The Decentralized Cloud: Providers as Relays, Storage, and Compute Markets

Service Markets with User-Owned Keys and Exit Rights

---

on Lux Network

Zach Kelling

Lux Industries

2025

Abstract

We present the Lux provider model, a decentralized cloud architecture in which infrastructure providers operate as service markets rather than data custodians. In the centralized cloud, the provider is the owner: it holds user keys, controls data at rest, and imposes switching costs that make exit prohibitively expensive. The Lux model inverts this relationship. Users hold their own encryption keys in capsule vaults. Providers compete on relay latency, storage durability, compute throughput, and recovery guarantees—never on data lock-in. Every service interaction produces a chain receipt on the Lux P-Chain, creating a tamper-evident audit trail for SLA enforcement without requiring trust in any single provider. We formalize five provider roles (relay, storage, gateway, recovery, compute), define a stake-weighted reputation protocol that measures provider quality without surveilling user traffic, introduce a bid/ask market for inference, storage, and bandwidth with on-chain settlement, and prove that the provider portability guarantee—users can switch providers without data loss or service interruption—holds under Byzantine fault assumptions with up to $f < n/3$ malicious providers. We compare the economic and sovereignty properties of the Lux provider model against AWS, GCP, Vercel, and Fly.io, demonstrating that decentralized service markets achieve comparable performance with strictly superior user sovereignty and exit rights.

1 Introduction

The cloud computing industry is built on a fundamental inversion of ownership. When a developer deploys an application to AWS, GCP, or Azure, the provider assumes custodial control over the application's data, encryption keys, network identity, and operational continuity. The user pays for access to their own data through the provider's proprietary APIs. Switching providers requires rewriting application code, migrating data through provider-controlled export tools (when they exist), and accepting downtime measured in days or weeks. This is not a service market. It is a custody arrangement.

The consequences are well-documented. Amazon Web Services holds approximately 31% of global cloud infrastructure spend [1]. Google Cloud and Microsoft Azure hold a combined 33%. Together, three companies control the operational infrastructure for the majority of the internet's applications. When AWS experiences an outage, large fractions of the internet become unavailable [2]. When a provider decides to change pricing, deprecate a service, or restrict access, customers have no recourse except migration—which the provider has made as expensive as possible.

This paper presents an alternative: the Lux provider model, in which cloud infrastructure providers are service markets, not data owners. The key insight is a separation of concerns:

(i) **Users own their keys.** All data is encrypted client-side with keys stored in capsule vaults. No provider ever holds plaintext data or decryption keys.

(ii) **Providers compete on service quality.** Relay latency, storage durability, compute throughput, and recovery speed are the competitive dimensions—not data lock-in.

(iii) **Exit is a first-class right.** Users can switch providers at any time without data loss, because providers never hold the only copy of user data in a format only they can read.

(iv) **SLAs are chain-enforced.** Service level agreements are encoded as smart contracts with stake-backed penalties for violations, verified by on-chain receipts.

The remainder of this paper formalizes this model. Section 2 defines the five provider roles. Section 3 introduces the reputation protocol. Section 4 describes SLA enforcement. Section 5 defines the compute and resource markets. Section 6 proves the portability guarantee. Section 7 compares against centralized clouds. Section 8 presents the economic model. Section 9 discusses related work. Section 10 concludes.

2 Provider Model

The Lux decentralized cloud defines five distinct provider roles. Each role represents a separable service that can be operated independently, composed with other roles, or bundled by a single operator. The critical property is that no single role requires custodial access to user data.

2.1 Relay Providers

A relay provider routes encrypted messages between capsules and between capsules and external services. Relays operate on ciphertext exclusively—they forward bytes without the ability to inspect, modify, or selectively drop content (detectable via receipt verification).

