conformance vectors it produces become the acceptance test for every later phase. Performance optimization (Phase 8) comes after security review (Phase 7) and not before, because optimizations made earlier would have to be re-audited.

These phases map onto the four gates of the Whitepaper v2.0. Its Gate 1 (testnet ready) corresponds to the end of our Phase 6; Gate 2 (mainnet ready) to Phase 10; and its Gate 3 (post-quantum validator signatures) is absorbed into Phases 3–5, because XENORRA now uses post-quantum consensus signatures from genesis.

XXV. CONCLUSION

We set out to design a layer-1 blockchain that is post-quantum-resistant from its first block, using only standardized primitives and an implementation path through maintained Rust libraries. Moving to post-quantum signatures changes which resource limits a ledger. ML-DSA verification is fast; ML-DSA signatures are large. Once that is accepted, a series of design choices follows: account-based state, so that each transaction carries one signature; witnesses segregated from bodies and pruned after finality, which cuts history growth by about 95%; compact block relay; validator sets that change only at epoch boundaries, so that light clients verify one large certificate per epoch instead of one per block; and a Tendermint-family consensus protocol, chosen because the communication advantage of linear protocols disappears when votes cannot be aggregated.

Two further choices follow from the threat itself. Consensus keys must be post-quantum from genesis, because classical validator keys would leave a chain's entire history open to rewriting once a cryptographically relevant quantum computer exists. And account control should not depend on a single

TABLE XXV
ROADMAP SUMMARY. EACH GATE IS A HARD PREREQUISITE FOR THE NEXT PHASE. DETAILS IN APPENDIX F.

#	Phase	Principal deliverable	Gate
1	Cryptographic and protocol research	This paper; research dossier; decision register	External review of crypto and consensus choices completed
2	Formal protocol specification	Normative spec; conformance vectors; TLA+/Quint consensus model	Spec approved; model checks pass; two reviewers sign off
3	Rust cryptographic foundation	`xenorra-primitives`, `xenorra-crypto`; differential fuzzing	All vectors pass; no differential mismatch in 7 days of fuzzing
4	Local blockchain prototype	Single-node chain: execution, state, storage, mempool	S1–S2 benchmarks published; supply invariant holds under property tests
5	Consensus and networking	Consensus engine, XenAuth, gossip, compact relay, sync	Deterministic simulation passes fault matrix; Malachite adoption decided
6	Multi-node testnet	Private testnet, 16–64 nodes, light client	S3, S6 pass; 30 days without unplanned halt
7	Security analysis and audits	Independent audits of crypto, consensus, node; bug bounty	All critical and high findings fixed and re-verified
8	Performance optimization	Parallel execution; tuned storage and network	S4, S5, S7 meet targets; no regression in S6
9	Public testnet	Open validator set; governance and economics live	90 days stable; parameters fixed from data; incident drills done
10	Mainnet readiness	Genesis, launch validator set, runbooks	Launch criteria met; legal review complete

hardness assumption, so XENORRA adds an SLH-DSA recovery key whose security rests on the hash function alone.

The design has clear limits. Without aggregation, it supports a few hundred validators. Its randomness beacon is biasable. Its performance estimates rest on published cycle counts. Several of its components depend on Rust libraries that are young and not yet independently audited. We have tried to state each of these openly, to mark each claim as proven, assumption-based, or awaiting measurement, and to specify the protocol precisely enough that independent teams can build it, test it against shared vectors, and check our work. The next steps are the Phase 2 specification review and the S1 micro-benchmarks, which will confirm or correct the numbers on which much of this paper depends.

ACKNOWLEDGMENT

The design in Sections IX–XV draws directly on the analysis of Ilie *et al.* [8], whose observation that post-quantum authorization data comes to dominate on-chain footprint shaped our witness-segregation and pruning rules. Any errors in how we have applied their ideas are ours.

REFERENCES

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[2] C. Gidney, "How to factor 2048 bit RSA integers with less than a million noisy qubits," arXiv:2505.15917 [quant-ph], May 2025. [Online]. Available: https://arxiv.org/abs/2505.15917

[3] R. Babbush, A. Zalcman, C. Gidney, M. Broughton, T. Khattar, H. Neven, T. Bergamaschi, J. Drake, and D. Boneh, "Securing elliptic curve cryptocurrencies against quantum vulnerabilities: Resource estimates and mitigations," arXiv:2603.28846 [quant-ph] (also IACR ePrint 2026/625), Mar. 2026. [Online]. Available: https://arxiv.org/abs/2603.28846

[4] National Institute of Standards and Technology, "Module-lattice-based key-encapsulation mechanism standard," U.S. Department of Commerce, Washington, D.C., Federal Information Processing Standards Publication FIPS 203, Aug. 2024. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.203

[5] ——, "Module-lattice-based digital signature standard," U.S. Department of Commerce, Washington, D.C., Federal Information Processing Standards Publication FIPS 204, Aug. 2024. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.204

[6] ——, "Stateless hash-based digital signature standard," U.S. Department of Commerce, Washington, D.C., Federal Information Processing Standards Publication FIPS 205, Aug. 2024. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.205

[7] D. Moody, R. Perlner, A. Regenscheid, A. Robinson, and D. Cooper, "Transition to post-quantum cryptography standards," National Institute of Standards and Technology, NIST Internal Report 8547, Oct. 2026, initial public draft Nov. 2024; final Oct. 27, 2026. [Online]. Available: https://doi.org/10.6028/NIST.IR.8547

[8] D. I. Ilie, U. Lee, I. D. Stewart, J. Zhu, E. Jones, and W. J. Knottenbelt, "Compact shielded CSV: Post-quantum, private, lightweight client-side validation blockchain," arXiv:2609.32015, Sep. 2026. [Online]. Available: https://arxiv.org/abs/2609.32015

[9] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on BFT consensus," arXiv:1807.04938, 2018. [Online]. Available: https://arxiv.org/abs/1807.04938

[10] M. Mosca, "Cybersecurity in an era with quantum computers: Will we be ready?" *IEEE Security & Privacy*, vol. 16, no. 5, pp. 38–41, Sep. 2018.

[11] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Advances in Cryptology – EUROCRYPT 2018*, ser. Lecture Notes in Computer Science. Springer, 2018, pp. 66–98.

[12] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining GHOST and Casper," arXiv:2003.03052, 2020. [Online]. Available: https://arxiv.org/abs/2003.03052

[13] M. A. Aardal, D. F. Aranha, K. Boudgoust, S. Kolby, and A. Takahashi, "Aggregating Falcon signatures with LaBRADOR," in *Advances in Cryptology – CRYPTO 2024, Part I*, ser. Lecture Notes in Computer Science, vol. 14920. Springer, 2024, pp. 71–106.

[14] J. Drake, D. Khovratovich, M. Kudinov, and B. Wagner, "Hash-based multi-signatures for post-quantum Ethereum," *IACR Commun. Cryptol.*, vol. 2, no. 1, 2025, full version: IACR ePrint 2025/055.

[15] J. Nick, L. Eagen, and R. Linus, "Shielded CSV: Private and efficient client-side validation," Cryptology ePrint Archive, Paper 2025/068, 2025. [Online]. Available: https://eprint.iacr.org/2025/068

[16] H. Beast, E. Heilman, and I. F. Duke, "Pay-to-Merkle-root (P2MR)," Bitcoin Improvement Proposal BIP-360 (Draft, v0.12.1), 2024, originally titled "Pay to Quantum Resistant Hash (P2QRH)". [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0360.mediawiki

[17] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan, "Proofs of useful work," Cryptology ePrint Archive, Paper 2017/203, 2017. [Online]. Available: https://eprint.iacr.org/2017/203

[18] S. Lee and H. Kim, "Inside Qubic's selfish mining campaign on Monero: Evidence, tactics, and limits," arXiv:2512.01437, 2025, v5; accepted at AFT 2026. [Online]. Available: https://arxiv.org/abs/2512.01437

[19] J. O'Connor, J.-P. Aumasson, S. Neves, and Z. Wilcox-O'Hearn, "BLAKE3: One function, fast everywhere," Specification, 2020. [Online]. Available: https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf

[20] R. Gelashvili, A. Spiegelman, Z. Xiang, G. Danezis, Z. Li, D. Malkhi, Y. Xia, and R. Zhou, "Block-STM: Scaling blockchain execution by turning ordering curse to a performance blessing," in *Proc. 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2023, pp. 232–244.

[21] A. Yakovenko, "Solana: A new architecture for a high performance blockchain v0.8.13," White paper, 2018. [Online]. Available: https://solana.com/solana-whitepaper.pdf

[22] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. 28th Annu. ACM Symp. Theory Comput. (STOC)*, 1996, pp. 212–219.

[23] D. J. Bernstein, "Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?" in *Workshop Record of SHARCS'09: Special-purpose Hardware for Attacking Cryptographic Systems*, 2009. [Online]. Available: https://cr.yp.to/hash/collisioncost-20090823.pdf

[24] V. Buterin, "Proof of stake: How I learned to love weak subjectivity," Ethereum Foundation Blog, Nov. 2014. [Online]. Available: https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity

[25] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on Bitcoin's peer-to-peer network," in *Proc. 24th USENIX Security Symposium*. USENIX Association, 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[26] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium: A lattice-based digital signature scheme," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 1, pp. 238–268, 2018.

