A Framework for Evaluating Cryptographic Agility in Deployed Systems

Tin Erispe

ETHPH

(Dated:April 2026)

Abstract

Motivation	Cryptographic primitives have finite security lifespans. The deprecations of MD5, SHA-1, and RSA-1024 demonstrate that systems must eventually migrate. Modern infrastructure, however, increasingly operates under structural constraints that resist change — most acutely in blockchain and zero-knowledge proof systems, where key cryptographic components are encoded in immutable onchain contracts.
Problem	No rigorous evaluation framework exists for measuring cryptographic agility in immutability-constrained systems. Existing guidance (RFC 7696 [2], NIST PQC migration documentation [23]) was designed for mutable, negotiable environments and does not account for governance costs, proof system substitution, or state continuity requirements specific to decentralized infrastructure.
Thesis	Cryptographic agility is an architectural property determined by the system layer at which the cryptographic primitive is anchored. Systems that anchor primitives at negotiation layers achieve agility structurally; systems that anchor primitives at execution layers do not. This distinction is not a matter of engineering quality — it is a consequence of where in the system the primitive dependency lives.
Approach	We propose the Cryptographic Agility Score (CAS) as an operationalization of this thesis: a nine-dimension semi-formal evaluation framework that measures where and how severely a system is constrained in its ability to migrate cryptographic primitives. We apply it comparatively to TLS cipher suite negotiation and ZK rollup proof system migration, and derive design principles from the structural differences observed.
Scope	This paper proposes the framework and provides a worked demonstration across two case studies. It does not claim empirical validation; inter-rater consistency and historical calibration against measured Time-to-Migrate values are identified as future work. Scores presented here represent the author's reasoned assessment against explicit criteria, not experimentally derived values.

Key Finding	TLS achieves high agility (20/27) because algorithm negotiation is colocated with the handshake — the protocol layer designed for parameter agreement. ZK rollups score critically low (8/27) not primarily due to poor engineering, but because their proof systems are expressions of fixed algebraic universes (field, curve, hash function) embedded throughout the constraint system. This is mathematical rigidity, not merely architectural rigidity. The two are structurally distinct and require different remediation strategies.

Findings at a Glance

The table below presents the complete CAS scoring for both case studies. Scores are ordinal (0 = no agility, 3 = full agility). Section 3 defines each dimension; Sections 4 and 5 justify each score. The composite is out of 27 following the addition of D9 (Identity Coupling).

#	Dimension	TLS	ZK Rollup	Key Contrast
D1	Substitutability	3	1	TLS ciphers are modular by spec. ZK verifier encodes the algorithm in bytecode.
D2	Migration Cost	2	1	TLS is tooled and documented. ZK requires governance vote, audit, and ceremony.
D3	State & Interface Continuity	3	1	TLS sessions are ephemeral. ZK must validate historical proofs across verifier versions.
D4	Trust Surface Delta	2	0	TLS CA hierarchy is stable. ZK upgrade proxy introduces admin key trust.
D5	Downgrade Resistance	2	2	TLS 1.3 transcript binding. ZK has no negotiation surface but no active defense either.
D6	Automation Depth	3	2	TLS is fully automated. ZK proving is automated; verifier upgrade requires governance tx.
D7	Negotiation Mechanism	3	0	TLS has extensible cryptographically-bound negotiation. ZK has none.

#	Dimension	TLS	ZK Rollup	Key Contrast
D8	Cryptographic Isolation	2	0	TLS separates key exchange, auth, and symmetric. ZK field assumptions permeate the circuit.
D9	Identity Coupling	3	0	TLS identity lives in certificates, not primitive. Ethereum address = hash(pubkey); migration requires rederiving identity.
—	**Composite CAS**	**20/27**	**8/27**	TLS: High Agility. ZK Rollups: Critical Constraint. Gap is structural, not incidental.

Detailed scoring rationale is in Sections 4 and 5. Composite scoring and interpretation bands are in Section 3.4.

1 Introduction

Cryptographic primitives age. The deprecation of MD5 began formally in 2004; MD5 appeared in production certificates until 2012. SHA-1 was theoretically broken in 2005, practically broken in 2017, and required years of coordinated effort across certificate authorities, browsers, and server operators to remove. RSA-1024 was deprecated by NIST in 2011 and persisted in enterprise infrastructure for over a decade afterward. These are not stories of negligence. They are stories of migration friction — the gap between knowing a primitive must change and being architecturally capable of changing it.

This paper argues that migration friction is primarily an architectural property, not an operational one. Systems fail to migrate on schedule not because operators are inattentive, but because the cryptographic primitive is anchored at a layer of the system that was not designed to be changed. The central thesis is: cryptographic agility is determined by where in the system the primitive dependency lives. A system that anchors its cryptographic primitive at a negotiation layer: one designed for parameter agreement, achieves agility structurally. A system that anchors it at an execution layer achieves neither agility nor a clear path to it.

This thesis is not merely descriptive. It has prescriptive force: it tells designers where to locate cryptographic dependencies, and it tells evaluators what to look for when assessing migration readiness. The Cryptographic Agility Score (CAS) proposed in this paper is an operationalization of this thesis — a structured instrument for observing where and how severely a system is constrained. CAS does not replace the architectural insight; it makes it measurable.

Two developments make this analysis urgent. First, the post-quantum transition: NIST finalized ML-KEM (FIPS 203 [4]), ML-DSA (FIPS 204 [5]), and SLH-DSA (FIPS 205 [6]) in 2024, initiating the largest coordinated cryptographic migration in the history of deployed systems. The question is no longer whether systems will migrate, but whether they are architecturally capable of doing so. Second, blockchain and ZK proof systems have introduced a class of infrastructure that is structurally hostile to change: smart contracts are immutable by default, and ZK verifier contracts encode specific proof systems at the bytecode level. These systems are not merely

difficult to migrate — they represent a new category of agility problem that existing frameworks do not address.

1.1 Contributions

This paper makes the following contributions:

1. An architectural thesis: cryptographic agility is determined by the system layer at which the primitive is anchored. This claim is the paper's primary contribution; CAS is its operationalization.

2. A nine-dimension semi-formal evaluation framework (CAS) for measuring migration readiness in deployed cryptographic systems, with explicit criteria for each score level and a discussion of dimension independence.

3. A worked demonstration of the framework across two structurally dissimilar case studies: TLS cipher suite negotiation and ZK rollup proof system migration. This constitutes the first systematic comparison of these systems under a unified agility lens.

4. A distinction between architectural rigidity and mathematical rigidity in ZK systems — a distinction with direct implications for remediation strategy.

5. Five design principles for agility-aware protocol engineering, and a diagnostic application of the framework to Ethereum's post-quantum migration problem.

1.2 Scope and Non-Goals

The framework is semi-formal: dimensions are precisely defined with explicit criteria, but we do not prove formal security theorems about agility. CAS scores presented in the case studies represent the author's reasoned assessment against explicit criteria. Inter-rater consistency testing and empirical calibration against historical migration data are identified as future work. We frame this paper as a framework proposal with a worked demonstration, not a validated empirical study.

1.3 Paper Organization

Section 2 reviews related work. Section 3 presents the CAS framework. Sections 4 and 5 apply the framework to TLS and ZK rollups. Section 6 derives design principles. Section 7 applies the framework diagnostically to Ethereum's PQC migration. Section 8 concludes with limitations and open problems.

The core claim: cryptographic agility is an architectural property determined by where the primitive is anchored in the system. CAS operationalizes this. TLS anchors primitives at the negotiation layer and scores 20/27. ZK rollups anchor them at the execution layer — and face mathematical, not merely architectural, rigidity. Score: 8/27.

