Information Dispersal with Provable Retrievability for Rollups

Kamilla Nazirkhanova, Joachim Neu, and David Tse

Stanford University
{nazirk,jneu,dntse}@stanford.edu

Abstract. The ability to verifiably retrieve transaction or state data stored off-chain is crucial to blockchain scaling techniques such as rollups or sharding. We formalize the problem and design a storage- and communication-efficient protocol using linear erasure-correcting codes and homomorphic vector commitments. Motivated by application requirements for rollups, our solution departs from earlier Verifiable Information Dispersal schemes in that we do not require comprehensive termination properties or retrievability from *any* but only from *some known* sufficiently large set of storage nodes. Compared to Data Availability Oracles, under no circumstance do we fall back to returning empty blocks. Distributing a file of 28.8 MB among 900 storage nodes (up to 300 of which may be adversarial) requires in total $\approx$ 95 MB of communication and storage and $\approx$ 30 s of cryptographic computation on a single-threaded consumer-grade laptop computer. Our solution requires no modification to on-chain contracts of Validium rollups such as StarkWare's StarkEx. Additionally, it provides privacy of the dispersed data against honest-but-curious storage nodes.

1 Introduction

1.1 Rollups

Ethereum, like many blockchains, suffers from poor transaction throughput and latency. To address this issue, various consensus-layer and *off-chain* scaling methods were introduced. While consensus-layer solutions such as sharding [15,14] or multi-chain protocols [2,25] aim at improving the base blockchain protocol, off-chain 'layer 2' solutions such as payment channels [7,17] and rollups [12] aim at moving transaction processing and storage off-chain. The base blockchain then serves only as a trust anchor, rollback prevention mechanism, and arbitrator in case of misbehavior and disputes among participants. *Rollups* in particular introduce an on-chain smart contract representing certain application logic, to and from which rollup users can transfer funds to enter and exit the rollup, and who watches over proper execution of the state machine that describes the rollup's application logic. Rollup users appoint an operator whose role is to execute the contract's state machine and keep track of updated state such as users'

KN and JN contributed equally and are listed alphabetically.

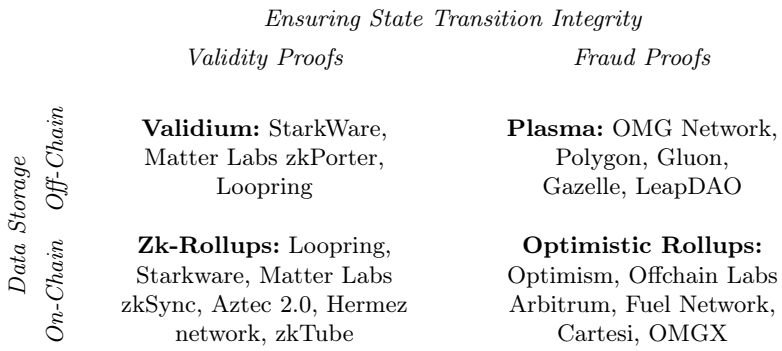

Fig. 1. Layer 2 and rollup projects grouped into four categories according to how validity of state transitions and data availability is ensured (fraud/validity proofs vs. data storage on/off chain). Source: https://ethereum.org/en/developers/docs/scaling/

balances. For this purpose, the operator collects transactions issued by users and executes them off the main chain, but periodically posts a state snapshot to the main chain in order to irrevocably confirm transaction execution and, thus, inherit the main chain's safety guarantee. To ensure liveness, rollup users need to be able to enforce application logic and to exit the rollup with their funds, even if the rollup operator turns uncooperative. To this end, if a user presents proof of their balance according to the latest state snapshot, then the on-chain contract will pay out their funds to the user and thus enforce the user's exit. Rollup designs differ in two crucial aspects. First, how to ensure that the state is only updated in accordance with the application logic. Second, how to guarantee that users are able to exit even if the operator turns malicious and withholds the information necessary for users to prove their balances on-chain.

1.2 Auditability of State Transitions and Data Availability

Rollup designs can be grouped into four categories, as illustrated in Figure 1, according to how they ensure validity of state transitions and availability of transaction information. For the problem of ensuring that application logic is followed, one approach is to use *fraud proofs*: anyone can re-execute the application logic on the inputs at hand and check that the state transitions are correct. If they are not, they present proof of a fraudulent state transition to the on-chain contract which will step in as an arbitrator and enforce application logic. Rollups using fraud proofs are called *optimistic rollups*. A second approach is based on *validity proofs*, where instead of detecting fraud after the fact, fraud is prevented from the get-go by requiring the operator to provide cryptographic proof [9,3] of proper state update. This approach is used in *zk-rollups*.