[27] R. Perlner, "FIPS 206 status update," Presentation, 6th NIST PQC Standardization Conference, Gaithersburg, MD, USA, Sep. 2025, fIPS 206 (FN-DSA) had not been published, even as an initial public draft, as of Nov. 2026. [Online]. Available: https://csrc.nist.gov/csrc/media/presentations/2025/fips-206-fn-dsa-(falcon)/images-media/fips_206-perlner_2.1.pdf

[28] W. Beullens, "Breaking Rainbow takes a weekend on a laptop," in *Advances in Cryptology – CRYPTO 2022, Part II*, ser. Lecture Notes in Computer Science, vol. 13508. Springer, 2022, pp. 464–479.

[29] W. Castryck and T. Decru, "An efficient key recovery attack on SIDH," in *Advances in Cryptology – EUROCRYPT 2023, Part V*, ser. Lecture Notes in Computer Science, vol. 14008. Springer, 2023, pp. 423–447.

[30] E. Kiltz, V. Lyubashevsky, and C. Schaffner, "A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model," in *Advances in Cryptology – EUROCRYPT 2018, Part III*, ser. Lecture Notes in Computer Science, vol. 10822. Springer, 2018, pp. 552–586.

[31] L. G. Bruinderink and P. Pessl, "Differential fault attacks on deterministic lattice signatures," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 3, pp. 21–43, 2018.

[32] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium: Algorithm specifications and supporting documentation (version 3.1)," NIST PQC Round 3 submission, Feb. 2021. [Online]. Available: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf

[33] J.-P. Aumasson, D. J. Bernstein, W. Beullens, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, and B. Westerbaan, "SPHINCS$^+$: Submission to the NIST post-quantum project, v.3.1," Jun. 2022. [Online]. Available: https://sphincs.org/data/sphincs+-r3.1-specification.pdf

[34] M. Ounsworth, J. Gray, M. Pala, J. Klaußner, and S. Fluhrer, "Composite module-lattice-based digital signature algorithm (ML-DSA) for use in X.509 public key infrastructure," RFC 10073, RFC Editor, Nov. 2026. [Online]. Available: https://www.rfc-editor.org/info/rfc10073

[35] K. Kwiatkowski, P. Kampanakis, B. E. Westerbaan, and D. Stebila, "Post-quantum traditional (PQ/T) hybrid key agreement mechanisms for TLS 1.3," RFC 10024, RFC Editor, Aug. 2026, defines X25519MLKEM768. [Online]. Available: https://www.rfc-editor.org/info/rfc10024

[36] Rust Secure Code Working Group, "RustSec advisory database," https://github.com/rustsec/advisory-db, 2026, accessed: Nov. 2026.

[37] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," White paper, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[38] Team Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and probabilistic leaderless BFT consensus through metastability," arXiv:1906.08936, 2019. [Online]. Available: https://arxiv.org/abs/1906.08936

[39] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. K. Alper, X. Luo, F. Shirazi, A. Stewart, and G. Wood, "Overview of Polkadot and its design considerations," arXiv:2005.13456, 2020. [Online]. Available: https://arxiv.org/abs/2005.13456

[40] Z. Gao, Y. Hu, and Q. Wu, "Jellyfish Merkle Tree," Diem Association, Technical Paper, Jan. 2021. [Online]. Available: https://developers.diem.com/papers/jellyfish-merkle-tree/2021-01-14.pdf

[41] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*, 2019, pp. 347–356.

[42] CometBFT Contributors, "CometBFT: Tendermint consensus in Go (fork of Tendermint Core), including consensus specification," GitHub repository, Apache-2.0, spec: https://github.com/cometbft/cometbft/tree/main/spec/consensus; accessed Nov. 2026. [Online]. Available: https://github.com/cometbft/cometbft

[43] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and Tusk: A DAG-based mempool and efficient BFT consensus," in *Proc. 17th European Conference on Computer Systems (EuroSys)*, 2022, pp. 34–50.

[44] K. Babel, A. Chursin, G. Danezis, A. Kichidis, L. Kokoris-Kogias, A. Koshy, A. Sonnino, and M. Tian, "Mysticeti: Reaching the latency limits with uncertified DAGs," in *Proc. Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2025.

[45] National Institute of Standards and Technology, "SHA-3 standard: Permutation-based hash and extendable-output functions," U.S. Department of Commerce, Washington, D.C., Federal Information Processing Standards Publication FIPS 202, Aug. 2015. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.202

[46] J. Kelsey, S. j. Chang, and R. Perlner, "SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash," National Institute of Standards and Technology, NIST Special Publication 800-185, Dec. 2016. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-185

[47] M. Palatinus, P. Rusnak, A. Voisine, and S. Bowe, "Mnemonic code for generating deterministic keys," Bitcoin Improvement Proposal BIP-39, 2013. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki

[48] libp2p Contributors, "rust-libp2p: The Rust implementation of the libp2p networking stack (v0.57.1)," https://github.com/libp2p/rust-libp2p, 2026, accessed: Nov. 2026.

[49] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras, "GossipSub: Attack-resilient message propagation in the Filecoin and ETH2.0 networks," arXiv:2007.02754, 2020. [Online]. Available: https://arxiv.org/abs/2007.02754

[50] Circle Internet Group, "Malachite: Flexible BFT consensus engine in Rust (arc-malachitebft v0.8.2)," https://github.com/circlefin/malachite, 2026, accessed: Nov. 2026. Originally developed by Informal Systems; alpha, not externally audited.

[51] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, Apr. 1988.

[52] P. Sheng, G. Wang, K. Nayak, S. Kannan, and P. Viswanath, "BFT protocol forensics," in *Proc. ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021, pp. 1722–1743.

[53] V. Buterin, E. Conner, R. Dudley, M. Slipper, I. Norden, and A. Bakhta, "EIP-1559: Fee market change for ETH 1.0 chain," Ethereum Improvement Proposal 1559, Apr. 2019. [Online]. Available: https://eips.ethereum.org/EIPS/eip-1559

[54] T. Roughgarden, "Transaction fee mechanism design for the Ethereum blockchain: An economic analysis of EIP-1559," arXiv:2012.00854, 2020. [Online]. Available: https://arxiv.org/abs/2012.00854

[55] RustCrypto Developers, "ml-dsa: Pure Rust implementation of ML-DSA (FIPS 204) (v0.1.2)," https://github.com/RustCrypto/signatures/tree/master/ml-dsa, 2026, accessed: Nov. 2026.

[56] Amazon Web Services, "aws-lc-rs: A cryptographic library using AWS-LC (v1.19.0)," https://github.com/aws/aws-lc-rs, 2026, accessed: Nov. 2026.

APPENDIX A
NORMATIVE PROTOCOL SPECIFICATION (V1, DRAFT)

This appendix is the normative core of the protocol. Where it conflicts with the body of the paper, this appendix governs. The key words MUST, MUST NOT, SHOULD, and MAY are to be read as requirements on conforming implementations. Constants and tags are listed in Appendix G.

A. Canonical encoding

All hashed or signed data uses the following encoding, which is a strict subset of Borsh. Implementations MUST produce exactly these bytes and MUST reject any input that is not the unique canonical encoding of a value.

1) Unsigned integers `u8`, `u16`, `u32`, `u64`, `u128` and signed `i32`: fixed width, little-endian.
2) `bool`: one byte, `0x00` or `0x01`. Any other value MUST be rejected.
3) Fixed arrays `[u8; N]`: the N raw bytes.
4) Variable sequences `Vec<T>`: a `u32` element count followed by the elements. Every sequence field has a maximum length (Appendix G); a decoder MUST reject a count above the maximum before allocating.
5) `Option<T>`: `0x00` for none, or `0x01` followed by T. Other tag values MUST be rejected.
6) Enumerations: a `u8` variant tag followed by the variant's fields. Unknown tags MUST be rejected.
7) Structures: fields in declaration order, with no padding.
8) A top-level decode MUST consume the entire input; trailing bytes MUST be rejected.

B. Hashing and signing

$\mathsf{H}_{\mathtt{tag}}(m) = \text{SHA3-256}(\mathsf{enc}_{u8}(|tag|) \| tag \| m)$, with tag ASCII and $|tag| \le 255$. Every signature in the protocol is

$$\sigma = \textsc{Sign}_{suite}\big(sk,\ \mathsf{H}_{\mathtt{tag}}(\mathsf{enc}(x)),\ ctx\big),$$

where x is the unsigned form of the object (all fields except the signature), and (tag, ctx) is the pair registered for that object type in Table XXX. For ML-DSA suites, SIGN is FIPS 204 ML-DSA.Sign (hedged) and VERIFY is ML-DSA.Verify, with the context string ctx. For SLH-DSA suites, the FIPS 205 equivalents are used with ctx.

