ACeD: Scalable Data Availability Oracle

Peiyao Sheng\(^1\)*, Bowen Xue\(^2\)*, Sreeram Kannan\(^2\), and Pramod Viswanath\(^1\)

\(^1\) University of Illinois, Urbana-Champaign IL 61801
\(^2\) University of Washington at Seattle, WA

Abstract. A popular method in practice offloads computation and storage in blockchains by relying on committing only hashes of off-chain data into the blockchain. This mechanism is acknowledged to be vulnerable to a stalling attack: the blocks corresponding to the committed hashes may be unavailable at any honest node. The straightforward solution of broadcasting all blocks to the entire network sidesteps this data availability attack, but it is not scalable. In this paper, we propose ACeD, a scalable solution to this data availability problem with \(O(1)\) communication efficiency, the first to the best of our knowledge. The key innovation is a new protocol that requires each of the \(N\) nodes to receive only \(O(1/N)\) of the block, such that the data is guaranteed to be available in a distributed manner in the network. Our solution creatively integrates coding-theoretic designs inside of Merkle tree commitments to guarantee efficient and tamper-proof reconstruction; this solution is distinct from Asynchronous Verifiable Information Dispersal [8] (in guaranteeing efficient proofs of malformed coding) and Coded Merkle Tree [31] (which only provides guarantees for random corruption as opposed to our guarantees for worst-case corruption). We implement ACeD with full functionality in 6000 lines of Rust code, integrate the functionality as a smart contract into Ethereum via a high-performance implementation demonstrating up to 10,000 transactions per second in throughput and 6000x reduction in gas cost on the Ethereum testnet Kovan. Our code is available in [1].

1 Introduction

Public blockchains such as Bitcoin and Ethereum have demonstrated themselves to be secure in practice (more than a decade of safe and live operation in the case of Bitcoin), but at the expense of poor performance (throughput of a few transactions per second and hours of latency). Design of high performance (high throughput and low latency) blockchains without sacrificing security has been a major research area in recent years, resulting in new proof of work \([5,14,29,30]\), proof of stake \([4,10,11,16,20]\), and hybrid \([7,24]\) consensus protocols. These solutions entail a wholesale change to the core blockchain stack and existing blockchains can only potentially upgrade with very significant practical hurdles (eg: hard fork of existing ledger). To address this concern, high throughput scaling solutions are explored via “layer 2” methods, including payment channels \([12,23]\) and state channels \([18,25,28]\). These solutions involve “locking” a part of the ledger on the blockchain and operating on this trusted, locked state on an application layer outside the blockchain; however the computations are required to be semantically closely tied to the blockchain (eg: using the same native currency for transactions) and the locked nature of the ledger state leads to limited applications (especially, in a smart contract platform such as Ethereum).

In practice, a popular form of scaling blockchain performance is via the following: a smaller blockchain (henceforth termed “side blockchain”) derives trust from a larger blockchain (henceforth termed “trusted blockchain”) by committing the hashes of its blocks periodically to the trusted blockchain (Fig.1a). The ordering of blocks in the side blockchain is now determined by

\* The first two authors contributed equally to this work.
the order of the hashes in the trusted blockchain; this way the security of the side blockchain is directly derived from that of the trusted blockchain. This mechanism is simple, practical and efficient – a single trusted blockchain can cheaply support a large number of side blockchains, because it does not need to store, process or validate the semantics of the blocks of the side blockchains, only storing the hashes of them. It is also very popular in practice, with several side blockchains running on both Bitcoin and Ethereum; examples include donation trace (Binance charity [15]) and diplomas and credentials verification (BlockCert used by MIT among others [17]).

For decentralized operations of this simple scaling mechanism, any node in the side blockchain should be able to commit hashes to the trusted blockchain. This simple operation, however, opens up a serious vulnerability: an adversarial side blockchain node can commit the hash of a block without transmitting the block to any other side blockchain node. Thus, while the hash is part of the ordering according to the trusted blockchain, the block corresponding to the hash is itself unavailable at any side blockchain node; this “stalling attack” is a serious threat to the liveness of the side blockchain. The straightforward solution to this data availability attack is to store all blocks on the trusted blockchain, but the communication and storage overhead for the trusted blockchain is directly proportional to the size of the side blockchains and the mechanism is no longer scalable.

We propose an intermediate “data availability oracle” layer that interfaces between the side blockchains and the trusted blockchain (Fig. 1c). The oracle layer accepts blocks from side blockchains, pushes verifiable commitments to the trusted blockchain and ensures data availability to the side blockchains. The $N$ oracle layer nodes work together to reach a consensus about whether the proposed block is retrievable (i.e., data is available) and only then commit it to the trusted blockchain. The key challenge is how to securely and efficiently share the data amongst the oracle nodes to verify data availability; if all oracle nodes maintain a copy of the entire side blockchain data locally (i.e., repetition), then that obviously leads to simple majority vote-based retrievability but is not scalable. If the data is dispersed among the nodes to reduce redundancy
ACeD: Scalable Data Availability Oracle 3

(we call this “dispersal”), even one malicious oracle node can violate the retrievability. Thus it appears there is an inherent trade-off between security and efficiency.

The main point of this paper is to demonstrate that the trade-off between security and efficiency is not inherent; the starting point of our solution is the utilization of an erasure code such that different oracle nodes receive different coded chunks. A key issue is how to ensure the integrity and correctness of the coded chunks. Intuitively we can use a Merkle tree to provide the proof of inclusion for any chunk, but a malicious block producer can construct a Merkle tree of a bunch of nonsense symbols so that no one can successfully reconstruct the block. To detect such attacks, nodes can broadcast what they received and meanwhile download chunks forwarded by others to decode the data and check the correctness. Such a method is used in an asynchronous verifiable information dispersal (AVID) protocol proposed by [8]. AVID makes progress on storage savings with the help of erasure code but sacrifices communication efficiency. Nodes still need to download all data chunks; hence, the communication complexity is $O(Nb)$. An alternative approach to detect incorrect coding attacks is via an incorrect-coding proof, which can be provided by any node who tries to reconstruct the data [2]; consider using an $(n, k)$ Reed-Solomon code (which is referred as 1D-RS), with $k$ coded symbols in the fraud proof, essentially not much better than downloading the original block ($n$ symbols).

In summary (Table 1), to find a scalable solution for the data availability oracle problem, erasure code based methods must defend against the incorrect coding attack while minimizing the storage and communication cost. 1D-RS has low communication complexity but when the storing node is adversarial, the storage and download overhead are factor $b$ worse than optimal. The storage and download overhead of AVID remains optimal even under adversarial storing node but the communication complexity is factor $N$ worse than optimal. A full analysis on each performance entry in the table is provided in Appendix E.

Our main technical contribution is a new protocol, called Authenticated Coded Dispersal (ACeD), that provides a scalable solution to the data availability oracle problem. ACeD achieves near optimal performance on all parameters: optimal storage, download and communication overhead under the normal path, and near-optimal (worse by a logarithmic factor) storage and download overhead when the storing node is adversarial, cf. Table 1. We state and prove the security of the data availability guarantee and efficiency properties of ACeD in a formal security model (Section §3).

**Table 1: Performance metrics for different data availability oracles ($N$: number of oracle nodes, $b$: block size).**

|                           | maximal adversary fraction | normal case | worst case | communication complexity |
|---------------------------|---------------------------|-------------|------------|-------------------------|
| uncoded (repetition)      | 1/2                       | $O(N)$      | $O(1)$     | $O(N)$                  | $O(Nb)$     |
| uncoded (dispersal)       | 1/N                       | $O(1)$      | $O(1)$     | $O(1)$                  | $O(b)$      |
| AVID [8]                  | 1/3                       | $O(1)$      | $O(1)$     | $O(1)$                  | $O(Nb)$     |
| 1D-RS                     | 1/2                       | $O(1)$      | $O(b)$     | $O(b)$                  | $O(b)$      |
| ACeD                      | 1/2                       | $O(1)$      | $O(log b)$ | $O(log b)$              | $O(b)$      |

* AVID is an asynchronous protocol.
* See metrics definition in Table 2

**Technical summary of ACeD.** There are four core components in ACeD, as is shown in Fig 1c, with the following highlights.
ACeD develops a novel coded commitment generator called *Coded Interleaving Tree* (CIT), which is constructed layer by layer in an interleaved manner embedded with erasure codes. The interleaving property avoids downloading extra proof thus minimizes the number of symbols needed to store.

- A dispersal protocol is designed to disperse tree chunks among the network with the least redundancy and we show how feasible dispersal algorithms ensure the reconstruction of all data.
- A hash-aware peeling decoder is used to achieve linear decoding complexity. The incorrect-coding proof is minimized to a *single parity equation*.

**Performance guarantees of ACeD.** Our main mathematical claim is that safety of ACeD holds as long as the trusted blockchain is secure, and ACeD is live as long as the trusted blockchain is live and a majority of oracle nodes are honest (i.e., follow protocol) (Section §5.1). ACeD is the first scalable data availability oracle that promises storage and communication efficiency while providing a guarantee for security with a provable bound and linear retrieval complexity; see Table 1 with details deferred to Section §5.2. The block hash commitment on the trusted blockchain and the size of fraud proof are both in constant size.

**Incentives.** From a rational action point of view, oracle nodes are naturally incentivized to mimic others’ decisions without storing/operating on their own data. This “information cascade” phenomenon is recognized as a basic challenge of actors abandoning their own information in favor of inferences based on actions of earlier people when making sequential decisions [13]. In the context of ACeD, we carefully use the semantics of the data dispersal mechanisms to design a probabilistic auditing mechanism that ties the vote of data availability to an appropriate action by any oracle node. This allows us to create a formal rational actor model where the incentive mechanism can be mathematically modeled: we show that the honest strategy is a strong Nash equilibrium; the details are deferred to Appendix §C.