2 Background and Related Work

This section reviews prior work across four areas: the existing definition and treatment of cryptographic agility, TLS cipher suite negotiation, ZK proof systems as deployed infrastructure, and

post-quantum migration guidance. We identify the specific gaps motivating the framework in Section 3.

2.1 Cryptographic Agility: Existing Definitions and Limitations

RFC 7696 [2] is the most widely cited normative treatment of cryptographic agility in protocol design. It defines algorithm agility as the ability of a protocol to support multiple cryptographic algorithms and to negotiate which to use for a given session or operation. It is a valuable foundation but carries two limitations material to this work. First, it assumes negotiability: the protocol must have a session-establishment phase in which algorithm selection can occur. This assumption holds for TLS, SSH, and IKE. It does not hold for smart contracts, ZK verifier contracts, or consensus-layer cryptography. Second, RFC 7696 treats agility as a design guideline rather than a measurable property of deployed systems. It provides no scoring mechanism, no comparative vocabulary, and no way to assess how agile an existing system is.

NIST SP 800-131A [3] addresses cryptographic algorithm transitions for federal systems. NIST SP 1800-38 [23] provides PQC migration guidance. Neither document addresses governance-mediated migration, immutable contract state, or proof system substitution. They were designed for mutable, patchable enterprise IT environments.

Acar et al. [11] examine cryptographic agility as a software engineering concern at the library level, framing it as an API design problem. This is adjacent to our D8 (Cryptographic Isolation) but operates below the protocol level. Bernstein and Lange [10] argue that algorithm agility can itself be a vulnerability surface — a point we take seriously in D5 (Downgrade Resistance) and in our treatment of TLS's design tradeoffs. Their argument motivates TLS 1.3's deliberate reduction of cipher suite optionality.

To the best of our knowledge, no prior work proposes a multi-dimensional evaluation framework for cryptographic agility applicable to immutability-constrained systems, nor has any prior work articulated the architectural thesis — that agility is determined by the layer at which the primitive is anchored — in a form applicable to ZK proof systems.

2.2 TLS Cipher Suite Negotiation

TLS negotiates cryptographic parameters during the handshake. In TLS 1.2, the ClientHello carries a list of supported cipher suites; the server selects one. This architecture is highly flexible but created a large negotiation surface that was repeatedly exploited: POODLE [12] (2014) downgraded to SSL 3.0, FREAK [14] (2015) forced export-grade RSA, LOGJAM (2015) targeted weak Diffie-Hellman parameters. These attacks are evidence that high agility scores do not imply security — they demonstrate the agility-downgrade tradeoff directly.

TLS 1.3 [1] is a deliberate architectural response. The cipher suite list was pruned to five strong AEAD suites; export ciphers, RC4, and 3DES were removed from the specification entirely. The Finished message commits to the full handshake transcript, making parameter tampering detectable. A downgrade sentinel mechanism embeds a detectable signal when TLS 1.3 is downgraded to 1.2 by an active attacker. TLS 1.3 achieves high agility not by maximising optionality but by removing weak options and hardening the negotiation mechanism. This distinction matters for how we frame TLS in the case study: it is the best available baseline, not a perfect system. Its high CAS score reflects genuine architectural strengths; its history reflects real costs of the agility model it employs.

2.3 ZK Proof Systems as Deployed Infrastructure

Zero-knowledge proof systems allow a prover to convince a verifier of a statement's truth without revealing information beyond validity [22]. Several families are deployed in production. Groth16 [7] requires a circuit-specific trusted setup and is grounded in the hardness of pairing-based problems over BN254 — a curve vulnerable to Shor's algorithm. PLONK [8] introduced a universal trusted setup; variants including TurboPlonk and Plonky2 extend it with custom gates and recursion. STARKs [9] require no trusted setup and rely on hash function security, making them post-quantum secure under standard assumptions.

The critical architectural fact is this: in deployed ZK rollups, the proof system is not a configuration parameter. The verifier contract deployed onchain implements a specific verification algorithm at the bytecode level. Changing the proof system requires deploying a new verifier contract, executing a governance process, and managing the transition of historical proofs. This is not a software update. It is closer to a protocol replacement.

But the constraint runs deeper than architecture. A ZK circuit is a constraint system defined over a specific finite field. The field is not a parameter — it is present in every constraint in the system. A circuit built for BN254 is not a circuit that calls BN254; it is a circuit that exists in the algebraic universe defined by BN254's scalar field. This is the mathematical rigidity we distinguish from architectural rigidity in Section 5. It has no direct analogue in TLS or any other deployed cryptographic protocol.

2.4 Post-Quantum Migration

NIST finalized its first PQC standards in 2024: ML-KEM (FIPS 203 [4]), ML-DSA (FIPS 204 [5]), and SLH-DSA (FIPS 205 [6]). These primitives carry substantially larger key and signature sizes than their classical counterparts — ML-DSA signatures are approximately 2.4 KB versus 64 bytes for ECDSA — which has direct implications for blockchain systems where calldata is gas-priced.

For Ethereum specifically, the primary quantum vulnerability is ECDSA over secp256k1. Every transaction is signed with ECDSA; every account address is derived from an ECDSA public key via address = keccak256(pubkey)[12:]. This address derivation creates an identity coupling problem — migrating the signature scheme requires re-deriving account identities — that we model explicitly as D9 in the CAS framework. It is not captured by any existing agility treatment.

The harvest-now-decrypt-later (HNDL) threat is acute for long-lived smart contract state: escrows, DAOs, vesting contracts. An adversary recording current state can retroactively break its security guarantees once a quantum computer is available. This motivates treating PQC migration as an active engineering problem, not a deferred one.

Existing frameworks assume mutable systems and provide no scoring mechanism. TLS is the best available agility baseline — but its history of downgrade attacks shows the cost of the model. ZK circuits face mathematical rigidity that has no precedent in prior agility literature. Ethereum's address derivation creates an identity coupling problem that existing frameworks do not model.

3 The Cryptographic Agility Score (CAS) Framework

We propose the Cryptographic Agility Score (CAS) as an operationalization of the architectural thesis: cryptographic agility is determined by where in the system the primitive is anchored. CAS decomposes this into nine independently motivated dimensions, each scored 0–3 on an ordinal scale with explicit criteria. The composite score, out of 27, is a diagnostic indicator of migration friction and a structured way to identify which dimensions are the binding constraints.

We use an ordinal rather than cardinal scale for three reasons. First, the criteria that determine agility are qualitative properties of system architecture, not continuously variable quantities; forcing them onto a cardinal scale would introduce false precision. Second, ordinal scaling has precedent in systems evaluation under similar conditions — CVSS uses ordinal aggregation for vulnerability scoring with comparable epistemological constraints [27]. Third, the gap between TLS and ZK rollup scores is large enough to be robust to reasonable weighting variations, which we discuss in Section 3.4.

3.1 Design Principles of the Framework

CAS was designed around four principles. Dimensions must be independently motivated: each must capture a distinct aspect of agility not fully explained by another dimension. We address dimension coupling directly in the boundary notes accompanying each definition. Metrics must be criteria-bound: each score level must be defined by observable system properties, not subjective judgment. The framework must accommodate immutability: it must produce meaningful differentiated scores for systems that structurally resist change. The composite score must be diagnostic: a low score should identify the specific bottleneck dimensions, not merely signal overall difficulty.