C. Core types

```
type Hash    = [u8; 32];
type Address = [u8; 32];          // Eq. (address): H_"XEN/addr"(u16 suite || pk0)
type Amount  = u128;              // base units
type SuiteId = u16;

struct Transaction {              // 134 bytes for a Transfer with empty memo
    version: u8,                  // = 1
    chain_id: u64,
    sender: Address,
    nonce: u64,
    max_fee_per_weight: u64,
    tip_per_weight: u64,
    weight_limit: u64,
    expiry_height: u64,           // last height at which tx may be included
    call: Call,
}
struct Witness {
    suite: SuiteId,
    public_key: Option<Vec<u8>>,  // required unless the signing key is registered
    signature: Vec<u8>,
}
enum KeySpec {                    // tag 0 / 1
    Hashed { suite: SuiteId, commitment: Hash },   // H_"XEN/key"(u16 suite || pk)
    Registered { suite: SuiteId, public_key: Vec<u8> },
}
enum Call {
    Transfer { recipient: Address, amount: Amount, memo: Vec<u8> },          // 0x00
    RegisterKey,                                                            // 0x01
    UnregisterKey,                                                          // 0x02
    RotateKey { new_key: KeySpec },                                         // 0x03
    SetRecovery { suite: SuiteId, commitment: Hash, delay: u64 },           // 0x04
    InitiateRecovery { new_key: KeySpec },   // signed by recovery key        0x05
    CancelRecovery,                                                         // 0x06
    FinalizeRecovery,                        // signed by recovery key       0x07
    CreateValidator { consensus_key: Vec<u8>, node_id: Hash,
                      commission_bps: u16, self_bond: Amount },             // 0x10
```

```
    EditValidator { commission_bps: Option<u16> },                          // 0x11
    Delegate { validator: Address, amount: Amount },                        // 0x12
    Undelegate { validator: Address, amount: Amount },                      // 0x13
    Redelegate { from: Address, to: Address, amount: Amount },              // 0x14
    Unjail,                                                                 // 0x15
    RotateConsensusKey { consensus_key: Vec<u8> },                          // 0x16
    BeaconCommit { epoch: u64, commitment: Hash },                          // 0x17
    BeaconReveal { epoch: u64, value: [u8; 32] },                           // 0x18
    Propose { action: GovAction, deposit: Amount },                         // 0x20
    Vote { proposal: u64, choice: u8 },                                     // 0x21
    Execute { proposal: u64 },                                              // 0x22
    // 0x30-0x3F identity, 0x40-0x4F verification: specified in the
    // identity-module specification (separate document).
}
struct Receipt {                    // 57 bytes
    status: u8,                     // 0 = success, else module error code
    weight_used: u64,
    fee_paid: u128,
    events_root: Hash,
}
```

a) Identifiers: $txid = \mathsf{H}_{\texttt{XEN/tx}}(\mathsf{enc}(tx))$; $wtxid = \mathsf{H}_{\texttt{XEN/wtx}}(txid \| \mathsf{enc}(w))$; block hash $= \mathsf{H}_{\texttt{XEN/header}}(\mathsf{enc}(header))$.

D. Consensus objects

```
struct Header {                     // 356 bytes; field order as in the header table
    version: u16, chain_id: u64, height: u64, epoch: u64, time_ms: u64,
    parent: Hash, last_commit: Hash, tx_root: Hash, witness_root: Hash,
    state_root: Hash, receipts_root: Hash, evidence_root: Hash,
    validators: Hash, next_validators: Hash, params: Hash, proposer: u16,
}
struct Vote {                       // 3,376 bytes with an ML-DSA-65 signature
    kind: u8,                       // 1 = prevote, 2 = precommit
    chain_id: u64, height: u64, round: u32,
    block_id: Hash,                 // all-zero = nil
    validator_index: u16,
    timestamp_ms: u64,
    signature: Vec<u8>,
}
struct VoteCertificate {            // POL certificate or commit certificate
    kind: u8, height: u64, round: u32, block_id: Hash,
    signers: Vec<u8>,               // bitmap, ceil(n/8) bytes, bit i = validator i
    entries: Vec<SignedEntry>,      // one per set bit, ascending index
}
struct SignedEntry { timestamp_ms: u64, signature: Vec<u8> }
struct Proposal {
    chain_id: u64, height: u64, round: u32,
    pol_round: i32,                 // -1 if none
    header: Header,
    short_ids: Vec<[u8; 8]>,        // per tx, from wtxid keyed by header hash
    pol_certificate: Option<VoteCertificate>,  // required iff pol_round >= 0
    timestamp_ms: u64,
    signature: Vec<u8>,
}
struct Block {
    header: Header,
    txs: Vec<Transaction>,
    witnesses: Vec<Witness>,        // witnesses[i] belongs to txs[i]
    evidence: Vec<Evidence>,
    last_commit: Option<VoteCertificate>,   // None only at height 1
}
enum Evidence {
    DuplicateVote { a: Vote, b: Vote },                          // 0
    DuplicateProposal { a: Proposal, b: Proposal },              // 1
    LockViolation { precommit: Vote, prevote: Vote },            // 2
    LightClientAttack { header: Header, cert: VoteCertificate }, // 3
}
```

a) Vote certificate validity: A certificate is valid for validator set V (of the epoch containing its height) iff: the bitmap has exactly $\lceil |V|/8 \rceil$ bytes and no bits set beyond $|V|$; the number of entries equals the number of set bits; for each set bit i, the entry's signature verifies under validator i's consensus key over the vote reconstructed from $(kind, chain_id, h, r, block_id, i, timestamp)$; and the summed power of the signers exceeds $2W/3$. Entries MUST be verified in any order but the result MUST NOT depend on the order.

E. Transaction validity

A pair (tx, w) is valid at height h against state S iff all of the following hold. Rules TV1–TV7 are *stateless*; TV8–TV13 depend on S.

TV1 Both decode canonically; $|\mathsf{enc}(tx)| + |\mathsf{enc}(w)| \leq$ `MAX_TX_BYTES`.

TV2 $tx.version = 1$ and $tx.chain_id$ equals the network's identifier.

TV3 $w.suite$ is in the suite table with status ACTIVE, or DEPRECATED and h before its retirement height, and is permitted for the signing role (Table VI).

TV4 $|w.signature|$ equals the suite's signature length; if present, $|w.public_key|$ equals the suite's public-key length.

TV5 $h \leq tx.expiry_height$.

TV6 $tx.weight_limit \geq weight(tx, w)$, computed by Equation (17) with the parameters at h.

TV7 All amounts and lengths in *call* are within their declared bounds.

TV8 *Signing key resolution.* If *call* is `InitiateRecovery` or `FinalizeRecovery`, the signing key is the account's recovery record: $w.suite$ equals its suite, $w.public_key$ is present, and $\mathsf{H}_{\texttt{XEN/key}}(\mathsf{enc}_{u16}(suite) \| pk) =$ its commitment. Otherwise: if the account has no key record, $w.public_key$ is present and Equation (4) holds for the sender; if the record is *Hashed*, $w.public_key$ is present and hashes to the commitment; if *Registered*, $w.public_key$ is absent and the stored key is used. The suite in w MUST equal the suite of the record.

TV9 VERIFY$(pk, txid, w.signature,$ `"xenorra/tx/v1"`$)$ accepts.

TV10 $tx.nonce$ equals the account's nonce.

TV11 $tx.max_fee_per_weight \geq b_h$, the base fee at h.

TV12 The account's balance is at least $weight_limit \cdot \min(max_fee, b_h + tip)$.

TV13 If the account's primary suite is FROZEN, only calls whose signing key resolves to a recovery record are valid.

F. Block validity

Block B at height h with parent P (header H_P, post-state S_P) is valid iff:

BV1 B decodes canonically; $|\mathsf{enc}(B)| \leq$ `MAX_BLOCK_BYTES`; $|B.txs| = |B.witnesses|$.

BV2 `version`, `chain_id` are correct; `height` $= H_P.height + 1$; `parent` = hash of H_P; `epoch` $= \lfloor (h-1)/E \rfloor$.

BV3 For $h > 1$: `last_commit` is a valid precommit certificate for H_P under the validator set of P, and the header field `last_commit` equals its hash. `time_ms` equals the power-weighted median of its entry timestamps (lower median on ties) and exceeds $H_P.time_ms$.

BV4 `tx_root` $= \mathrm{MTH}([txid_i])$; `witness_root` $= \mathrm{MTH}([wtxid_i])$; `evidence_root` $= \mathrm{MTH}([\mathsf{H}_{\texttt{XEN/evid}}(\mathsf{enc}(e_j))])$.

BV5 `validators` equals H_P.`next_validators`; `proposer` is the output of Algorithm 3 for the deciding round.

BV6 Each evidence item is valid (signatures verify, conflict is real, offender was in the relevant set, evidence age within `EVIDENCE_MAX_AGE`) and was not previously included.

BV7 Each (tx_i, w_i) satisfies TV1–TV9 and TV13 against the state immediately before it in block order (Section XIV-E for key resolution), and TV10–TV12 at the moment it is applied.

BV8 $\mathrm{STF}(S_P, B)$ completes, its total weight is at most $2W^*$, and the resulting state root, receipts root (MTH over $\mathsf{H}_{\texttt{XEN/rcpt}}(\mathsf{enc}(receipt_i))$), and next validator set hash equal the header's `state_root`, `receipts_root`, and `next_validators`.

BV9 `params` equals the hash of the consensus parameters and suite table in effect after STF.