**Algorithm to system design and implementation.** We design an efficient system architecture implementing the ACeD components. Multithreaded erasure code encoding and block pipelining designs significantly parallelize the operations leading to a high performing architecture. We implement this design in roughly 6000 lines of code in Rust and integrate ACeD with Ethereum (as the trusted blockchain). We discuss the design highlights and implementation optimizations in Section §6.

**Evaluation.** ACeD is very efficient theoretically, but also in practice. Our implementation of ACeD is run by lightweight oracle nodes (e.g., up to 6 CPU cores) communicating over a wide area network (geographically situated in three continents) and is integrated with the Ethereum testnet Kovan with full functionality for the side blockchains to run Ethereum smart contracts. Our implementation scales a side blockchain throughput up to 10,000 tx/s while adding a latency of only a few seconds. Decoupling computation from Ethereum (the trusted blockchain) significantly reduces gas costs associated with side blockchain transactions: in our experiments on a popular Ethereum app Cryptokitties, we find that the gas (Ethereum transaction fee) is reduced by a factor of over 6000. This is the focus of Section §7.

We conclude the paper with an overview of our contributions in the context of the interoperability of blockchains in Section §8.

## 2 Related Work

**Blockchain Scaling.** Achieving the highest throughput and lowest latency blockchains which consensus can operate on for a given network of nodes (known as *vertical scaling*), has been a major focus area; this research has resulted in new new proof of work [5,14,29,30], proof of stake [4,10,11,16,20], and hybrid [7,24] consensus protocols. Scaling throughput linearly with
the number of nodes in the network (known as horizontal scaling) is the focus of sharding designs: partition the blockchain and parallelize the computation and storage responsibilities \([21,22,26]\).

Both horizontal and vertical scaling approaches directly impact the core technology ("layer 1") of the blockchain stack and their implementation in existing public blockchains is onerous. An alternate approach is to lock parts of the state of the blockchain and process transactions associated with this locked state in an application layer: this approach includes payment channels \([12,23]\) and state channels \([18,25,28]\). In this paper we are concerned with a third (and direct) form of blockchain scaling: we interact very lightly with the trusted blockchain (only storing hashes of each block of the side blockchains) but design and implement an oracle that allows for scalable secure interactions between the side blockchains and the trusted blockchain.

**Data Availability.** Blockchain nodes that do not have access to all of the data (eg: light nodes in simple payment verification clients in Bitcoin) are susceptible to the data availability attack. One approach is to have the light nodes rely on full nodes to notify the misbehavior of a malicious block proposer (which has withheld the publication of the full block). Coding the blockchain data to improve the efficiency of fraud proofs was first suggested in \([2]\) (using 2D Reed-Solomon codes), was strongly generalized by \([31]\) via a cryptographic hash accumulator called Coded Merkle Tree (CMT) to generate the commitment of the block. Light nodes randomly sample symbols through anonymous channels and by collecting sampling results, an honest full node can either reconstruct the block or provide incorrect-coding proof. CMT reduces the proof size compared to \([2]\). However, the sampling mechanism is probabilistic (and not appropriate for implementation in the oracle setting of this paper); further CMT requires anonymous communication channels. In this paper, we propose an alternate coded Merkle tree construction (CIT) that is specific to the oracle operation: the method is equipped with a “push” scheme to efficiently and deterministically disseminate data among oracle nodes (no communication anonymity is needed).

Another perspective to understand the data availability problem is to find a secure, efficient way of dispersing data in the network and prove it is permanently retrievable. Such a problem has been discussed under the context of asynchronous verifiable information dispersal (AVID) \([8]\), which applies erasure code to convert a file into a vector of symbols which later are hashed into a fingerprint. The symbols together with the fingerprint are disseminated to nodes. Then each node checks the hash and broadcasts its received copy for guaranteeing retrievability. AVID improves storage saving but sacrifices communication efficiency; the performance is summarized in Table 1 with details in Appendix E. The key insight of ACeD is that by using codes designed to have short fraud proofs, it is possible to avoid the additional echo phase in AVID where honest nodes flood the chunks to each other and thus provide near-optimal communication efficiency.

### 3 System and Security Model

The system is made up of three components: a trusted blockchain (that stores commitments and decides ordering), clients (nodes in side blockchains who propose data), and an intermediate oracle layer ensuring data availability (see Fig.1c).

#### 3.1 Network Model and Assumptions

There are two types of nodes in the network: oracle nodes and clients.

*Oracle nodes* are participants in the oracle layer. They receive block commitment requests from clients, including block headers, and a set of data chunks. After verifying the integrity and correctness of the data, they vote to decide whether the block is available or not and submit the results to the trusted blockchain.
Clients propose blocks and request the oracle layer to store and commit the blocks. They periodically update the local ledger according to the commitment states from the trusted blockchain and ask oracle nodes for the missing blocks on demand.

One of the key assumptions of our system is that the trusted blockchain has a persistent order of data and liveness for its service. Besides, we assume that in the oracle layer, honest nodes take the majority. For clients, we only assume that at least one client is honest (for liveness). Oracle nodes are connected to all clients. The network is synchronous, and the communication is authenticated and reliable.

3.2 Oracle Model

The oracle layer is introduced to offload the storage and ensure data availability. The network of oracle layer consists of $N$ oracle nodes, which can interact with clients to provide data availability service. There exists an adversary that is able to corrupt up to $\beta N$ oracle nodes. Any node if not corrupted is called honest.

The basic data unit for the oracle layer is a block. A data availability oracle comprises of the following primitives which are required for committing and retrieving a block $B$.

1. **Generate chunks**: When a client wants to commit a block $B$ to the trusted blockchain, it runs $(\text{generate\_commitment}(B, M))$ to generate a commitment $c$ for the block $B$ and a set of $M$ chunks $c_1, c_M$.
2. **Disperse chunks**: There is a dispersal protocol $\text{disperse}(B, (c_1, \ldots, c_M), N)$ which can be run by the client and specifies which chunks need to be sent to which of the $N$ oracle nodes.
3. **Oracle finalization**: The oracle nodes run a finalization protocol to finalize and accept certain blocks whose commitments are written into the trusted blockchain.
4. **Retrieve Data**: Clients can initiate a request $(\text{retrieve}, c)$ for retrieving a set of chunks for any commitment $c$ that has been accepted by the oracle.
5. **Decode Data**: There is a primitive $\text{decode}(c, \{c_i\}_{i \in S})$ that any client can run to decode the block from the set of chunks $\{c_i\}_{i \in S}$ retrieved for the commitment. The decoder also returns a proof that the decoded block $B$ is related to the commitment.

We characterize the security of the oracle model and formally define data availability oracle as follows,

**Definition 1.** A data availability oracle for a trusted blockchain accepts blocks from clients and writes commitments into the trusted blockchain with the following properties:

1. **Termination**: If an honest client initiates a disperse request for a block $B$, then block $B$ will be eventually accepted and the commitment $c$ will be written into the trusted blockchain.
2. **Availability** If a dispersal is accepted, whenever an honest client requests for retrieval of a commitment $c$, the oracle is able to deliver either a block $B$ or a null block $\emptyset$ and prove its relationship to the commitment $c$.
3. **Correctness**: If two honest clients on running $(\text{retrieve}, c)$ receives $B_1$ and $B_2$, then $B_1 = B_2$. If the client that initiated the dispersal was honest, we require furthermore that $B_1 = B$, the original dispersed block.

A naive oracle satisfying all above expectations is trivial to construct, e.g., sending all oracle nodes a full copy of data. However, what we want is a scalable data availability oracle. To better understand this motivation, in the next section, we will introduce some metrics to concretize the oracle properties.
Table 2: System Performance Metrics

| Metric                        | Formula           | Explanation                                      |
|-------------------------------|-------------------|--------------------------------------------------|
| Maximal adversary fraction    | $\beta$           | The maximum number of adversaries is $\beta N$.  |
| Storage overhead              | $\frac{D_{\text{store}}}{D_{\text{info}}}$ | The ratio of total storage used and total information stored. |
| Download overhead             | $\frac{D_{\text{download}}}{D_{\text{data}}}$ | The ratio of the size of downloaded data and the size of reconstructed data. |
| Communication complexity      | $D_{\text{msg}}$ | Total number of bits communicated.               |

3.3 Performance Metrics

The central system goals are scalability and data persistence. We define four key metrics of system performance; see Table 2. Let $N$ be the number of nodes and $b$ be the block size. “Maximal adversary fraction” measures the fault tolerant capability of the model. Consider the simplest scaling solution that disperses data among the network without repetition; here the model can not tolerate even one corrupted node. “Storage overhead” measures the ratio of total storage cost and actual information stored. Consider the oracle where everyone keeps a full copy of data, the storage overhead is $O(N)$, indicating that the storage cost increases linearly with the size of the network. ACeD achieves $O(1)$ storage when the client is honest, and $O(\log b)$ when the client is corrupted. “Download overhead” measures the ratio of downloaded data size and reconstructed information size. When applying 1D-RS code, in the worst case, the adversary sends a block with incorrect encoding, oracle nodes need to download $O(b)$ data for fraud proof. ACeD achieves near-optimal overhead, only $O(\log b)$ proof needs to be downloaded. “Communication complexity” measures the number of bits communicated; AVID [8] adopts a dispersal protocol with message complexity $O(N^2)$ for two-round broadcasting so the communication complexity here is $O(Nb)$; the communication cost linearly increases with the size of the network and the size of block. For a given block ACeD achieves $O(b)$ communication efficiency. We formally prove all these properties of ACeD in Section 5. A complete analysis for different protocols on all performance metrics are referred to Appendix E.

4 Technical Description of ACeD

In this section, we describe the four components of ACeD: CIT, dispersal protocol, retrieval protocol and block peeling decoder.