Definition 1 (Relay Provider)**.** *A relay provider $\mathcal{R}$ is a network node that accepts encrypted payloads $\mathsf{Enc}(k, m)$ from source capsule C_s and delivers them to destination capsule C_d (or external endpoint E) within a committed latency bound $\tau_{\mathcal{R}}$. The relay produces a delivery receipt $\sigma_{\mathcal{R}} = \mathsf{Sign}(sk_{\mathcal{R}}, (h(m), t_{recv}, t_{deliver}))$ anchored on the P-Chain.*

Relay providers are analogous to CDN edge nodes and load balancers in the centralized cloud, with two differences: (i) they cannot read the traffic they route, and (ii) their performance is independently verifiable via chain receipts.

The relay selection algorithm uses a stake-weighted latency score:

$$\mathsf{Score}(\mathcal{R}) = \frac{w_{\text{stake}} \cdot S_{\mathcal{R}}/S_{\text{total}} + w_{\text{perf}} \cdot (1 - \bar{\tau}_{\mathcal{R}}/\tau_{\max})}{w_{\text{stake}} + w_{\text{perf}}} \tag{1}$$

where $S_{\mathcal{R}}$ is the relay's staked amount, $\bar{\tau}_{\mathcal{R}}$ is its rolling average latency, and $\tau_{\max}$ is the network-wide maximum tolerated latency.

2.2 Storage Providers

Storage providers persist encrypted blobs on behalf of capsules. They guarantee durability (data survives hardware failure) and availability (data is retrievable within committed latency bounds) but never hold decryption keys.

Definition 2 (Storage Provider)**.** *A storage provider $\mathcal{S}$ accepts encrypted blobs $b = \mathsf{Enc}(k, d)$ with a committed durability class $\delta \in \{1, 2, 3\}$ (single-replica, geo-redundant, erasure-coded) and availability SLA $\alpha_{\mathcal{S}}$ (fraction of time the blob must be retrievable). The provider produces periodic proof-of-storage attestations:*

$$\pi_t = \mathsf{PoSt}(\mathcal{S}, b, t) \tag{2}$$

verified on-chain without revealing blob contents.

The proof-of-storage mechanism adapts the Filecoin proof-of-spacetime construction [3] to operate over encrypted blobs where the prover cannot inspect the underlying data. The key modification is that challenge responses are computed over the ciphertext directly:

$$\mathsf{PoSt}(\mathcal{S}, b, t) = \mathsf{MerkleProof}(b[c_1], b[c_2], \ldots, b[c_k]) \tag{3}$$

where $c_1, \ldots, c_k$ are pseudorandom challenge indices derived from the block hash at epoch t.

2.3 Gateway Providers

Gateway providers expose capsule services to the external internet. They terminate TLS connections, translate HTTP requests into capsule protocol messages, and route responses back to clients. Gateways are the public interface of the decentralized cloud.

Definition 3 (Gateway Provider)**.** *A gateway provider $\mathcal{G}$ maps external requests to capsule endpoints:*

$$\mathcal{G} : (\mathsf{domain}, \mathsf{path}, \mathsf{headers}) \mapsto (\mathsf{capsule\_id}, \mathsf{action}, \mathsf{params}) \tag{4}$$

The gateway authenticates external callers via standard mechanisms (API keys, OAuth, JWTs) but delegates authorization to the capsule's access control policy. Gateway providers never decrypt capsule data.

The gateway role is analogous to API Gateway services (AWS API Gateway, Cloudflare Workers, Vercel Edge Functions) but with a critical difference: the gateway does not execute application logic. It routes requests to capsules that execute logic locally with their own keys.

2.4 Recovery Providers

Recovery providers hold encrypted key shares for capsule disaster recovery. Under normal operation, recovery providers are inert. When a user loses access to their primary device, recovery providers participate in a threshold reconstruction protocol to restore capsule access.

Definition 4 (Recovery Provider)**.** *A recovery provider $\mathcal{P}$ holds a Shamir share s_i of a capsule's recovery key, where the shares are distributed under a (t, n) threshold scheme. Recovery requires t of n providers to participate in a reconstruction protocol:*

$$k_{recovery} = \mathsf{Reconstruct}(s_{i_1}, s_{i_2}, \ldots, s_{i_t}) \tag{5}$$

The reconstruction is mediated by the T-Chain threshold protocol, ensuring no single provider (and no subset smaller than t) can recover the key unilaterally.