G. State-transition summary

For every transaction, APPLYTX performs, in order: (1) re-check TV10–TV12; (2) debit $weight_limit \cdot p$ where $p = \min(max_fee, b_h + tip)$; (3) increment nonce; (4) open an overlay and execute the call; (5) on success merge the overlay, on failure discard it and record the error code; (6) compute $weight_used$; refund $(weight_limit - weight_used) \cdot p$; burn $weight_used \cdot b_h \cdot (1 - \phi)$; credit $weight_used \cdot b_h \cdot \phi$ to the treasury and $weight_used \cdot (p - b_h)$ to the proposer; (7) emit the receipt. All products are computed in `u128` with overflow treated as failure of the transaction at step (1).

Table XXVI gives the preconditions and effects of the account and staking calls. The *signer* is the sender account.

TABLE XXVI: Preconditions and effects of selected calls.

Call	Preconditions (else failure)	Effects
`Transfer`	$amount > 0$; balance $\geq$ amount; recipient balance after $\geq$ existential deposit or recipient exists	Debit sender, credit recipient; create recipient account with no key record if absent
`RegisterKey`	Key record is *Hashed*; witness carried the public key; balance $\geq$ deposit for $\lvert pk \rvert$ bytes	Record becomes *Registered*; reserve deposit
`UnregisterKey`	Key record is *Registered*	Record becomes *Hashed* with commitment of the stored key; release deposit
`RotateKey`	New suite permitted for spending; deposit available if *Registered*	Replace key record; adjust deposit; cancel any pending recovery
`SetRecovery`	Suite permitted for recovery; $delay \geq$ `MIN_RECOVERY_DELAY`	Set recovery record (suite, commitment, delay); clear pending recovery
`InitiateRecovery`	Signed by recovery key (TV8); no pending recovery	Pending recovery $\leftarrow (new_key, h + delay)$
`CancelRecovery`	Signed by primary key; pending recovery exists	Clear pending recovery
`FinalizeRecovery`	Signed by recovery key; pending recovery exists and $h \geq$ its maturity	Install pending key; clear pending
`CreateValidator`	Not already a validator; consensus key decodes for an ACTIVE consensus suite; $self_bond \geq$ `MIN_SELF_BOND`	Move self-bond to bonded; register candidate (effective next epoch)
`Delegate`	Validator exists and not tombstoned; balance $\geq$ amount	Move amount to bonded under (delegator, validator)
`Undelegate`	Delegation $\geq$ amount	Move amount to an unbonding entry maturing at $h + U$; remains slashable until maturity
`Unjail`	Signer is jailed validator; jail period elapsed; not tombstoned	Clear jailed flag (eligible next epoch)
`RotateConsensusKey`	Signer is validator operator (operator suite)	New key effective from next epoch
`BeaconCommit/Reveal`	Signer is active validator; correct epoch; reveal matches commitment	Record commitment / reveal

a) BeginBlock: (1) Distribute the issuance for the parent block to the validators that signed its commit certificate, pro rata by power, with each validator's commission deducted before the remainder is shared with its delegators. (Tips were already credited to proposers in APPLYTX.) (2) For each evidence item: compute the correlated slash fraction (Equation (11)) over the window `SLASH_WINDOW`, slash bonded and unbonding stake of the offender and its delegators, burn the slashed amount, jail, and for safety offences tombstone. (3) At the first block of an epoch, release matured unbonding entries.

b) EndBlock: (1) Record precommit participation from `last_commit`; jail validators below the downtime threshold. (2) Update the base fee by Equation (20). (3) At the last block of an epoch: compute powers by Equation (10), select the next set, write `next_validators`; settle the beacon; apply scheduled governance actions and suite status changes.

H. Consensus message validity

CV1 A vote is accepted only if its chain identifier matches, its height is the current or next height, its validator index is in the set for that height, and its signature verifies under `("XEN/vote", "xenorra/vote/v1")`.

CV2 A proposal is accepted only from PROPOSER(h, r); its signature verifies under `("XEN/proposal", "xenorra/proposal/v1")`; if $pol_round \geq 0$ then $pol_round < r$ and `pol_certificate` is a valid prevote certificate for the proposed block at pol_round.

CV3 A second vote by the same validator for the same $(h, r, kind)$ with a different `block_id` is not counted; the pair is stored as `DuplicateVote` evidence.

CV4 Thresholds "$> 2/3$" and "$> 1/3$" compare summed power with $2W/3$ and $W/3$ using integer arithmetic: $3\sum w_i > 2W$ and $3\sum w_i > W$.

I. Conformance vectors

The `xenorra-vectors` crate publishes vectors as JSON files with hexadecimal byte fields. Each vector names an input, the expected output or the expected error class. A conforming implementation MUST pass all of them. Categories:

1) *Encoding*: every type, including boundary lengths; rejection of non-canonical booleans, option tags, unknown enum tags, oversize counts, and trailing bytes.
2) *Hashing*: $\mathsf{H}_{\mathtt{tag}}$ for each registered tag; Merkle roots for list lengths 0–17 and powers of two to 2^{12}; sparse-Merkle membership and non-membership proofs.
3) *Signatures*: the NIST ACVP vectors for ML-DSA-44/65/87 and SLH-DSA-SHAKE-128s with non-empty context strings; negative cases, including wrong context, truncated and extended signatures and keys, malformed hint encodings, repeated or unordered hint indices, coefficients of $\mathbf{z}$ out of range, and signatures valid for a different public key.
4) *Key derivation*: mnemonic to seed to ξ to public key along fixed paths (Equations (5)–(7)).
5) *Transactions*: each TV rule with a passing and a failing case.
6) *State transitions*: initial state, block, expected post-state root and receipts, for every call and failure path.
7) *Consensus*: proposer sequences for given power distributions over 10,000 rounds; certificate validity; evidence validity; BFT time computation.

APPENDIX B
RUST INTERFACES AND EXECUTION BOUNDARIES

The listings below fix the boundaries between crates. They are interface sketches for the specification. Bodies are omitted, and error types are abbreviated. The deterministic crates expose synchronous, side-effect-free APIs; the asynchronous shell owns all I/O.

A. Cryptography (`xenorra-crypto`)

```rust
/// Role a key plays; the suite table records which suites each role permits.
pub enum KeyRole { Spending, Recovery, Consensus, Operator, Node, Governance }

pub enum SuiteStatus { Reserved, Active, Deprecated { retire_at: u64 }, Retired, Frozen }

pub struct SuiteInfo {
    pub id: SuiteId,
    pub pk_len: usize,
    pub sig_len: usize,
    pub roles: &'static [KeyRole],
}

/// The only signature interface other crates may call.
pub trait SignatureVerifier: Send + Sync {
    /// `msg` is the 32-byte domain-separated digest; `ctx` the FIPS 204/205 context.
    fn verify(&self, suite: SuiteId, pk: &[u8], msg: &Hash, ctx: &[u8], sig: &[u8])
        -> Result<(), VerifyError>;
}

pub trait Signer {
    fn suite(&self) -> SuiteId;
    fn public_key(&self) -> &[u8];
    fn sign(&self, msg: &Hash, ctx: &[u8]) -> Result<Vec<u8>, SignError>; // hedged
}

/// Cache of verified witnesses, keyed by wtxid. Shared by mempool and chain.
pub trait SigCache: Send + Sync {
    fn insert_verified(&self, wtxid: Hash);
    fn is_verified(&self, wtxid: &Hash) -> bool;
}

/// Hardened-only HD derivation (Eqs. hd-master .. hd-seed).
pub fn derive_seed(master: &MasterKey, path: &[u32], suite: SuiteId) -> Zeroizing<[u8; 32]>;
```

B. Primitives and encoding (`xenorra-primitives`)

```rust
/// Implemented for every consensus type; the only path to bytes that are hashed or signed.
pub trait Canonical: Sized {
    fn encode(&self, out: &mut Vec<u8>);
    fn decode(input: &[u8]) -> Result<Self, DecodeError>;   // rejects non-canonical input
}
```

```rust
pub fn hash_tagged(tag: &'static str, data: &[u8]) -> Hash;  // Eq. (domain-hash)
pub fn merkle_root(leaves: &[Hash]) -> Hash;                 // Eq. (mth)

pub struct SignedTx { pub tx: Transaction, pub witness: Witness }
impl SignedTx {
    pub fn txid(&self) -> Hash;
    pub fn wtxid(&self) -> Hash;
}
```

C. State and storage (`xenorra-state`, `xenorra-storage`)

```rust
/// Read access to versioned tree nodes; implemented by xenorra-storage.
pub trait NodeReader {
    fn get_node(&self, key: &NodeKey) -> Result<Option<Node>, StorageError>;
}

pub struct StateView<'a, R: NodeReader> { reader: &'a R, root: Hash, version: u64 }
impl<'a, R: NodeReader> StateView<'a, R> {
    pub fn get(&self, key: &StateKey) -> Result<Option<Vec<u8>>, StateError>;
    pub fn prove(&self, key: &StateKey) -> Result<SparseMerkleProof, StateError>;
}

/// Batched tree update producing new nodes and stale-node records; no I/O.
pub fn apply_updates<R: NodeReader>(base: &StateView<R>, updates: BTreeMap<StateKey, Option<Vec<u8>>>,
                                    version: u64) -> Result<TreeUpdate, StateError>;

/// Storage engine abstraction (RocksDB in production, in-memory for tests).
pub trait KvStore: Send + Sync {
    fn get(&self, cf: Column, key: &[u8]) -> Result<Option<Vec<u8>>, StorageError>;
    fn write(&self, batch: WriteBatch) -> Result<(), StorageError>;   // atomic
    fn delete_range(&self, cf: Column, from: &[u8], to: &[u8]) -> Result<(), StorageError>;
}
```