![Diagram](image.png)

Fig. 2: The pipeline for a block to be committed in trusted blockchain.
4.1 Coded Interleaving Tree

The first building block of ACeD is a coded commitment generator which takes a block proposed by a client as an input and creates three outputs: a commitment, a sequence of coded symbols, and their proof of membership POM – see Figure 2. The commitment is the root of a coded interleaving tree (CIT), the coded symbols are the leaves of CIT in the base layer, and POM of a symbol including the Merkle proof (all siblings’ hashes from each layer) and a set of parity symbols from all intermediate layers.

The construction process of CIT is illustrated in figure 3. Suppose a CIT for a block with size \( b \) has \( \ell \) layers, and let’s denote the size of base layer symbols as \( c \) and the number of data symbols in the base layer before encoding as \( s_{\ell} = b/c \), we first apply erasure codes with coding ratio \( r \leq 1 \) to generate \( m_{\ell} = s_{\ell}/r \) coded symbols. Then by aggregating the hashes of every \( q \) coded symbols we get \( m_{\ell}/q \) data symbols for the parent layer (layer \( \ell - 1 \)), which is further encoded to \( m_{\ell - 1} = m_{\ell}/(qr) \) coded symbols. We aggregate and code the symbols iteratively until the number of symbols in a layer decays to \( t \).

For all layers \( j \) except for root, \( 1 \leq j \leq \ell \), denote the set of all coded symbols as \( M_j \) with size \( m_j \), which contains two disjoint sets of symbols: systematic symbol \( S_j \) and parity symbols \( P_j \). \( S_j \) is called systematic symbol because it is used to generate the parity symbols, and hence the size of systematic symbol is \( s_j = rm_j \). Specifically we set \( S_j = [0, rm_j) \) and \( P_j = [rm_j, m_j) \).

Given a block of \( s_{\ell} \) data symbols in the base layer, the aggregation rule for the \( k \)-th systematic symbol of layer \( j - 1 \) is defined as follows:

\[
Q_{j-1}[k] = \{ H(M_j[x]) \mid x \in [0, M_j), k = x \mod rm_{j-1} \} \tag{1}
\]

\[
M_{j-1}[k] = H(\text{concat}(Q_{j-1}[k])) \tag{2}
\]

where \( 1 \leq j \leq \ell \) and \( H \) is a hash function. \( Q[k] \) is the set of hashes that will be used to generate \( k \)-th symbol in the parent layer and \( \text{concat} \) represents the string concatenation function which will concatenate all elements in an input set.

Fig. 3: CIT construction process for \( s = 4 \) systematic symbols, applied with erasure codes of coding ratio \( r = \frac{1}{4} \). The batch size \( q = 8 \) and the number of hashes \( t \) in root is 4. Circled symbols constitute a base layer coded symbol and POM.
Generating a POM for a base layer symbol can be considered as a layer by layer sampling process as captured by the following functions:

\[ f_\ell : [m_\ell] \rightarrow \left( \frac{m_\ell - 1}{2} \right), \cdots, f_2 : [m_2] \rightarrow \left( \frac{m_1}{2} \right); \]

each function maps a base layer symbol to two symbols of the specified layer: one is a systematic symbol and the other is a parity symbol. We denote the two symbols with a tuple of functions \( f_j(i) = (p_j(i), e_j(i)) \), each is defined as follows:

\[ p_j(i) = i \mod rm_j - 1; \quad e_j(i) = rm_j - 1 + (i \mod (1 - r)m_j - 1) \quad (3) \]

where \( p_j : [m_\ell] \rightarrow [0, rm_j - 1] \) and \( e_j : [m_\ell] \rightarrow [rm_j - 1, m_j - 1] \).

Ideally, the set of sampling functions should guarantee that if at least \( \eta \leq 1 \) ratio of distinct base layer symbols along with their POM are sampled, then in every intermediate layer, at least \( \eta \) ratio of distinct symbols can be picked out from the collected POM to ensure reconstruction (Lemma 2). An efficient sampling scheme requires a key property, sibling property, that all sampled symbols at each layer have a common parent so that no extra Merkle proof is needed. (see Lemma 3 for a formal statement).

As described above, CIT can be represented by parameters \( T = (c, t, r, \alpha, q, d) \). All parameters have been defined except \( \alpha \), which is the undecodable ratio of an erasure code and \( d \) is the number of symbols in a failed parity equation, both of which will be used in decoder.

### 4.2 Dispersal Protocol

Given the commitment, coded symbols and POM generated by CIT, the dispersal protocol is used to decide the chunks all oracle nodes need to store. Consider a simple model where there are \( M = b/(cr) \) chunks to be distributed to \( N \) nodes (each node may receive \( k \) chunks) such that any \( \gamma \) fraction of nodes together contains \( \eta \) fraction of chunks. The efficiency of the dispersal protocol is given by \( \lambda = M/(Nk) \). A chunk dispersal protocol can be represented by a certain triple \( (\gamma, \eta, \lambda) \). In the ACeD model, \( \gamma \) is constrained by security threshold \( \beta \) since the clients can only expect to collect chunks from at most \( (1 - \beta)N \) nodes. \( \eta \) is constrained by the undecodable ratio \( \alpha \) of the erasure code since the total chunks a client collects should enable the reconstruction. So for a given erasure code, there is a trade-off between dispersal efficiency and security threshold. Our main result is the demonstration of a dispersal protocol with near optimal parameters. Due to space limitations we have deferred the proof to Appendix §A.1.

**Lemma 1.** If \( \frac{\gamma}{\lambda} < \eta \), then \( (\gamma, \eta, \lambda) \) is not feasible. If \( \frac{\gamma}{\lambda} > \log(\frac{1}{1-\eta}) \) then \( (\gamma, \eta, \lambda) \) is feasible and there exists a chunk dispersal protocol with these parameters.

In the dispersal phase, all oracle nodes wait for a data commitment \( c, k \) assigned chunks and the corresponding POM. The dispersal is accepted if \( \gamma + \beta \) fraction of nodes vote that they receive the valid data.

### 4.3 Retrieval Protocol and Block Decoding

When a client wants to retrieve the stored information, the retrieval protocol will ask the oracle layer for data chunks. Actually, given erasure codes with undecodable ratio \( \alpha \), an arbitrary subset of codes with the size of over ratio \( 1 - \alpha \) is sufficient to reconstruct the whole block. When enough responses are collected, a hash-aware peeling decoder introduced in [31] will be used to reconstruct the block. The decoding starts from the root of CIT to the leaf layer and for each layer, it keeps checking all degree-0 parity equations and then finding a degree-1 parity
equation to solve in turn. Eventually, either all symbols are decoded or there exists an invalid parity equation. In the second case, a logarithmic size incorrect-coding proof is prepared, which contains the inconsistent hash, \( d \) coded symbols in the parity equation and their Merkle proofs. After an agreement is reached on the oracle layer, the logarithmic size proof is stored in the trusted blockchain to permanently record the invalid block. Oracle nodes then remove all invalid symbols to provide space for new symbols.

4.4 Protocol Summary

In summary, an ACeD system with \( N \) oracle nodes and block size \( b \) using CIT \( T \) and dispersal protocol \( D \) can be represented by parameters \((b, N, T, D)\), where \( T = (c, t, r, \alpha, q, d) \) and \( D = (\gamma, \eta, \lambda) \). The pipeline to commit a block to the trusted blockchain is as follows (see Figure 2).

- A client proposes a block of size \( b \), it first generates a CIT with base layer symbol size \( c \), number of hashes in root \( t \), coding ratio \( r \) and batch size \( q \). There are \( M = b/(cr) \) coded symbols in the base layer. And then it disperses \( M \) coded symbols, their POM and the root of CIT to \( N \) oracle nodes using the dispersal protocol \( D = (\gamma, \eta, \lambda) \).
- Oracle nodes receive dispersal request, they accept chunks and the commitment, verify the consistency of data, POM and root, vote their results. A block is successfully committed if there are at least \( \beta + \gamma \) votes. Upon receiving retrieval requests, oracle nodes send the stored data to the requester. Upon receiving fraud proof of a block, oracle nodes delete the stored data for that block.
- Other clients send retrieval requests to the oracle nodes on demand. Upon receiving at least \( \eta \geq 1 - \alpha \) fraction of chunks from at least \( \gamma \) oracle nodes, they reconstruct the block, if a coding error happens, the proof of error will be sent to the trusted blockchain.

5 Performance Guarantees of ACeD

**Theorem 1.** Given an adversarial fraction \( \beta < \frac{1}{2} \) for an oracle layer of \( N \) nodes, ACeD is a data availability oracle for a trusted blockchain with \( O(b) \) communication complexity, \( O(1) \) storage and download overhead in the normal case, and \( O(\log b) \) storage and download overhead in the worst case.

This result follows as a special case of a more general result below (Theorem 2).

**Proof.** Suppose \( \chi \) is an ACeD data availability oracle with parameters \((b, N, T, D)\) where \( T = (c, t, r, \alpha, q, d) \) and \( D = (\gamma, \eta, \lambda) \). There are at most \( \beta < \frac{1}{2} \) fraction of adversarial nodes in the oracle layer. Then by setting \( r, q, d, t = O(1), c = O(\log b), b \gg N \), \( \chi \) is secure as long as \( \beta \leq \frac{1}{2}(1 - \lambda \log(\frac{1}{\alpha})) \); the communication complexity of \( \chi \) is \( O(b) \) because

\[
Nyt + \frac{b}{\lambda r} + \frac{(2q - 1)by}{cr\lambda} \log_{qr}\frac{b}{crt} = O(N) + O(b) + O(b) = O(b)
\]

the storage and download overhead in the normal case is \( O(1) \), because

\[
\frac{Nyt}{b} + \frac{1}{\lambda r} + \frac{(2q - 1)y}{cr\lambda} \log_{qr}\frac{b}{crt} = O(1) + O(1) + O\left(\frac{1}{\log b} \log\left(\frac{b}{\log b}\right)\right) = O(1)
\]

the storage and download overhead in the normal case is \( O(\log b) \), because

\[
\frac{c(d - 1)}{y} + d(q - 1) \log_{qr}\frac{b}{crt} = O(\log b) + O(\log_{qr}\left(\frac{b}{\log b}\right)) = O(\log b).
\]
A complete description of the security and performance guarantees of ACeD is below.