To ensure that rollup users are able to track proper execution of application rules and to prove their balances, there are again two approaches. The relevant information could be made available either on the main chain, perhaps in a con-

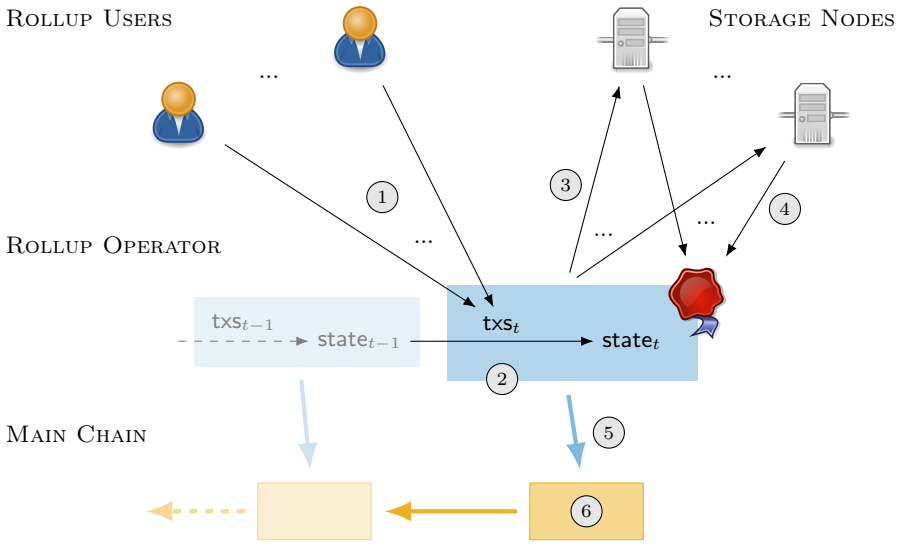

Fig. 2. Normal operating mode of a *Validium rollup*, *i.e.*, a zk-rollup with off-chain storage: (1) Rollup operator collects transactions from rollup users. (2) Transactions are executed by operator off-chain and new state of rollup is calculated. (3) Transaction and state data is dispersed to storage nodes by operator. (4) Storage nodes confirm receipt. Operator collects sufficient number of confirmations into certificate of data retrievability. (5) Operator sends commitment to state and certificate of retrievability to main chain. (6) State snapshot and certificate of retrievability are verified and if valid accepted by main chain.

densed form, or stored off the main chain but with some credible assurance that the data is in fact available for users to retrieve. For the latter purpose, *Validium rollups*, *i.e.*, zk-rollups with off-chain storage such as StarkWare's 'StarkEx', introduce a committee of trusted storage nodes. For the normal operating mode of a Validium rollup see Figure 2. The rollup operator deposits a copy of the relevant data with each storage node, who in turn confirm receipt. A state snapshot is accepted by the main chain only if enough storage nodes have confirmed receipt of the corresponding full data. As long as enough storage nodes remain honest and available, rollup users can always turn to them to obtain the data necessary to prove fraud or balances, should the rollup operator withhold it.

This solution, however, is not communication- or storage-efficient. The operator has to send a copy of the entire data to every storage node which in turn stores an entire copy. Therefore, this solution is not scalable and works only for a relatively small number of storage nodes (*e.g.*, a current application of the StarkWare Validium rollup uses 8 storage nodes [10]), which leads to heavy centralization. Furthermore, the privacy of user data is violated, as storage nodes can view the entire state.

1.3 Information Dispersal with Provable Retrievability

A more communication- and storage-efficient solution is provided by Verifiable Information Dispersal (VID) as embodied by Asynchronous Verifiable Information Dispersal (AVID [5]) and its successors AVID-FP [11] and AVID-M [24]. Generally speaking, AVID schemes encode the input data block into chunks and every storage node has to store only one chunk rather than the full block. The correctness of the dispersal is verifiable, meaning that the consistency of chunks is ensured.

A VID scheme consists of two protocols, Disperse and Retrieve, satisfying [11]:

(a) **Termination.** If Disperse(B) is initiated by an honest client, then Disperse(B) is eventually completed by all honest storage nodes.
(b) **Agreement.** If some honest storage node completes Disperse(B), all honest storage nodes eventually complete Disperse(B).
(c) **Availability.** If 'enough' honest storage nodes complete Disperse(B), an honest client that initiates Retrieve() eventually reconstructs some block B'.
(d) **Correctness.** After 'enough' honest storage nodes complete Disperse(B), all honest clients that initiate Retrieve() eventually retrieve the same block B'. If the client that initiated Disperse(B) was honest, then $B' = B$.

Although existing VID schemes can be used to ensure data availability for rollups, they miss properties that are required for this application, while having others that are not needed, resulting in unnecessary complexity (*cf.* Figure 4). For Validium rollups, it is crucial that the on-chain rollup contract can verify (④, ⑤) the *retrievability* of the underlying data before accepting a new state update, to ensure that users have access to the data required to enforce the contract (or be able to exit) on-chain in case of an uncooperative operator. For this purpose, consistent retrieval of 'some' block $B' \neq B$ is not enough, the *retrievability* (③) of the original block B needs to be ensured. Oppositely, comprehensive termination properties (②) such as Termination and Agreement are not needed, and some VID schemes (here AVID) provide properties exceeding VID that are not required for the rollup application (①)

We introduce *Semi-AVID with Provable Retrievability* (Semi-AVID-PR) to capture the requirements in the rollup application. Besides Disperse and Retrieve, Semi-AVID-PR provides Commit to succinctly and unequivocally identify data blocks and Verify to verify certificates of retrievability. The requirements are:

(a) **Binding.** Commit is a binding deterministic commitment to a block of data.
(b) **Correctness.** If an honest client initiates Disperse(B), then eventually it obtains a valid certificate of retrievability for Commit(B).
(c) **Soundness.** If an honest client invokes Retrieve(P, C) with a valid certificate of retrievability P for commitment C, then eventually it obtains a block B such that Commit(B) $= C$.