D. Execution (`xenorra-execution`)

```rust
/// Inputs the STF may read besides state: nothing else (no clock, no randomness).
pub struct BlockEnv { pub height: u64, pub epoch: u64, pub time_ms: u64,
                      pub proposer: u16, pub base_fee: u64, pub params: ConsensusParams }

pub struct StfOutput { pub updates: BTreeMap<StateKey, Option<Vec<u8>>>,
                       pub receipts: Vec<Receipt>, pub next_validators: ValidatorSet,
                       pub next_params: ConsensusParams }

/// Deterministic state-transition function, Eq. (stf). Signatures must already be
/// verified (by key resolution in block order) before calling.
pub fn execute_block<R: NodeReader>(state: &StateView<R>, env: &BlockEnv,
                                    block: &Block, evidence_ok: &[bool])
    -> Result<StfOutput, BlockError>;

/// Static access set of a call, used for weight (Eq. weight) and parallel scheduling.
pub fn access_set(tx: &Transaction) -> AccessSet;
```

E. Consensus (`xenorra-consensus`)

```rust
/// Inputs to the consensus state machine.
pub enum Input {
    Proposal(SignedProposal, Option<FullBlock>),
    Vote(SignedVote),
    BlockValidated { id: Hash, valid: bool },  // result of valid(v) from the chain crate
    TimeoutFired { height: u64, round: u32, step: Step },
    Committed { height: u64 },                 // storage confirmed the commit
}

/// Outputs: actions for the shell to perform.
pub enum Action {
    BuildProposal { height: u64, round: u32, valid_value: Option<Hash> },
    Validate(Hash),
    SignAndBroadcast(UnsignedMessage),         // routed to the remote signer
    ScheduleTimeout { height: u64, round: u32, step: Step, after_ms: u64 },
    Commit { block: Hash, certificate: VoteCertificate },
```

```
    ReportEvidence(Evidence),
}

/// Pure state machine (Algorithm 4). No I/O, no clock, no randomness.
pub struct Tendermint { /* height, round, step, locked/valid, vote sets */ }
impl Tendermint {
    pub fn new(validators: ValidatorSet, height: u64, params: TimeoutParams) -> Self;
    pub fn handle(&mut self, input: Input) -> Vec<Action>;
}

pub fn proposer(set: &mut ProposerPriorities, round: u32) -> u16;   // Algorithm 3
```

F. Signer protocol (`xenorra-signer`)

```
pub struct HighWaterMark { pub height: u64, pub round: u32, pub step: u8, pub block_id: Hash }

/// Persisted before any signature is released (fsync). Refuses any request that is
/// below the mark, or equal to it with a different block_id.
pub trait DoubleSignGuard {
    fn check_and_advance(&mut self, req: &SignRequest) -> Result<(), GuardError>;
}
```

G. Execution boundaries

Three boundaries are enforced in code review and continuous integration. (1) `xenorra-consensus`, `xenorra-execution`, `xenorra-state`, and `xenorra-primitives` do not depend on `tokio`, `std::time`, `rand`, or any networking crate. (2) Only `xenorra-crypto` depends on cryptographic backends. (3) Only `xenorra-primitives` implements `Canonical`. A dependency-graph check (for example with `cargo-deny` bans) fails the build if these rules are broken.

APPENDIX C
PROOFS AND PROOF SKETCHES

Throughout, "negligible" is with respect to the security parameter of the signature and hash primitives, and $\mathcal{A}$ is quantum polynomial-time. Validator powers are integers and thresholds are evaluated exactly as in rule CV4.

A. Preliminaries

Lemma 3 (Quorum intersection)**.** *If $Q_1, Q_2 \subseteq \mathcal{V}$ with $w(Q_1) > 2W/3$ and $w(Q_2) > 2W/3$, then $w(Q_1 \cap Q_2) > W/3$. Under Assumption 1, $Q_1 \cap Q_2$ contains an honest validator.*

Proof. $w(Q_1 \cap Q_2) = w(Q_1) + w(Q_2) - w(Q_1 \cup Q_2) > 4W/3 - W = W/3 > w(B)$, so $Q_1 \cap Q_2 \not\subseteq B$. □

Lemma 4 (Attribution)**.** *Under Assumption 3, except with negligible probability, every consensus message that verifies under an honest validator's consensus key was produced by that validator for that object type.*

Proof sketch. Suppose $\mathcal{A}$ produces, with non-negligible probability, a message m of type τ with a valid signature under honest validator i's key, where i never signed m as type τ. Build a forger $\mathcal{F}$ that receives an ML-DSA challenge key, guesses i uniformly among the n validators, embeds the key as i's consensus key, answers i's signing needs through the signing oracle, and outputs $\mathcal{A}$'s forgery. The digest signed is H

By Lemma 4, we may treat honest validators' votes and proposals as unforgeable in what follows, losing only a negligible term.

B. Agreement (Theorem 1)

Proof sketch. The argument is that of Tendermint [9], restated with weights. Suppose honest p commits v at height h in round r and honest q commits $v' \neq v$ at height h in round r'. Each holds a precommit certificate of power $> 2W/3$.

Case $r = r'$. By Lemma 3 the two certificates share an honest signer, who must have precommitted twice in round r for different values. Honest validators precommit at most once per round, which is a contradiction.

Case $r < r'$ (the other case is symmetric). Let C be the precommit certificate for v at r, and let $L = C \setminus B$ be its honest signers; $w(L) > 2W/3 - W/3 = W/3$. Each member of L precommitted v at r and so set $lockedValue = v$, $lockedRound = r$. We show by induction on $s \in (r, r']$ that no prevote certificate for any $u \neq v$ exists at round s and that every member of L is still locked on v at the start of round $s + 1$.

An honest member of L prevotes $u \neq v$ in round s only through one of the two prevote rules of Algorithm 4. The first requires $lockedRound = -1$ or $lockedValue = u$, both false. The second requires a prevote certificate for u at some vr with $lockedRound \leq vr < s$, so $r \leq vr < s$. For $vr > r$ this is excluded by the induction hypothesis. For $vr = r$: honest members of L precommitted v at r only after seeing a prevote certificate for v at r, and a prevote certificate for u at r would intersect it (Lemma 3) in an honest validator that prevoted twice in round r, which is impossible. Members of L prevote v or $\perp$ in round s, and prevotes for u have power at most $W - w(L) < 2W/3$. No prevote certificate for u exists at s, so no honest validator precommits u or relocks on u at s, which completes the induction.

In round r', a precommit certificate for v' needs power $> 2W/3$. Honest validators precommit v' only after seeing a prevote certificate for v' at r', which does not exist. So all precommitters for v' are Byzantine, with power $< W/3$, which is a contradiction.

Our modifications do not alter this argument. (M1) changes the signature scheme, which is covered by Lemma 4. (M2) adds a requirement on proposals, which only removes executions. (M3)–(M4) change how $valid(v)$ is computed, not the voting rules. (M5) fixes the validator set within an epoch, which the argument assumes anyway. Agreement across heights follows because each height's decision is final and the next height starts from it. □

C. Accountable safety (Theorem 2)

We assume honest nodes retain every proposal, vote, and certificate they receive for heights within the evidence window, which conforming implementations are required to do. The theorem does not use Assumption 1.

Proof sketch. Suppose valid precommit certificates C for v at (h, r) and C' for $v' \neq v$ at (h, r') are observed. By Lemma 3 (which needs only the weight condition), $F_0 = \text{signers}(C) \cap \text{signers}(C')$ has $w(F_0) > W/3$.

If $r = r'$, each member of F_0 signed two precommits for different values in the same round. The two signed votes form `DuplicateVote` evidence, checkable by anyone holding the validator set. Set $F = F_0$.

If $r < r'$, each member of F_0 precommitted v at r and v' at r'. Under the protocol, a validator locked on v at round r may precommit $v' \neq v$ at a later round r' only after seeing a prevote certificate for v' at r'. Two situations arise. (i) No honest node holds a prevote certificate for v' at r'. Then the accusation against every member of F_0, consisting of their precommit for v at r and their precommit for v' at r', cannot be refuted by any message an honest node possesses. We set $F = F_0$. Here, and only here, accountability depends on availability: a member of F_0 who was in fact honest would have received such a certificate and retained it. Honest validators can always refute a false accusation, while the guilty cannot. (ii) A prevote certificate P' for v' at r' exists. Then $\text{signers}(P') \cap \text{signers}(C)$ has power $> W/3$, and each member prevoted v' at r' while locked on v since r. By the prevote rules, this is permitted only if the proposal for v' at r' carried a POL certificate for v' at some round vr with $r \leq vr < r'$. Because of (M2), that certificate is part of the signed proposal and is observed with it. If no such proposal exists, these signers form a `LockViolation` set. If it exists, repeat the argument with vr in place of r'. Rounds strictly decrease and are bounded below by r. At $vr = r$ the POL certificate for v' at r intersects the prevote certificate for v at r, which members of C needed before precommitting, in a set of power $> W/3$ whose members prevoted twice in round r. That is `DuplicateVote` evidence. In every branch we obtain a set of power $> W/3$ with transferable evidence. The signatures in the evidence are attributable by Lemma 4. □

Remark 1. Branch (i) is why Sheng *et al.* classify Tendermint's forensic support as dependent on the information available [52]. (M2) closes the gap for re-proposed values, and retention of prevotes by honest nodes closes it for fresh proposals. Machine-checking this argument is open problem OP13.