**Theorem 2.** ACeD is a data availability oracle for the trusted blockchain tolerating at most $\beta \leq 1/2$ fraction of adversarial oracle nodes in an oracle layer of $N$ nodes. The ACeD is characterized by the system parameters $(b,N,T,D)$, where $T = (c,t,r,\alpha,q,d)$ and $D = (\gamma,\eta,\lambda)$. $y$ is a constant size of a hash digest, then

1. ACeD is secure under the conditions that

$$\beta \leq \frac{1-\gamma}{2}; \quad \frac{\gamma}{\lambda} > \log\left(\frac{1}{1-\eta}\right); \quad \eta \geq 1 - \alpha$$

2. Communication complexity is

$$Nyt + \frac{b}{\lambda r} + \frac{(2q-1)b}{cr\lambda} \log_{qr} \frac{b}{c \tau r}$$

3. In normal case, both the storage and download overhead are

$$\frac{Nyt}{b} + \frac{1}{\lambda r} + \frac{(2q-1)y}{cr\lambda} \log_{qr} \frac{b}{c \tau r}$$

4. In worst case, both storage and download overhead are

$$\frac{c(d-1)}{y} + \frac{d(q-1)\log_{qr} b}{c \tau r}$$

**Proof.** We prove the security and efficiency guarantees separately.

### 5.1 Security

To prove that ACeD is secure as long as the trusted blockchain is persistent and

$$1 - 2\beta \geq \gamma; \quad \frac{\gamma}{\lambda} > \log\left(\frac{1}{1-\eta}\right); \quad \eta \geq 1 - \alpha$$

we prove the following properties as per Definition 1.

- **Termination.** In ACeD, a dispersal is accepted only if there is a valid commitment submitted to the trusted blockchain. Suppose an honest client requests for dispersal but the commitment is not written into the trusted blockchain, then either the commitment is not submitted or the trusted blockchain is not accepting new transactions. Since $1 - 2\beta \geq \gamma$, thus $\beta + \gamma \leq 1 - \beta$, even if all corrupted nodes remain silent, there are still enough oracle nodes vote that the data is available and the commitment will be submitted, hence the trusted blockchain is not live, which contradicts our assumption.

- **Availability.** If a dispersal is accepted, the commitment is on the trusted blockchain and $\beta + \gamma$ oracle nodes have voted for the block. Since the trusted blockchain is persistent, whenever a client wants to retrieve the block, it can get the commitment and at least $\gamma$ nodes will respond with stored chunks. On receiving chunks from $\gamma$ fraction of nodes, for a CIT applying an erasure code with undecodable ratio $\alpha$ and a feasible dispersal algorithm $(\gamma,\eta,\lambda)$ (Lemma 1), because $\eta \geq 1 - \alpha$, the base layer is reconstructable. Then we prove the following lemma ensures the reconstruction of all intermediate layers.

**Lemma 2.** (Reconstruction) For any subset of base layer symbols $W_\ell$, denote $W_j := \bigcup_{i \in W_\ell} f_j(i)$ as the set of symbols contained in POM of all symbols in $W_\ell$. If $|W_\ell| \geq \eta m_\ell$, then $\forall j \in [1,\ell]$, $|W_j| \geq \eta m_j$. 

The proof of Lemma 2 utilizes the property when generating POM given base layer symbols. (See details in Appendix §A.2.) Thus the entire tree is eventually reconstructed and the oracle can deliver a block $B$, and the proof for $B$’s relationship to commitment $c$ is the Merkel proof in CIT. If a client detects a failed parity equation and outputs a null block $\emptyset$, it will generate an incorrect-coding proof.

Correctness. Suppose for a given commitment $c$, two honest clients reconstruct two different blocks $B_1$ and $B_2$, the original dispersed block is $B$.

1. If the client that initiated the dispersal was honest, according to the availability property, $B_1, B_2 \neq \emptyset$, both clients can reconstruct the entire CIT. If $B_1 \neq B_2$, the commitment $c_1 \neq c_2$, which contradict our assumption that the trusted blockchain is persistent.

2. If the client that initiated the dispersal was adversary and one of reconstructed blocks is empty, w.l.o.g suppose $B_1 = \emptyset$, the client can generate a fraud proof for the block. If $B_2 \neq \emptyset$, the entire CIT is reconstructed whose root is commitment $c_2$. Since there’s no failed equation in the CIT of $B_2$, $c_1 \neq c_2$, which contradict our assumption that the trusted blockchain is persistent.

3. If the client that initiated the dispersal was adversary and $B_1, B_2 \neq \emptyset$, both clients can reconstruct the entire CIT. If $B_1 \neq B_2$, the commitment $c_1 \neq c_2$, which contradict our assumption that the trusted blockchain is persistent.

Thus we have $B_1 = B_2$, and if the client that initiated the dispersal was honest, $B_1 = B$.

5.2 Efficiency

Prior to computing the storage and communication cost for a single node to participate dispersal, we first claim a crucial lemma:

Lemma 3. For any functions $p_j(i)$ and $e_j(i)$ defined in equation (3), where $1 \leq j \leq \ell$, $0 \leq i < m\ell$, $p_j(i)$ and $e_j(i)$ are siblings.

Lemma 3 indicates that in each layer, there are exactly two symbols included in the POM for a base layer symbol and no extra proof is needed since they are siblings (see proof details in §A.3). For any block $B$, oracle nodes need to store two parts of data, the hash commitment, which consists of $t$ hashes in the CIT root, and $k$ dispersal units where each unit contains one base layer symbol and two symbols per intermediate layer. Denote the total storage cost as $X$, we have

$$X = ty + kc + k[y(q - 1) + yq] \frac{b}{c \lambda} \log \frac{qr}{b}$$

where $y$ is the size of hash, $b$ is the size of block, $q$ is batch size, $r$ is coding rate, and $c$ is the size of a base layer symbol. Notice that $k = \frac{b}{N \lambda c}$, we have

$$X = ty + \frac{b}{N \lambda c} + \frac{(2q - 1)by}{N \lambda c} \log \frac{b}{c \lambda r}.$$ 

It follows that the communication complexity is $NX$. In the normal case, each node only stores $X$ bits hence the storage overhead becomes $\frac{NX}{t}$, and similarly when a client downloads data from $N$ nodes, its overhead is $\frac{NX}{t}$. In the worst case, we use proof of error to notify all oracle nodes. The proof for a failed parity equation which contains $d$ coded symbols consist of $d - 1$ symbols and their Merkle proofs, denote the size as $P$, we have

$$P = (d - 1)c + dy(q - 1) \frac{b}{c \lambda r}.$$ 

The storage and download overhead in this case is $\frac{P}{y}$, the ratio of the proof size and the size of reconstructed data, a single hash $y$. 


6 Algorithm to System Design and Implementation

ACeD clients are nodes associated with a number of side blockchains; the honest clients rely on ACeD and the trusted blockchain to provide an ordering service of their ledger (regardless of any adversarial fraction among the peers in the side blockchain). A client proposes a new block to all peers in the side blockchain by running ACeD protocol. An honest client confirms to append the new block to the local side blockchain once the block hash is committed in the trusted blockchain and the full block is downloaded. As long as there is a single honest client in the side blockchain, we have the following claim:

Claim. Once a block is confirmed by an honest client, security is guaranteed as long as the trusted blockchain is safe, even if the oracle layer is dishonest majority. Liveness is guaranteed when the trusted blockchain is live and the oracle layer is honest majority.

The claim indicates that in side blockchains, the safety of a confirmed block only relies on the trusted blockchain because the commitment on it is irrefutable once the trusted blockchain is safe, and the honest client has already downloaded the full block. So even if the oracle layer is occupied by dishonest majority, those confirmed blocks are still safe. However the liveness relies on the liveness of both ACeD and the trusted blockchain. As for the side blockchain network, because data persistence is guaranteed by ACeD, any client inside the network can safely conduct a transaction and reconstruct the ledger without worrying about a dishonest majority; similarly a valid transaction will eventually join the ledger as long as there is a single honest client who can include the transaction into a block.

Next we discuss practical parameter settings for ACeD. We use these parameter choices to design and implement an ACeD oracle layer that interfaces with Ethereum as the trusted blockchain.

Parameter Specifications. We study an ACeD system with $N = 9000$ oracle nodes with adversarial fraction $\beta = 0.49$; the block size $b$ is 12 MB and therefore $b \gg N$. In each block, the base layer symbol size $c$ is $2000 \log b \approx 48$ kB, which corresponds to $\frac{b}{c} = 256$ uncoded symbols in the base layer. Within the setup, we construct a CIT of five layers with parameters: number of root symbols $t = 16$, hash size $y = 32$ bytes, coding ratio $r = 0.25$, aggregation batch size $q = 8$ and 4 erasure codes of size $(256,128,64,32)$ for each non-root layer. For selecting erasure codes properties, we use codes with undecodable ratio $\alpha = 0.125$, maximal parity equation size $d = 8$. In the dispersal protocol, we use $\eta = 0.875 = 1 - \alpha$, which translates to a dispersal efficiency $\lambda \leq \frac{1 - \frac{1}{\log \frac{1}{1 - \alpha}}}{150} = 1/150$, and therefore each Oracle node needs to store roughly 17 symbols. With ACeD characterized by those parameters, the total communication cost for a client to commit a 12 MB block is roughly 5.38 GB; this represents a 0.5N factor boost over storing just one block. In the normal path, after accepting the 12 MB block, each Oracle node only has to store 448 kB of data, a 3.7% factor of the original data; if there is an incorrect-coding attack, Oracle layer will together upload data of size 339 kB incorrect-coding proof to the trusted blockchain. To download a block in the normal path, a client can use a naive method of collecting all symbols from all Oracle nodes. Despite the conservative approach at block reconstruction, the entire download takes 5.38 GB. A simple optimization involving selectively querying the missing symbols can save significant download bandwidth: a client only needs 896 coded symbols in the base layer to reconstruct the block; thus, in the best case only 42 MB is needed to directly receive those symbols. When there is an incorrect-coding attack, a new user only needs to download the fraud proof which has been generated by other nodes (either client or Oracle nodes); the proof is only of size of 339 kB. Table 3 tabulates the performance metric of ACeD (and baseline schemes) with these parameter settings.