Recovery providers are compensated via a retainer fee (paid per epoch for availability) plus a recovery fee (paid per successful reconstruction). The retainer creates economic incentive to maintain share availability even during long periods of inactivity.

2.5 Compute Providers

Compute providers execute workloads on behalf of capsules. Workloads include inference (running ML models on user data), transformation (processing encrypted data), and general-purpose computation. Compute providers operate in one of two modes:

1. **Plaintext compute in TEE.** The capsule sends encrypted data to a compute provider running inside a Trusted Execution Environment (Intel SGX, AMD SEV, ARM CCA). Data is decrypted inside the enclave, processed, and re-encrypted before leaving.

2. **Encrypted compute via FHE.** The capsule sends CKKS- or TFHE-encrypted data to the provider, which performs computation over ciphertext. Results are returned encrypted and decrypted by the capsule.

Definition 5 (Compute Provider)**.** *A compute provider $\mathcal{C}$ accepts a workload specification $W = (f, x, \mathsf{mode})$ where f is the function to compute, x is the (encrypted) input, and $\mathsf{mode} \in \{\mathsf{TEE}, \mathsf{FHE}\}$. The provider returns $y = f(x)$ (encrypted under the capsule's key) along with an execution receipt:*

$$\sigma_{\mathcal{C}} = \mathsf{Sign}(sk_{\mathcal{C}}, (h(W), h(y), t_{start}, t_{end}, \mathsf{attestation})) \tag{6}$$

where $\mathsf{attestation}$ is a TEE remote attestation (in TEE mode) or an FHE correctness proof (in FHE mode).

3 Provider Reputation Without Surveillance

A decentralized provider market requires a reputation system: users must be able to distinguish reliable providers from unreliable ones. However, the standard approach to reputation—collecting and analyzing user feedback, monitoring traffic patterns, inspecting request/response content—is incompatible with the privacy guarantees of the capsule model.

We introduce a reputation protocol based exclusively on verifiable on-chain signals that reveal nothing about user data or behavior.

3.1 Reputation Inputs

The reputation score for a provider $\mathcal{P}$ is computed from four on-chain signals:

1. **Receipt completion rate.** The fraction of committed service interactions for which $\mathcal{P}$ produced a valid chain receipt within the committed time bound:

$$r_{\text{receipt}} = \frac{|\{\sigma : \sigma \text{ valid and on-time}\}|}{|\{\text{committed interactions}\}|} \tag{7}$$

2. **Stake duration.** The number of consecutive epochs for which $\mathcal{P}$ has maintained stake above the minimum threshold:

$$r_{\text{tenure}} = \min\left(1, \frac{T_{\text{staked}}}{T_{\max}}\right) \tag{8}$$

3. **Slashing history.** The cumulative fraction of stake that has been slashed over the provider's lifetime:

$$r_{\text{slash}} = 1 - \frac{\sum_t \Delta S_t^{\text{slash}}}{S_{\text{initial}}} \tag{9}$$

4. **Proof-of-service attestations.** For storage providers, the fraction of proof-of-spacetime challenges passed. For compute providers, the fraction of TEE attestations verified. For relay providers, the fraction of latency probes within SLA bounds.

3.2 Reputation Aggregation

The composite reputation score is:

$$\mathsf{Rep}(\mathcal{P}) = \prod_{i \in \{\text{receipt,tenure,slash,service}\}} r_i^{w_i} \tag{10}$$

where w_i are governance-set weights satisfying $\sum w_i = 1$. The multiplicative form ensures that a zero score in any dimension (e.g., total slashing) yields a zero composite score, regardless of performance in other dimensions.