D. Liveness (Theorem 3)

Proof sketch. The original proof [9] shows that after GST, once timeouts exceed the time needed for a round's messages to reach all honest validators, there is a round in which all honest validators participate, the proposer is honest, and every honest validator prevotes and precommits the proposed value. Round synchronization uses the rule that moves a validator forward on seeing $> 1/3$ power at a higher round. Locked validators accept the proposal because an honest proposer re-proposes its $validValue$ with $validRound$. Since weighted round-robin gives every validator a proposer slot within every window of rounds (each validator's priority grows by its power each round and is reduced only when it proposes), honest proposers recur. We check our modifications. (M2): an honest proposer that sets $validValue$ has received the $> 2/3$ prevotes justifying it and can attach them. (M3)–(M4): validation and reconstruction add a bounded delay after GST, since an honest proposer serves missing transactions on request; timeouts that grow linearly in the round eventually exceed it. Transaction inclusion follows because an honest proposer includes any valid mempool transaction whose fee meets the base fee, subject to block limits, and honest proposers recur. □

E. Authorization (Theorem 4)

Proof sketch. By induction over committed blocks, we show that every key record and recovery record in state was installed by a transaction authorized under the rules. The base case is genesis, which installs no key records. Consider a committed block containing (tx, w) that modifies account a. By Validity, TV8 and TV9 hold, so w carries a valid signature on $txid$ under a public key pk bound to a in one of three ways: (1) $\mathsf{H}_{\texttt{XEN/addr}}(suite \| pk) = a$; (2) $\mathsf{H}_{\texttt{XEN/key}}(suite \| pk)$ equals the stored commitment; (3) pk is the stored registered key. In cases (1) and (2), if pk differs from the key the owner generated, $\mathcal{A}$ has found a second preimage or a collision for SHA3-256 under a fixed domain tag, contradicting Assumption 4. In case (3), the stored key was installed by an authorized transaction, by the induction hypothesis. So pk is the owner's key. If the owner never signed $txid$ under context `"xenorra/tx/v1"`, the signature is a forgery, contradicting Assumption 3 (or Assumption 5 for recovery-signed calls). If the owner signed a different transaction with the same $txid$, that is a SHA3 collision. Calls that install key or recovery records are authorized, which preserves the induction hypothesis. □

F. No replay (Lemma 1)

Proof. APPLYTX increments the sender's nonce whenever a transaction passes TV10, whether or not its call succeeds, and nonces never decrease. TV10 requires the transaction's nonce to equal the current nonce, so after the first application the same transaction fails TV10 forever. On a chain with a different identifier it fails TV2. The chain identifier is inside the signed body, so the transaction cannot be modified to pass TV2 there without invalidating the signature (Theorem 4). □

G. Supply conservation (Proposition 1)

Proof. Let Σ denote the left-hand side of Equation (15). We show that each operation changes Σ by exactly the change in $I_h - Burn_h$. *Fees*: APPLYTX debits ℓp (with $\ell = weight_limit$), refunds $(\ell - u)p$ (with $u = weight_used$), credits $ub_h\phi$ to the treasury and $u(p - b_h)$ to the proposer, and burns $ub_h(1 - \phi)$. Net change of Σ: $-\ell p + (\ell - u)p + ub_h\phi + u(p - b_h) = -ub_h(1-\phi)$, which equals the increase in $Burn$. *Calls*: `Transfer` moves value between balances. `Delegate`, `Undelegate`, `Redelegate`, and unbonding maturity move value between balance, bonded, and unbonding terms. `RegisterKey`, `UnregisterKey`, and deposits move value between balance and deposit terms. Key, recovery, beacon, and governance-vote calls move no value. Each of these leaves Σ unchanged. *Issuance*: each block credits exactly the amount by which I_h increases. *Slashing*: decreases bonded or unbonding stake by an amount added to $Burn$. *Failure*: a failed call's overlay is discarded, so only the fee changes apply, and these are covered above. All arithmetic is checked, so no wrap-around can create value. Each new call type must extend this case analysis. □

H. Recoverability (Lemma 2)

Proof sketch. With the primary suite frozen, TV13 rejects every transaction for the account except those signed by its recovery key, so `RotateKey`, `SetRecovery`, and `CancelRecovery` cannot be executed by anyone. `InitiateRecovery` and `FinalizeRecovery` require a valid SLH-DSA signature under the committed recovery key. By Assumption 5 and the binding argument of Theorem 4, only the recovery-key holder can produce them. The holder initiates; no other party can cancel or replace the pending recovery; after D_{rec} blocks the holder finalizes. Inclusion of each transaction follows from Liveness. □

APPENDIX D
USEFUL COMPUTATION: EVALUATION AND AN OPTIONAL SERVICE DESIGN

The project brief asked whether useful computation, as promoted by Qubic and pursued by Pearl, could be incorporated into XENORRA without compromising determinism, security, or fairness. This appendix records the evaluation and the resulting design.

A. Requirements for computation that secures consensus

For computation to serve as Sybil resistance or leader election, as proof of work does, it must satisfy five properties.

1) *Efficient verification*: checking a claimed result must cost far less than producing it.
2) *Unforgeability of effort*: there must be no shortcut that yields valid results without the claimed work.
3) *Unpredictability and freshness*: work must be bound to the current chain state, so that it cannot be precomputed or reused.
4) *Progress-free or fair reward*: the probability of winning must be proportional to work contributed, without large advantages for whoever happens to hold particular inputs.
5) *Determinism*: honest verifiers must agree bit-for-bit on validity.

Hash-based proof of work meets all five. Useful workloads typically fail at least one. Training a neural network has no cheap verification of "the model improved by this much" unless verification repeats much of the work. Many optimization

problems allow shortcuts for some instances. Floating-point workloads on GPUs are not bit-reproducible across devices without deliberate effort, as *Hawkeye* documents. Recent work tackles these obstacles directly, for example with proofs of useful work from arbitrary matrix multiplication and earlier from specific fine-grained hard problems [17], and economic analyses have begun to examine how such schemes change miners' incentives. These are promising research results, but none has been deployed at scale with independent security analysis, and none fits a BFT proof-of-stake design, where Sybil resistance comes from stake.

B. Assessment of the supplied references

TABLE XXVII: Claims in the supplied references and their evidential status.

Claim	Status	Notes
A ∼150 KB trinary network reached zero errors on two-number addition across seven difficulty levels	Reported experimental result	Not independently replicated; code release described as planned; the article itself calls it interpolation
The network generalizes to unseen problems	Partly reported	Within the training range only (interpolation); extrapolation stated as not yet done
Path to general intelligence	Project thesis	The article: "a thesis. It is not a result"
Use for trading, noise removal, pattern detection	Project claim	No results reported
Qubic consensus: 451 of 676 computors	Documented design	No peer-reviewed security analysis found; arbitrator role is a centralization point
Qubic held majority Monero hashrate (Aug. 2025)	Disputed claim	Measurement study (AFT 2026) estimates 23.4% average share, brief hourly peaks near 50%, consistent with selfish mining; yield 4% below honest mining [18]
Pearl: PoUW layer-1 from AI computation	Project design	Competing protocol; whitepaper not peer reviewed
Pearl: GPU kernel and hashing speedups	Self-reported benchmarks	Not independently reproduced; GPU BLAKE3 claim appears only in a social-media post
GPU non-determinism can be characterized and reproduced	Peer-reviewed (MLSys 2026)	Supports keeping GPUs out of consensus-critical paths

C. Decision

a) Decision U1: no useful computation in consensus: XENORRA derives Sybil resistance from stake and agreement from BFT voting. Neither benefits from useful computation, and adding it would introduce non-determinism, verification asymmetries, and a dependence on rented hardware that the Monero episode shows can be redirected [18].

D. Optional verifiable-computation service

Useful computation can still be offered as an application service, treated like the identity verifier market: results are claims made by bonded parties and carry no consensus guarantee. The design has five elements.

1) *Task specification.* A requester registers a task with a content hash of a deterministic workload (integer or fixed-point arithmetic, pinned runtime image), the inputs' commitment, a reward, and a redundancy level k.
2) *Assignment.* k bonded workers are drawn using the beacon and stake caps. The beacon's bias is tolerable here, because integrity comes from redundancy and challenges, and assignment need not be secret.
3) *Commit–reveal.* Workers commit to result hashes, then reveal. Agreement of at least t of k results finalizes the task.
4) *Challenge.* During a window, any bonded party can dispute a result by posting a bond. Disputes are resolved by re-execution by a fresh set of workers, or, for workloads that admit it, by a succinct proof. Hash-based proof systems are preferred here; pairing-based verifiable computation such as Pinocchio is efficient but not post-quantum. Trusted execution environments can add evidence, but have repeatedly been broken by microarchitectural attacks, so they supplement redundancy and cannot replace it. Losers forfeit their bonds.
5) *No feedback into consensus.* Task outcomes never affect block validity, leader selection, or validator rewards.