In the uncoded repetition protocol, each node stores the identical block, so the storage cost for both cases is 12MB. In the uncoded dispersal protocol, because all Oracle nodes are honest,
the block is equally divided into $N$ nodes in both cases. In the AVID protocol, each message has the size $\frac{b}{N} + y(\log_2 N + 1) = 4.37$ kB, so the total amount of messages exchanged during the broadcasting stage equals to 354GB; AVID does not commit an invalid block, so only the block hash is required to show the block is invalid. When $\beta = 0.33$ 1D-RS uses the same erasure code as AVID to assign each node a symbol, so their normal case is identical; when there is an incorrect-coding attack, a node needs to download the whole block to reconstruct the block, therefore in the worst case both storage and download are 13.4MB. To tolerate $\beta = 0.49$, 1D-RS needs to decrease the coding ratio to 0.02 while increasing the symbol size, because 200 nodes are required to start the decoding. A lower coding ratio requires nodes to store more (67.1) kB data; download cost are saved by reducing the number of Merkle proof, because only 200 symbols are needed for decoding. In the end, we use the same method to evaluate ACeD with $\beta = 0.33$, both normal case storage and communication complexity improves due to a better dispersal efficiency.

|                        | maximal adversary fraction | storage cost $^*$ | download cost $^*$ $^\dagger$ | worst case         | communication complexity |
|------------------------|----------------------------|------------------|-------------------------------|---------------------|--------------------------|
| uncoded (repetition)   | 0.49                       | 12MB             | 12MB                          | 12MB                | 108GB                    |
| uncoded (dispersal)    | 0                          | 1.3kB            | 12MB                          | 1.2kB               | 12MB                     |
| AVID [8]               | 0.33                       | 4.37kB           | 13.4MB                        | 32B                 | 354GB                    |
| 1D-RS                  | 0.33                       | 4.37kB           | 13.4MB                        | 12.1MB              | 39.4MB                   |
| ACeD                   | 0.33                       | 67.1kB           | 12.1MB                        | 12.1MB              | 604MB                    |
| ACeD                   | 0.49                       | 448kB            | 42MB                          | 339kB               | 5.38GB                   |

* cost is derived by $\frac{b}{N}$ overhead
$^\dagger$ best case

The ACeD Oracle layer interacts with the side blockchains and the trusted blockchain. For a high performance (high throughput, low gas), the oracle layer and its interaction protocols have to be carefully designed for software implementation. In this section, we discuss our system design decisions that guided the implementation.

**Architecture.** We use Ethereum as the trusted blockchain. The oracle nodes interact with Ethereum via a special (ACeD) smart contract; the contract is owned by a group of permissioned Oracle nodes, who individually cannot update the contract, except that a majority of the Oracle nodes together agree and perform a single action on the smart contract. The contract only changes its state after accepting a multisignature of majority votes [6]. We implement both the oracle nodes and side blockchain nodes in RUST; the architecture is depicted in Figure 4. There are four important modules:

1. **Smart contract manager,** when used by a side blockchain node, is responsible for fetching states from the trusted blockchain; when used by an oracle node, it writes state to the trusted blockchain after the block manager has collected sufficient signatures. 2. **Block manager,** when used by a side blockchain node, creates a block (triggered by a scheduler), and distributes coded symbols to oracle nodes; when used by an oracle node and has received messages from side blockchain nodes, the module stores the received symbols and provides them on demand; When used by an oracle committee node, the module collects signatures and send an Ethereum transaction once there are sufficient signatures. 3. **Scheduler,** when used by a side blockchain node, decides when a node can transmit its block (and coded symbols) to oracle nodes; when used by an oracle node, the module decides who the correct block proposer is. 4)
Fig. 4: The left figure depicts a block diagram of a side node; the right figure depicts an oracle node. The sharp rectangle represents an active thread; round rectangles are system states.

**Trusted Chain manager**, when used by a side node, periodically pulls state changes from the trusted blockchain; if a new block is updated, it collects block symbols from all oracle nodes; the module also maintains a blockchain data structure.

A detailed discussion of the design and implementation optimizations of these four blocks is provided in Appendix §D, which also discusses the data structures that maintain the state of ACeD.

**Implementation Optimizations.** The key challenge to a high performance implementation of ACeD is the large number of interactions among the various system blocks. Since the trusted blockchain is immutable, we optimize the oracle and side blockchain client implementations by utilizing parallelism and pipelining as much as possible. (1) **LDPC parallelization**: we bring state of the art techniques from wireless communication designs and encoding procedures (Appendix A of [27]) to our implementation of encoding a block by distributing the workload to many worker threads; we find that the encoding time is almost quartered using 4 threads. A detailed description is deferred to Appendix §B. (2) **Block Pipelining**. We pipeline signature aggregation, block dispersal and retrieval so that multiple blocks can be handled at the same time. We also pipelined the trusted chain manager so that a side node can simultaneously request multiple blocks from oracle nodes. The local state variable, (block id, hash digest), is validated against the smart contract State whenever a new block is reconstructed locally.

## 7 Evaluation

We aim to answer the following questions through our evaluation. (a) What is the highest throughput (block confirmation rate) that ACeD can offer to the side blockchains in a practical integration with Ethereum? (b) What latency overhead does ACeD pose to block confirmation? (c) How much is the gas cost reduced, given the computation (smart contract execution) is not conducted on the trusted blockchain? (d) What are the bottleneck elements to performance?

**Testbed.** We deploy our implementation of ACeD on Vultr Cloud hardware of 6 core CPU, 16GB memory, 320GB SSD and a 40Gbps network interface (for both oracle and side blockchain nodes). To simplify the Oracle layer, we consider all of oracle nodes are committee members, which is reflected in the experiment topology: all the oracle nodes are connected as a complete graph, and each side blockchain node has TCP connections to each of the oracle nodes. We deploy a smart contract written in Solidity using the Truffle development tool on a popular Ethereum testnet: Kovan. Oracle nodes write their smart contracts using Ethereum accounts created with MetaMask, each loaded with sufficient Keth to pay the gas fee of each Ethereum transaction. We
use an LDPC code of $\frac{b}{c} = 128$ input symbols with a coding rate of $r = 0.25$ to construct CIT with $q = 8, d = 8, \alpha = 0.125, \eta = 0.875, t = 16$. We fix the transaction length to be 316 bytes, which is sufficient for many Ethereum transactions.

**Experiment Settings.** We consider four separate experiments with varying settings of four key parameters: the number of oracle nodes, the number of side blockchain nodes, block generation rate, and block size. The experiment results are in Figure 5.

| # side blockchain nodes | # oracle nodes | Block size(MB) | Block generation rate(sec/blk) |
|-------------------------|----------------|---------------|-----------------------------|
| A                       | 5              | 5,10,15,25    | 4                           | 5                           |
| B                       | 5,10,20        | 10            | 4                           | 5                           |
| C                       | 5              | 10            | 4,8,16                      | 5                           |
| D                       | 3,5,8,10       | 10            | 4                           | 8.33,5,3.125,2.5            |

Table 4: Four different experiments varying the parameters of ACeD.

(1) **Throughput.** We measure the rate of state update in the trusted blockchain as viewed from each oracle blockchain nodes; the throughput is then the rate averaged across time and across oracle blockchain nodes. The throughput performance changes with four parameters: In experiments A and B, the throughput is not noticeably impacted as the number of oracles or side blockchain nodes increases. In experiment C, the block size has roughly a linear effect on throughput, which continues until coding the block is too onerous (either it costs too much memory or takes too long). In experiment D, we fix the product of the block generating rate and the number of side blockchain node to constant, while increasing the block generation rate, the throughput increases linearly; the performance will hit a bottleneck when the physical hardware cannot encode a block in a time smaller than the round time.

(2) The **latency** of ACeD is composed of three major categories: block encoding time, oracle vote collection time and time to update the trusted blockchain. We find latency stays relatively constant in all experiments (the exception is experiment C where block encoding is onerous).

(3) **Gas saving.** ACeD transactions cost significantly less gas than a regular Ethereum transaction, independent of the computational complexity of the transaction itself. The cost of an ACeD transaction voted by 10 Oracles nodes on a 4MB block costs on average 570K gas. Based on a historical analysis of 977 Cryptokitties transactions, we estimate a 220 byte transaction costing roughly 180,000 gas: ACeD gas savings are thus over a factor 6000. We emphasize that the saving in gas is independent of block generation rate or the block size, since the smart contract only needs the block header and a list of the oracle signatures.

8 Conclusion and Discussion

Interoperability across blockchains is a major research area. Our work can be viewed as a mechanism to transfer the trust of an established blockchain (eg: Ethereum) to many small (side) blockchains. Our proposal, ACeD, achieves this by building an oracle that guarantees data availability. The oracle layer enabled by ACeD is run by a group of permissioned nodes which are envisioned to provide an interoperability service across blockchains; we have provided a detailed specification of ACeD, designed incentives to support our requirement of an honest majority among the oracle nodes, and designed and evaluated a high performance implementation.

