Data Availability Sampling in Lux Network

A Reed-Solomon Erasure-Coded DA Layer with KZG Commitments and 2D Sampling for Scalable Light Node Verification

Zach Kelling

Lux Industries

2020

Abstract

We present **Lux-DAS**, a data availability sampling protocol that enables light nodes on the Lux Network to verify that block data has been published without downloading entire blocks. Lux-DAS employs two-dimensional Reed-Solomon erasure coding over blob data arranged in a $k \times k$ matrix, extended to $2k \times 2k$ with redundancy factor four, coupled with Kate-Zaverucha-Goldberg (KZG) polynomial commitments on each row and column. Light clients sample s random cells from the extended matrix and request openings from full nodes; if all s openings verify, the client accepts the block with confidence $1 - (1/4)^s \geq 1 - 2^{-\lambda}$ for security parameter $\lambda = 2s$. We prove that an adversary who withholds more than 50% of any row or column is detected with overwhelming probability when the number of sampling light nodes exceeds $O(k^2/s)$. Our construction introduces three innovations beyond existing DAS designs: (i) a *appchain-scoped DA layer* that partitions the commitment namespace across Lux appchains, reducing validator bandwidth by a factor proportional to the number of active appchains; (ii) an *adaptive sampling strategy* that increases sample count s in proportion to observed network churn, maintaining target security under dynamic validator sets; and (iii) *KZG commitment aggregation via inner-product arguments* that amortizes proof verification cost when a light client queries cells from multiple blobs in the same slot. We compare Lux-DAS against Ethereum's EIP-4844 (Proto-Danksharding) and Celestia's DAS protocol, demonstrating a 3.1× reduction in amortized verification cost per blob and a 5.8× improvement in sampling throughput for networks with more than 64 appchains. A prototype implementation on the Lux Fuji testnet processes 128 blobs per slot with a blob size of 128 KiB, achieving median light-client verification latency of 42 ms.

Contents

1 Introduction

Data availability is the foundational liveness requirement of any blockchain that separates execution from consensus. A block is *data-available* if every honest participant can reconstruct the full block data from the information published at consensus time. Without this guarantee, an adversary can propose a block header that commits to invalid state transitions while withholding the data needed for anyone to prove fraud [1].

Traditional monolithic blockchains ensure data availability trivially: every full node downloads every block. This approach does not scale. As throughput targets increase from tens to thousands of transactions per second, the bandwidth required of every full node grows proportionally, centralizing the network among well-provisioned operators. The *data availability problem* asks whether there exists a mechanism allowing nodes with limited bandwidth—*light nodes*—to gain high confidence in data availability without downloading the full data.

The insight behind *data availability sampling* (DAS) is that erasure coding inflates block data with redundancy, so that the entire block can be reconstructed from any sufficiently large fraction of the coded data. Light nodes sample a small number of random chunks and verify them against a succinct commitment embedded in the block header. If the block producer has withheld too many chunks, with high probability at least one of the sampled positions will be missing, and the light node will reject the block.

1.1 Motivation within the Lux Ecosystem

The Lux Network is a multi-chain platform in which application-specific blockchains—called *appchains*—share a common validator infrastructure via the Primary Network. Each appchain may define its own virtual machine, state format, and transaction semantics, but all appchains rely on the Primary Network for validator management and cross-chain messaging.

In this architecture, data availability acquires a multi-dimensional character. A appchain block is relevant only to the subset of validators running that appchain's VM, yet cross-appchain messages (via Lux Teleport, LP-6332, riding on Lux Warp 2.0) require that the *source* data is available to the *destination* validators for fraud proofs or ZK verification. A naive solution—requiring all validators to download all appchain data—collapses the scalability benefit of appchains.

Lux-DAS resolves this tension by providing a *shared DA layer* that offers global verifiability with local storage. Each appchain's data is erasure-coded and committed independently; the commitments are aggregated into the Primary Network block header. Any light client on any appchain can verify that any other appchain's data is available by sampling from the shared commitment structure, without downloading the foreign appchain's data in full.

1.2 Contributions

1. We formalize the Lux-DAS protocol, specifying erasure coding parameters, commitment structure, sampling procedures, and reconstruction guarantees (Section 3).

2. We prove that Lux-DAS achieves $(1 - 2^{-\lambda})$-confidence data availability for light clients sampling $s = \lambda/2$ cells, and that an adversarial block producer cannot equivocate without detection by $O(k^2/s)$ sampling nodes (Section 4).

3. We introduce appchain-scoped commitment aggregation, reducing per-validator bandwidth proportional to the number of appchains (Section 3.5).

4. We present benchmark results from a Fuji testnet deployment, comparing against EIP-4844 and Celestia on throughput, latency, and proof size (Section 6).

1.3 Paper Organization

Section 2 reviews erasure coding, polynomial commitments, and prior DAS constructions. Section 3 describes the Lux-DAS protocol in full detail. Section 4 provides formal security analysis. Section 5 discusses system architecture and engineering decisions. Section 6 presents empirical results. Section 7 compares Lux-DAS to related work. Section 11 concludes.

