

Multi-System Verification Model

Ritual

April 20, 2026

Abstract

No single proof system dominates all others across all three dimensions of soundness, completeness, and latency. This is the fundamental reason for using multiple proof systems rather than selecting the “best” one. Delegated computations are verified through a product lattice [6] of non-interactive proof systems [7] with heterogeneous soundness-completeness-latency profiles. Applications declare safety requirements as upsets in the lattice. The product lattice allows users to dictate their own safety requirements: an application can require “at least one hardware attestation and one ZK proof,” or “any two of three proof systems,” without the protocol imposing a single global choice. Let $V_1, \dots, V_m$ be proof systems with independent failure modes and soundness parameters $s_1, \dots, s_m$. The probability that an invalid result is accepted by all m systems is at most $\prod_{i=1}^m s_i$.

1 Introduction

Symphony [25] is an execution-aware SMR framework.

Standard SMR treats transactions as opaque: validity is structural (signatures, well-formedness). Symphony’s delegated execution [21] requires application-specific verification (ZK proofs, hardware attestations, or combinations thereof), and different applications have different safety needs. The product lattice allows users to dictate their own safety requirements: an application can require “at least one hardware attestation and one ZK proof,” or “any two of three proof systems,” without the protocol imposing a single global choice. This flexibility is the motivation for the construction below.

The verification state vector over proof systems, introduced in Section 2.3, is the data structure through which applications interact with delegated computation results. Verification states form a boolean hypercube $\{0, 1\}^m$ with componentwise ordering. This standard structure gives the paper a natural partial order (“at least this much verification”) and monotonicity properties.

This paper makes the following contributions:

- Applications declare safety requirements as upsets in the lattice (Section 2.3).
- We prove soundness composition for independent proof systems under explicit independence assumptions (Section 2.3).
- We develop the operational semantics: how verification state evolves, how applications declare and consume safety requirements, how the no-rollback semantics works, and how proof system degradation is handled (Section 3).

Section 2 establishes the formal framework: system model, proof systems, and the product lattice. Section 3 develops the product lattice verification model. Section 4 establishes safety and liveness. Section 5 discusses limitations. Section 6 reviews related work. Section 7 concludes.

2 Formal Framework

2.1 System Model

We consider a distributed system of n stake-weighted replicas (validators), of which at most $f < n/3$ are Byzantine. The system operates under partial synchrony [10]: there exists a Global Stabilization Time (GST) and a message delay bound Δ , both unknown to the protocol, such that after GST all messages between correct replicas are delivered within Δ . Before GST, messages may be delayed arbitrarily but are not lost.

Each replica i holds stake $w_i > 0$. A set of replicas constitutes a *quorum* if it holds more than two thirds of the total stake. Byzantine replicas control less than one third of the total stake.

The system runs a *state machine replication (SMR)* protocol [19]. SMR (equivalently, atomic broadcast) is the task of maintaining a totally-ordered log of state transitions such that all correct replicas agree on the same sequence of transitions and apply them in the same order. Formally: given a deterministic state transition function $\text{STF} : \mathcal{S} \times \mathcal{T} \rightarrow \mathcal{S}$, where $\mathcal{S}$ is the state space and $\mathcal{T}$ the set of transactions, SMR guarantees that (1) all correct replicas deliver the same sequence of transactions, and (2) the resulting state is uniquely determined by applying STF to the initial state in that order.

2.2 Non-Interactive Proof Systems

Symphony’s verification layer is built on proof systems. We adopt the standard definition [7, 13] with a deliberate architectural restriction to non-interactive systems [7].

Definition 2.1 (Non-Interactive Proof System). *A non-interactive proof system for a relation $\mathcal{R} \subseteq \mathcal{Z} \times \mathcal{W}$ between statements and witnesses is a pair of probabilistic polynomial-time algorithms (Prove, Verify) such that:*

- $\text{Prove}(z, w) \rightarrow \pi$: *given statement z and witness w , produces a proof π in a single message.*
- $\text{Verify}(z, \pi) \rightarrow \{0, 1\}$: *accepts or rejects without further interaction.*

Definition 2.2 (Completeness and Soundness). *A non-interactive proof system $V = (\text{Prove}, \text{Verify})$ satisfies:*