3.2 Dimensions and Metrics

The nine dimensions are defined below. Each includes a formal definition, rationale for independent inclusion, a boundary note clarifying the distinction from the most closely adjacent dimension, and a four-level scoring rubric.

D1. Substitutability

Definition. The degree to which a cryptographic primitive can be replaced with an alternative without requiring redesign of the protocol logic that depends on it.

Rationale. Substitutability is the most direct expression of agility. A system with high substitutability treats the primitive as a modular component behind a stable interface; low substitutability means the primitive is baked into the protocol logic and changing it requires rewriting dependent components.

Boundary with adjacent dimensions. D1 asks whether substitution is architecturally possible. D8 (Cryptographic Isolation) asks how wide the blast radius is when substitution occurs. A system can score high on D1 (substitution is defined) and low on D8 (it still requires touching 40 components). Both matter independently.

Score	Criteria
0	The primitive is hardcoded into the protocol logic. Substitution requires full protocol redesign with no abstraction layer between the primitive and its dependents.
1	Substitution is theoretically possible but requires significant engineering effort and redeployment of core components. No defined migration path exists.
2	A migration path exists and is documented. Substitution requires coordination and redeployment but not protocol redesign.
3	The primitive is fully abstracted behind a stable interface. Substitution is a configuration or parameter change with no impact on dependent logic.

D2. Migration Cost

Definition. The total coordination overhead required to execute a cryptographic migration, including governance processes, tooling, redeployment procedures, and key rotation operations.

Rationale. Even when substitution is architecturally possible, migration may be practically infeasible due to coordination costs: governance votes, security audits, key ceremonies, operator coordination. Key rotation is included as a sub-metric because it is the most frequent migration event in practice.

Boundary with adjacent dimensions. D2 measures process overhead: governance, audits, ceremonies, and operator coordination. D6 (Automation Depth) measures the technical automation of the migration's execution steps. A system can have a lightweight governance process (high D2) but require manual technical execution (low D6), or vice versa.

Score	Criteria
0	Migration requires multi-party governance with no defined process, multiple independent security audits, and manual coordination across heterogeneous operator sets with no tooling.
1	Migration has a defined process but requires significant governance coordination, at least one full security audit, and manual intervention by multiple independent parties.
2	Migration process is defined and tooled. Requires governance approval and audit but can be executed by a small coordinated team with documented procedures.
3	Migration is largely automated or requires only routine operational procedures. Governance overhead is minimal; tooling handles key rotation and redeployment.

D3. State and Interface Continuity

Definition. The degree to which a system can maintain the validity of historical cryptographic artifacts — signatures, proofs, commitments — and preserve dependent interface contracts after a migration.

Rationale. State continuity is a problem unique to systems with persistent cryptographic state. After migration, the system must still validate artifacts produced under the old primitive. Interface continuity extends this to downstream systems that consume the system's cryptographic outputs.

Boundary with adjacent dimensions. D3 concerns the treatment of cryptographic artifacts produced before migration. D7 (Negotiation Mechanism) concerns whether the protocol can select between primitives at runtime. The two are distinct: a system with no negotiation mechanism (D7=0) can still achieve high D3 by maintaining backward-compatible validation infrastructure.

Score	Criteria
0	Historical artifacts become invalid after migration. No mechanism exists to validate pre-migration state. Dependent interfaces break without independent migration.
1	Historical artifacts remain technically accessible but require a separate legacy validation path that is not officially maintained or audited.
2	A defined dual-validation period exists during which both old and new primitives are supported. Legacy artifacts remain valid for a specified window.
3	Historical artifacts remain fully valid indefinitely through a maintained backward-compatible validation layer. Dependent interfaces are unaffected.

D4. Trust Surface Delta

Definition. The change in the set of trust assumptions a system's users must accept as a result of a cryptographic migration.

Rationale. A migration that introduces new trusted parties, new governance keys, or new cryptographic assumptions expands the trust surface. Proxy upgrade patterns introduce admin keys that did not previously exist. This captures the security cost of agility mechanisms themselves.

Score	Criteria
0	Migration introduces new trusted parties or new unmitigated cryptographic assumptions. Trust surface is materially and persistently expanded.
1	Migration expands the trust surface in limited, documented ways. New assumptions are bounded and time-limited (e.g., a multisig with a defined sunset).
2	Migration does not introduce new trusted parties. New cryptographic assumptions are additive and publicly documented.
3	Migration is trust-neutral. The post-migration trust surface is identical to or strictly smaller than the pre-migration system's.

D5. Downgrade Resistance

Definition. The degree to which the system prevents an adversary from forcing use of a weaker cryptographic primitive than currently preferred.

Rationale. Agility and downgrade resistance are in fundamental tension: supporting multiple primitives for migration creates a surface on which an adversary might force selection of a weaker option. This tension is empirically documented in TLS's downgrade attack history. Note that systems with no negotiation mechanism trivially avoid negotiation-layer downgrade but also have no active defense.

Score	Criteria
0	The system has a negotiation mechanism exploitable by an active adversary to force weaker primitives. No downgrade protection exists.
1	Downgrade protection exists at the policy layer but can be circumvented by implementation flaws or misconfiguration.
2	Downgrade protection is enforced at the protocol layer through cryptographic binding. Requires active attack to circumvent.
3	Downgrade is structurally impossible. The protocol either does not negotiate, or negotiation is designed such that any modification is detectable and rejected.

D6. Automation Depth

Definition. The proportion of the migration process that can be executed programmatically without human intervention.

Rationale. Manual migration steps are security events. Human coordination introduces timing uncertainty, social engineering vectors, and execution risk. Automation depth also determines how quickly a system can respond to an emergency migration triggered by a newly discovered vulnerability.

Boundary with adjacent dimensions. D6 measures whether the technical steps of migration can be automated. D2 (Migration Cost) measures the governance and process overhead around those steps. A lightweight governance process (high D2) combined with manual technical execution (low D6) is a different failure mode from heavy governance (low D2) with good automation (high D6).

Score	Criteria
0	Migration requires manual intervention at every stage: governance, tooling, redeployment, and validation. No automation exists.
1	Some tooling exists for specific steps but coordination and decisions require significant manual intervention.
2	Migration tooling covers the majority of technical steps. Human intervention required only at defined governance decision points.
3	Migration is fully automated end-to-end, or human intervention is limited to a single well-defined approval step with no manual technical execution.

D7. Negotiation Mechanism

Definition. Whether the protocol has a native, in-band channel through which communicating parties can agree on which cryptographic primitive to use for a given session or operation.

Rationale. A negotiation mechanism is a structural prerequisite for online agility — the ability to transition primitives at runtime without a coordinated offline migration event. In its absence, all migration is necessarily offline and batch. This dimension is independent of D2: a system can have a well-defined governance process (low D2 cost) but still lack a protocol-level negotiation channel (D7=0).

Boundary with adjacent dimensions. D7 asks whether the protocol can select between primitives at runtime. D3 (State Continuity) asks whether historical artifacts remain valid after a migration has occurred. The two are orthogonal: D7 concerns the live negotiation channel; D3 concerns the treatment of pre-migration state.

Score	Criteria
0	No negotiation mechanism exists. Algorithm is fixed at deployment and can only be changed through an offline migration event.
1	A negotiation mechanism exists but is informal or out-of-band and not cryptographically bound.
2	A formal negotiation mechanism exists but is limited to a predefined static set of alternatives. Adding new alternatives requires protocol modification.
3	A fully extensible, cryptographically-bound negotiation mechanism exists. New primitives can be added and negotiated without protocol modification.

D8. Cryptographic Isolation

Definition. The degree to which cryptographic primitive dependencies are architecturally contained, such that a primitive substitution has a bounded and predictable blast radius across the system.