ACeD solves the data availability problem with near optimal performance on all metrics. However, when instantiating the system with specific parameters, we observe that some metrics have to be computed with constant factors which give rise to acceptable but non-negligible performance loss. The critical bottleneck for ACeD is the lower undecodable ratio (the fraction of
ACeD: Scalable Data Availability Oracle

Fig. 5: Throughput (left) and Latency (right) for the 4 experiments A, B, C, D.

symbols in the minimum stopping set) compared to 1D-RS coding; this undermines dispersal efficiency and increases the communication cost under the existence of strong adversary. Therefore, finding a deterministic LDPC code with a higher undecodable ratio will immediately benefit the practical performance of ACeD; the construction of such LDPC codes is an exciting direction of future work and is of independent interest.

9 Acknowledgements

This research is supported in part by a gift from IOHK Inc., an Army Research Office grant W911NF1810332 and by the National Science Foundation under grant CCF 1705007.

References

1. ACeD library, https://github.com/simplespy/ACeD.git
2. Al-Bassam, M., Sonnino, A., Buterin, V.: Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities. arXiv preprint arXiv:1809.09044 (2018)
3. Alon, N., Spencer, J.H.: The probabilistic method. John Wiley & Sons (2004)
4. Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 913–930 (2018)
5. Bagaria, V., Kannan, S., Tse, D., Fanti, G., Viswanath, P.: Prism: Deconstructing the blockchain to approach physical limits. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 585–602 (2019)
6. Breidenbach, L., Daian, P., Juels, A., Sirer, E.G.: An in-depth look at the parity multisig bug. 2017. URL: http://hackingdistributed.com/2017/07/22/deepdive-parity-bug (2017)
7. Buterin, V., Griffith, V.: Casper the friendly finality gadget. arXiv preprint arXiv:1710.09437 (2017)
8. Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: 24th IEEE Symposium on Reliable Distributed Systems (SRDS’05). pp. 191–201. IEEE (2005)
9. Csiszár, I.: The method of types [information theory]. IEEE Transactions on Information Theory 44(6), 2505–2523 (1998)
10. Daian, P., Pass, R., Shi, E.: Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In: International Conference on Financial Cryptography and Data Security. pp. 23–41. Springer (2019)
11. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 66–98. Springer (2018)
12. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Symposium on Self-Stabilizing Systems. pp. 3–18. Springer (2015)
13. Easley, D., Kleinberg, J., et al.: Networks, crowds, and markets, vol. 8. Cambridge university press Cambridge (2010)
14. Fitzi, M., Gazi, P., Kiayias, A., Russell, A.: Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. IACR Cryptol. ePrint Arch. 2018, 1119 (2018)
15. Foti, A., Marino, D.: Blockchain and charities: A systemic opportunity to create social value. In: Economic and Policy Implications of Artificial Intelligence, pp. 145–148. Springer (2020)
16. Gilad, Y., Hemo, R., Miccai, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles. pp. 51–68 (2017)
17. Jirgensons, M., Kapenieks, J.: Blockchain and the future of digital learning credential assessment and management. Journal of Teacher Education for Sustainability 20(1), 145–156 (2018)
18. Kalodner, A., Goldfelder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: 27th {USENIX} Security Symposium (USENIX Security 18). pp. 1353–1370 (2018)
19. Kamath, A., Motwani, R., Palem, K., Spirakis, P.: Tail bounds for occupancy and the satisfiability threshold conjecture. Random Structures & Algorithms 7(1), 59–80 (1995)
20. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Annual International Cryptology Conference. pp. 357–388. Springer (2017)
21. Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., Ford, B.: Omniledger: A secure, scale-out, decentralized ledger via sharding. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 583–598. IEEE (2018)
22. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharding protocol for open blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 17–30 (2016)
23. Miller, A., Bentov, I., Kumaresan, R., McCorry, P.: Sprites: Payment channels that go faster than lightning. CoRR abs/1702.05812 306 (2017)
24. Pass, R., Shi, E.: Hybrid consensus: Efficient consensus in the permissionless model. In: 31st International Symposium on Distributed Computing (DISC 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
25. Poon, J., Buterin, V.: Plasma: Scalable autonomous smart contracts. White paper pp. 1–47 (2017)
26. Rana, R., Kannan, S., Tse, D., Viswanath, P.: Free2shard: Adaptive-adversary-resistant sharding via dynamic self allocation. arXiv preprint arXiv:2005.09610 (2020)
27. Richardson, T., Urbanke, R.: Modern coding theory. Cambridge university press (2008)
28. Teutsch, J., Reitwießner, C.: A scalable verification solution for blockchains. arXiv preprint arXiv:1908.04756 (2019)
29. Yang, L., Bagaria, V., Wang, G., Alizadeh, M., Tse, D., Fanti, G., Viswanath, P.: Prism: Scaling bitcoin by 10,000 x. arXiv preprint arXiv:1909.11261 (2019)
30. Yu, H., Nikolić, I., Hou, R., Saxena, P.: Ohie: Blockchain scaling made simple. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 90–105. IEEE (2020)
31. Yu, M., Sahraei, S., Li, S., Avestimehr, S., Kannan, S., Viswanath, P.: Coded merkle tree: Solving data availability attacks in blockchains. In: International Conference on Financial Cryptography and Data Security. pp. 114–134. Springer (2020)
A Proof of Lemmas and Theorems

A.1 Proof of Lemma 1

Consider a simple model where there are $N$ nodes and there are $M$ chunks to be distributed. We want to distribute these chunks to the nodes (each node may receive many chunks) such any $\gamma$ fraction of nodes together contains $\eta$ fraction of chunks. The efficiency of the chunk dispersal protocol is given by $\lambda$ (which is $M$ divided by the product of $N$ and the number of chunks-per-node).

Definition 2. The chunk dispersal algorithm is said to achieve parameters set $(M, N, \gamma, \eta, \lambda)$ if there is a collection of sets $C = \{A_1, \ldots, A_N\}$ such that $A_i \subseteq [M]$, $|A_i| = \frac{M}{N \lambda}$. Also, for any $S \subseteq [N]$ with $|S| = \gamma N$, it holds that $|\bigcup_{i \in S} A_i| \geq \eta M$.

To simplify the 5 dimensional region, we consider the tradeoff on the three quantities: a certain triple $(\gamma, \eta, \lambda)$ is said to be achievable if for any $N$, there exists $M$ such that the chunk dispersal algorithm can achieve parameters set $(M, N, \gamma, \eta, \lambda)$.

We show the existence of (several) chunk dispersal algorithms using the probabilistic method. Consider a randomized chunk dispersal design, where each element of each $A_i$ is chosen i.i.d. uniformly randomly from the set of $M$ possible chunks. This gives raise to a randomized code $C$.

We prove the following statement:

$$P\{C \text{ is NOT a } (M, N, \gamma, \eta, \lambda) \text{ code} \} \leq e^{-n}$$ (7)

Let $k = M/(N \lambda)$ be the number of chunks at a given node.

$$P\{C \text{ is not a valid code} \} = P(\exists S \text{ with } |S| = \gamma N : |\bigcup_{i \in S} A_i| \leq \eta M)$$ (8)

$$\leq \sum_{S \subseteq [M] : |S| = \gamma N} P(|\bigcup_{i \in S} A_i| \leq \eta M)$$ (9)

$$\leq e^{NH_e(\gamma)} P(|\bigcup_{i \in S} A_i| \leq \eta M)$$ (10)

where we have used that $(\frac{N}{\gamma N}) \leq 2^{NH(\gamma)} = e^{NH_e(\gamma)}$, with $H(.)$ being the binary entropy function measured in bits (logarithm to the base 2) and $H_e$ being the binary entropy measured in natural logarithm (nats), a standard inequality from the method of types.

The key question now is for a fixed $S$, we need to obtain an upper bound on $P(|\bigcup_{i \in S} A_i| \leq \eta M)$. We observe that under the random selection, we are choosing $\gamma / \lambda M$ chunks (sampled randomly with replacement). The mathematical question now becomes, if we choose $\rho M$ chunks randomly (with replacement) from $M$ chunks, what is the probability that we get at least $\eta M$ distinct chunks.

Lemma 4. We sample $\rho M$ chunks from a set of $M$ chunks randomly with replacement. Let $Y$ be the number of distinct chunks. Let $x = (1 - \eta) \exp(\rho)$.

$$P\{Y < \eta M \} \leq e^{-(1-\eta)(\frac{(x-1)^2}{x+1})}$$ (11)

Proof. Let $Z = M - Y$ be the number of chunks not sampled. Then $\mu := E[Z] = (1 - \frac{1}{M})^{\rho M} \approx Me^{-\rho}$ (when $M$ is large). We can write $Z = \sum_{i=1}^{M} Z_i$, where $Z_i$ is the binary indicator random
variable indicating that chunk $i$ is not sampled. We note that a direct application of tail bounds for i.i.d. random variables does not work here. However, there are results analyzing this problem in the context of balls-in-bins and they show that the following tail bound does indeed hold (see Theorem 2 and Corollary 1 in [19]). We can write the tail bound on $Z$ as follows:

$$P(Z > ℓμ) \leq e^{-\frac{(e^ε-1)^2}{2+ε} μ} \forall ℓ > 1$$  \hspace{1cm} (12)

Let $\bar{η} = 1 - η$ and $ℓ = e^ε\bar{η}$, we get

$$P(Z > \bar{η}M) \leq e^{-\frac{(e^ε\bar{η}-1)^2}{2+ε} M} \text{ if } e^ε\bar{η} > 1$$  \hspace{1cm} (13)

Define the function, $f(η, ρ) = \frac{(e^ε\bar{η}-1)^2}{2+ε}$ (we note that $f(·)$ is a positive function), we have

$$P(Y < ηM) \leq e^{-f(η, ρ)M} \text{ if } ρ > \log(\frac{1}{1-η}).$$

Continuing from before, we get

$$\mathbb{P}(C \text{ is not a valid code}) \leq e^{N.H_c(γ)-M.f(η, ρ)} \text{ if } \frac{γ}{λ} > \log \left( \frac{1}{1-η} \right)$$  \hspace{1cm} (14)

If we fix a $N$ and take $M$ large enough, we can make the right hand side large enough to make the probability that $C$ is not a valid code vanish to as small as we want. This result has two interpretations: (1) as long as the probability of being a valid code is strictly greater than zero, then it proves the existence of a deterministic chunk distribution algorithm, (2) if we can make the probability of being a invalid code arbitrarily close to zero, then we can use the randomized algorithm directly with a vanishing probability of error. Fixing a $N$ and taking $M$ large can make the RHS arbitrarily small and thus the stronger second interpretation can be used. This concludes the proof of the theorem.