- *Completeness with parameter $c_V \in [0, 1]$: for all $(z, w) \in \mathcal{R}$,*

$$\Pr[\text{Verify}(z, \text{Prove}(z, w)) = 1] \geq c_V.$$

- *Soundness with parameter $s_V \in [0, 1]$: for all z such that no w satisfies $(z, w) \in \mathcal{R}$, and for all (possibly adversarial) proofs π^* ,*

$$\Pr[\text{Verify}(z, \pi^*) = 1] \leq s_V.$$

We extend the standard definition with one parameter relevant to the blockchain setting:

Definition 2.3 (Extended Proof System). *An extended proof system is a tuple*

$$V = (\text{Prove}, \text{Verify}, s_V, c_V, \ell_V)$$

where s_V and c_V are soundness and completeness parameters, and ℓ_V is the expected proving latency (time for Prove to produce π).

Latency is an engineering parameter. The purpose of this extension is to make the soundness-completeness-latency tradeoff explicit. Failure correlation between proof systems (whether compromise of one implies compromise of another) is a property of proof system *pairs*, not of individual systems, and is analyzed in Section 3.3.

The non-interactivity constraint. Symphony restricts to non-interactive proof systems. This excludes interactive verification schemes, including optimistic fraud proofs as used in Optimism [17] and Arbitrum [1]. The restriction is a deliberate architectural choice with three consequences:

1. *No synchrony assumptions in verification.* Interactive fraud proofs require a synchrony assumption: the honest challenger must submit a fraud proof within a bounded challenge window. This introduces a synchrony requirement beyond what the underlying SMR protocol demands (which operates under partial synchrony [10]). Non-interactive proof systems require no additional synchrony: a proof is valid or invalid regardless of when it is checked.
2. *Simpler peer-to-peer layer.* Non-interactive proofs are data: they can be gossiped, stored, and verified independently by any replica at any time. Interactive verification requires a protocol layered on top of the SMR protocol, with its own liveness and safety requirements. Symphony avoids this additional protocol complexity.
3. *Bounded verification latency.* The time from proof generation to verification acceptance is $\ell_V + \delta$, where δ is the network propagation delay. There are no interactive rounds, no challenge periods, and no dispute games. For fraud proofs, the equivalent latency includes the full challenge period (typically days in production systems).

The tradeoff is that some computations efficiently verifiable via interactive protocols may not be efficiently verifiable non-interactively [12]. In practice, the proof systems relevant to Symphony—hardware attestations, SNARKs [14], STARKs [4]—are all non-interactive.

2.3 Product Lattice over Proof Systems

Given m non-interactive proof systems $\{V_1, \dots, V_m\}$, we define the *verification state* of a delegated computation result o as a vector:

$$\sigma(o) = (v_1, \dots, v_m) \in \{0, 1\}^m$$

where $v_i = 1$ if and only if proof system V_i has verified the result (i.e., a quorum of assigned validators has accepted the proof under V_i).

The set $\{0, 1\}^m$ with componentwise ordering forms a boolean hypercube with a natural partial order. For two states $\sigma = (v_1, \dots, v_m)$ and $\sigma' = (v'_1, \dots, v'_m)$:

$$\sigma \leq \sigma' \iff v_i \leq v'_i \text{ for all } i \in [m].$$

The lattice has 2^m elements, a bottom element $\perp = (0, \dots, 0)$ (no proof system has verified), and a top element $\top = (1, \dots, 1)$ (all proof systems have verified). The meet and join operations are componentwise min and max, respectively.

Boolean hypercubes with componentwise ordering are a standard construction in computer science, appearing in information flow security, abstract interpretation, and distributed knowledge models.

Definition 2.4 (Application Safety Requirement). *An application’s safety requirement is an upset (upward-closed set) $U \subseteq \{0, 1\}^m$: if $\sigma \in U$ and $\sigma \leq \sigma'$, then $\sigma' \in U$. A result o is accessible to an application with requirement U if and only if $\sigma(o) \in U$ and its status (Definition 2.6) is SETTLED.*

Upsets are the natural formalism for “at least this much verification.” For example, with $m = 3$ proof systems (hardware attestation, SNARK, STARK):

- “At least one hardware attestation and one cryptographic proof” defines the upset $U = \{(1, 1, 0), (1, 0, 1), (1, 1, 1)\}$ and its upward closure.
- “At least two of three” defines a different upset.
- “All three” defines the singleton upset $\{(1, 1, 1)\}$.