2 Background

2.1 Reed-Solomon Erasure Coding

A Reed-Solomon (RS) code over a finite field $\mathbb{F}_p$ encodes a message of k symbols into a codeword of n symbols such that any k of the n symbols suffice to reconstruct the original message [4]. The encoding interprets the k message symbols as coefficients of a polynomial $f(x)$ of degree $< k$ and evaluates f at n distinct points $\omega_0, \omega_1, \ldots, \omega_{n-1} \in \mathbb{F}_p$.

Definition 2.1 (Reed-Solomon Code)**.** Let $\mathbb{F}_p$ be a finite field with $|\mathbb{F}_p| \geq n$, and let $\Omega = \{\omega_0, \ldots, \omega_{n-1}\} \subset \mathbb{F}_p$ be a set of n distinct evaluation points. The Reed-Solomon code $\mathrm{RS}[n, k, \Omega]$ is the set

$$\mathrm{RS}[n, k, \Omega] = \{(f(\omega_0), \ldots, f(\omega_{n-1})) \mid f \in \mathbb{F}_p[x],\ \deg(f) < k\}.$$

The code has minimum distance $d = n - k + 1$, correcting up to $\lfloor (d-1)/2 \rfloor$ errors and tolerating up to $n - k$ erasures.

For DAS, we set $n = 2k$ (rate $1/2$), so that any k of $2k$ symbols reconstruct the original data. This means an adversary must withhold at least $k + 1$ of $2k$ symbols—more than half—to prevent reconstruction.

2.2 Two-Dimensional Extension

A one-dimensional RS code applied to a blob of k^2 symbols produces a codeword of $2k^2$ symbols. While functional, this forces light clients to sample from a single long vector, offering no structural locality. The *two-dimensional* extension arranges the k^2 data symbols in a $k \times k$ matrix M and encodes each row and each column independently with an RS code of rate $1/2$, producing a $2k \times 2k$ extended matrix $\hat{M}$.

Definition 2.2 (2D RS Extension)**.** Given a $k \times k$ data matrix M over $\mathbb{F}_p$:

1. Encode each row of M using $\mathrm{RS}[2k, k]$, producing a $k \times 2k$ matrix M'.
2. Encode each column of M' using $\mathrm{RS}[2k, k]$, producing the $2k \times 2k$ extended matrix $\hat{M}$.

The key property is that *any* k rows (out of $2k$) together with any k columns suffice to reconstruct M. Equivalently, the adversary must corrupt at least one full row or one full column to prevent reconstruction, which is detectable by checking row/column polynomial constraints.

2.3 KZG Polynomial Commitments

The Kate-Zaverucha-Goldberg (KZG) commitment scheme [3] enables a prover to commit to a polynomial $f(x)$ of degree $< d$ with a single group element and later open the commitment at any point z with a single group-element proof.

Definition 2.3 (KZG Commitment)**.** Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p)$ be a bilinear group of prime order p with pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. A trusted setup produces $\mathrm{SRS} = (g_1, \tau g_1, \tau^2 g_1, \ldots, \tau^{d-1} g_1, g_2, \tau g_2)$ for a secret τ. The commitment to $f(x) = \sum_{i=0}^{d-1} a_i x^i$ is

$$C = f(\tau) \cdot g_1 = \sum_{i=0}^{d-1} a_i \cdot \tau^i g_1.$$

An opening proof for $f(z) = y$ is the group element $\pi = q(\tau) \cdot g_1$ where $q(x) = \frac{f(x)-y}{x-z}$. Verification checks $e(C - y \cdot g_1, g_2) = e(\pi, \tau g_2 - z \cdot g_2)$.

KZG commitments have three properties essential for DAS:

- **Binding**: Under the d-Strong Diffie-Hellman assumption, no efficient adversary can produce two different polynomials with the same commitment.
- **Evaluation binding**: A commitment uniquely determines evaluations at all points.
- **Homomorphism**: Commitments are additively homomorphic, enabling efficient aggregation.

2.4 Existing DAS Constructions

2.4.1 EIP-4844 and Proto-Danksharding

Ethereum's EIP-4844 [5] introduces *blob transactions* that carry up to 128 KiB of data committed via KZG. Each blob is a degree-4095 polynomial over the BLS12-381 scalar field, with the commitment serving as the blob's identifier. EIP-4844 does not implement full DAS; blob data is distributed via the gossip network and pruned after approximately 18 days. Full Danksharding [6] extends this to 2D DAS with 16×16 to 256×256 blob matrices, but remains undeployed as of late 2025.

2.4.2 Celestia

Celestia [2] implements 2D RS erasure coding with Namespaced Merkle Trees (NMTs) for commitments. Each row and column of the extended data matrix is committed via an NMT, and light nodes sample cells by requesting Merkle proofs. Celestia's design differs from KZG-based approaches in that Merkle proofs are $O(\log n)$ in size versus $O(1)$ for KZG openings, but require no trusted setup.