Rationale. Poor isolation is the hidden multiplier of migration cost. In a system where a hash function is called through a single interface, substitution requires changing one module. In a system where it is called in 40 components, substitution requires auditing each. In ZK circuits, poor isolation is particularly severe: field assumptions, hash functions, and curve parameters are embedded in the constraint system in ways invisible at the API level.

Boundary with adjacent dimensions. D8 asks about the scope of change required when substitution occurs. D1 (Substitutability) asks whether substitution is architecturally possible at all. A system can have a clear substitution path (D1=2) but require touching many components to execute it (D8=1). The distinction matters for estimating migration effort.

Score	Criteria
0	Primitive dependencies are pervasive and undocumented. No abstraction layer exists. Substitution scope is the entire system.
1	Some abstraction exists but dependencies leak across module boundaries. Substitution requires significant cross-cutting changes.
2	Primitives are largely contained behind defined interfaces. Substitution is bounded to a defined component set. A CBOM is feasible.
3	Complete isolation. All dependencies are routed through a single interface layer. Substitution is a one-point change.

D9. Identity Coupling

Definition. The degree to which persistent system identities — addresses, accounts, certificates, session handles — are cryptographically bound to the specific primitive being migrated.

Rationale. This dimension captures a failure mode absent from prior agility frameworks. When an identity is derived from or bound to a specific cryptographic primitive, migrating that primitive requires re-deriving the identity itself. In TLS, session identity is ephemeral and certificates can be reissued with new algorithms. In Ethereum, address = keccak256(pubkey)[12:]: the address is a direct function of the ECDSA public key. Changing the signature scheme requires a new key pair, which produces a new address — destroying all historical associations between the old address and its accumulated state, permissions, and social identity. This is a migration barrier with no analogue in any other dimension.

Score	Criteria
0	System identities are cryptographically derived from or bound to the primitive being migrated. Migration requires re-deriving all identities, destroying historical associations and accumulated state.
1	Identities are partially coupled. A migration path exists that preserves some identity associations, but requires a defined transition period and explicit user action.
2	Identities are indirectly coupled through a layer that can be migrated independently. Identity re-derivation is not required but a non-trivial migration step exists.
3	Identities are not bound to the primitive. Migration has no effect on existing identities, sessions, or accumulated state.

3.3 Dimension Independence

A legitimate concern for any multi-dimensional framework is whether the dimensions are genuinely independent or whether high correlation between them means the framework is overcounting. We address this directly.

Three pairs warrant explicit attention. D1 (Substitutability) and D8 (Isolation) are related but measure different things: D1 is a binary question about architectural possibility, D8 is a quantitative question about blast radius. A system can have D1=3 (a defined substitution interface) and D8=0 (50 undocumented callers of the primitive). They do not covary in general. D2 (Migration Cost) and D6 (Automation Depth) decompose migration friction along orthogonal axes: D2 captures process overhead (governance, audits, ceremonies) and D6 captures technical automation. A highly automated migration process (D6=3) can still carry heavy governance overhead (D2=1). D3 (State Continuity) and D7 (Negotiation Mechanism) are independent by construction: D3 concerns historical artifacts, D7 concerns live session negotiation. A system with no negotiation channel (D7=0) can still maintain historical artifacts through a backward-compatible validation layer (D3=2).

The remaining dimensions — D4 (Trust Surface Delta), D5 (Downgrade Resistance), D9 (Identity Coupling) — are not structurally adjacent to each other or to the above pairs. Some empirical

correlation is expected across well-designed systems (e.g., systems with good isolation tend to have lower migration cost) but this reflects real-world co-occurrence of good design, not logical dependence between the dimensions.

3.4 Composite Scoring, Weighting, and Sensitivity

The composite CAS is the unweighted sum across all nine dimensions, yielding a score out of 27. Equal weighting is the conservative default: any alternative weighting imposes a judgment about relative importance that varies by system and threat model. Practitioners applying the framework to a specific deployment should consider domain-appropriate weights with explicit justification.

The sensitivity question — whether the TLS vs. ZK rollup finding is robust to weighting variation — can be assessed directly. TLS scores 20/27 and ZK rollups score 8/27 under equal weights. The gap is 12 points. For this gap to reverse under alternative weights, ZK rollups would need to score significantly higher than TLS on at least some dimensions. They do not: the only dimension on which ZK rollups score equal to TLS is D5 (both score 2). On all other dimensions, TLS scores higher. The finding is robust to any non-degenerate positive weighting scheme.

Weighting Scenario	TLS Score	ZK Score	Gap	Interpretation
Equal weights (baseline)	20/27	8/27	+12	TLS High; ZK Critical
D7 weighted ×3 (negotiation emphasis)	26/33	8/33	+18	Gap widens
D5 weighted ×3 (security emphasis)	24/33	12/33	+12	Gap unchanged
D8 weighted ×3 (isolation emphasis)	26/33	8/33	+18	Gap widens
D5 weight 0 (ignore downgrade)	18/24	6/24	+12	Gap unchanged

Composite interpretation bands follow:

Score Range	Band	Interpretation
19 – 27	High Agility	Migration feasible with manageable coordination. T2M primarily determined by deployment logistics.
10 – 18	Moderate Agility	Migration feasible but requires significant effort. Low-scoring dimensions are primary bottlenecks.
0 – 9	Critical Constraint	Structural barriers to migration. T2M driven by architectural limitations requiring design-level intervention.

CAS has nine dimensions scored 0–3. Equal weighting is the default; the TLS/ZK gap is robust to all reasonable weighting variations. Three dimension pairs (D1/D8, D2/D6, D3/D7) are adjacent but independently motivated — boundary notes in Section 3.3 distinguish them. High CAS predicts low migration friction. It says nothing about cryptographic soundness.

4 Case Study I: TLS Cipher Suite Negotiation

TLS is the best available empirical example of protocol-level cryptographic agility. We use it as a comparative baseline, not as a model to be uncritically emulated. Its high CAS score reflects genuine architectural strengths; its history of downgrade attacks reflects the real cost of the negotiation model it employs. The case study is grounded primarily in TLS 1.3 (RFC 8446 [1]), with notes on TLS 1.2 where the contrast is instructive.

4.1 System Overview

TLS operates in two phases relevant to agility: the handshake and the record layer. The handshake is the agility mechanism — cipher suite selection is a handshake-layer decision. The record layer is algorithmically agnostic: it applies whatever algorithm the handshake selected.

In TLS 1.2, the ClientHello carries a list of supported cipher suites specifying key exchange, authentication, bulk cipher, and MAC algorithms. The server selects from this list. This architecture maximises optionality but the cipher suite list peaked at over 300 entries, many weak or legacy. POODLE [12], FREAK [14], and LOGJAM all exploited the negotiation surface to force weaker primitives. This is the empirical evidence that algorithm agility has a security cost.

TLS 1.3 [1] restructured negotiation. The cipher suite list was pruned to five strong AEAD suites. Key exchange, authentication, and symmetric cipher were decoupled into separate negotiation fields. The Finished message covers the full handshake transcript. A downgrade sentinel embeds a detectable signal when TLS 1.3 is forced to TLS 1.2. TLS 1.3 achieves high agility not by maximising options but by removing weak ones and hardening the negotiation mechanism — the exact design strategy P3 formalises in Section 6.

4.2 CAS Framework Application

D1 Substitutability — Score: 3/3

TLS cipher suites are modular by specification. Adding a new key exchange group requires only a new NamedGroup identifier and library implementation; the record layer is unaffected. The abstraction between cipher suite selection and record layer operation is formally specified and cleanly maintained. Score: 3.