### A.2 Proof of Lemma 2

**Proof.** By definition, $f_j(i)$ can be decomposed to two functions, $p_j(i)$ and $e_j(i)$. According to equation 3 in [19], $p_j(i)$ is computed by modulo, so there are at most $c = \frac{m_j}{m_j}$ base symbols mapping to one symbol in layer $j$. In the worst case, $γm_j$ distinct base layer symbols map to $\frac{γm_j}{c_1} = γr(m_j)$ distinct symbols in layer $j$ in the range $[0, rm_j]$. Similarly, there are $d = \frac{M_j}{(1-r)m_j} = \frac{M_j(γqr)^{-1}}{(1-r)m_j} = \frac{(γqr)^{-1}}{1-r}$ base layer symbols mapping to one symbol at layer $j$, and therefore in the worst case there are $\frac{2m_j}{d} = γ(1-r)m_j$ distinct symbols in the range $[rm_j, m_j]$. Since two range do not overlap, in total there are $γm_j$ distinct symbols for each layer. Hence it completes the proof.

### A.3 Proof of Lemma 3

**Proof.** First we show for every $i \in [m_j]$ and $2 \leq j < ℓ$, $p_j(i)$ and $e_j(i)$ are sibling. According to the definition,

$$p_j(i) = i \mod rm_{j-1} = i \mod qr^2m_{j-2}$$  \hspace{1cm} (15)

$$p_{j-1}(i) = i \mod rm_{j-2} \text{ (by definition) }$$  \hspace{1cm} (16)

$$= p_j(i) \mod rm_{j-2} \text{ (discuss below) }$$  \hspace{1cm} (17)

To prove equation 17, suppose $i = p_j(i) + xqr^2m_{j-2}$ and $i = p_{j-1}(i) + yrm_{j-2}$ for some integer $x, y$. By rearranging terms, $p_j(i) - p_{j-1}(i) = rm_{j-2}(y-xqr)$. If $p_j(i) - p_{j-1}(i) \neq 0$, we mod $rm_{j-2}$
on both sides, and therefore \( p_{j-1}(i) = p_j(i) \mod rm_{j-2} \). If \( p_j(i) - p_{j-1}(i) = 0 \), then obviously \( p_{j-1}(i) = p_j(i) \mod rm_{j-2} \). Therefore in any case, equation \( 17 \) holds, which means \( p_{j-1}(i) \) is the parent of \( p_j(i) \).

Then let’s see another function \( e_j(i) \), by definition we have

\[
e_j(i) = rm_{j-1} + (i \mod (1 - r)m_{j-1}) \\
= rm_{j-2}qr + (i \mod (1 - r)qrm_{j-2})
\]

To calculate its parent according to aggregation rules, we get similar result that \( e_j(i) - p_{j-1}(i) = rm_{j-2}(y - xq(1 - r)) \) thus

\[
e_j(i) = i \mod (1 - r)qrM_{j-2} \\
= p_{j-1}(i) \mod rm_{j-2}
\]

\( p_{j-1}(i) \) is also the parent of \( e_j(i) \). Hence it completes the proof.

### B Efficient Encoding for LDPC Codes

A block of transactions is coded to symbols before distributing them to the Oracle nodes. We have implemented a CIT library in RUST; compared to CMT (coded Merkle tree) library, the major distinction is in the data interleaving algorithm. We find that the encoding time of CMT is very time consuming: on a Mac 2.6 GHz 6-Core Intel Core i7 computer, the library required about 34 seconds to encode a 4 MB block on a basis of 128 symbols with 0.25 coding ratio. The issue is due to a huge amount of XOR operations required for encoding the base layer. In our implementation, we have developed two strategies to overcome the problem by adding parallelism to the encoding process and by devising a code transformation that reduces coding complexity. Parallelization involves two components: a light weight read-only hash map and a multiple-thread environment for partitioning the workload. An inverse map is created whose key is the index of input symbols, and value is a list of parity symbols to be XOR with according to the encoding matrix. We use a single-producer multiple-workers paradigm where the producer thread uses the inverse map to properly distribute the input symbol. Worker threads partition the parity set such that each thread is responsible for an equal number of parity symbols. The input symbol is communicated through a message queue, and once a worker receives the input symbol, it XORRes the symbol locally. The producer distributes all input symbols and sends a final control message to all workers to collect the result. The optimization brings out a significant speedup, reducing the encoding time of a 4MB block from 31 second to about 10 sec using 4 threads. Adding as many as 10 threads further reduces the encoding time to 8.5 seconds, and eventually hitting physical (CPU) limits. In the second strategy, we attempted to change Coding matrix to reduce the number of XOR computation at the root. The conventional encoding complexity of LDPC codes is \( O(n^2) \) where \( n \) is the length of coded symbols. We could transform the original encoding matrix to a upper triangular form that has a computation complexity of \( O(n) \). But after a careful analysis, we discover an extra constant factor overkills the reduced coding complexity, so in the end we rely solely on parallelism to reduce coding latency.

### C Incentive Analysis

#### C.1 Motivation and Countermeasures

For oracle nodes, downloading and checking data requires nontrivial computation effort, selfish oracle nodes have an incentive to follow others’ decisions and skip verification to compete for
more reward by voting for more blocks. Such a phenomenon, known as information cascade in the literature, sets forth that people tend to make decisions according to the inferences based on actions of earlier people. As a consequence, an oracle node may submit vote for a block but can not provide any data when a retrieval request is received. To solve this problem, we introduce an audit scheme. When a block is added to the side blockchain, with probability $p_a$, a voted node will be selected to submit received data chunks. Our dispersal protocol guarantees that each oracle node ought to store a specific subset of data chunks. For those committed blocks which are not successfully appended in the side blockchain, all voted nodes should submit data or lose their stake.

Even if there are several nodes in the aggregation committee, only the first one who submits the valid signature can claim the reward, thus rational committee members would choose to submit votes containing the first bath of signatures. And since only those oracle nodes whose signatures are used to aggregate the final signature would receive a reward, nodes with the largest network latency may never get any income. Furthermore, adversaries can violate arbitrary subset of honest nodes’ interests by excluding them intentionally. We can not influence the adversarial behavior, but to encourage selfish nodes to include more signatures, the committee reward will be proportional to the number of signatures in final aggregation.

C.2 Model

We consider a network to be a set of $N$ configured oracle nodes (oracle nodes) $O = \{o_1, o_2, \cdots, o_N\}$. Suppose $\beta N$ nodes in the network are Byzantine, who behave arbitrarily. And that every non-Byzantine nodes are Rational, they follow the protocol if it suits them, and deviate otherwise. In the registration phase, each oracle node needs to deposit an amount of stake $stk_o$ for registering a pair of valid BLS key set.

There is a set of blocks proposed by side nodes. To propose a block, the proposer needs to deposit an amount of stake $stk_b$ and will receive block reward $B$ if successful. A fraction of block reward $\eta B$ will be used for rewarding oracle nodes. There is a small committee $C \subset O$ in the oracle layer. If a block is committed, the committee member who submits the commitment will get an extra reward, which comes from a constant submission fee $r_m$ paid by other oracle nodes.

Basically there are three general actions a node can take, cooperate, offline and defect, denoted as $D, O, C$ respectively. We define $All-C, All-O$ and $All-D$ representing the strategy that all nodes choose to be cooperative, offline and defective. We design the utility functions for block proposer, oracle nodes and committee members when a block is successfully committed in Table 5.

| Strategy | Block Proposer | Oracle Nodes | Committee Member |
|----------|----------------|--------------|------------------|
| $D$      | $-stk_b$       | $-p_a stk_b + (1 - p_a) \frac{\eta B}{k} - r_m$ | $-stk_m$ |
| $O$      | 0              | 0            | 0                |
| $C$      | $(1 - \eta)B$ | $\frac{\eta B}{k} - r_m - c_k$ | $kr_m - c_m$ |

Table 5: Utility functions for different types of nodes. $k$ is the number of signatures aggregated in committed signature.

C.3 Incentive Compatibility

A protocol is incentive compatible if rational actors see no increase in their utility from unilaterally deviating from the protocol. To prove that, there are several things the rational participants may care about.
For oracle nodes, there is a reward for having a block it successfully voted for added to the side blockchain. They can’t do much to influence the reward, except creating more voting accounts which means to store more data and deposit more stake. For block producer, there is a reward \((1 - \eta)B\) when the proposed block is added to the side blockchain. Rational side nodes can’t gain more rewards by deviating the protocol thus have no reason to violate the protocol. For committee member, they claim rewards when a vote is accepted and more participants in aggregation bring more reward with the submission fee.

We prove that All-C strategy is a Nash Equilibrium below.