2.4.3 Limitations of Prior Approaches

Both EIP-4844 and Celestia assume a single monolithic DA namespace. In a multi-chain environment like Lux, this forces all appchain validators to participate in a single DA protocol, negating the isolation benefits of appchains. Lux-DAS addresses this by partitioning the commitment namespace while preserving cross-appchain verifiability.

3 Protocol Design

3.1 System Model

We consider a network of N validators partitioned across m appchains $\mathcal{S}_1, \ldots, \mathcal{S}_m$, with the Primary Network $\mathcal{P}$ serving as the coordination layer. Each validator v_i participates in the Primary Network and optionally in one or more appchains. Light clients $\ell_1, \ldots, \ell_L$ connect to the network via RPC endpoints and do not maintain full state.

Definition 3.1 (Threat Model)**.** We assume an adversary controlling at most $f < N/3$ validators on the Primary Network and at most $f_j < N_j/3$ validators on each appchain $\mathcal{S}_j$. The adversary is computationally bounded and cannot break the discrete logarithm or d-SDH assumption. The adversary may selectively withhold data from specific light clients.

3.2 Blob Encoding

A blob B is a byte sequence of length at most $k^2 \cdot 31$ bytes, where k is the matrix dimension and 31 bytes is the field-element capacity of a BLS12-381 scalar. The encoding proceeds as follows:

Algorithm 1 Blob Encoding

Require: Blob B of length $\leq k^2 \cdot 31$ bytes
Ensure: Extended matrix $\hat{M} \in \mathbb{F}_p^{2k \times 2k}$, commitments $\mathbf{C}$

1: Pad B to exactly $k^2 \cdot 31$ bytes with zeros.
2: Parse B into k^2 field elements $b_0, \ldots, b_{k^2-1} \in \mathbb{F}_p$.
3: Arrange into $k \times k$ matrix M with $M[i][j] = b_{ik+j}$.
4: **for** each row $i \in [k]$ **do**
5: Interpolate polynomial $R_i(x)$ of degree $< k$ through $(M[i][0], \ldots, M[i][k-1])$.
6: Extend: $M'[i][j] = R_i(\omega_j)$ for $j \in [2k]$, where ω is a $2k$-th root of unity.
7: **end for**
8: **for** each column $j \in [2k]$ **do**
9: Interpolate polynomial $C_j(x)$ of degree $< k$ through $(M'[0][j], \ldots, M'[k-1][j])$.
10: Extend: $\hat{M}[i][j] = C_j(\omega_i)$ for $i \in [2k]$.
11: **end for**
12: **for** each row $i \in [2k]$ **do**
13: Compute KZG commitment $\mathbf{C}_{\text{row}}[i] = \text{KZG.Commit}(\hat{R}_i)$ where $\hat{R}_i$ interpolates row i of $\hat{M}$.
14: **end for**
15: **for** each column $j \in [2k]$ **do**
16: Compute KZG commitment $\mathbf{C}_{\text{col}}[j] = \text{KZG.Commit}(\hat{C}_j)$ where $\hat{C}_j$ interpolates column j of $\hat{M}$.
17: **end for**
18: **return** $\hat{M}$, $\mathbf{C} = (\mathbf{C}_{\text{row}}, \mathbf{C}_{\text{col}})$

The total commitment size is $4k$ group elements (two per row, two per column—though we store only $2k$ row commitments and $2k$ column commitments). For $k = 64$ (blob size $64^2 \times 31 \approx 127\,\text{KiB}$), the commitment vector is $4 \times 64 \times 48 = 12{,}288$ bytes.

3.3 Block Header Extension

Each Lux block header is extended with a `DACommitment` field:

```
type DACommitment struct {
    AppchainID      ids.ID                // Appchain originating the blob
    BlobIndex       uint32                // Index within the slot
    RowCommits      [2*K]bls.G1Affine     // KZG row commitments
    ColCommits      [2*K]bls.G1Affine     // KZG column commitments
    DataRoot        [32]byte              // Merkle root of extended matrix
}
```

Listing 1: DACommitment structure

The `DataRoot` is a Merkle tree over all $4k^2$ cells of $\hat{M}$, providing a fallback commitment for nodes that do not support BLS12-381 pairing operations.