Proposition 2.5 (Soundness Composition for Independent Proof Systems). *Let $V_1, \dots, V_m$ be proof systems whose soundness failures (acceptance of an invalid statement) are mutually independent events, with soundness parameters $s_1, \dots, s_m$. The probability that an invalid result is accepted by all m systems is at most $\prod_{i=1}^m s_i$.*

Proof. By independence, the event that V_i accepts an invalid result is independent across i . The joint acceptance probability is the product of marginal acceptance probabilities, each bounded by s_i . $\square$

For correlated proof systems (e.g., multiple hardware attestations from the same vendor), independence does not hold and the joint analysis requires explicit modeling of the correlation structure, as standard in reliability theory [3]. The lattice structure makes the independence assumptions explicit: applications that require high combined soundness should specify proof systems with independent failure modes, which correspond to orthogonal dimensions in the lattice.

2.4 Extended State

Symphony extends the standard SMR model [19] with additional state components:

Definition 2.6 (Symphony State). *The state S consists of:*

- *Execution state S_{exec} : account balances, contract storage (standard).*
- *Async result registry S_{async} : for each pending or settled delegated computation, a record $(id, \theta, x, status, y, \sigma)$ where $status \in \{PENDING, SETTLED, DISPUTED, EXPIRED\}$, y is the output (if settled), and $\sigma \in \{0, 1\}^m$ is the verification state; θ and x are the parameters and input of the delegated computation [21].*

3 Multi-System Verification Model

This section develops the operational semantics: how verification state evolves, how applications declare and consume safety requirements, how the no-rollback semantics works, and how proof system degradation is handled.

3.1 Lattice Structure and Verification State

Recall from Section 2.3 that the verification state of a delegated computation result o is:

$$\sigma(o) = (v_1, \dots, v_m) \in \{0, 1\}^m$$

where $v_i = 1$ indicates that proof system V_i has verified the result (committee quorum attained). The product lattice $(\{0, 1\}^m, \leq)$ with componentwise ordering has 2^m elements.

For $m = 2$ (hardware attestation V_1 and SNARK V_2), the lattice has four elements (Figure 1):

$$\perp = (0, 0) \quad (0, 1) \quad (1, 0) \quad \top = (1, 1)$$

and the Hasse diagram is a diamond. For $m = 3$, the lattice is a cube with 8 elements and 12 edges.



Figure 1: Product lattice over proof systems. Verification state advances monotonically as independent proof systems (e.g., trusted execution environment (TEE) attestation and zero-knowledge proof (ZKP)) provide attestations (“failed proofs are slashed” in the figure refers to the executor of a rejected result). Applications declare safety requirements as upsets (upward-closed regions) in the lattice; a result becomes accessible when its state enters the required upset.

3.2 Soundness-Completeness-Latency Pareto Frontier

No single proof system dominates all others across all three dimensions of soundness, completeness, and latency. This is the fundamental reason for using multiple proof systems rather than selecting the “best” one.

Hardware attestations achieve high completeness (if the hardware is functioning, the attestation is always produced), low latency (milliseconds), but correlated soundness (a vendor-wide vulnerability compromises all attestations from that vendor simultaneously). They dominate the latency-completeness frontier but not the soundness frontier.

Cryptographic proofs (SNARKs, STARKs) achieve high soundness (negligible error under computational assumptions, independent across instances), but lower completeness (the prover may fail for very large computations) and high latency (minutes to hours for large workloads). They dominate the soundness frontier but not the latency frontier.

The product lattice exposes this tradeoff to applications. An application that needs fast finality can declare a safety requirement of $(1, 0)$ (hardware attestation sufficient). An application that needs cryptographic soundness can require $(0, 1)$ (SNARK sufficient) or $(1, 1)$ (both). The lattice is the mechanism through which applications navigate the Pareto frontier.

3.3 Failure Correlation Analysis

The soundness composition proposition (Proposition 2.5) requires independence of failure modes. In practice, independence must be assessed per proof system pair.