Theorem 1 (Reputation Sybil Resistance)**.** *Under the stake-weighted reputation model, creating k Sybil identities with equal stake S/k each yields a strictly lower aggregate reputation than a single identity with stake S, provided $w_{tenure} > 0$.*

Proof. Each Sybil identity starts with $T_{\text{staked}} = 0$, yielding $r_{\text{tenure}} = 0$ and therefore $\mathsf{Rep} = 0$ for all new identities. The original identity retains its accumulated tenure. Since reputation is not transferable (it is bound to the staking identity on-chain), the attacker cannot redistribute reputation across Sybil identities. The aggregate service capacity of the k Sybil identities is bounded by S (total stake is conserved), and each must independently accumulate tenure, creating an unavoidable cold-start penalty. □

3.3 Privacy-Preserving Feedback

Users may optionally submit encrypted quality ratings using the CKKS homomorphic encryption scheme. Ratings are encrypted under the governance committee's public key and aggregated in ciphertext:

$$\mathsf{Enc}(\bar{q}) = \frac{1}{N} \sum_{i=1}^{N} \mathsf{Enc}(q_i) \tag{11}$$

The governance committee threshold-decrypts the aggregate rating (requiring t-of-n committee members) to obtain the average quality score without learning any individual rating. This provides a soft signal that complements the hard on-chain metrics.

4 SLA Enforcement via Chain Receipts

Service level agreements in the centralized cloud are legal contracts enforced through billing disputes and, occasionally, litigation. They are not technically enforced. A provider that violates an SLA may issue a service credit—typically a fraction of the monthly bill—at its own discretion.

In the Lux provider model, SLAs are smart contracts with stake-backed penalties.

4.1 SLA Contract Structure

An SLA between user U and provider $\mathcal{P}$ for service $\mathcal{S}$ is encoded as a smart contract on the C-Chain:

$$\mathsf{SLA}(U, \mathcal{P}, \mathcal{S}) = (\tau_{\max}, \alpha_{\min}, \delta_{\min}, p_{\text{epoch}}, \sigma_{\text{penalty}}) \tag{12}$$

where:

- $\tau_{\max}$: maximum response latency
- $\alpha_{\min}$: minimum availability (fraction of time)
- $\delta_{\min}$: minimum durability class
- p_{epoch}: payment per epoch
- σ_{penalty}: penalty function for violations

4.2 Violation Detection

SLA violations are detected through three mechanisms:

1. **Missing receipts.** If a provider commits to a service interaction but fails to produce a chain receipt within $\tau_{\max}$, the absence is detected by the SLA contract's monitoring logic.
2. **Failed proofs.** If a storage provider fails a proof-of-spacetime challenge, the failure is recorded on-chain and triggers the penalty function.
3. **Latency probes.** The network runs periodic latency probes from geographically distributed verifier nodes. Probe results are submitted on-chain and compared against the committed $\tau_{\max}$.

4.3 Penalty Execution

When a violation is detected, the penalty function executes automatically:

$$\sigma_{\text{penalty}}(v) = \min\left(\beta \cdot S_{\mathcal{P}}, \gamma \cdot p_{\text{epoch}} \cdot \mathsf{severity}(v)\right) \tag{13}$$

where β is the maximum slash fraction (governance-set, typically 5%), $S_{\mathcal{P}}$ is the provider's total stake, γ is a multiplier (typically 2× to 10× the epoch payment), and $\mathsf{severity}(v)$ maps violation types to severity scores.

Slashed funds are split: 50% to the affected user as compensation, 50% to the protocol treasury. This creates a direct economic incentive for users to report violations and for providers to maintain SLA compliance.

Proposition 2 (Rational SLA Compliance)**.** *A rational provider complies with SLA terms if and only if:*

$$\mathbb{E}[\textit{revenue from compliance}] > \mathbb{E}[\textit{savings from violation}] - \mathbb{E}[\sigma_{\textit{penalty}}] \tag{14}$$

Under the Lux penalty model with $\gamma \geq 2$, compliance is the dominant strategy for any provider with stake exceeding the minimum threshold, provided violation detection probability exceeds $1/\gamma$.

5 Compute and Resource Markets

The Lux provider model organizes compute, storage, and bandwidth as continuous double-auction markets with on-chain settlement. Providers post asks (offers to sell service at a given price). Users post bids (willingness to pay for service at a given quality level). A matching engine on the C-Chain clears the market each epoch.