We propose a communication- and storage-efficient Semi-AVID-PR protocol with practical computational cost, which is compatible with the established on-chain smart contracts of and thus can be readily adopted for existing Validium

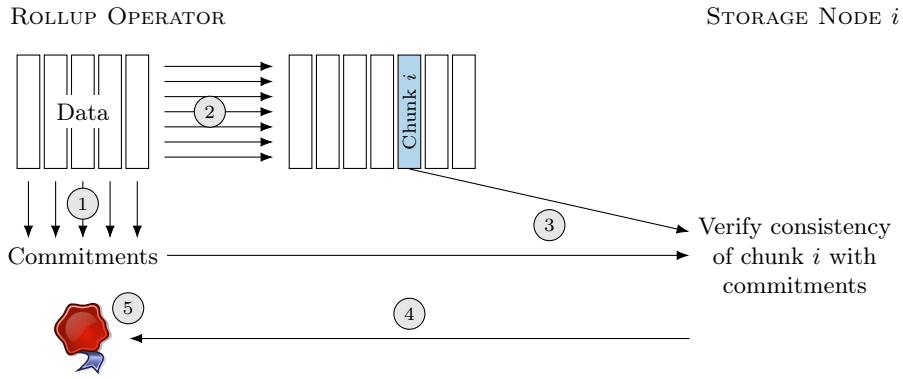

Fig. 3. Dispersal in Semi-AVID-PR. (1) Client arranges data in matrix and computes vector commitments of columns. (2) Client encodes data row-wise. (3) Commitments and chunk i are sent to storage node i. (4) If chunk i is consistent with commitments, storage node acknowledges receipt. (5) Enough acknowledgements form certificate of retrievability.

rollups, *e.g.*, such as StarkWare's StarkEx. A high-level illustration of Disperse of our scheme is provided in Figure 3. In our protocol, the rollup operator computes commitments to chunks of the initial data and encodes it using an erasure-correcting code. Encoded chunks are dispersed among the storage nodes along with the commitments. Similarly to AVID-FP, the commitments allow storage nodes to verify the consistency of their local chunk with the file for which they are about to acknowledge the receipt of a chunk. If their chunk is consistent, a storage node confirms receipt to the operator. Upon collecting enough confirmations, the operator can produce a certificate of retrievability for the respective file, which is later verified by the main chain before accepting the new state snapshot. Furthermore, our scheme can provide privacy of the data against honest-but-curious storage nodes because storage nodes can only view some coded chunk of the block, and additional blinding can be used (details in Appendix B).

1.4 Related Work

The AVID protocol [5] satisfies not only the VID properties, but furthermore guarantees that eventually a dispersed file can be retrieved from *any* subset containing k honest storage nodes (*cf.* Figure 4, (1)), rather than from only *some fixed known* subset, as for Semi-AVID-PR. This is achieved by an additional round of echoing chunks which leads to a high communication cost for AVID (*cf.* Table 1), whose communication complexity is $O(n|B| + n^2|C|)$, where $|B|$ is the block size, n is the number of storage nodes, and $|C|$ is the size of a commitment, *e.g.*, 32 B. Semi-AVID-PR's communication and storage complexity is $O(|B| + n^2|C|)$.

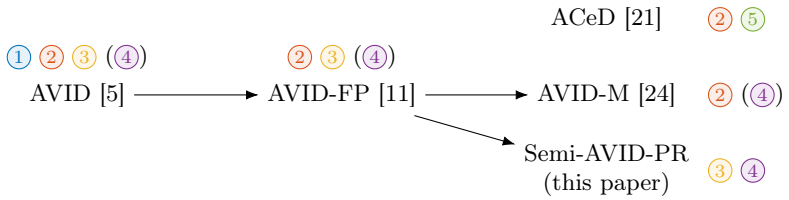

Fig. 4. Related protocols and supported properties: ① Retrieval from *any* sufficiently large set of storage nodes ② Comprehensive termination guarantees ③ Retrievability guaranteed ④ Issues certificates of retrievability ⑤ Dispersal verifiable on-chain

AVID-FP [11], a successor of AVID, brings the communication complexity to $O(|B| + n^3|C|)$ (which is an improvement for large $|B|$ and moderate n) using homomorphic fingerprinting (as in Semi-AVID-PR) so that storage nodes can verify their chunks without echoing them. Since chunks can be verified, retrievability is guaranteed (③). A round of AVID (with the fingerprints) is still used to achieve comprehensive termination properties (②).

A recent advancement of VID, AVID-M [24], improves the communication complexity to $O(|B| + n^2|C|)$ by reducing the size of fingerprints using Merkle trees [16]. However, chunks cannot be verified anymore, and AVID-M retrieves as empty block any data that was maliciously encoded during dispersal, so that retrievability is no longer guaranteed. This makes AVID-M less suitable for application to rollups.

The situation is similar for Data Availability Oracles such as ACeD [21]. If a block was invalidly encoded during dispersal by a malicious client, then Data Availability Oracle ensure consistency across retrieving clients, but no guarantee is provided about how the retrieved content relates to the dispersed content.

Furthermore, Data Availability Oracles and protocols from the VID family differ in terms of how the rollup's on-chain contract can verify completion of the dispersal. While Semi-AVID-PR issues certificates of retrievability (④) that can be independently verified (*e.g.*, by a smart contract), and AVID, AVID-FP and AVID-M can readily be extended with such a functionality (by collecting signatures from storage nodes that have completed a dispersal), Data Availability Oracles report data availability directly on-chain to the smart contract (⑤).

Since AVID, AVID-FP and AVID-M all perform a round of AVID to achieve comprehensive termination properties, the AVID family is limited to an adversarial resilience $t < n/3$, compared to $t < n/2$ for Semi-AVID-PR and ACeD.