Independent failures. Hardware attestation (TDX) and cryptographic proof systems (SNARKs, STARKs) fail for unrelated reasons: the former via hardware compromise (physical attack, firmware vulnerability), the latter via cryptographic assumption failure. These failures are independent under any reasonable threat model.

Correlated failures. Multiple hardware attestations from the same vendor are correlated: a firmware vulnerability affects all instances simultaneously. The correlation structure is *per-vendor*: attestations from different vendors (Intel SGX, AMD SEV, ARM TrustZone) are approximately independent.

This analysis follows standard reliability theory [3]. Common-mode failure analysis identifies correlated failure channels and ensures that the lattice dimensions assigned to correlated systems are not treated as independent in the safety analysis.

Definition 3.1 (Failure Correlation Matrix). *For m proof systems, the failure correlation matrix $\rho \in [-1, 1]^{m \times m}$ has entries ρ_{ij} denoting the correlation between failure events of systems V_i and V_j . Mutual independence implies $\rho_{ij} = 0$ for $i \neq j$; the converse does not hold, so ρ is a diagnostic, and the product bound of Proposition 2.5 requires mutual independence.*

Applications requiring high combined soundness should specify proof systems from orthogonal dimensions in the lattice — those with $\rho_{ij} \approx 0$. The lattice structure makes this explicit: the soundness guarantee for a lattice position $\sigma = (v_1, \dots, v_m)$ depends on the correlation structure among the proof systems with $v_i = 1$.

3.4 Monotonic State Advancement

Verification state transitions are monotonically increasing in the lattice.

Invariant 3.2 (Monotonicity). *For any result o , the verification state $\sigma(o)$ can only advance: v_i can transition from 0 to 1, never from 1 to 0. No rollback of verification attestations occurs.*

This invariant holds because:

1. Each $v_i = 0 \rightarrow 1$ transition is guarded by a committee quorum attestation. Once a quorum of validators has attested, the attestation is recorded on-chain and is immutable (backed by the SMR protocol’s finality).
2. There is no mechanism for “un-verifying”: proof system verification is a one-way gate.

The lattice has two absorbing regions:

- **Top:** $\sigma(o) = \top = (1, \dots, 1)$. All proof systems have verified. This is the terminal state for a correctly executed computation.
- **Disputed:** a special status entered when proof systems disagree (one accepts, another rejects). If V_i has verified ($v_i = 1$) but V_j rejects (v_j cannot advance to 1 because the committee quorum rejected the proof), the result’s status transitions from **Settled** to **Disputed**. Disputed results are quarantined: no application can read them regardless of their lattice position. Formally, accessibility (Definition 2.4) is conditioned on both the lattice position and the result status: a result o is accessible to an application with requirement U if and only if $\sigma(o) \in U$ and $\text{status}(o) = \text{SETTLED}$.

3.5 Temporal Consistency

Different proof systems complete verification at different times. Hardware attestations complete in milliseconds; SNARK proofs may take minutes to hours. During this interval, different applications observe different lattice positions depending on their declared safety requirements.

Symphony provides three consistency guarantees, each grounded in the distributed database literature:

Monotonic reads [20]. Once an application observes a result o at lattice position σ , it never subsequently observes o at a position $\sigma' < \sigma$. This follows directly from the monotonicity invariant: the on-chain state only advances.

Snapshot isolation [5]. An application’s view of the result registry is consistent with a specific block height. Within a single transaction, the application sees a snapshot of all results at their lattice positions as of that block.

Bounded staleness [2]. The gap between the “latest” lattice position of a result (the position that would be observed if all pending verifications completed instantly) and the application-visible position is bounded by the latency of the slowest required proof system. For a result o and a requirement U with set of minimal elements $\min U$, where proof system V_i has latency ℓ_i , the bounded staleness is the latency of the slowest proof system still missing on the fastest path into U (and 0 if $\sigma(o) \in U$):

$$\delta_{\text{stale}}(o) = \min_{u \in \min U} \max_{i: u_i=1, v_i=0} \ell_i$$

This quantity decreases monotonically as proof systems complete. An application requiring only hardware attestation experiences staleness $\leq \ell_{\text{HW}} \approx$ milliseconds. An application requiring a SNARK proof experiences staleness $\leq \ell_{\text{ZKP}} \approx$ minutes to hours.