This addresses the "verifier's dilemma" identified by Truebit, in which no one has an incentive to check results that are almost always correct, through redundancy at assignment and occasional forced audits paid from task fees. Whether such a service attracts demand is open problem OP14. Its security is economic, bounded by bonds, and must be described that way to users.

APPENDIX E
ARCHITECTURE DECISION REGISTER

Each entry records the seven attributes required by the design method of Section VII: selected option (Sel.), alternatives (Alt.), justification (Why), security implications (Sec.), performance implications (Perf.), implementation complexity (Cx.), and known limitations (Lim.). Section references point to the full discussion.

TABLE XXVIII: Decision register.

C1 Default signature scheme (Section IX)	
Sel.	ML-DSA-65 for transactions, votes, proposals, node identity; ML-DSA-87 for operator, governance, treasury keys.
Alt.	ML-DSA-44; FN-DSA-512; SLH-DSA; XMSS/LMS; hybrid ML-DSA+Ed25519 by default.
Why	Standardized (FIPS 204); category-3 margin for irrecoverable long-lived keys; mature implementations; single scheme on consensus path.
Sec.	EUF-CMA from MLWE/SelfTargetMSIS (QROM, non-tight); hedged signing resists faults.
Perf.	3,309-byte signatures: bandwidth and storage dominate cost; verification $\approx$60 µs.
Cx.	Moderate; libraries available but young.
Lim.	About 4$\times$ larger than FN-DSA; no aggregation; depends on lattice assumptions.
C2 Recovery keys (Section IX)	
Sel.	Optional SLH-DSA-SHAKE-128s recovery key, stored as 32-byte commitment, with delay D_{rec}.
Alt.	No recovery; social recovery only; ML-DSA recovery key; stateful hash-based recovery.
Why	Independent, hash-only assumption keeps account control if lattices fall.
Sec.	Survives an ML-DSA break; delay protects against stolen recovery key.
Perf.	7,856-byte signature, $\sim$0.2 s signing; used rarely.
Cx.	Second signature implementation; recovery state machine.
Lim.	Accounts without recovery keys remain exposed (OP3); shared mnemonic gives no theft protection.
C3 Hybrid signatures (Section IX)	
Sel.	Optional composite suite (reserved), not default.
Alt.	Mandatory hybrid for all or for high-value operations.
Why	Classical half adds nothing once a CRQC exists; SLH-DSA hedge persists.
Sec.	No reliance on classical schemes for any default path.
Perf.	Avoids $\sim$100 extra bytes per transaction.
Cx.	Low (reserved suite only).
Lim.	Institutions requiring hybrids must opt in.
C4 Hash function (Section IX)	
Sel.	SHA3-256 with domain tags; SHAKE256; KMAC256.
Alt.	SHA-256; BLAKE3; Poseidon-family hashes.
Why	NIST standard; same Keccak permutation as ML-DSA and SLH-DSA-SHAKE; one hash family to audit.
Sec.	$\sim$128-bit collision and preimage security against known quantum attacks.
Perf.	Slower than BLAKE3, but hashing is a minor cost.
Cx.	Low.
Lim.	Not friendly to arithmetic-circuit proof systems if succinct proofs are added later.
C5 Key-record modes (Section IX)	
Sel.	Hashed (default) or Registered (with storage deposit), chosen by owner.
Alt.	Always store full key; always carry full key.
Why	Trades prunable witness bytes for permanent state, priced by the party who benefits.
Sec.	Key hidden until first spend in either mode.
Perf.	Hashed: +1,956 bytes per tx; Registered: +1,952 bytes of state per account.
Cx.	Low.
Lim.	Users must understand the choice; wallets should automate it.
C6 Wallet derivation (Section IX-F)	
Sel.	BIP-39 mnemonic; hardened-only KMAC256 derivation to ML-DSA seed ξ.
Alt.	BIP-32-style public derivation; random keys with backup files.
Why	ML-DSA has no public derivation; deterministic seeds allow backup with a mnemonic.
Sec.	No xpub exposure risk.
Perf.	Negligible.
Cx.	Low.
Lim.	No watch-only address generation from public data.
N1 Transport (Section X)	
Sel.	QUIC + TLS 1.3 with X25519MLKEM768; libp2p discovery and gossip; XenAuth ML-DSA peer authentication.
Alt.	Bespoke transport; Noise; PQ Noise.
Why	Mature stack with PQ key exchange; adds PQ identity that libp2p lacks.
Sec.	Confidentiality against harvest-now-decrypt-later; channel-bound PQ authentication.
Perf.	$\sim$1.1 KB extra per handshake; one signature each way.
Cx.	Moderate (custom auth layer).
Lim.	Depends on rustls/quinn advisory hygiene; libp2p identity remains classical underneath.

N2 Gossip policy (Section X)

Sel.	Verify before relay; vote bitmaps; announce-then-pull for transactions.
Alt.	Relay before verify; eager flooding.
Why	Prevents amplification of invalid signatures; cuts redundant 3.3 KB transfers.
Sec.	DoS resistance.
Perf.	Extra round trip for pulled messages.
Cx.	Moderate.
Lim.	Per-hop verification cost.

N3 Block propagation (Section X)

Sel.	Compact proposals with 8-byte keyed short identifiers.
Alt.	Full blocks; block parts only.
Why	Transactions already in mempools need not be resent.
Sec.	Keyed identifiers prevent precomputed collisions.
Perf.	$\sim$5 KB instead of $\sim$2 MB per proposal in good case.
Cx.	Moderate.
Lim.	Falls back to full fetch on misses.

K1 Consensus protocol (Section XI)

Sel.	Tendermint with modifications (M1)–(M5).
Alt.	HotStuff/HotStuff-2/Jolteon; Narwhal/Bullshark; Mysticeti; Gasper-style; Avalanche.
Why	Linear protocols lose their advantage without aggregation; Tendermint is mature, analyzed, accountable.
Sec.	Safety $< 1/3$; accountable safety with (M2).
Perf.	$\approx 2n$ votes per height per node; three message delays in good case.
Cx.	Moderate; reference implementations exist.
Lim.	Halts under prolonged asynchrony; predictable leaders; validator ceiling of a few hundred.

K2 Proposer selection (Section XI)

Sel.	Deterministic weighted round-robin.
Alt.	PQ VRF; RANDAO-style beacon; round-robin by index.
Why	No standardized PQ VRF; beacon biasable; weighted RR is unbiasable and fair by power.
Sec.	Predictable leaders enable targeted DoS.
Perf.	None.
Cx.	Low.
Lim.	DoS risk to be measured (OP12).

K3 Slashing (Section XI-J)

Sel.	Correlated slashing, $\kappa = 3$; jailing for downtime.
Alt.	Fixed-fraction slashing.
Why	Cheap isolated mistakes, full loss for coordinated attacks.
Sec.	Raises cost of safety attacks to full stake of $\geq 1/3$.
Perf.	None.
Cx.	Low.
Lim.	Window and κ need tuning.

L1 State commitment (Section XIII)

Sel.	Binary sparse Merkle tree (Jellyfish-style), SHA3-256.
Alt.	Hexary MPT; Verkle; no authenticated state.
Why	Compact proofs incl. non-membership for revocation; simple spec.
Sec.	Hash-based, PQ-safe (Verkle is not).
Perf.	$\sim$27 hashes per update and proof.
Cx.	Moderate.
Lim.	More hashing than hexary trees.

E1 Execution (Section XIV)

Sel.	Native modules; sequential semantics; parallel signature verification.
Alt.	EVM; Wasm; Move; RISC-V VM at launch.
Why	Smaller audit surface; targeted use cases are well understood.
Sec.	No contract exploits; MEV limited.
Perf.	Verification dominates and is parallel.
Cx.	Lower than a VM.
Lim.	Every new feature needs a protocol upgrade (OP11).

S1 Storage engine (Section XV)

Sel.	RocksDB behind a key–value trait; witnesses and certificates in separate column families.
Alt.	MDBX; redb; fjall; sled.
Why	Production-proven LSM suited to versioned trees; range-delete pruning.
Sec.	Not consensus-critical.
Perf.	Compaction write amplification.
Cx.	C++ dependency.
Lim.	Upstream crate releases are slow; active fork exists.