5.1 Market Structure

Each resource type has an independent order book:

Table 1: Resource market structure for the Lux decentralized cloud.

Market	Unit	Quality Dimensions	Settlement
Inference	GPU-second	Latency, throughput, model	Per-request
Storage	GB-month	Durability, availability, region	Per-epoch
Bandwidth	GB-transfer	Latency, throughput, region	Per-request
Recovery	Share-epoch	Availability, threshold	Per-epoch
Gateway	Request	Latency, concurrent connections	Per-request

5.2 Inference Market

The inference market is the most complex, as it must match heterogeneous compute capabilities (GPU type, VRAM, supported models) with heterogeneous demand (model size, batch size, latency requirements).

An inference ask has the form:

$$\mathsf{Ask}_{\text{infer}} = (\mathsf{gpu\_type}, \mathsf{vram}, \mathsf{models}[], \tau_{\max}, p_{\text{per\_token}}) \tag{15}$$

An inference bid has the form:

$$\mathsf{Bid}_{\text{infer}} = (\mathsf{model}, \mathsf{max\_tokens}, \tau_{\max}, p_{\max}, \mathsf{mode}) \tag{16}$$

where $\mathsf{mode} \in \{\mathsf{TEE}, \mathsf{FHE}, \mathsf{plaintext}\}$ specifies the privacy level. TEE and FHE modes command a premium (typically 1.3× and 3× respectively) due to the computational overhead.

5.3 Market Clearing

The matching engine runs a modified second-price auction each epoch. For each resource type, bids and asks are sorted by price. Matches are made greedily by descending bid price against ascending ask price. The clearing price for each match is the ask price (second-price semantics), incentivizing truthful bidding:

$$p_{\text{clear}}(b, a) = p_a \quad \text{where } p_b \geq p_a \tag{17}$$

Theorem 3 (Truthful Bidding)**.** *Under second-price semantics with on-chain settlement, truthful bidding (reporting true valuation) is a weakly dominant strategy for users.*

Proof. Standard Vickrey auction argument. If a user bids above true valuation and wins, they may pay more than the resource is worth. If they bid below and lose, they miss a profitable trade. Bidding true valuation maximizes expected surplus in all cases. □

5.4 Dynamic Pricing

The base price for each resource type adjusts each epoch based on utilization, following an EIP-1559-inspired mechanism:

$$p_{t+1}^{\text{base}} = p_t^{\text{base}} \cdot \left(1 + \frac{1}{8} \cdot \frac{U_t - U_{\text{target}}}{U_{\text{target}}}\right) \tag{18}$$

where U_t is the utilization ratio (fraction of available capacity consumed) and $U_{\text{target}} = 0.5$ is the target utilization. When utilization exceeds 50%, prices rise; when it falls below, prices decrease. The 1/8 dampening factor prevents price oscillation.

6 Provider Portability

The fundamental sovereignty guarantee of the Lux provider model is portability: a user can switch from provider $\mathcal{P}_1$ to provider $\mathcal{P}_2$ without data loss, service interruption, or requiring cooperation from $\mathcal{P}_1$.

6.1 Portability Protocol

The provider switch protocol proceeds in four phases:

1. **Capability announcement.** User announces intent to switch by registering $\mathcal{P}_2$ as an authorized provider for the capsule on the I-Chain.
2. **Data replication.** User's capsule client retrieves all encrypted blobs from $\mathcal{P}_1$ and uploads them to $\mathcal{P}_2$. Since blobs are encrypted under the user's key (not the provider's key), no re-encryption is needed.
3. **Receipt migration.** All chain receipts referencing $\mathcal{P}_1$ remain on-chain and are augmented with a pointer to $\mathcal{P}_2$ for future interactions.
4. **SLA transfer.** The SLA contract is updated to reference $\mathcal{P}_2$. Outstanding payment obligations to $\mathcal{P}_1$ are settled through the epoch boundary.