3.6 Irrevocable Execution with Economic Safety

Delegated computation results are not rolled back, even if a proof system later rejects the result. This is a deliberate design choice with consequences for the safety model.

Why no rollback. Rolling back a settled result would require reverting all downstream state transitions that consumed the result. In a blockchain with composable contracts, a single result may be consumed by hundreds of downstream transactions within seconds of settlement. Rollback would create a cascade of reversions, violating the finality expectations of users and applications that interacted with the post-settlement state. This is the same argument that motivates irrevocable transactions in the saga model [11]: compensating actions are preferable to rollback when the cost of rollback exceeds the cost of the incorrect state.

Economic safety argument. In the absence of rollback, safety is maintained economically. If a result y^* passes hardware attestation ($v_1 = 1$) but is later rejected by the SNARK proof ($v_2 = 0$), the result enters the **Disputed** state. The executor (the single staked node that performed the delegated computation [21]) who submitted y^* is slashed: their bond b (the stake the executor locks for the computation) is distributed to affected parties (users who consumed y^* in downstream transactions, the protocol treasury, the committee that detected the dispute).

A rational executor does not submit incorrect results when $b \geq \text{MEV}(W, S)$, where $\text{MEV}(W, S)$ denotes the maximal extractable value [9] that an incorrect result for the delegated workload W (the parameterized, possibly randomized computation the executor evaluated [21]) yields at state S . This is mechanism design [16, 18], not cryptographic safety. The assumptions are: rational adversary, correct bond sizing, live slashing mechanism.

Interaction with the lattice. The no-rollback semantics is reflected in the lattice: the **Disputed** state is absorbing (no transition out of it without protocol governance intervention). Applications that consumed the disputed result at a lower lattice position (e.g., based on hardware attestation only) accepted the risk encoded in their safety requirement. Applications that required the full lattice ($\top$) never consumed the result (it never reached $\top$) and are unaffected.

3.7 Application Safety Declarations

Applications declare their safety requirements as upsets in the lattice (Definition 2.4).

Declaration mechanism. A smart contract registers a safety requirement by calling a registry function:

`declareSafety( $U$ )` where $U \subseteq \{0, 1\}^m$ is an upset.

In practice, upsets are specified by their minimal elements (the antichain generating the upset), which is compact: for $m = 2$, the antichain has at most 2 elements.

Join propagation. When contract A calls contract B and B consumes a delegated computation result, the effective safety requirement is $U_A \cap U_B$, the join (least upper bound) of A 's and B 's requirements in the order in which a smaller upset is a stronger requirement. This ensures that a caller never receives data verified at a level below its own requirement, even through transitive calls.

Explicit downcasting. A contract can explicitly lower its effective safety requirement for a specific read. This is logged on-chain (for auditability), scoped to the current transaction (does not affect the contract's declared requirement), and does not propagate to callers (the caller still sees the result through the lens of its own requirement). Downcasting is the mechanism by which applications opt into faster finality at reduced safety.

3.8 Proof System Degradation Protocol

When a proof system is compromised (e.g., a hardware vendor discloses a firmware vulnerability, or a cryptographic assumption is broken), the protocol enters a degraded mode for that system.

Detection. Proof system compromise is detected externally (vendor disclosure, academic publication) and reported via a governance transaction. The protocol does not attempt automatic detection of proof system compromise; this is a deliberate limitation (automatic detection of, e.g., a broken cryptographic assumption is an open problem).

Response.

1. The compromised proof system V_j is marked as **Degraded** in the protocol state.
2. Applications that required V_j (i.e., every minimal element u of their upset has $u_j = 1$) are notified: their effective safety requirement can no longer be met by existing proofs. They must either:
 - Accept a reduced safety level (explicitly downcast past the degraded dimension).
 - Wait for re-verification under a replacement proof system (if one is added).
 - Trigger a compensating action.
3. Settled results whose verification state has $v_j = 1$ as its only nonzero component are flagged for quarantine via exclusion guarantees [24]: transactions consuming these results are excluded until the results are re-verified.
4. New proofs under the degraded system are not accepted until the degradation is resolved.

This generalizes certificate revocation [15] and key compromise response to the proof system setting.

4 Safety and Liveness

4.1 Safety

Safety means that no correct replica accepts an invalid state transition.