D2 Migration Cost — Score: 2/3

TLS migration cost is driven by certificate reissuance and client/server update cycles, not by governance processes. Major library updates propagate through OS package management. Certificate migration depends on CA and browser vendor deprecation timelines, which operate on multi-year schedules. Key rotation is automated via ACME (RFC 8555 [24]). Score: 2 — well-tooled but certificate infrastructure introduces coordination overhead.

D3 State and Interface Continuity — Score: 3/3

TLS has no persistent cryptographic state in the relevant sense. Individual sessions are ephemeral; there are no historical proofs or signatures that must remain validatable after migration. Certificate revocation (CRL, OCSP) handles premature invalidation. The TLS API contract is stable across cipher suite migrations. Score: 3.

D4 Trust Surface Delta — Score: 2/3

Cipher suite migration does not introduce new trusted parties. The CA hierarchy is unaffected by algorithm changes. Introduction of new signature algorithms for certificate authentication requires CAs to generate new key material and clients to trust new certificate types — a limited, time-bounded expansion during the transition window. Score: 2.

D5 Downgrade Resistance — Score: 2/3

TLS 1.3 addresses downgrade directly through Finished message transcript binding and the downgrade sentinel mechanism. However, the sentinel depends on correct client implementation and does not prevent a TLS 1.2 session from being established with a legacy server. We score this a 2 rather than 3 because structural elimination of downgrade would require removing TLS 1.2 support entirely, which is an operational decision beyond the protocol's control. The TLS 1.2 era is historical evidence that score 2 is not guaranteed — it was achieved only through the deliberate reduction of optionality in TLS 1.3. Score: 2.

D6 Automation Depth — Score: 3/3

TLS negotiation is fully automated. No human intervention is required during cipher suite selection. Certificate renewal via ACME [24] is automated. Library updates distributing new cipher suite support use standard package management. The only manual steps are policy decisions (removing deprecated suites from configuration) and certificate reissuance, both well-tooled. Score: 3.

D7 Negotiation Mechanism — Score: 3/3

TLS has a fully extensible, cryptographically-bound negotiation mechanism. The extensions framework allows new capabilities to be advertised and negotiated without modifying the core protocol. Negotiated parameters are bound into the Finished message transcript hash, making negotiation tamper-evident. Score: 3.

D8 Cryptographic Isolation — Score: 2/3

TLS achieves reasonable isolation through its layered architecture: key exchange, authentication, and symmetric encryption are distinct algorithm slots with defined interfaces (e.g., EVP in OpenSSL). Isolation is not complete at the implementation level — some implementations couple cipher suites to optimised code paths (e.g., AES-NI). At the specification level the isolation is clean; at the implementation level it is moderate. Score: 2.

D9 Identity Coupling — Score: 3/3

TLS session identity is ephemeral; no session identity persists between handshakes. Long-term identity lives in certificates, which can be reissued with new signature algorithms while the subject name and organisational identity remain stable. The identity is not derived from the primitive; it is attested by the primitive. Migrating the signature algorithm requires certificate reissuance, not identity re-derivation. Score: 3.

4.3 Composite Score and Interpretation

TLS 1.3 achieves a composite CAS of 20/27, firmly in the High Agility band. Four dimensions score below 3: D2 (Migration Cost, 2), D4 (Trust Surface Delta, 2), D5 (Downgrade Resistance, 2), and D8 (Cryptographic Isolation, 2). None represents a structural barrier; all are operational constraints with known mitigation paths.

The critical insight is not that TLS scores well, but why. Algorithm selection occurs at the handshake layer — the protocol moment specifically designed for parameter agreement. The record layer is algorithmically agnostic. This colocation of agility mechanism and negotiation layer is the structural source of TLS's high score across D1, D6, D7, and D9. It is also the property that ZK rollups most conspicuously lack.

It is worth restating: TLS's high agility score did not come free. The optionality that enables high D1 and D7 scores was also the attack surface exploited by POODLE, FREAK, and LOGJAM. TLS 1.3's improvement on D5 came at the deliberate cost of reduced optionality. The framework captures this tradeoff rather than obscuring it.

TLS 1.3 scores 20/27. Its high agility is structural — primitives are negotiated at the handshake, not encoded in the execution layer. Its history shows the cost of the model: high optionality enabled downgrade attacks until TLS 1.3 eliminated weak options. The score reflects genuine strengths. The history is an honest accounting of the tradeoffs.

5 Case Study II: ZK Rollup Proof System Migration

ZK rollups are the most cryptographically constrained production systems currently in operation. This section applies the CAS framework to proof system migration in ZK rollups, drawing on Polygon zkEVM [19], zkSync Era [20], and StarkNet [21] as representative instances. We argue that low scores here reflect two distinct phenomena: architectural rigidity, which is a consequence of design choices that could in principle be changed, and mathematical rigidity, which is a consequence of how ZK proof systems are constructed and has no direct analogue in any other deployed system.

5.1 System Overview

A ZK rollup achieves L2 scalability by batching transactions offchain, generating a cryptographic proof of correct execution, and submitting that proof to an L1 verifier contract. The verifier contract is the cryptographic anchor: it checks proof validity and finalises the corresponding state transition on L1. The rollup's entire security guarantee rests on the soundness of this verification step.

The verifier contract encodes a specific proof verification algorithm at the bytecode level. A Groth16 [7] verifier implements pairing checks over BN254. A STARK [9] verifier implements FRI-based polynomial commitment verification. These are not interchangeable. Changing the proof system requires deploying a new verifier contract, executing a governance upgrade, migrating the prover infrastructure, and resolving the treatment of proofs generated under the old system during the transition window.

5.2 CAS Framework Application

D1 Substitutability — Score: 1/3

Proof system substitution requires deploying an entirely new onchain verifier contract and a governance process to update the bridge to recognise it. There is no abstraction layer between the proof system and the verifier — the algorithm is the contract. Polygon's [19] incremental verifier upgrades demonstrate that substitution is technically possible, but each migration is effectively a from-scratch deployment with no defined reusable interface. Score: 1.

D2 Migration Cost — Score: 1/3

Migration requires a full security audit of the new verifier circuit (typically $200K–$500K and several months), a governance process involving the rollup's multisig or DAO, coordination with L1 bridge contract maintainers, and a new trusted setup ceremony for Groth16-based [7] systems. No rollup has completed a full proof system migration in production. Score: 1.

D3 State and Interface Continuity — Score: 1/3

After migration, the new verifier cannot verify proofs generated under the old system. A mandatory dual-validation period requires maintaining both contracts; the bridge must know which verifier to call for a given batch. No rollup currently has a specified protocol for this transition. The problem is acknowledged in the engineering community but no maintained solution exists at the protocol level. Score: 1.

D4 Trust Surface Delta — Score: 0/3

Verifier contract upgrades are gated behind proxy upgrade patterns requiring a privileged admin key — typically a multisig. Pre-migration, users trust only the cryptographic construction. Post-migration, users also trust the integrity of the multisig key holders. StarkNet [21] and Polygon zkEVM [19] have governance timelocks as partial mitigations, which reduce the risk but do not eliminate the structural trust surface expansion. Score: 0.

D5 Downgrade Resistance — Score: 2/3

ZK rollups have no negotiation mechanism and therefore cannot be subjected to negotiation-layer downgrade attacks. A prover cannot be forced to submit a weak proof because the verifier contract is fixed and will reject non-conforming proofs. However, a governance-level compromise of the upgrade multisig could replace the verifier with one implementing a weaker proof system

— a downgrade of the most serious kind. We score 2: resistant to protocol-level downgrade; vulnerable to governance-level downgrade. Score: 2.