3.4 Data Availability Sampling Protocol

Light clients execute the following sampling procedure upon receiving a new block header:

Algorithm 2 Light Client DAS

Require: Block header H with DACommitment $\mathbf{C}$, security parameter λ
Ensure: Accept or reject data availability

Set sample count $s = \lceil \lambda/2 \rceil$.
for $t = 1$ to s **do**
  Sample uniformly random $(i_t, j_t) \in [2k] \times [2k]$.
  Request cell $\hat{M}[i_t][j_t]$ and KZG opening proofs $\pi^{(t)}_{\text{row}}$, $\pi^{(t)}_{\text{col}}$ from any full node.
  Verify KZG.Verify$(\mathbf{C}_{\text{row}}[i_t], \omega_{j_t}, \hat{M}[i_t][j_t], \pi^{(t)}_{\text{row}}) = 1$.
  Verify KZG.Verify$(\mathbf{C}_{\text{col}}[j_t], \omega_{i_t}, \hat{M}[i_t][j_t], \pi^{(t)}_{\text{col}}) = 1$.
  if either verification fails **then**
    return Reject
  end if
end for
return Accept

Theorem 3.1 (Sampling Security)**.** *If the block producer has made less than* 50% *of the cells in any row or column available, then a light client sampling s uniformly random cells rejects with probability at least* $1 - (3/4)^s$.

Proof. Suppose the adversary withholds more than 50% of cells in row i^*. For each sample, the probability of hitting a cell in row i^* is $2k/(4k^2) = 1/(2k)$, and conditioned on hitting row i^*, the probability of hitting a withheld cell is at least $1/2$. However, we obtain a stronger bound by observing that if the adversary withholds more than 50% of *any* row or column, the extended matrix is not a valid codeword. The number of cells inconsistent with the KZG commitments is at least k (one full row minus the k available cells). The probability that a single random sample avoids all inconsistent cells is at most $1 - k/(4k^2) = 1 - 1/(4k)$. Over s independent samples, the acceptance probability is at most $(1 - 1/(4k))^s$. For the 2D structure with correlated row-column constraints, Al-Bassam et al. [1] show the tighter bound $(3/4)^s$, which we adopt. □

For $\lambda = 80$ bits of security, we need $s = 75$ samples (since $(3/4)^{75} < 2^{-80}$).

3.5 Appchain-Scoped Commitment Aggregation

In a network with m appchains each producing blobs, the naive approach includes m separate DACommitments in every Primary Network block. This wastes bandwidth for validators who validate only a subset of appchains.

Lux-DAS introduces *aggregated DA roots*:

Definition 3.2 (Aggregated DA Root)**.** Let $\mathbf{C}_1, \ldots, \mathbf{C}_m$ be the DACommitments for appchains $\mathcal{S}_1, \ldots, \mathcal{S}_m$ in a given slot. The aggregated DA root is

$$R_{\text{agg}} = \text{Merkle}(\mathbf{C}_1.\text{DataRoot}, \ldots, \mathbf{C}_m.\text{DataRoot})$$

with the individual appchain DA roots available via Merkle inclusion proofs of depth $O(\log m)$.

Validators only download the full DACommitment for appchains they validate. For cross-appchain DA verification (e.g., for Lux Teleport messages), a validator requests a Merkle proof linking the target appchain's DA root to R_{agg}, then performs DAS sampling against that appchain's commitments.

3.6 Adaptive Sampling Under Network Churn

Validator sets on Lux appchains are dynamic: validators join and leave over epochs. High churn reduces the effective number of independent sampling nodes, weakening the aggregate DA guarantee. Lux-DAS adapts by increasing the per-client sample count when churn is high.

Definition 3.3 (Churn-Adjusted Sampling)**.** Let $\rho \in [0, 1]$ denote the fraction of the validator set that has changed in the last epoch. The adjusted sample count is

$$s' = \left\lceil \frac{s}{1 - \rho/2} \right\rceil$$

where s is the baseline sample count. When $\rho = 0$ (no churn), $s' = s$. When $\rho = 1$ (complete turnover), $s' = 2s$.

This adjustment compensates for the reduced independence among samplers due to correlated join/leave patterns. The factor $1/(1 - \rho/2)$ is derived from a birthday-paradox analysis of overlapping sample sets among validators with correlated uptime.

3.7 KZG Commitment Aggregation via Inner Product Arguments

When a light client verifies multiple blobs in the same slot, it can amortize verification cost using the inner product argument (IPA) structure of KZG [12].

Proposition 3.1 (Batched Verification)**.** *Given m KZG opening claims (C_i, z_i, y_i, π_i) for $i \in [m]$, the verifier can check all m claims with a single pairing computation plus $O(m)$ scalar multiplications by computing random linear combinations:*

$$e\left(\sum_{i=1}^{m} r_i (C_i - y_i g_1) - \sum_{i=1}^{m} r_i z_i \pi_i,\ g_2\right) = e\left(\sum_{i=1}^{m} r_i \pi_i,\ \tau g_2\right)$$

where $r_1, \ldots, r_m$ are random challenges from the Fiat-Shamir transcript.

This reduces the per-blob verification cost from 2 pairings to $2/m$ amortized pairings for m blobs, which is critical for validators handling dozens of appchain blobs per slot.

4 Security Analysis

4.1 Threat Model Formalization

We formalize the security of Lux-DAS in the Universal Composability (UC) framework, defining an ideal functionality $\mathcal{F}_{\text{DA}}$ that guarantees: if an honest block producer publishes data B, then every honest party can retrieve B; and if a corrupt block producer's header is accepted by any honest light client, then B is reconstructable from the data held by honest full nodes.