Under *full-network verification*, every correct validator independently runs **Verify** on every proof. Under committee-based verification, each proof is instead checked by a committee of validators sampled uniformly at random from the validator set, using a simple majority quorum. A committee is *compromised* if it contains a Byzantine majority; the probability of this event is bounded by a hypergeometric analysis [22].

4.1.1 Delegated Execution: Multiple Independent Proof Systems

Under full-network verification of each proof system, the composed safety analysis is clean: each system's acceptance depends only on its own soundness, so by Proposition 2.5 an application requiring verification by all m systems accepts an incorrect result with probability at most $\prod_{i=1}^m s_i$, provided the soundness failures are mutually independent. Mutual independence holds when the proof systems have different soundness bases (e.g., hardware integrity vs. cryptographic assumption) and do not share implementation infrastructure. In practice, shared host infrastructure (hypervisors, compilers, OS kernels) can correlate failures that are nominally orthogonal. The product bound is an idealized limit; the correlation structure must be assessed per deployment.

With committee-based shard verification. If shard-level committees are used (a proof is split into shards, each verified by its own sampled committee [22]), the per-system acceptance probability becomes $s_i + p_i$ (soundness failure or committee compromise), and the composed bound becomes $\prod_{i=1}^m (s_i + p_i)$. In this regime, committee compromise probabilities p_i may dominate the soundness terms, and committee sizing becomes the binding constraint for practical safety.

4.1.2 Delegated Execution: Correlated Proof Systems

When proof systems have correlated failure modes (e.g., multiple hardware attestations from the same vendor), the product bound does not apply.

Lemma 4.1 (Correlated Safety Bound). *Let $V_1, \dots, V_m$ be proof systems with joint soundness failure probability $s_{1,\dots,m}$ (the probability that all systems simultaneously accept an invalid proof). For an application requiring all m systems:*

$$\Pr[\text{incorrect accepted}] \leq s_{1,\dots,m} + \Pr[\text{at least one committee compromised}]$$

where $s_{1,\dots,m} \leq \min_i s_i$ but may be much larger than $\prod_i s_i$ if failures are positively correlated.

Proof. An incorrect result is accepted only if, for every i , V_i accepts an invalid proof or the committee verifying it for V_i is compromised. If no committee is compromised, all m systems accept an invalid proof, which has probability at most $s_{1,\dots,m}$; otherwise at least one committee is compromised. A union bound over the two events gives the claim. $\square$

The joint analysis requires explicit modeling of the correlation structure, standard in reliability theory [3]. For the common case of one hardware attestation vendor and one cryptographic proof system, the two systems have independent failure modes: hardware compromise and cryptographic assumption failure are unrelated events, so the product bound (Proposition 2.5) applies.

4.2 Liveness

Liveness means that every correct transaction eventually makes progress.

4.2.1 Prover Liveness

After the executor submits a result, proof generation for each proof system proceeds independently. A proof system may fail to produce a proof within the expected latency (prover crash, resource exhaustion, proof system bug).

The result remains at its current lattice position. Applications that require a higher lattice position than currently achieved simply wait. If the proving deadline expires:

1. The proof escrow for that proof system (the payment held for its prover) is returned to the payer (no penalty for the prover, as proof failure is not necessarily Byzantine behavior).
2. The result’s lattice position is frozen at its current state.
3. Applications requiring the unachieved proof system dimension must either (a) wait for a re-proving attempt, (b) fall back to a lower safety requirement, or (c) trigger a compensating action via the saga mechanism (the compensating actions of two-phase delegated execution [21]).

5 Discussion

5.1 Limitations

Verification window for high-latency proof systems. For large workloads, SNARK proving may take hours. During this verification window, safety depends on the hardware attestation and the economic safety model. This is a constraint of current proof technology. As proof systems improve, the window narrows; the architecture accommodates this progression without structural change.

Correlated proof system failures. The product lattice provides composed safety for independent proof systems, but correlated failures (e.g., a vulnerability affecting all hardware vendors simultaneously) can compromise multiple lattice dimensions. The degradation protocol (Section 3.8) mitigates this but cannot eliminate systemic risk.

6 Related Work

Non-interactive proof systems [7, 13] are the verification primitive. SNARKs [14] achieve constant-size proofs and constant verification time under knowledge assumptions. STARKs [4] achieve transparent setup and post-quantum security at the cost of larger proofs. IPA-based systems [8] offer an intermediate tradeoff.