Theorem 4 (Lossless Provider Switch)**.** *Given a capsule C with data encrypted under key k stored at provider $\mathcal{P}_1$, and a target provider $\mathcal{P}_2$ with sufficient storage capacity, the provider switch protocol completes with zero data loss and bounded downtime $\Delta t \leq 2\tau_{\max}$, even if $\mathcal{P}_1$ is uncooperative, provided the encrypted blobs were replicated to at least $f + 1$ storage providers under the capsule's durability policy (where f is the maximum number of Byzantine providers).*

Proof. The proof follows from two properties:

Data independence. All blobs stored at $\mathcal{P}_1$ are encrypted under k, which $\mathcal{P}_1$ does not possess. The blobs are self-contained: their integrity can be verified by the user via Merkle proofs committed on-chain during upload. Therefore, any copy of the blob (from $\mathcal{P}_1$, from a replica provider, or from the user's local cache) is equally valid.

Availability under Byzantine faults. Under the capsule's durability policy, blobs are replicated to n storage providers with $n \geq 2f + 1$. If $\mathcal{P}_1$ is uncooperative (refuses to serve reads), the user retrieves blobs from the remaining $n - 1 \geq 2f$ honest providers. Since at most f of these may also be Byzantine, at least $f + 1$ honest providers remain, which suffices for complete data retrieval.

Bounded downtime. During the switch, the capsule client reads from $\mathcal{P}_1$ (or replicas) and writes to $\mathcal{P}_2$ in parallel. The maximum downtime is bounded by the time to detect $\mathcal{P}_1$ failure ($\leq \tau_{\max}$) plus the time to reroute to $\mathcal{P}_2$ ($\leq \tau_{\max}$). □

6.2 No Re-Encryption Requirement

A critical property distinguishing the Lux model from centralized cloud migration is the absence of re-encryption. In AWS or GCP, data at rest is encrypted with provider-managed keys (AWS KMS, Google Cloud KMS). Migrating data requires decrypting with the source provider's key and re-encrypting with the destination provider's key—a process that exposes plaintext during transfer and requires active cooperation from the source provider.

In the Lux model, data is encrypted under the user's capsule key. The provider never holds a decryption key. Migration is a simple byte-copy of ciphertext blobs. No key rotation, no re-encryption, no plaintext exposure.

7 Comparison with Centralized Clouds

Table 2: Sovereignty comparison between Lux provider model and centralized cloud platforms.

Property	**Lux**	**AWS**	**GCP**	**Vercel**	**Fly.io**
User holds encryption keys	Yes	No	No	No	No
Provider-independent data	Yes	No	No	No	Partial
On-chain SLA enforcement	Yes	No	No	No	No
Switch without data loss	Yes	No	No	No	Partial
Switch without code rewrite	Yes	No	No	No	Partial
Verifiable provider quality	Yes	No	No	No	No
Compute privacy (TEE/FHE)	Yes	Partial	Partial	No	No
Open market pricing	Yes	No	No	No	No
Censorship resistance	Yes	No	No	No	No
Decentralized availability	Yes	No	No	No	Partial

7.1 Performance Comparison

The decentralized model introduces overhead from encryption, chain receipts, and market clearing. We quantify these costs:

Table 3: Performance overhead of the Lux provider model relative to direct cloud access.

Operation	**Centralized**	**Lux (TEE)**	**Lux (FHE)**
Object read (64 KB)	2 ms	3 ms	N/A
Object write (64 KB)	5 ms	8 ms	N/A
Inference (7B model, 100 tokens)	1.2 s	1.4 s	4.8 s
Receipt anchoring	N/A	50 ms	50 ms
Provider switch (1 TB)	12–48 h	2–6 h	2–6 h

The overhead is modest for TEE mode (15–60% depending on operation) and significant for FHE mode (3×–4× for compute). The receipt anchoring cost (50 ms amortized over batched receipts) is negligible for most applications. Provider switch time is substantially faster than centralized migration because no re-encryption is required and parallel transfer from multiple replicas is supported.