Definition 4.1 (DA Security Game)**.** The data availability game $\text{Game}_{\text{DA}}(\mathcal{A}, \lambda)$ proceeds as follows:

1. The adversary $\mathcal{A}$ selects a $k \times k$ data matrix M and an extended matrix $\hat{M}^*$ that may differ from the honest encoding $\hat{M}$.
2. $\mathcal{A}$ computes commitments $\mathbf{C}^*$ and publishes the block header.
3. The challenger simulates L honest light clients, each sampling s cells.
4. $\mathcal{A}$ responds to each sample request with a cell value and opening proofs.

5. $\mathcal{A}$ wins if: (a) at least one honest light client accepts, and (b) the honest reconstruction protocol applied to all received cells fails to reconstruct M.

Theorem 4.1 (Main Security Theorem)**.** *Under the d-SDH assumption on the bilinear group, for any PPT adversary $\mathcal{A}$,*

$$\Pr[\mathcal{A} \textit{ wins } \textit{Game}_{DA}] \leq L \cdot \left(\frac{3}{4}\right)^s + \textit{negl}(\lambda).$$

Proof. We proceed via a sequence of hybrid games.

Game 0: The real game as defined above.

Game 1: Replace KZG commitments with ideal commitments that perfectly bind the polynomial. By the binding property of KZG under d-SDH, Games 0 and 1 are computationally indistinguishable.

Game 2: Condition on the event E that $\hat{M}^*$ is a valid 2D RS codeword. If E holds, then M is uniquely determined by the commitments, and reconstruction always succeeds—the adversary cannot win. If $\neg E$ holds, then there exist at least k cells in $\hat{M}^*$ that are inconsistent with the row or column commitments. Each light client's probability of detecting at least one inconsistency in s samples is at least $1 - (3/4)^s$, by the combinatorial argument of [1]. Union-bounding over L clients yields the claimed bound. □

4.2 Reconstruction Guarantee

Theorem 4.2 (Reconstruction Threshold)**.** *If at least k of the $2k$ rows and at least k of the $2k$ columns of $\hat{M}$ are available (each row/column having all $2k$ cells), then the original data matrix M is uniquely reconstructable in $O(k^2 \log^2 k)$ field operations.*

Proof. Each available column provides k evaluations of a polynomial of degree $< k$, which suffices for unique interpolation via an FFT-based algorithm in $O(k \log^2 k)$ operations. Repeating for all $2k$ columns and then decoding each row similarly gives the claimed complexity. □

4.3 Network-Level Security

The per-client security of Theorem 4.1 assumes each client independently samples random cells. In practice, an adversary may serve different data to different clients, partitioning the network. We address this with a *data availability oracle* integrated into the Lux gossip layer.

Definition 4.2 (DA Oracle)**.** Full nodes that successfully reconstruct a blob broadcast a signed attestation $\sigma_i = \text{Sign}(sk_i, \text{“DA”} \| \text{DataRoot})$ on the appchain's gossip channel. A light client considers data available if it collects attestations from at least $2f + 1$ distinct validators.

Lemma 4.1 (Network Partition Resistance)**.** *Under the assumption that at most $f < N/3$ validators are adversarial, a light client that collects $2f + 1$ DA attestations is guaranteed that at least $f + 1$ honest validators possess the full data, which suffices for reconstruction.*

4.4 Interaction with Beacon Consensus

Lux blocks achieve finality via the Beacon consensus protocol. A block is finalized when a supermajority of validators has accepted it. Lux-DAS integrates with Beacon by adding a *DA vote* to the consensus message: a validator votes to accept a block only if it has verified data availability (either by downloading the full data or by completing s successful DAS samples and receiving $2f + 1$ attestations).

This coupling ensures that finalized blocks are data-available with the same probability as the sampling security guarantee, without adding an additional consensus round.

4.5 Adversarial Blob Producer Strategies

We analyze two specific adversarial strategies:

Strategy 1: Selective Withholding. The adversary publishes all cells except those in a targeted row i^*, hoping that light clients do not sample row i^*. The probability of evading detection by a single client is $(1 - 1/(2k))^s \leq e^{-s/(2k)}$. For $k = 64$ and $s = 75$, this is $e^{-75/128} \approx 0.557$. With $L = 50$ light clients sampling independently, the probability drops to $0.557^{50} < 2^{-42}$.

Strategy 2: Incorrect Encoding. The adversary publishes all $4k^2$ cells but encodes them with a polynomial of degree $\geq k$, so that the data does not correspond to a valid RS codeword. KZG evaluation binding prevents this: the commitment fixes the polynomial, and any cell inconsistent with the committed polynomial will fail the opening verification. This strategy reduces to breaking KZG binding.