Hardware-based attestation (Intel SGX, AMD SEV, ARM TrustZone) provides a different soundness model: hardware integrity rather than computational hardness. Symphony treats hardware attestations and cryptographic proofs as independent dimensions in the product lattice, exploiting their orthogonal failure modes.

Predictable validity is the property Symphony requires of the underlying SMR protocol [23]: the proposer can exactly compute the state resulting from confirmation of their block. Bounded staleness [2] parameterizes the relaxation for DAG-based protocols. Session guarantees [20] (monotonic reads, read-your-writes) are provided by the product lattice’s monotonicity invariant (Section 3.4). Snapshot isolation [5] describes the per-block consistency model.

Common-mode failure analysis [3] underpins the correlation analysis of proof systems (Section 3.3). Certificate revocation (X.509 CRL, OCSP) [15] is the precedent for the proof system degradation protocol (Section 3.8).

7 Conclusion

Verification is multi-dimensional (product lattice over heterogeneous proof systems) rather than a single proof system. The lattice is the mechanism through which applications navigate the Pareto frontier. The lattice structure makes the independence assumptions explicit: applications that require high combined soundness should specify proof systems with independent failure modes, which correspond to orthogonal dimensions in the lattice. Delegated computation results are not rolled back; when a proof system later rejects a result, safety is maintained economically, which is mechanism design, not cryptographic safety. The degradation protocol generalizes certificate revocation [15] and key compromise response to the proof system setting.

References

- [1] Offchain Labs. Arbitrum. 2021.
- [2] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, I. Stoica. Probabilistically bounded staleness for practical partial quorums. *Proc. VLDB Endow.* 5(8):776–787, 2012.
- [3] R. E. Barlow, F. Proschan. *Statistical Theory of Reliability and Life Testing*. 1975.
- [4] E. Ben-Sasson, I. Bentov, Y. Horesh, M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. IACR ePrint 2018/046, 2018.
- [5] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O’Neil, P. O’Neil. A critique of ANSI SQL isolation levels. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 1–10, 1995.
- [6] G. Birkhoff. *Lattice Theory*. AMS, 3rd ed., 1967.
- [7] M. Blum, P. Feldman, S. Micali. Non-interactive zero-knowledge and its applications. In *Proc. 20th ACM Symp. Theory of Computing (STOC)*, pp. 103–112, 1988.
- [8] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, C. Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Proc. EUROCRYPT*, pp. 327–357, 2016.
- [9] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, A. Juels. Flash Boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *Proc. IEEE Symp. Security and Privacy*, pp. 910–927, 2020.
- [10] C. Dwork, N. Lynch, L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM* 35(2):288–323, 1988.
- [11] H. Garcia-Molina, K. Salem. Sagas. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 249–259, 1987.
- [12] O. Goldreich, S. Micali, A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. ACM* 38(3):691–729, 1991.
- [13] S. Goldwasser, S. Micali, C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.* 18(1):186–208, 1989.

- [14] J. Groth. On the size of pairing-based non-interactive arguments. *EUROCRYPT*, 2016.
- [15] R. Housley, W. Ford, W. Polk, D. Solo. Internet X.509 public key infrastructure certificate and CRL profile. RFC 2459, 1999.
- [16] L. Hurwicz. On informationally decentralized systems. 1972.
- [17] Optimism Foundation. 2021.
- [18] T. Roughgarden. Transaction fee mechanism design. In *EC*, 2021.
- [19] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.* 22(4):299–319, 1990.
- [20] D. B. Terry, A. J. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, C. H. Hauser. Session guarantees for weakly consistent replicated data. In *Proc. 3rd Int. Conf. Parallel and Distributed Information Systems (PDIS)*, pp. 140–149, 1994.
- [21] Ritual. Superposition of Execution Models. 2026.
- [22] Ritual. Verifiable Computation with Distributed Proving and Verification. 2026.
- [23] Ritual. Extended External Validity. 2026.
- [24] Ritual. Inclusion and Exclusion Guarantees. 2026.
- [25] Ritual. Symphony: Execution-Aware Consensus and Proposer Disaggregation. 2026.