Sampling Based Data Availability Checks The data availability problem is not unique to rollups, but arises in other scaling approaches such as sharding or light clients as well. Solutions like [1,26] provide interactive protocols based on random sampling of chunks of erasure-coded data. A block is deemed available if all randomly sampled chunks are available, with the assumption being that if enough nodes' random queries are answered, then enough chunks are available to restore the block. However, this interactive technique is not feasible for rollups

since the on-chain contract cannot engage in random sampling to convince itself of data availability. Instead, the assurance of data availability could either be made on-chain (as by Data Availability Oracles) or in the form of a verifiable certificate of retrievability.

1.5 Outline

Cryptographic essentials and erasure-correcting codes are reviewed in Section 2. Model and formal properties of Semi-AVID-PR for the application in rollups are introduced in Section 3. We describe our Semi-AVID-PR protocol in Section 4 and argue its security in Section 5. We close with an evaluation of storage- and communication-efficiency in comparison to other schemes in Section 6.

Application of the Semi-AVID-PR construction to data availability sampling is discussed in Appendix A. Use of blinding to protect privacy of dispersed data against an honest-but-curious storage node is discussed in Appendix B.

2 Preliminaries

In this section, we briefly recapitulate tools from cryptography and erasure-correcting codes used throughout the paper.

2.1 Basics & Notation

Let $\mathbb{G}$ be a cyclic group (denoted multiplicatively, *i.e.*, with group operation '$\cdot$') of prime order $q \geq 2^{2\lambda}$ with generator $g \in \mathbb{G}$, where λ denotes the security parameter used subsequently for all primitives. The function $H(x) \triangleq g^x$ is a bijection between the finite field $\mathbb{Z}_q$ (*i.e.*, integers modulo q) and $\mathbb{G}$. It has the *linear homomorphism* property

$$\forall n \geq 1 \colon \forall c_1, ..., c_n \in \mathbb{Z}_q \colon \forall x_1, ..., x_n \in \mathbb{Z}_q \colon \quad H\left(\sum_{i=1}^{n} c_i x_i\right) = \prod_{i=1}^{n} H(x_i)^{c_i}, \quad (1)$$

which this paper makes ample use of.

An efficiently computable hash function CRHF is *collision resistant* (CRHF) if for any probabilistic poly-time (PPT) adversary $\mathcal{A}$,

$$\Pr(\, (m_0 \neq m_1) \wedge (\mathsf{CRHF}(m_0) = \mathsf{CRHF}(m_1)) \mid (m_0, m_1) \leftarrow \mathcal{A}\,) = \mathrm{negl}(\lambda), \quad (2)$$

where $\mathrm{negl}(\lambda)$ is a negligible function that decays faster than every polynomial.

A commitment scheme $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ is *binding* if for any PPT adversary $\mathcal{A}$,

$$\Pr\left(\begin{array}{l} \mathsf{Verify}(\mathsf{pp}, C, v_0) = \top \wedge \\ \mathsf{Verify}(\mathsf{pp}, C, v_1) = \top \wedge \\ (v_0 \neq v_1) \end{array} \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (C, v_0, v_1) \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right) = \mathrm{negl}(\lambda). \quad (3)$$

We call a signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *secure* if it is existentially unforgeable under a chosen message attack [4].

We denote by $[\boldsymbol{x}]_i$ the i-th entry of a vector $\boldsymbol{x}$.

2.2 Reed-Solomon Codes

A *linear* (n, k)*-code* is a linear mapping $\mathbb{Z}_q^k \to \mathbb{Z}_q^n$ with $n \geq k$. It can be represented by a $k \times n$ *generator matrix* $\boldsymbol{G}$, with the encoding operation then $\boldsymbol{c}^\top = \boldsymbol{G}.\mathsf{Encode}(\boldsymbol{u}^\top) \triangleq \boldsymbol{u}^\top \boldsymbol{G}$ to obtain a length-n row vector of codeword symbols $\boldsymbol{c}^\top$ from a length-k row vector of information symbols $\boldsymbol{u}^\top$.

A linear code is *maximum distance separable* (MDS) if any k columns of its generator matrix $\boldsymbol{G}$ are linearly independent, *i.e.*, any $k \times k$ submatrix of $\boldsymbol{G}$ is invertible. Thus, any set of codeword symbols c_{i_j} from k distinct indices i_j can be used to uniquely decode using the relation

$$\boldsymbol{u}^\top \underbrace{\left[\boldsymbol{g}_{i_1} \ldots \boldsymbol{g}_{i_k}\right]}_{\triangleq \boldsymbol{G}_{\text{reduced}}} \overset{!}{=} \underbrace{\left[c_{i_1} \ldots c_{i_k}\right]}_{\triangleq \boldsymbol{c}_{\text{reduced}}^\top} \iff \boldsymbol{u}^\top = \boldsymbol{G}.\mathsf{Decode}(((i_j, c_{i_j}))_{j=1}^k) \triangleq \boldsymbol{c}_{\text{reduced}}^\top \boldsymbol{G}_{\text{reduced}}^{-1}, \quad (4)$$

where $\boldsymbol{g}_i$ corresponds to the i-th column of the generator matrix $\boldsymbol{G}$.