M1 Issuance (Section XVI)	
Sel.	Issuance curve as a function of staking ratio, parameters from modelling.
Alt.	Fixed inflation; fixed supply.
Why	Steers staking ratio toward security target.
Sec.	Security budget scales with bonded value.
Perf.	None.
Cx.	Low.
Lim.	Parameters open (OP8).
M2 Fees (Section XVI)	
Sel.	Weight-based EIP-1559 base fee, burned (share ϕ to treasury); tips to proposer.
Alt.	First-price auction; multidimensional market.
Why	Predictable fees; proven incentive properties; simple.
Sec.	Removes proposer incentive to manipulate base fee.
Perf.	One weight dimension may misprice resources.
Cx.	Low.
Lim.	May need extra dimensions (OP7).
R1 Crypto implementation (Section XIX)	
Sel.	aws-lc-rs ML-DSA primary; RustCrypto `ml-dsa` and `fips204` differential; `fips205` for SLH-DSA.
Alt.	RustCrypto only; libcrux; pqcrypto; liboqs.
Why	Most optimized maintained option on a FIPS path; differential testing catches divergence.
Sec.	Mitigates verify-path bugs that would split consensus.
Perf.	Optimized assembly paths.
Cx.	C dependency; test infrastructure.
Lim.	No audited pure-Rust option yet (OP6).
R2 Consensus implementation (Section XIX)	
Sel.	Own sans-I/O engine; Malachite as reference and test oracle.
Alt.	Adopt Malachite directly; port CometBFT.
Why	Malachite is alpha and unaudited; own engine matches (M1)–(M5) exactly.
Sec.	Smaller trusted base under our control.
Perf.	Neutral.
Cx.	Higher initial effort.
Lim.	Revisit at Phase 5 gate.
U1 Useful computation (Appendix D)	
Sel.	Excluded from consensus; optional bonded verifiable-compute service.
Alt.	Useful PoW consensus; hybrid PoS/uPoW.
Why	Verification asymmetry, determinism, and rented-hardware risk.
Sec.	Consensus unaffected by compute fraud.
Perf.	None on consensus.
Cx.	Service is optional.
Lim.	Service security is economic only.

APPENDIX F
ROADMAP DETAIL

TABLE XXIX: Phases, deliverables, dependencies, risks, and acceptance criteria.

Phase	Deliverables	Dependencies	Principal risks	Acceptance criteria
1. Cryptographic and protocol research	This paper; source-verified research dossier; decision register; threat model	None	Missing a relevant attack or standard change; over-commitment to young primitives	External cryptographer and distributed-systems reviewer report no unresolved blocking issue; all references verified
2. Formal protocol specification	Normative spec (Appendix A expanded, incl. identity modules); conformance vectors; TLA+/Quint model of consensus with (M1)–(M5); spec change process	Phase 1	Ambiguities that implementations resolve differently; model too abstract to catch real bugs	Two independent reviewers can derive identical test outputs from the spec alone; model checking finds no agreement or accountability violation up to configured bounds
3. Rust cryptographic foundation	`xenorra-primitives`, `xenorra-crypto`, `xenorra-vectors`; differential fuzzing harness; KMAC HD derivation	Phase 2 vectors	Library verify-path bugs; side channels; dependency churn	100% of vectors pass on x86-64 and ARM64; no differential mismatch in 7 consecutive days of fuzzing; S1 micro-benchmarks published
4. Local blockchain prototype	State tree, storage, execution of native modules, mempool, single-node block production; CLI and wallet	Phase 3	State-tree performance; storage write amplification	Supply invariant holds over 10^6 random property-test transactions; S2 benchmarks published; state-transition vectors pass

Phase	Deliverables	Dependencies	Principal risks	Acceptance criteria
5. Consensus and networking	Consensus engine; signer with double-sign guard; XenAuth; gossip; compact relay; block and state sync; deterministic simulator	Phase 4; Malachite evaluation	Liveness bugs under partitions; gossip bandwidth higher than modelled	Simulator passes the fault matrix (crash, partition, equivocation, delay) over 10^4 seeded runs with zero safety violations; Malachite adoption decision recorded
6. Multi-node testnet	Private testnet of 16–64 nodes across regions; light client; monitoring; incident runbooks	Phase 5	Operational issues; performance far from estimates	S3 and S6 executed; 30 days without unplanned halt; measured bandwidth within $2\times$ of Eq. (23) or model revised
7. Security analysis and audits	Independent audits of crypto backends (incl. ML-DSA and SLH-DSA verifiers), consensus, networking, node; public bug bounty	Phase 6	Critical findings requiring redesign; audit capacity	All critical and high findings fixed and re-verified by auditors; audit reports published
8. Performance optimization	Parallel execution with differential testing; storage and network tuning; weight calibration	Phase 7	Optimizations introducing consensus divergence	S4, S5, S7 meet targets O4–O6 or targets revised with justification; no regression in S6; optimized code re-reviewed
9. Public testnet	Open validator onboarding; governance and fee market live; identity modules; economic parameter study	Phase 8	Low participation; parameter mis-setting; spam	90 days stable; parameters in Table XII fixed from data and reviewed; incident and suite-freeze drills completed
10. Mainnet readiness	Genesis configuration; launch validator set meeting independence criteria; legal review; operator documentation	Phase 9	Regulatory constraints; concentration of stake at launch	Launch criteria met; legal review complete; independent reproduction of S1, S3, S5 published

APPENDIX G
PROTOCOL CONSTANTS, TAGS, AND PARAMETERS

A. Domain tags and context strings

Every hash uses a tag (Equation (3)). Every signature uses a tag for the digest and a FIPS 204/205 context string. Tags and contexts MUST NOT be reused across object types.

TABLE XXX: Registered hash tags and signature contexts.

Tag	Signature context	Used for
`XEN/tx`	`xenorra/tx/v1`	Transaction identifier and transaction signatures
`XEN/wtx`	–	Witnessed transaction identifier
`XEN/vote`	`xenorra/vote/v1`	Prevotes and precommits
`XEN/proposal`	`xenorra/proposal/v1`	Proposals
`XEN/auth`	`xenorra/p2p/v1`	XenAuth handshake transcript
`XEN/freeze`	`xenorra/freeze/v1`	Emergency suite-freeze messages
`XEN/attest`	`xenorra/attest/v1`	Verifier and AI-assisted attestations
`XEN/header`	–	Block hash
`XEN/addr`	–	Addresses, Eq. (4)
`XEN/key`	–	Key and recovery commitments
`XEN/node`	–	Node identifiers
`XEN/mt`	–	Merkle lists, Eq. (12)
`XEN/smt`	–	Sparse Merkle nodes, Eq. (14)
`XEN/val`	–	Sparse Merkle leaf values
`XEN/skey`	–	State keys, Eq. (13)
`XEN/rcpt`	–	Receipts
`XEN/evid`	–	Evidence items
`XEN/rand`	–	Beacon commitments
`XEN/beacon`	–	Beacon output
`XEN/shortid`	–	Compact-relay short identifier key derivation

KMAC customization strings: `xenorra-hd-v1`, `xenorra-hd-child`, `xenorra-keygen`. TLS exporter label: `EXPORTER-xenorra-auth`. HD path purpose index: `7363'`.

B. Parameters

Table XXXI lists protocol parameters. Values marked *testnet* are starting points for Phase 6 with the rationale given; they are not mainnet values. Parameters marked *benchmark* or *model* must not be set until the corresponding evidence exists.

TABLE XXXI: Protocol parameters.

Parameter	Initial value	Rationale
`MAX_BLOCK_BYTES`	2 MiB (testnet)	Planning configuration of Section XX; revised by S5
Target block interval Δ_b	2 s (testnet)	Budget for propagating and validating 2 MiB blocks across regions
`MAX_TX_BYTES`	64 KiB	Must fit SLH-DSA-256s signatures (29,792 bytes) plus body and DID payloads
Epoch length E	43,200 blocks (testnet)	About one day at 2 s; one certificate per day for light clients
Timeouts $\tau^0_{\text{prop}}, \delta_{\text{prop}}$	3 s, 0.5 s (testnet)	Within the range used by CometBFT deployments [42]; propose timeout covers reconstruction and validation
Timeouts $\tau^0_{\text{pv}}, \tau^0_{\text{pc}}, \delta$	1 s, 1 s, 0.5 s (testnet)	As above
W_{prune}	2×`EVIDENCE_MAX_AGE`	Witnesses and certificates kept while evidence can still be submitted
`EVIDENCE_MAX_AGE`	$< U$, *model*	Must expire before unbonding completes so offenders remain slashable
Unbonding period U	*model*	Trade-off between long-range security and liquidity
Trusting period T_{trust}	$2U/3$ (testnet)	Strictly less than U with margin for evidence propagation
D_{rec}	7 days (testnet)	Time for an owner to notice and cancel a malicious recovery
`MIN_RECOVERY_DELAY`	1 day (testnet)	Lower bound on user-chosen D_{rec}
κ (correlation multiplier)	3	Full slash when $\geq 1/3$ of stake is slashed in the window
`SLASH_WINDOW`	1 epoch (testnet)	Groups coordinated offences
Downtime threshold θ, window L	*benchmark*	From testnet distribution of honest downtime
$N_{\max}$	100 (testnet)	Objective O6; revised by S7
$c_{\max}$	*model*	Concentration target
Nonce look-ahead	16	Allows pipelined submissions without unbounded mempool state per sender
Replacement bump	10% of tip	Prevents cheap signature-verification churn
Short identifier length	8 bytes	Collision probability negligible at block sizes considered; keyed per block
Max vector lengths	per field, in spec	e.g. memo $\leq$ 256 bytes; DID document $\leq$ 16 KiB; evidence per block $\leq$ 64
α weight coefficients	*benchmark*	S1/S2 calibration
W^* target weight	*benchmark*	Validation within propose timeout with margin
Issuance parameters	*model*	Table XII
δ_{byte} storage deposit	*model*	Table XII