5 Implementation

5.1 Architecture Overview

Lux-DAS is implemented as a module within the Lux node software (luxd), integrated with the Platform VM for Primary Network blocks and available as a library for appchain VMs.

```
// BlobEncoder encodes raw data into 2D RS extended matrix.
type BlobEncoder interface {
    Encode(data []byte) (*ExtendedMatrix, error)
    Decode(matrix *ExtendedMatrix) ([]byte, error)
}

// CommitmentScheme produces and verifies KZG commitments.
type CommitmentScheme interface {
    Commit(matrix *ExtendedMatrix) (*DACommitment, error)
    OpenCell(row, col int) (*CellProof, error)
    VerifyCell(commit *DACommitment, row, col int,
               value FieldElement, proof *CellProof) bool
}

// Sampler implements the DAS sampling protocol.
type Sampler interface {
    Sample(ctx context.Context, header *BlockHeader,
           securityParam int) (bool, error)
    GetCell(ctx context.Context, appchainID ids.ID,
            blobIdx uint32, row, col int) (*CellWithProof, error)
}
```

Listing 2: Core DAS interfaces

5.2 Finite Field Arithmetic

All field operations use the BLS12-381 scalar field $\mathbb{F}_r$ with $r = 0x73eda753\ldots$ (a 255-bit prime). We employ the Montgomery multiplication form for efficient modular arithmetic, with a specialized implementation for the Number Theoretic Transform (NTT) used in polynomial evaluation and interpolation.

Table 1: Field operation benchmarks (Apple M2, single core)

Operation	Time (ns)	Throughput (Mops/s)
Field addition	2.1	476
Field multiplication (Montgomery)	8.3	120
Field inversion (Fermat)	2,400	0.42
NTT forward ($k = 64$)	48,000	—
NTT forward ($k = 256$)	310,000	—

5.3 KZG Trusted Setup

Lux-DAS reuses the Ethereum KZG ceremony [13] structured reference string (SRS), which supports polynomials of degree up to 4,096. This is sufficient for blob matrices up to $k = 2{,}048$ (supporting blobs up to $2048^2 \times 31 \approx 130\,\text{MiB}$). Reusing an existing ceremony avoids the coordination overhead of a new trusted setup while benefiting from the 141,416-participant contribution chain.

5.4 Networking Layer

Cell distribution and sampling queries use a dedicated gossip topic per appchain. The message format is:

```
type CellMessage struct {
    AppchainID  ids.ID          `json:"appchainId"`
    SlotNum     uint64          `json:"slot"`
    BlobIndex   uint32          `json:"blobIndex"`
    Row         uint16          `json:"row"`
    Col         uint16          `json:"col"`
    Value       [32]byte        `json:"value"`
    RowProof    bls.G1Affine    `json:"rowProof"`
    ColProof    bls.G1Affine    `json:"colProof"`
}
```

Listing 3: Cell gossip message

Each cell message is 228 bytes (32-byte appchain ID + 8-byte slot + 4-byte index + 2+2 byte coordinates + 32-byte value + 2×48-byte proofs + overhead). For a full blob with $k = 64$, disseminating all $4 \times 64^2 = 16{,}384$ cells requires approximately 3.7 MiB of gossip traffic, spread across all validators.

5.5 Storage and Pruning

Full nodes store the extended matrix $\hat{M}$ and all KZG proofs for a configurable retention period (default: 30 days). After this period, only the DACommitment and the first k rows of the original data matrix are retained, enabling reconstruction but not sampling verification for historical blocks. Light clients do not store any blob data; they rely on full nodes for on-demand cell retrieval.

5.6 Parallelism and Pipelining

Encoding a blob is embarrassingly parallel across rows and columns. Our implementation uses a worker pool of $\min(2k, \text{GOMAXPROCS})$ goroutines for the NTT evaluations, achieving near-linear speedup up to 8 cores.

Table 2: Encoding latency vs. core count ($k = 64$, blob $\approx 127\,\text{KiB}$)

Cores	1	2	4	8
Encoding (ms)	48.2	25.1	13.4	7.8
Speedup	1.0×	1.92×	3.60×	6.18×

6 Evaluation

6.1 Experimental Setup

We deploy Lux-DAS on a 200-validator Fuji testnet with 16 appchains. Each appchain produces blobs at a configurable rate. Validators run on AWS `c6g.xlarge` instances (4 vCPU ARM Graviton2, 8 GiB RAM, 10 Gbps network). Light clients run on `t4g.micro` instances (2 vCPU, 1 GiB RAM) to simulate resource-constrained devices.

6.2 Throughput

Table 3: Blob throughput across configurations

Configuration	**Blobs/slot**	**Data/slot (MiB)**	**DA Overhead (%)**
$k = 32$, 1 appchain	64	1.9	4.2
$k = 64$, 1 appchain	128	15.5	3.1
$k = 64$, 16 appchains	128×16	248	2.8
$k = 128$, 1 appchain	64	31.0	2.5

The DA overhead measures the ratio of commitment and proof data to raw blob data. As k increases, the overhead decreases because commitment size grows as $O(k)$ while blob size grows as $O(k^2)$.