Reed-Solomon codes [20] are an important class of MDS codes. Here, an information vector $\boldsymbol{u}^\top$ is associated with a polynomial $U(X) = \sum_{j=1}^k [\boldsymbol{u}^\top]_i X^{i-1}$ and the codeword vector is obtained by evaluating $U(X)$ at n distinct locations $\alpha_1, ..., \alpha_n$, $\boldsymbol{c}^\top = (U(\alpha_1), ..., U(\alpha_n))^\top$. This corresponds to a generator matrix $\boldsymbol{G}_{\text{RS}}$ with columns $\boldsymbol{g}_{\text{RS},i} = (\alpha_i^0, ..., \alpha_i^{k-1})$.

2.3 Linear Vector Commitment Schemes

A *vector commitment* (VC) scheme $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{OpenVector}, \mathsf{VerifyVector}, \mathsf{OpenEntry}, \mathsf{VerifyEntry})$ [6,16] for vectors of length L allows to commit to an element of $\mathbb{Z}_q^L$. Later, the commitment can be opened either for the full vector, or to individual entries of the vector. The VC is binding if a commitment cannot be opened to values that are inconsistent with the committed vector. Ideally, the proof for the opening of an entry of the vector is short and computationally easy to generate and verify.

For this manuscript of particular interest are linearly homomorphic (also simply called *linear*) VCs (LVC) with

$$\forall \alpha, \beta \in \mathbb{Z}_q \colon \forall \boldsymbol{v}, \boldsymbol{w} \in \mathbb{Z}_q^L \colon \mathsf{Commit}(\alpha \boldsymbol{v} + \beta \boldsymbol{w}) = \alpha \mathsf{Commit}(\boldsymbol{v}) + \beta \mathsf{Commit}(\boldsymbol{w}). \quad (5)$$

Kate-Zaverucha-Goldberg (KZG) polynomial commitments [13] (here the 'basic' variant $\mathsf{PolyCommit}_{\mathbf{DL}}$ of [13] as KZG) can be readily turned into an example linear VC, which we use subsequently and introduce here briefly. From a vector $\boldsymbol{u}$ of length L interpolate a polynomial $U(X)$ of degree $(L-1)$ such that $U(i) = [\boldsymbol{u}]_i$ for $i = 1, ..., L$. Commit to $\boldsymbol{u}$ by $\mathsf{KZG.Commit}(U)$.[1] The vector opening can be verified by recomputing the commitment. The entry $[\boldsymbol{u}]_i$ can be opened and the opening verified using $\mathsf{KZG.CreateWitness}$ and $\mathsf{KZG.VerifyEval}$ for the corresponding $U(X)$ at $X = i$, respectively.

To see that the resulting VC Commit is linear, consider this. During trusted setup, $\mathsf{KZG.Setup}$ samples $r \xleftarrow{\text{R}} \mathbb{Z}_q$ and computes public parameters $(g^{r^0}, ..., g^{r^{L-1}})$.

[1] The polynomial interpolation can be avoided by preprocessing the public parameters of KZG to obtain them in the Lagrange polynomial basis.

KZG.Commit computes the commitment to a polynomial $U(X)$ of degree $(L-1)$ with coefficients $\gamma_0, ..., \gamma_{L-1}$ as $g^{U(r)}$ which, due to the linear homomorphism of $H(x) = g^x$ discussed above, can be obtained from the public parameters as

$$\mathsf{KZG.Commit}(\gamma_0, ..., \gamma_{L-1}) = \mathsf{KZG.Commit}(U) = \prod_{j=0}^{L-1} (g^{r^j})^{\gamma_j}. \tag{6}$$

Since interpolation of coefficients $\boldsymbol{\gamma} = (\gamma_0, ..., \gamma_{L-1})$ of $U(X)$ from a vector $\boldsymbol{u}$ such that $U(i) = [\boldsymbol{u}]_i$ for $i = 1, ..., L$ is linear and invertible, Commit is linear.

3 Model

The system under discussion consists of n *storage nodes* $S_1, ..., S_n$ and some *clients*. A PPT adversary can corrupt protocol participants adaptively, *i.e.*, as the protocol execution progresses. Corrupt participants surrender their internal state to the adversary immediately and from thereon behave as coordinated by the adversary. We denote by f the number of storage nodes corrupted over the course of the execution, and by t the design resilience, *i.e.*, our construction is parametric in t and satisfies the desired security properties in all executions with $f \leq t$. Protocol participants can send each other messages (a priori without sender identification) which undergo delay controlled by the adversary, subject to the constraint that every message has to arrive eventually. We design a scheme with the following interface and security properties.

Definition 1 (Syntax of Semi-AVID-PR). *A* Semi-AVID (Asynchronous Verifiable Information Dispersal) Scheme with Provable Retrievability *(Semi-AVID-PR) consists of two algorithms,* Commit *and* Verify*, and three protocols,* Setup*,* Disperse *and* Retrieve*.*

- *The protocol* Setup *is run by a temporary trusted party and all storage nodes, at the beginning of time (*i.e.*, before adversarial corruption). It takes as input the security parameter 1^λ and outputs global public parameters* pp*, and local secret parameters $\mathsf{sp}_1, ..., \mathsf{sp}_n$, one for each storage node.*
  The public parameters pp *are common knowledge and input to all other algorithms and protocols. The secret parameters $\mathsf{sp}_1, ..., \mathsf{sp}_n$ are part of the state of a storage node and as such available to that node during* Disperse *and* Retrieve *invocations. Explicit mention is subsequently omitted for brevity.*
- *The algorithm* Commit *takes as input a block B of data, and returns a commitment to the data.*
- *The protocol* Disperse *is run by a client and all storage nodes. It takes as input a block B of data at the client, and outputs $\perp$ or a* certificate of retrievability *for* Commit(B) *to the client.*
- *The algorithm* Verify *takes as input a certificate of retrievability P and a commitment C, and returns $\top$ or $\perp$.*
- *The protocol* Retrieve *is run by a client and all storage nodes. It takes as input a certificate of retrievability P and a commitment C at the client, and outputs $\perp$ or a block B of data to the client.*