7.2 Lock-In Analysis

We formalize the switching cost for each platform:

Definition 6 (Switching Cost)**.** *The switching cost* $\mathsf{SC}(\mathcal{P}_1, \mathcal{P}_2)$ *from provider* $\mathcal{P}_1$ *to* $\mathcal{P}_2$ *is the total cost (in time, engineering effort, and data risk) of migrating an application with* D *bytes of data,* A *API surface area, and* I *infrastructure dependencies.*

For centralized clouds, SC is dominated by API incompatibility (rewriting application code to use $\mathcal{P}_2$'s proprietary APIs) and data re-encryption. For the Lux model, SC is dominated by data transfer time, which is $O(D/\mathsf{bandwidth})$ with no API rewrite required (providers implement the same open protocol).

8 Economic Model

8.1 Provider Economics

A provider's profit in epoch t is:

$$\Pi_{\mathcal{P},t} = \underbrace{\sum_i p_i \cdot q_i}_{\text{service revenue}} - \underbrace{c_{\text{infra}} + c_{\text{stake}}}_{\text{costs}} - \underbrace{\sigma_{\text{penalty},t}}_{\text{slash losses}} \tag{19}$$

where p_i and q_i are the price and quantity for service i, c_{infra} is infrastructure cost (hardware, bandwidth, electricity), c_{stake} is the opportunity cost of locked stake, and $\sigma_{\text{penalty},t}$ is any slashing penalty incurred.

8.2 Stake-Weighted Provider Selection

Users select providers through a stake-weighted lottery modified by reputation:

$$\Pr[\text{select } \mathcal{P}] = \frac{S_{\mathcal{P}} \cdot \mathsf{Rep}(\mathcal{P})}{\sum_{\mathcal{P}'} S_{\mathcal{P}'} \cdot \mathsf{Rep}(\mathcal{P}')} \tag{20}$$

This creates a positive feedback loop: providers with high stake and high reputation receive more traffic, earn more revenue, and can afford to stake more. The reputation component prevents pure plutocracy—a high-stake provider with poor service quality loses reputation and traffic despite its stake advantage.

8.3 Slashing Conditions

Providers are slashed for four categories of violation:

1. **SLA breach.** Failing to meet committed latency, availability, or durability targets. Penalty: 1%–5% of stake per violation, scaling with severity.

2. **Data loss.** Failing a proof-of-spacetime challenge, indicating data loss or unavailability. Penalty: 5%–10% of stake.

3. **Equivocation.** Producing conflicting receipts for the same service interaction (evidence of fraud). Penalty: 100% of stake (full slashing).

4. **Censorship.** Selectively refusing to serve specific capsules (detected via statistical analysis of receipt patterns). Penalty: 10%–50% of stake.

Proposition 5 (Economic Finality of Slashing)**.** *Under the Lux slashing model, a rational provider with stake S and expected epoch revenue R will not engage in equivocation if:*

$$S > \frac{R \cdot T_{remaining}}{\Pr[detection]} \tag{21}$$

where $T_{remaining}$ is the remaining staking epochs. For the minimum stake requirement and typical revenue, this condition is satisfied when $\Pr[detection] > 0.01$.

8.4 Market Equilibrium

In the long run, the provider market converges to a competitive equilibrium where provider margins approach zero economic profit (revenue equals cost including opportunity cost of capital):

$$\lim_{t \to \infty} \Pi_{\mathcal{P},t} = 0 \quad \text{for all } \mathcal{P} \text{ in competitive equilibrium} \tag{22}$$

This is the standard result for competitive markets. The Lux model achieves this equilibrium faster than centralized markets because: (i) provider switching costs are near zero, removing the friction that sustains supranormal profits; (ii) reputation is public and verifiable, reducing information asymmetry; and (iii) market clearing is automated and transparent, eliminating price discrimination.

8.5 Network Effects Without Lock-In

The Lux provider market exhibits network effects (more providers attract more users, which attract more providers) without the lock-in that typically accompanies network effects in centralized platforms. The mechanism is the open protocol: all providers implement the same interface, so a user who joins the network to access one provider can seamlessly use any other. The network effect accrues to the protocol, not to any individual provider.