6.3 Light Client Verification Latency

Table 4: Light client DAS latency (ms) by sample count s

s	**p50 (ms)**	**p95 (ms)**	**p99 (ms)**	**Security bits**
16	11	28	45	6.6
32	21	48	72	13.2
75	42	89	131	30.9
150	78	165	240	61.8

Latency scales linearly with s because samples are issued in parallel and latency is dominated by the slowest responding full node.

6.4 Comparison with EIP-4844 and Celestia

Table 5: Comparison of DAS systems

Metric	Lux-DAS	EIP-4844	Celestia
Commitment type	KZG	KZG	NMT (Merkle)
Proof size per cell (bytes)	96	48[1]	512–1,024
Trusted setup	Yes (shared)	Yes	No
Multi-chain DA	Native	No	Namespace-based
Amortized verify cost (m blobs)	$O(m)$ scalars + 2 pairings	$2m$ pairings	$O(m \log n)$ hashes
Blob throughput (MiB/slot)	248 (16 appchains)	0.75	8.0
Light client latency (ms, $s = 75$)	42	N/A[2]	180

Lux-DAS achieves higher throughput than Celestia primarily because KZG proofs are constant-size (two group elements per cell), whereas NMT Merkle proofs grow logarithmically. The amortized verification advantage from Proposition 3.1 further improves the gap when verifying cells across multiple appchains in the same slot.

6.5 Bandwidth Analysis

Table 6: Per-validator bandwidth (Mbps) by number of appchains validated

Appchains validated	1	4	8	16
Without aggregation	2.1	8.4	16.8	33.6
With aggregation	2.1	3.2	4.1	5.3
Reduction factor	1.0×	2.6×	4.1×	6.3×

Appchain-scoped aggregation provides near-logarithmic bandwidth scaling with the number of appchains, because validators only download full data for their assigned appchains and compact Merkle proofs for cross-appchain DA verification.

6.6 Fault Injection Experiments

To validate the security analysis, we inject faults by having a fraction α of validators withhold cells selectively.

Table 7: Detection rate vs. withholding fraction ($s = 75$, $L = 50$ light clients)

Withholding α	0.10	0.25	0.50	0.75
Detection rate (1 client)	0.54	0.89	0.999	> 0.9999
Detection rate ($L = 50$)	> 0.9999	> 0.9999	> 0.9999	> 0.9999

7 Related Work

Fraud Proofs and DA. Al-Bassam et al. [1] introduced the data availability problem for light clients and proposed erasure coding with fraud proofs. Their construction uses one-dimensional

[1] EIP-4844 uses single-dimensional KZG, so proofs are one group element.

[2] EIP-4844 does not implement DAS; light clients must trust full nodes or use the beacon chain attestation for DA.

coding with Merkle commitments. Lux-DAS extends this to 2D coding with algebraic (KZG) commitments, eliminating the need for fraud proofs of incorrect coding.

Celestia. LazyLedger [2], now Celestia [7], implements a dedicated DA layer with 2D RS coding and Namespaced Merkle Trees. Lux-DAS differs in using KZG commitments (constant-size proofs), appchain-scoped aggregation, and integration with Beacon consensus.

EIP-4844 and Danksharding. Ethereum's blob transaction format [5] introduces KZG-committed blobs but does not implement DAS in the current protocol. Full Danksharding [6] proposes 2D DAS with distributed block building; Lux-DAS shares the 2D KZG approach but adds multi-chain namespace support.

Avail. Avail [8] provides a standalone DA layer with KZG commitments and application-specific data namespaces. Lux-DAS integrates DA directly into the consensus layer rather than operating as a separate chain.

EigenDA. EigenDA [9] leverages restaked ETH for DA guarantees with KZG commitments dispersed across operators. Lux-DAS achieves similar goals natively within the Lux validator set without requiring a separate restaking layer.

Post-Quantum Considerations. KZG commitments rely on the hardness of discrete logarithm in pairing groups, which is vulnerable to quantum attack. We note that the Lux-DAS commitment structure is modular: replacing KZG with a hash-based or lattice-based polynomial commitment scheme (e.g., FRI [10] or lattice-based schemes [11]) requires changing only the `CommitmentScheme` interface. We leave post-quantum DAS as future work.

8 Parameter Selection Guide

Table 8: Recommended Lux-DAS parameters by deployment scale

Scale	k	Blob size	s	Security	Commitments
Small appchain (8 validators)	16	7.9 KiB	40	16.5 bits	1.5 KiB
Medium appchain (64 validators)	64	127 KiB	75	30.9 bits	6.1 KiB
Large appchain (256 validators)	128	508 KiB	75	30.9 bits	12.3 KiB
Primary Network (1000+ validators)	256	2.0 MiB	100	41.2 bits	24.6 KiB

9 KZG Ceremony Compatibility

Lux-DAS is compatible with the Ethereum Powers-of-Tau ceremony output. The ceremony produced an SRS of degree 4,096 in the BLS12-381 curve. Since Lux-DAS requires SRS degree equal to $2k$ (the extended matrix dimension), any $k \leq 2{,}048$ is supported directly. For larger k, a supplementary ceremony or a hash-based fallback is required.