Definition 2 (Security of Semi-AVID-PR). *A Semi-AVID-PR scheme is* secure with resilience t *if for all executions with $f \leq t$ and polynomially (in the security parameter λ) many invocations of the scheme it satisfies (except with probability negligible in λ):*

1. ***Binding.*** Commit *implements a binding deterministic commitment to a block B of data.*
2. ***Correctness.*** *If an honest client invokes* $\mathsf{Disperse}$ *with a block B of data, then eventually it outputs a certificate of retrievability P with the property that* $\mathsf{Verify}(P, \mathsf{Commit}(B)) = \top$.
3. ***Soundness.*** *For a certificate of retrievability P and a commitment C, if* $\mathsf{Verify}(P, C) = \top$*, then if an honest client invokes* $\mathsf{Retrieve}$ *with P and C, then eventually it outputs a block B of data such that* $\mathsf{Commit}(B) = C$.

A few remarks are due on this formulation. Unlike earlier formulations of AVID [5,11,24], our formulation does not have independent session identifiers. Instead, the scheme provides a binding commitment scheme which is used to establish a link between the data in question, invocations of the protocols, and certificates of retrievability. The completion of dispersal of a block and the possibility to retrieve content matching a commitment are tied together through the Binding property of the commitment scheme and can be proven to a third party using the certificate of retrievability. This can be seen as following the paradigm shift from location-addressed to content-addressed storage and is particularly suitable for applications such as rollups or sharding where one wants to succinctly but unequivocally identify *what* content is being referenced rather than *where to find it*. In terms of the original four properties of AVID schemes [5], our Correctness property takes the place of the Termination and Agreement properties, and our Soundness property takes the place of the Availability and Correctness properties. Above weakenings (hence the name 'Semi'-AVID) allow us to achieve greater resilience up to $t < n/2$ rather than $t < n/3$ as for AVID, AVID-FP or AVID-M.

4 Protocol

We provide a construction of Semi-AVID-PR from a binding deterministic linear vector commitment scheme LVC, a maximum distance separable (n, k)-code Code, a collision resistant hash function CRHF, and a secure digital signature scheme Sig. Our construction satisfies the properties laid out in Section 3 as shown in Section 5. Moreover, it is storage- and communication-efficient and incurs practically moderate cost for cryptographic computations as demonstrated in Section 6. It is easy to extend our scheme with blinding such that an honest-but-curious storage node cannot learn anything about the dispersed data from its chunk (see Appendix B).

Pseudocode of our construction is provided in Figure 5. See also Figure 6 for an illustration of the $\mathsf{Disperse}$ protocol (*cf.* Figure 3). Our approach is related to AVID-FP [11] in that we also use the linear homomorphism between the LVC

Algorithm 1 $\mathsf{Setup}(1^\lambda)$

At the trusted party: $\mathsf{pp}_{\mathsf{LVC}} \leftarrow \mathsf{LVC.Setup}(1^\lambda)$
At each storage node i**:** $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$
return $\mathsf{pp} = (\mathsf{pp}_{\mathsf{LVC}}, \mathsf{pk}_1, ..., \mathsf{pk}_k), \mathsf{sp}_1 = \mathsf{sk}_1, ..., \mathsf{sp}_k = \mathsf{sk}_k$

Algorithm 2 $\mathsf{Commit}(B)$

$\boldsymbol{U} \leftarrow \mathrm{AsMatrix}_{L\times k}(B)$
$(h_1, ..., h_k) \leftarrow \mathsf{LVC.Commit}^{\otimes k}(\boldsymbol{U})$
return $\mathsf{CRHF}(h_1 \| ... \| h_k)$

Algorithm 3 $\mathsf{Verify}(P, C)$

$\hat{q} \leftarrow \left| \left\{ i \,\middle|\, \begin{array}{l} (i \mapsto \sigma) \in P \colon m \leftarrow (\mathtt{stored}, C) \\ \wedge\, \mathsf{Sig.Verify}(\mathsf{pk}_i, m, \sigma) = \top \end{array} \right\} \right|$
if $\hat{q} \geq q$ **return** $\top$
return $\bot$

Algorithm 4 $\mathsf{Disperse}(B)$

At the client:
$\boldsymbol{U} \leftarrow \mathrm{AsMatrix}_{L\times k}(B)$
$(h_1, ..., h_k) \leftarrow \mathsf{LVC.Commit}^{\otimes k}(\boldsymbol{U})$
$\boldsymbol{C} \leftarrow \mathsf{Code.Encode}^{\otimes L}(\boldsymbol{U})$
Send $((h_1, ..., h_k), \boldsymbol{c}_i)$ to all storage nodes i
At storage node i**:**
$\hat{h} \leftarrow [\mathsf{Code.Encode}(h_1, ..., h_k)]_i$
if $\hat{h} \neq \mathsf{LVC.Commit}(\boldsymbol{c}_i)$ **abort**
$C \leftarrow \mathsf{CRHF}(h_1 \| ... \| h_k)$
Store $C \mapsto ((h_1, ..., h_k), \boldsymbol{c}_i)$
Send $\sigma_i \triangleq \mathsf{Sig.Sign}(\mathsf{sk}_i, (\mathtt{stored}, C))$ to client
At the client:
Wait for σ_{i_j} from q unique $\{i_j\}_{j=1}^q$ with $\mathsf{Sig.Verify}(\mathsf{pk}_{i_j}, (\mathtt{stored}, C), \sigma_{i_j}) = \top$
return $\bigcup_{j=1}^q \{i_j \mapsto \sigma_{i_j}\}$