9 Related Work

Decentralized cloud computing has been explored across several dimensions. Filecoin [3] provides decentralized storage with proof-of-spacetime verification but does not address compute, relay, or recovery services. Akash Network [4] offers a decentralized compute marketplace but lacks client-side encryption and chain-enforced SLAs. Golem [5] and iExec [6] provide decentralized compute but focus on batch processing rather than real-time service markets.

The capsule architecture for user-owned encrypted data builds on local-first software principles [7] and the Solid project's user-controlled data pods [8]. The Lux model extends these with chain-enforced SLAs, provider reputation, and compute markets.

Threshold cryptography for recovery draws on Shamir's secret sharing [9] and recent work on threshold ECDSA [10] for distributed key management. The Lux recovery provider model applies these primitives in a market context where recovery availability is economically incentivized.

Provider reputation without surveillance relates to privacy-preserving reputation systems studied by Androulaki et al. [11] and Kerschbaum [12]. The Lux approach uses on-chain verifiable signals rather than encrypted feedback, avoiding the complexity of fully homomorphic reputation aggregation while providing a practical privacy/accuracy tradeoff.

10 Conclusion

The centralized cloud model treats providers as data custodians who extract rent from switching costs. The Lux provider model treats providers as service markets competing on quality, with users retaining ownership of their keys, their data, and their exit rights.

The key contributions of this paper are:

1. **Five-role provider taxonomy.** Relay, storage, gateway, recovery, and compute as separable, composable service roles, none of which require custodial access to user data.

2. **Surveillance-free reputation.** A provider reputation protocol based entirely on verifiable on-chain signals—receipt completion, stake tenure, slashing history, proof-of-service—that requires no inspection of user traffic.

3. **Chain-enforced SLAs.** Service level agreements as smart contracts with automatic stake-backed penalties, replacing the legal fiction of centralized SLA credits.

4. **Resource markets.** Continuous double-auction markets for inference, storage, bandwidth, recovery, and gateway services with second-price settlement and EIP-1559-style dynamic pricing.

5. **Provable portability.** A formal proof that users can switch providers with zero data loss and bounded downtime, even under Byzantine fault assumptions.

The decentralized cloud is not a replacement for AWS. It is a different category of infrastructure: one where the user is sovereign, the provider is a vendor, and exit is not a migration—it is a market transaction.

References

[1] Synergy Research Group, "Cloud infrastructure services market reaches $76 billion in Q4 2024," *Synergy Research Group Report*, 2024.

[2] Amazon Web Services, "Summary of the AWS service event in the Northern Virginia (US-EAST-1) region," *AWS Post-Event Summary*, December 2021.

[3] J. Benet and N. Greco, "Filecoin: A decentralized storage network," *Protocol Labs Technical Report*, 2017.

[4] G. Gadiraju and A. Prem, "Akash Network: Decentralized cloud compute marketplace," *Akash Network Whitepaper*, 2020.

[5] The Golem Project, "The Golem Project: Crowdfunding whitepaper," *Golem Factory Technical Report*, 2016.

[6] G. Fedak et al., "iExec: Blockchain-based decentralized cloud computing," *iExec Whitepaper*, 2017.

[7] M. Kleppmann, A. Wiggins, P. van Hardenberg, and M. McGranaghan, "Local-first software: You own your data, in spite of the cloud," in *Onward!*, ACM, 2019.

[8] T. Berners-Lee, "Solid: A platform for decentralized social applications based on Linked Data," *MIT CSAIL Technical Report*, 2016.

[9] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[10] R. Gennaro and S. Goldfeder, "One round threshold ECDSA with identifiable abort," *IACR ePrint 2020/540*, 2020.

[11] E. Androulaki, S. G. Choi, S. M. Bellovin, and T. Malkin, "Reputation systems for anonymous networks," in *PETS*, Springer, 2008.

[12] F. Kerschbaum, "A verifiable, centralized, coercion-free reputation system," in *ACM WPES*, 2009.

[13] T. Roughgarden, "Transaction fee mechanism design for the Ethereum blockchain: An economic analysis of EIP-1559," *arXiv preprint arXiv:2012.00854*, 2020.

[14] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *Polkadot Whitepaper*, 2016.