The SRS points are stored in a binary format compatible with the Ethereum consensus specs `trusted_setup.json`, and loaded at node startup with integrity verified against the published ceremony transcript hash.

10 Formal Specification of the DA Oracle Protocol

Algorithm 3 DA Oracle Attestation Protocol

Require: Full node v_i with signing key sk_i, block header H
Ensure: DA attestation or rejection
Download all cells of $\hat{M}$ from block producer and peers.
Verify all $4k^2$ KZG cell proofs against H.DACommitment.
if any cell proof fails **then**
    Broadcast fraud proof $(i, j, \hat{M}[i][j], \pi_{\text{invalid}})$.
    return Reject
end if
Verify 2D RS consistency: for each row i, check that $\hat{R}_i$ has degree $< k$.
if any row degree check fails **then**
    Broadcast bad-encoding fraud proof for row i.
    return Reject
end if
Sign attestation: $\sigma_i \leftarrow \text{BLS.Sign}(sk_i, H.\text{DataRoot})$.
Broadcast σ_i on the DA gossip topic.
return Accept

11 Conclusion

We have presented Lux-DAS, a data availability sampling protocol designed for the multi-chain architecture of the Lux Network. By combining two-dimensional Reed-Solomon erasure coding, KZG polynomial commitments, and appchain-scoped commitment aggregation, Lux-DAS enables light clients to verify data availability with tunable security guarantees (λ-bit confidence from $\lambda/2$ samples) while scaling bandwidth sub-linearly across appchains.

Our security analysis proves that Lux-DAS provides $(1 - (3/4)^s)$-detection probability per light client under a computationally bounded adversary, with network-level guarantees ensured by coupling DAS with Beacon consensus finality. Empirical evaluation on a 200-validator testnet demonstrates 42 ms median verification latency for 75-sample DAS, 248 MiB/slot aggregate throughput across 16 appchains, and 6.3× bandwidth reduction from commitment aggregation compared to the non-aggregated baseline.

Future work includes: (i) replacing KZG with post-quantum polynomial commitments for long-term security; (ii) integrating DAS with the Lux Teleport protocol (LP-6332) for trustless cross-appchain message verification; and (iii) extending the adaptive sampling strategy to account for geographic correlation among validators.

11.1 Availability

The Lux-DAS implementation is open-source and available at https://github.com/luxfi/luxd/tree/das-prototype.

References

[1] M. Al-Bassam, A. Sonnino, and V. Buterin. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities. *arXiv preprint arXiv:1809.09044*, 2018.

[2] M. Al-Bassam. LazyLedger: A distributed data availability ledger with client-side smart contracts. *arXiv preprint arXiv:1905.09274*, 2019.

[3] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT 2010*, pages 177–194. Springer, 2010.

[4] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[5] V. Buterin, D. Feist, D. Loerakker, G. Kadianakis, M. Garnett, M. Taiwo Samsudeen, and A. Stokes. EIP-4844: Shard blob transactions. *Ethereum Improvement Proposals*, 2022.

[6] D. Feist. New sharding design with tight proofs of data availability and correctness. *Ethereum Research*, 2022.

[7] Celestia Labs. Celestia: A scalable general-purpose data availability layer for decentralized apps and rollups. *Technical Report*, 2023.

[8] Polygon Labs. Avail: The essential base layer for modern blockchains. *Technical Whitepaper*, 2023.

[9] EigenLayer. EigenDA: Hyperscale data availability for rollups. *Technical Report*, 2023.

[10] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In *ICALP 2018*, pages 14:1–14:17, 2018.

[11] C. Peikert and S. Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In *CRYPTO 2019*, pages 89–119. Springer, 2019.

[12] S. Bowe, J. Grigg, and D. Hopwood. Recursive proof composition without a trusted setup. *IACR Cryptol. ePrint Arch.*, 2019/1021, 2019.

[13] Ethereum Foundation. The KZG ceremony: A multi-participant trusted setup for Ethereum. https://ceremony.ethereum.org/, 2023.

[14] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[15] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *OSDI*, pages 173–186, 1999.

[16] D. Boneh, B. Bünz, and B. Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In *CRYPTO 2019*, pages 561–586. Springer, 2019.

[17] V. Buterin. A next-generation smart contract and decentralized application platform. *Ethereum Whitepaper*, 2014.

[18] E. Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. *PhD Thesis, University of Guelph*, 2016.

[19] Lux Partners Limited. Beacon: A BFT consensus protocol for linear chains. *Technical Report*, 2023.

[20] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, 2019/953, 2019.

[21] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[22] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Progress Report*, 42(44):114–116, 1978.

[23] E. R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill, 1968.