D6 Automation Depth — Score: 2/3

Proof generation is highly automated — the prover is an offchain software system that generates proofs without human intervention, and replacing the prover software is an operational change. The verifier contract deployment and governance transaction are not automated. Score: 2 — proving infrastructure is automated; verifier upgrade governance is not.

D7 Negotiation Mechanism — Score: 0/3

ZK rollups have no in-protocol mechanism for negotiating proof system selection. The proof system is fixed at verifier contract deployment. There is no session establishment phase, no prover advertisement of supported systems, and no verifier selection logic. All proofs must conform to the single algorithm in the current verifier. Score: 0.

D8 Cryptographic Isolation — Score: 0/3

This dimension receives its lowest score for ZK circuits — and the reason is mathematical, not merely architectural. A ZK circuit is a constraint system defined over a specific finite field. The field is not a parameter; it is present in every constraint. A circuit for BN254 exists in the algebraic universe defined by BN254's scalar field [7, 8]. Every field arithmetic operation, every hash gadget, every elliptic curve operation in the circuit is an expression within that universe. Changing the field requires re-expressing every constraint. This is not a matter of poor modular design — it is a property of how constraint systems are constructed. Score: 0.

D9 Identity Coupling — Score: 0/3

Ethereum account addresses are derived as address = keccak256(pubkey)[12:]. The address is a direct function of the ECDSA public key. Migrating the signature scheme requires generating a new key pair, which produces a new address. All historical state, token balances, permissions, contract ownership, and social identity associated with the old address must be explicitly migrated to the new address — a process that requires user action and has no atomic execution guarantee. For smart contracts, the situation is more complex: a contract's address is derived from its deployment transaction and cannot be changed without redeployment, and redeployment does not inherit the original contract's state. Score: 0.

5.3 Architectural Rigidity vs. Mathematical Rigidity

The case study exposes a distinction not captured in prior agility literature: the difference between architectural rigidity and mathematical rigidity. This distinction matters because the two phenomena have different causes and require different remediation strategies.

Architectural rigidity in ZK rollups arises from design choices that could, in principle, be made differently. The verifier contract being the agility bottleneck is an architectural choice: the proof system is encoded in the execution layer rather than abstracted above it. The absence of a verifier registry is an architectural gap: no mechanism exists to map batch identifiers to verifier contract addresses, which would allow multiple proof systems to coexist. The lack of a dual-validation protocol for historical proofs is an architectural oversight: the problem is known but no specification exists. These are hard problems, but they are engineering problems. They admit engineering solutions.

Mathematical rigidity is categorically different. A ZK circuit is not software that calls a cryptographic library. It is a constraint system that exists within a specific algebraic universe: a finite field, an elliptic curve, a hash function optimised for that field. Changing the proof system is not a module swap — it is a re-expression of the computation in a different algebraic universe. The field size determines the size of every arithmetic operation. The hash function determines the structure of every Merkle proof gadget. The elliptic curve determines every signature verification constraint. None of these are parameters that can be changed without rewriting the circuit from scratch.

The practical implication is that D8 (Cryptographic Isolation) is not merely low for ZK circuits — it is structurally near-zero for any ZK system built on current proof system designs. Improving D8 for ZK rollups requires not better software engineering but new proof system research: proof systems that are algebraically agnostic, or that can compile circuits to multiple target fields without manual re-specification. This is an active research area but no production system currently exhibits it.

This distinction also reframes the critique of ZK rollup agility. Low CAS scores in ZK rollups are not primarily evidence of poor engineering decisions (though architectural rigidity is addressable). They are evidence that current ZK proof system designs do not admit cryptographic agility as a design property. The framework is measuring a real constraint, not penalising a correctable oversight.

5.4 Steelman: The ZK Engineer's Position

A ZK system designer presented with the above analysis would offer a legitimate response: the verifier contract is onchain not because of inattention to agility but because of verification cost, proof size, and gas economics. Onchain verification provides trustless finality — any full node can independently verify the proof without trusting the rollup operator. Moving the verification algorithm offchain or behind an abstraction layer would reintroduce trust assumptions the system was designed to eliminate.

This is a valid design constraint and we acknowledge it directly. The CAS framework is not arguing that ZK rollups should sacrifice security for agility. It is arguing that the current architecture has made a design choice — encoding the algorithm at the execution layer — that makes cryptographic migration structurally difficult, and that this choice has costs that should be explicitly accounted for in system design and risk assessment. The verifier registry pattern proposed in Section 6 preserves onchain verification while adding an abstraction layer above the algorithm identity. It is not a security regression; it is a structural improvement that separates what is verified from how it is verified.

5.5 Comparative Summary

#	Dimension	TLS	TLS Notes	ZK	ZK Notes
D1	Substitutability	3	Modular by specification; clean interface separation.	1	Algorithm is the contract; no abstraction layer; full redeployment required.
D2	Migration Cost	2	Tooled; primary cost is certificate coordination.	1	Audit, governance vote, ceremony, bridge migration; no automated path.

#	Dimension	TLS	TLS Notes	ZK	ZK Notes
D3	State & Interface Continuity	3	Ephemeral sessions; no persistent cryptographic state.	1	Old verifier must validate historical proofs; no maintained dual-validation protocol.
D4	Trust Surface Delta	2	CA hierarchy stable; no new trusted parties.	0	Proxy upgrade introduces admin key; users must trust multisig holders.
D5	Downgrade Resistance	2	Transcript binding; downgrade sentinel; no weak suites in TLS 1.3.	2	No negotiation surface; but governance compromise could replace verifier.
D6	Automation Depth	3	Fully automated; ACME [24] for certificates.	2	Proving automated; verifier upgrade requires manual governance transaction.
D7	Negotiation Mechanism	3	Extensible, cryptographically-bound extensions framework.	0	No negotiation mechanism; algorithm fixed at deployment.
D8	Cryptographic Isolation	2	Layered spec; moderate implementation coupling.	0	Field assumptions permeate every circuit constraint; mathematical rigidity.
D9	Identity Coupling	3	Certificates reissuable; identity not derived from primitive.	0	address = hash(pubkey); migration destroys address identity and accumulated state.
	Composite CAS	20/27	High Agility. T2M: weeks to months.	8/27	Critical Constraint. T2M: years.

ZK rollups score 8/27. Six dimensions score 0 or 1. The critical distinction: some of these low scores (D1, D2, D3, D7) reflect architectural rigidity that engineering can address. D8 and D9 score 0 due to mathematical rigidity — a property of how ZK proof systems are constructed, not how they are deployed. These require different remediation strategies.

6 Design Principles for Agility-Aware Protocol Engineering

The following five principles are derived from the structural differences observed in the case studies. They are transferable to any system that must eventually migrate its cryptographic primitives. Each principle is grounded in the CAS analysis and motivated by specific patterns of high or low scoring.

P1: Colocate Agility Mechanisms with Negotiation, Not Execution

Algorithm selection must occur at the protocol layer designed for parameter agreement, not at the execution layer. TLS achieves agility because cipher suite selection is a handshake-layer

decision; the record layer is agnostic. ZK rollups face agility constraints because the proof system is encoded at the execution layer (the verifier contract), which has no negotiation channel. This is the central structural finding of the comparative analysis.