Algorithm 5 $\mathsf{Retrieve}(P, C)$

At the client:
Extract from P any q unique $\left\{ i \,\middle|\, \begin{array}{l} (i \mapsto \sigma) \in P \colon m \leftarrow (\mathtt{stored}, C) \\ \wedge\, \mathsf{Sig.Verify}(\mathsf{pk}_i, m, \sigma) = \top \end{array} \right\}$
Send C to all storage nodes i
At storage node i**:**
Load $C \mapsto ((h_1, ..., h_k), \boldsymbol{c}_i)$
Send $(i, (h_1, ..., h_k), \boldsymbol{c}_i)$ to client
At the client:
Wait for $(h_1, ..., h_k)$ such that $C = \mathsf{CRHF}(h_1 \| ... \| h_k)$
$\hat{\boldsymbol{h}} \leftarrow \mathsf{Code.Encode}(h_1, ..., h_k)$
Discarding any i with $[\hat{\boldsymbol{h}}]_i \neq \mathsf{LVC.Commit}(\boldsymbol{c}_i)$, wait for k remaining unique $\{i_j\}_{j=1}^k$
return $\mathsf{Code.Decode}(((i_j, \boldsymbol{c}_{i_j}))_{j=1}^k)$

Fig. 5. Pseudocode of our Semi-AVID-PR construction

and the erasure-correcting code. More specifically, during $\mathsf{Disperse}$, the input file B is arranged as a matrix $\boldsymbol{U}$ and a commitment h_i is taken per column $\boldsymbol{u}_i$. Vectorization of the k column commitments of $\boldsymbol{U}$ is denoted as $\mathsf{LVC.Commit}^{\otimes k}(\boldsymbol{U})$. The matrix is encoded row-wise into a coded matrix $\boldsymbol{C}$, of which each column $\boldsymbol{c}_i$ constitutes the chunk for storage server i. Vectorization of the L row encodings of $\boldsymbol{U}$ is denoted as $\mathsf{Code.Encode}^{\otimes L}(\boldsymbol{U})$. Now, due to the linear homomorphism, the commitment of the encodings $\boldsymbol{c}_i$ is equal to the encoding of the commitments h_i. This allows storage nodes to easily verify the consistency of their chunk with the uncoded data (*i.e.*, the verifiability property in AVID). For this check, a storage node only needs to know the commitments h_i of the uncoded data, which keeps the communication-overhead of the scheme low. Our approach differs from AVID-FP in that AVID-FP still performs a round of AVID (for the

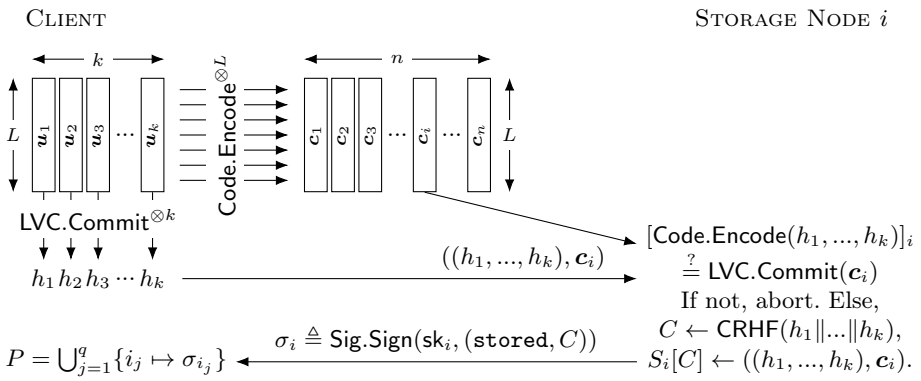

Fig. 6. Disperse protocol of Semi-AVID-PR (*cf.* Figure 3). Client arranges data in $L \times k$ matrix $\boldsymbol{U}$, computes commitments $h_1, ..., h_k$ column-wise and $L \times n$ coded matrix $\boldsymbol{C}$ row-wise. Commitments and i-th column $\boldsymbol{c}_i$ of $\boldsymbol{C}$ are sent to storage node i. Upon verification, storage node computes commitment C to the data, stores commitments and chunk, and acknowledges receipt of chunk to client. Client forms certificate of retrievability P from q unique server identifiers i_j and their receipts σ_{i_j}.

commitments) in order to satisfy the full AVID requirements (in particular Termination and Agreement), while our Semi-AVID-PR scheme satisfies only the weaker Correctness property (*cf.* Figure 4).

Our construction is parametric in the design resilience t, the quorum size q for certificates of retrievability, the code dimension k and the length of chunks L. The analysis of Section 5 reveals that $q \leq (n-t)$, $0 < (q-t)$ and $k \leq (q-t)$ are necessary. So given any $t < n/2$ and target file size $|B|$ (in field elements), choose $q \triangleq (n-t)$, minimize storage overhead with $k \triangleq n - 2t$, and set $L \triangleq |B|/k$.