**Theorem 3.** There exists a strategy \(\{e^*_i\}_{i \in [1, n]}\) for all rational oracle nodes in system to reach Bayesian Nash Equilibrium: \(\forall k \in [1, n], k \text{ is rational,} \)

\[
E[U_k(e^*_k) | \{e^*_i\}_{i \in [1, n]}] \geq E[U_k(e_k) | \{e^*_i\}_{i \neq k, e_k}]
\]

and the following statements hold:

1. All-O strategy is a Bayesian Nash Equilibrium.
2. All-C strategy is a Bayesian Nash Equilibrium.

**Proof.** A protocol is a Byzantine Nash Equilibrium if rational actors see no increase in their utility from unilaterally deviating from the protocol. Firstly, consider \(\{e^*_i\}_{i \in [1, n]} = \text{All-O}\), it can be easily derived that \(E[\text{All-O}] = 0\). Suppose any voter \(k\) wants to change the protocol, no matter it chooses to cooperate or defect, it can not change the consensus thus gets no reward and even pays some cost for computation when choosing to cooperate, thus All-O is a Bayesian Nash Equilibrium.

Suppose a block is eventually committed. The expected utility for All-C is,

\[
E[\text{All-C}] = \frac{\eta B}{k} - r_m - c_s
\] (18)

The expected utility for \(k\) to choose to offline and defect is,

\[
E[e_k = \text{O}] = 0
\] (19)

\[
E[e_k = \text{D}] = -p_a stk_v + (1 - p_a) \frac{\eta B}{k} - r_m
\] (20)

Letting \(E[\text{All-C}] > E[e_k = \text{D}]\) and \(E[\text{All-C}] > E[e_k = \text{O}]\) derives that All-C strategy is a Bayesian Nash Equilibrium when \(p_a(stk_v + 1) - c_s > 0\) and \(\frac{\eta B}{k} > r_m + c_s\). Since both \(c_s\) and \(r_m\) are constant, we can always choose stake value to make it reach the equilibrium.

### D Design and Implementation of ACeD Modules

The **Smart Contract manager** is an event loop which waits for signal from other managers. Its main purpose is to communicate with the smart contract by either sending Ethereum transaction to write new states on the smart contract, or calling the smart contract to read the current state.

The **Trusted Chain manager** is an event loop used by any side nodes for periodically pulling the latest state of the main chain by asking the smart contract manager. Every side node uses it to monitor any state update from the Oracle nodes. An update occurs when the smart contract block id is higher than the block id maintained in the blockchain. Each side blockchain node queries the trusted blockchain periodically to be aware of the state change. When a new state is received, the side node queries all Oracle nodes using the block id, for assembling the new
Table 6: List of symbols used in incentive analysis

| Symbol | Definition |
|--------|------------|
| $r_m$  | Rewards per signature for a committee member |
| $B_j$  | Block rewards for block $j$ |
| $c_s$  | Costs for oracle nodes to verify data chunks |
| $c_m$  | Costs for committee members to aggregate signatures |
| $stk_o$ | Unit stake of oracle nodes |
| $stk_m$ | Unit stake of committee members |
| $stk_b$ | Stake of block proposer |
| $\beta$ | Proportion of byzantine nodes |
| $\theta$ | Threshold of aggregated signature |
| $\eta$ | Proportion of block rewards as block proposition cost |
| $p_a$ | Probability to audit when a new block is added |

block. It then checks Merkle proof in the header to check block integrity. Then the module uses the new hash digest in the header to verify the integrity of the header. The computation is done by hashing the concatenation of the previous hash digest and the hash digest of the new block header $H$; then compare it with the hash digest in the smart contract. The trusted blockchain manager continues until it completes to the latest state and has a consistent view of blockchain compared to the states on the trusted blockchain.

The scheduler runs an endless loop which measures the current time to determine the correct side node proposer at its moment. It uses the current UNIX EPOCH to compute time elapsed from a time reference registered at the contract creation, and divides Slot Duration, a system parameter, to get the current slot id. Therefore both Oracle and side node can use slot id and the token ring to determine the valid block proposer at current time. When a node proposes, its block id is set to the current slot id. After receiving block symbols using dispersal protocol, oracle nodes verify the message sender using the identical scheduling rule. In the special situation while a block is pending in the network and a new block is proposed in the next slot, the pending block might be updated or rejected by the smart contract due to block id rule constraint by the smart contract. Because a block-id is accepted only if it is monotonically increasing, the releasing of the pending block does not hurt system in either case: If the pending block arrived to smart contract after the new block, the pending block is excluded because it has a lower block id than the new block; if the pending block is included, there might be some conflicting transactions in two blocks, but since the transactions in the pending block cannot be arbitrarily updated by adversaries after its release, the adversaries cannot adaptively damage the system. This property allows an optimization to improve throughput (discussed later).

The block manager is an event loop that sends and receives message among peers. When an oracle node receives header and coded symbols from a side node, it stores them into the symbols database, and submits its signature of the header to its Committee nodes after it receives a sufficient amount of symbols. If an oracle node is chosen as a committee node, the node receives signature and aggregate them until the number suggests there is a sufficient amount of symbols stored at honest nodes. The block manager asks smart contract manager to send an Ethereum transaction to write the new state on the smart contract.

The states of ACeD are maintained in three data structures:

1. block database, residing on persistent storage, is used by Side nodes to store decoded blocks reconstructed from the Oracle nodes
2. symbols database, residing on persistent storage, is used by oracle nodes to store coded symbols from the Side nodes.
The block database is a key-value persistent storage, where the key is block id and the value is a block $B = (H, C)$ which contains a header $H$ and content $C$. The header is a tuple of $(\text{blockid}, D)$ where $D$ is the digest of the content $C$. Instead of relying on a cryptographic puzzle for accepting blocks, each oracle node uses a deterministic token ring available from the smart contract to decide which block symbols to accept. The token ring is a list of registered side chain nodes eligible for proposing blocks. A block is considered valid only if it is received at the correct time from the correct side node.

**E  Performance table**

We compared five data dispersal protocols and analyzed their performances using maximal adversarial fraction and communication complexity; we further articulate the storage overhead for the Oracle layers and download overhead for each client into two cases depending on whether there is invalid coding attacks in the system. The definition of those metrics is located in Table 3 of section 2. Because an adversarial client can produce any arbitrary block whose information in the denominator in 2 is 0, we therefore used different metrics to compute the performance whose details are scattered in each following paragraph.

In the uncoded repetition data dispersal protocol, a block proposer client sends the original block to all oracle nodes to ensure the data availability. The system needs more than $1/2$ fraction of the votes for committing a block digest of constant $y$ bits into the trusted blockchain. The total amount of bits communicated equals to $Nb$, where $b$ is block size and $N$ is total number of Oracle nodes. Hence the communication complexity is $Nb$. In the normal case when a block is coded correctly, storage overhead is $Nb/b$ whereas the download overhead is $b/b$. When an adversary uploaded an invalid block by distributing different blocks to each node, the worst case for storage overhead is identical to the normal case, because every Oracle nodes have to store the data block to make sure data is available when later adversaries Oracle nodes decide to make their block available. The download efficiency is $O(1)$ because as long as a client contacts an honest Oracle node, it is sufficient to check the data availability.

In the uncoded data dispersal protocol, the block proposer client divides the original block into $N$ chunks, one for each Oracle node. The system cannot tolerate any adversary because any withheld chunk can make the block unavailable. The total communication complexity is $b$ since a block is equally distributed. In both normal and worst case, the storage overhead is $b/b$ because every node only store $b/N$ chunks. Similarly in both normal and worst cases, the download overhead is $b/b$ because a client downloads at most $b/N$ chunks from $N$ Oracle nodes.

In the 1D-RS data dispersal protocol, at least $1/2$ fraction of Oracle nodes need to be honest for them to agree to commit a block digest. The communication complexity is $O(b)$ using a chunk dispersal protocol discussed in section 4. In the normal case, each node only stores $O(b/N)$ number of chunks, and therefore the storage overhead is $O(1)$; similarly when a client node wants to download the block, it collects all chunks from the Oracles node with download overhead $O(1)$. In the worst case, when an adversarial client disperses chunks with invalid coding, Oracles nodes in the end would produce an invalid coding proof for proving that the block is invalid to all querying clients. In 1D-RS, the fraud-proof consists of all chunks as one needs to decode itself to detect inconsistency. Therefore the overall storage is $O(N(b/N + \log b))$, when $b$ is large ($b/N >> \log b$), the overall coding proof size is $O(b)$. For computing the storage overhead, we consider the total information stored in the denominator to be the size of block digest, which is used by a client to start the block retrieval. When a client requests for an invalid coded block, the client needs to download the incorrect-coding proof to convince itself that the block is invalid, hence the download overhead is the same as storage overhead.
In the AVID protocol, Oracle nodes run asynchronously with security threshold $1/3$ \cite{8}. During the block dispersal, every oracle node needs to recollect sufficient symbols for reconstructing the original blocks, hence the total communication complexity is $O(Nb)$. In the normal case, after block is checked valid at the dispersal phase, each node discard chunks except the ones sent from the block producer, hence the storage overhead is $O(1)$; similarly when another client downloads the data, it only needs to download $O(b)$ data, hence its download overhead is $O(1)$. When there is an invalid encoding attack, it can be detected by AVID, and then nodes would discard the block. So in the worst case, storage overhead of AVID has the same performance as its normal case. Similarly, since AVID can detect invalid encoding before committing to the trusted blockchain, the worst case download overhead is $O(1)$, the same as its normal case.

In the ACeD dispersal protocol, at least $1/2$ fractions of Oracle nodes need to be honest to commit the block digest. The communication complexity is $O(b)$ by using the dispersal protocol in section 4. In the normal case, each node only stores $O(b/N)$ chunks hence the storage overhead is $O(1)$, and similarly the download overhead is $O(1)$. In the worst case, when an invalid block is dispersed, Oracle layer generates an incorrect-coding proof of size $O(\log b)$ as shown in Theorem \cite{4}. The incorrect-coding proof is then stored in the trusted blockchain. When a clients query for such block, the proof is replied so that the client can be convinced that the block is invalid. For computing the storage and download overhead in the worst case, we consider both the stored information and the size of reconstructed data to be a single hash which is the pointer to the data.