The practical implication for ZK rollup design is a verifier registry: an onchain directory mapping proof system identifiers or batch ranges to verifier contract addresses, with the bridge routing proof verification to the appropriate verifier per batch. This preserves onchain verification (the security property) while separating algorithm identity from algorithm execution (the agility requirement). A registry architecture improves D1, D3, and D7 without security regression.

P2: Treat State Continuity as a First-Class Design Constraint

Systems with persistent cryptographic state must specify a backward-compatibility protocol before they need it. The absence of a maintained dual-validation mechanism in current ZK rollups means that proof system migration carries undefined risk to historical batch validity. State continuity (D3) should be a documented protocol invariant, not an informal operational practice.

Concretely: a ZK rollup specification should include a proof archive policy — a commitment to maintaining the old verifier contract as callable for a specified period with a defined sunset — as a protocol-level guarantee. This is cheap to specify before a migration event and extremely expensive to retrofit during one.

P3: Minimise Governance Surface in the Migration Path

Every human coordination step in a migration is a liveness risk and a social engineering vector. The design goal is not to eliminate governance — some migrations require human judgment — but to reduce governance to a single well-defined approval step with all technical execution automated around it. TLS achieves this: negotiation is automatic, certificate renewal is automated via ACME [24], and cipher suite updates propagate through package management. ZK rollup verifier upgrades require a multisig governance transaction that cannot currently be reduced to an automated step.

This principle has a corollary relevant to emergency migrations: a system whose migration requires three independent parties to coordinate on a specific schedule is a system that cannot respond quickly to a newly discovered vulnerability. Migration paths should be designed with emergency cadence in mind, not just planned migration cadence.

P4: Design for Parallel Operation During Transition Windows

Cryptographic migration should never require a hard cutover. Hard cutovers — moments at which the old primitive is deactivated and the new one simultaneously activated — are the highest-risk events in a migration timeline: partial deployment, implementation bugs, or coordination failures at that moment can cause complete system failure. TLS avoids this by supporting multiple cipher suites in parallel. ZK rollups currently require a hard cutover to the new verifier.

The verifier registry pattern (P1) directly enables parallel operation: multiple verifier contracts can coexist at different registry entries. New batches are submitted to the new verifier; historical batches remain resolvable through the old verifier. Migration is a gradual shift, not a simultaneous swap.

P5: Enumerate Cryptographic Dependencies Before Designing Migration Paths

A migration plan that does not enumerate all primitive dependencies will encounter unanticipated blast radius. In ZK circuits, dependencies are pervasive and often implicit: field size, hash function, elliptic curve, and pairing assumptions are embedded in the constraint system at a level not visible at the API surface. Before designing a migration path, produce a Cryptographic Bill of Materials (CBOM) — a comprehensive inventory of every component that depends on the primitive, its dependency type (direct, structural, or implicit), and the change required for each in a migration scenario.

CBOM is an emerging practice being formalised through NIST's cryptographic discovery efforts [23]. For ZK circuits, CBOM generation requires tooling that does not yet broadly exist — automatically enumerating which constraints depend on which field assumptions is an open engineering problem. Its absence is itself a migration risk.

Five principles: negotiate at the handshake layer, not the execution layer (P1); specify state continuity before you need it (P2); minimise governance steps, design for emergency cadence (P3); never require hard cutover (P4); enumerate all cryptographic dependencies before planning migration (P5).

7 Implications for Post-Quantum Migration on Immutable Ledger Systems

The post-quantum transition applies NIST's 2024 finalization of ML-KEM, ML-DSA, and SLH-DSA to a class of systems — immutable ledger systems — for which no migration guidance exists. This section applies the CAS framework diagnostically to Ethereum as a representative case [18], identifies which dimensions are the binding constraints for each proposed migration path, and derives targeted recommendations.

7.1 The Quantum Threat Surface on Ethereum

Ethereum's primary quantum vulnerability is ECDSA over secp256k1. Every transaction is signed with ECDSA; every account address is address = keccak256(pubkey)[12:]. Shor's algorithm [25] breaks the discrete logarithm problem underlying secp256k1, enabling signature forgery and account impersonation. The identity coupling problem (D9) makes this migration uniquely difficult: unlike TLS certificate replacement, Ethereum address migration requires re-deriving the identity itself, with no atomic mechanism for transferring accumulated state.

A secondary vulnerability affects ZK rollups using pairing-based proof systems (Groth16 [7], PLONK [8] over BN254): their security rests on the hardness of the discrete logarithm in BN254, which Shor's algorithm [25] breaks. STARK-based systems [9] (StarkNet [21]) are not affected; their security relies on hash function collision resistance. Keccak-256 is weakened only quadratically by Grover's algorithm [26], reducing effective security from 256 to 128 bits — generally acceptable for current threat models.

The harvest-now-decrypt-later threat is acute for long-lived smart contract state. An adversary recording current transactions can retroactively break their authentication once a quantum computer is available. This motivates treating PQC migration as an active engineering problem with a non-zero urgency today, not a future-dated one.

7.2 CAS Diagnostic: Ethereum PQC Migration Paths

We assess three migration paths against the CAS dimensions most relevant to the quantum threat.

7.3 Path A: EIP-7702 and Account Abstraction

EIP-7702 [15] and EIP-4337 [16] enable programmable signature validation: an account can verify ML-DSA or SLH-DSA signatures by deploying a validation contract implementing the post-quantum algorithm. This substantially improves D1 (substitutability becomes per-account configuration) and D7 (signature scheme selection is a validation contract parameter). D8 improves because verification logic is isolated in a dedicated contract. However, D9 (Identity Coupling) remains unresolved: ECDSA-derived addresses still anchor account identity. Users adopting account abstraction can verify PQC signatures, but their address is still derived from an ECDSA key, retaining the quantum vulnerability for the identity itself. This path is opt-in and does not migrate the protocol-level ecrecover precompile.

7.4 Path B: New Precompiles for Post-Quantum Signature Verification

Adding ML-DSA and SLH-DSA precompiles analogous to ecrecover [17] improves D1 for smart contracts (which can call the new precompile) and, if ecrecover is maintained, achieves reasonable D3 (historical ECDSA signatures remain verifiable). D9 remains unaddressed: addresses are still ECDSA-derived. D7 improves marginally — contracts can choose which precompile to call — but the protocol itself still has no negotiation mechanism. This path is additive: it does not replace ECDSA, it supplements it. The long-term D9 problem is deferred rather than solved.

7.5 Path C: Protocol-Level Signature Scheme Migration via Hard Fork

A hard fork replacing ECDSA in transaction signing is the only path that fully addresses D9: if accounts are migrated to a new address scheme derived from a PQC key, the identity coupling is resolved. D2 (Migration Cost) is extreme — coordinating a validator-level hard fork affecting every Ethereum user is the highest-cost migration event in Ethereum's history. D3 requires a specified transition protocol that explicitly does not yet exist. The key migration problem — every user must generate a new PQC key pair and migrate their account before a transition deadline — is a coordination problem at the scale of all Ethereum users, with no precedent.

Applying P2 (state continuity as first-class constraint) and P3 (minimise governance surface) to Path C: this path should be specified now, even if execution is years away. The transition protocol — the dual-acceptance window, the sunset timeline, the account migration mechanism — should be a ratified specification before any implementation begins. The absence of this specification is currently the primary D3 risk for Ethereum's PQC migration.

7.6 Recommendations

Deploy account abstraction as the near-term cryptographic agility layer. EIP-7702 [15] and EIP-4337 [16] should be framed explicitly as agility infrastructure for Ethereum accounts, not merely as UX improvements. This enables PQC signature schemes to be deployed and tested at scale before any protocol-level change.

Add PQC precompiles early to reduce future migration cost. Deploying ML-DSA and SLH-DSA precompiles now reduces D2 for a future hard fork by establishing gas cost baselines, implementation experience, and application developer familiarity before the migration is urgent.

Specify the Path C transition protocol before beginning execution. Following P2, the dual-acceptance window, account migration mechanism, and ECDSA sunset timeline should be ratified specifications. The identity re-derivation problem (D9) requires a protocol design that handles accumulated state atomically or provides an explicit migration contract — neither of which currently exists.

Treat pairing-based ZK rollup PQC migration as a separate and harder problem. Rollups using Groth16 or PLONK over BN254 face a quantum threat not addressed by account-level PQC migration. The architectural and mathematical rigidity constraints from Section 5 apply in full. Migration to STARK-based proof systems is the most direct remediation and should be treated as a proof system migration event subject to the full CAS analysis.

Ethereum's quantum exposure: ECDSA (all paths) and pairing-based ZK proofs (Groth16/PLONK rollups). Account abstraction addresses D1 and D7 but not D9. New precompiles improve D1 and D3 but defer D9. Only a hard fork resolves D9 — and it requires a transition protocol that doesn't yet exist. STARK rollups are already PQC-secure; pairing-based rollups are not.

8 Conclusion

This paper proposed and demonstrated the Cryptographic Agility Score (CAS) as an operationalization of a central architectural thesis: cryptographic agility is determined by the system layer at which the primitive is anchored. Systems that anchor primitives at negotiation layers achieve agility structurally; systems that anchor them at execution layers do not.

Applying CAS to TLS and ZK rollups produced a composite score gap of 12 points (20/27 versus 8/27) that is robust to all reasonable weighting variations. The gap is not primarily a commentary on engineering quality. TLS achieves high agility because its architecture separates negotiation from execution by design; ZK rollups face critical constraints because their proof systems are expressions of fixed algebraic universes that permeate the constraint system — a form of mathematical rigidity with no precedent in prior cryptographic agility literature.

The distinction between architectural rigidity (addressable through the verifier registry pattern and related mechanisms) and mathematical rigidity (requiring new proof system research for algebraically-agnostic constructions) is the paper's deepest finding. It changes the remediation question from 'how do we engineer better upgrade paths?' to 'what would a proof system designed for agility look like?'

Five design principles derived from the comparison provide actionable guidance: locate agility mechanisms at the negotiation layer, specify state continuity before it is needed, minimise governance surface in migration paths, design for parallel operation rather than hard cutover, and enumerate cryptographic dependencies before planning migration.

8.1 Limitations

CAS scoring is ordinal, not cardinal. Composite scores support comparison and bottleneck identification; they do not produce quantitative migration effort estimates. Equal weighting is a conservative default; specific deployment contexts may justify alternative weights. The framework does not model adversarial scenarios during the migration process itself — the window of dual-primitive operation creates an expanded attack surface that CAS does not capture.

Case study scores represent the author's reasoned assessment against explicit criteria. No inter-rater consistency testing has been performed. Scores are applied to publicly documented system architectures; implementation details not captured in public materials may affect results. ZK rollup architecture is evolving rapidly; some constraints identified here may be addressed by developments after this writing.

8.2 Open Problems

Several open problems are identified for future work. A formal security definition of cryptographic agility — capturing it as a game-based property resistant to adversarial manipulation of the migration process — would provide the theoretical foundation this framework lacks. Empirical calibration of CAS against historical migration data (TLS 1.2 to 1.3 adoption curves, SHA-1 deprecation timelines, Ethereum hard fork coordination history) would test whether composite scores predict Time-to-Migrate quantitatively. The verifier registry pattern requires formal specification and security analysis before deployment. CBOM tooling for ZK circuits — automated enumeration of implicit field, curve, and hash function dependencies in constraint systems — is an open engineering problem whose absence is itself a migration risk. Finally, the question of what a mathematically agile proof system would look like — one that can be compiled to multiple target fields without manual re-specification — is an open research problem in proof system design.

9 References

References

[1] Rescorla, E. (2018). The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. Internet Engineering Task Force.

[2] Housley, R. (2015). Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms. RFC 7696. Internet Engineering Task Force.

[3] Barker, E. and Roginsky, A. (2019). Transitioning the Use of Cryptographic Algorithms and Key Lengths. NIST Special Publication 800-131A Rev. 2.

[4] National Institute of Standards and Technology. (2024). Module-Lattice-Based Key-Encapsulation Mechanism Standard. FIPS 203.

[5] National Institute of Standards and Technology. (2024). Module-Lattice-Based Digital Signature Standard. FIPS 204.

[6] National Institute of Standards and Technology. (2024). Stateless Hash-Based Digital Signature Standard. FIPS 205.

[7] Groth, J. (2016). On the Size of Pairing-Based Non-interactive Arguments. EUROCRYPT 2016, LNCS 9666, pp. 305–326.

[8] Gabizon, A., Williamson, Z. J., and Ciobotaru, O. (2019). PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint 2019/953.

[9] Ben-Sasson, E., Bentov, I., Horesh, Y., and Riabzev, M. (2018). Scalable, Transparent, and Post-Quantum Secure Computational Integrity. Cryptology ePrint 2018/046.

[10] Bernstein, D. J. and Lange, T. (2017). Post-quantum cryptography. Nature, 549(7671), 188–194.

[11] Acar, T., Belenkiy, M., Bellare, M., and Cash, D. (2010). Cryptographic Agility and Its Relation to Circular Encryption. EUROCRYPT 2010, LNCS 6110, pp. 403–422.

[12] Möller, B., Duong, T., and Kotowicz, K. (2014). This POODLE Bites: Exploiting the SSL 3.0 Fallback. Google Security Advisory.

[13] Beurdouche, B. et al. (2015). A Messy State of the Union: Taming the Composite State Machines of TLS. IEEE Symposium on Security and Privacy, pp. 535–552.

[14] Adrian, D. et al. (2015). Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. ACM CCS 2015, pp. 5–17.

[15] Ethereum Improvement Proposal 7702. (2024). Set EOA Account Code for One Transaction. Ethereum Foundation.

[16] Ethereum Improvement Proposal 4337. (2021). Account Abstraction Using Alt Mempool. Ethereum Foundation.

[17] Ethereum Improvement Proposal 3. (2015). Addition of CALLDEPTH opcode / ecrecover precompile. Ethereum Foundation.

[18] Buterin, V. (2023). Ethereum post-quantum roadmap considerations. Ethereum Research Forum.

[19] Polygon zkEVM Technical Documentation. (2023). Polygon Technology.

[20] zkSync Era Documentation. (2023). Matter Labs.

[21] StarkNet Documentation. (2023). StarkWare Industries.

[22] Boneh, D. and Shoup, V. (2023). A Graduate Course in Applied Cryptography. Version 0.6.

[23] National Institute of Standards and Technology / NCCoE. (2023). Migration to Post-Quantum Cryptography: Cryptographic Discovery. NIST SP 1800-38B (Preliminary Draft).

[24] Barnes, R. et al. (2019). Automatic Certificate Management Environment (ACME). RFC 8555.

[25] Shor, P. W. (1994). Algorithms for Quantum Computation: Discrete Logarithms and Factoring. FOCS 1994, pp. 124–134.

[26] Grover, L. K. (1996). A Fast Quantum Mechanical Algorithm for Database Search. STOC 1996, pp. 212–219.

[27] Chen, L. et al. (2016). Report on Post-Quantum Cryptography. NIST Internal Report 8105.