During Setup (Algorithm 1), a trusted party performs the setup of the LVC and each storage node generates a cryptographic identity for Sig. The public parameters of the LVC and the public keys of the storage nodes become common knowledge, each storage node stores its secret key.

The Commitment of a block B (Algorithm 2) is computed by arranging B as an $L \times k$ matrix $\boldsymbol{U}$, then computing the commitments h_i as $\mathsf{LVC.Commit}(\boldsymbol{u}_i)$ for each of the k columns $\boldsymbol{u}_i$ of $\boldsymbol{U}$, and finally $\mathsf{CRHF}(h_1 \| ... \| h_k)$ is the commitment.

To Disperse a block B (Algorithm 4, Figures 3, 6), the client first computes $\boldsymbol{U}$ and the commitments h_i as for Commit. Then, $\boldsymbol{U}$ is encoded row-wise using Code.Encode to obtain an $L \times n$ coded matrix $\boldsymbol{C}$. Each column $\boldsymbol{c}_i$ of $\boldsymbol{C}$ is sent to storage node i together with $(h_1, ..., h_k)$. Each storage node i verifies its chunk using the linearly homomorphic property (aborting if violated)

$$[\mathsf{Code.Encode}(h_1, ..., h_k)]_i \stackrel{?}{=} \mathsf{LVC.Commit}(\boldsymbol{c}_i), \tag{7}$$

before computing the file's commitment $C \triangleq \mathsf{CRHF}(h_1 \| ... \| h_k)$ and storing commitments and chunk indexed by C. The storage node then acknowledges receipt of the chunk by sending a signature on $(\texttt{stored}, C)$ to the client. Upon collecting

valid signatures σ_{i_j} from q unique storage nodes i_j, the client collects them into a certificate of retrievability P.

To Verify a certificate of retrievability P for a commitment C (Algorithm 3), one counts whether P contains valid signatures on $(\texttt{stored}, C)$ from at least q unique storage nodes.

Finally, to Retrieve a file based on a certificate of retrievability P for a commitment C, the client first extracts any q unique storage nodes for which P contains a valid signature on $(\texttt{stored}, C)$. The client then requests the chunks of C from these storage nodes. The storage nodes reply with the commitments and chunks they have stored for C. The client first waits until some commitments $(h_1, ..., h_k)$ satisfy $C = \mathsf{CRHF}(h_1 \| ... \| h_k)$. Then, the client discards any chunks that do not satisfy the homomorphic property (7). Upon receiving valid chunks from k unique storage nodes, the client uses Code.Decode to decode the file.

To protect the data against honest-but-curious storage nodes (*i.e.*, assuming storage nodes do not collude – clearly, a sufficiently large set of storage nodes can retrieve the data, which is a design goal of Semi-AVID-PR), a hiding LVC scheme can be used and $\boldsymbol{U}$ can be augmented by the client with a column drawn uniformly a random, to blind the encoded chunks.

5 Security Argument

Theorem 1. *The Semi-AVID-PR construction of Section 4 is* secure with resilience t *for any* $t < n/2$ *as defined in Section 3.*

Proof. **Binding.** Commit is deterministic because LVC.Commit and CRHF are. For binding, assume for contradiction that Commit was not binding, *i.e.*, there was an adversary $\mathcal{A}$ that can produce blocks $B \neq B'$ and a commitment C such that $\mathsf{Commit}(B) = \mathsf{Commit}(B') = C$ with non-negligible probability. Then, $\mathcal{A}$ can be turned into either an adversary $\mathcal{B}_{\mathsf{CRHF}}$ against the CRHF property of CRHF or into an adversary $\mathcal{B}_{\mathsf{LVC.Commit}}$ against the Binding property of LVC.Commit, as follows. Since $B \neq B'$, for their respective representations as $L \times k$ matrices, $\boldsymbol{U} \neq \boldsymbol{U}'$. Either $\boldsymbol{h} \triangleq \mathsf{LVC.Commit}^{\otimes k}(\boldsymbol{U}) \neq \mathsf{LVC.Commit}^{\otimes k}(\boldsymbol{U}') \triangleq \boldsymbol{h}'$ but $\mathsf{CRHF}(\boldsymbol{h}) = \mathsf{CRHF}(\boldsymbol{h}')$, a break of CRHF (which $\mathcal{B}_{\mathsf{CRHF}}$ would output); or $\boldsymbol{h} = \boldsymbol{h}'$, so that for some i, $\mathsf{LVC.Commit}(\boldsymbol{u}_i) = \mathsf{LVC.Commit}(\boldsymbol{u}'_i)$ but $\boldsymbol{u}_i \neq \boldsymbol{u}'_i$, a break of LVC.Commit (which $\mathcal{B}_{\mathsf{LVC}}$ would output). If $\mathcal{B}_{\mathsf{CRHF}}$ and $\mathcal{B}_{\mathsf{LVC}}$ broke their primitives only with negligible probability, then $\mathcal{A}$ could not break Commit with non-negligible probability. So either $\mathcal{B}_{\mathsf{CRHF}}$ or $\mathcal{B}_{\mathsf{LVC}}$ break the respective primitive with non-negligible probability, a contradiction. So Commit is binding.

Correctness. Since the client is honest and LVC is linearly homomorphic, the consistency check in Algorithm 4 l. 8 passes at all honest storage nodes. So the client receives signatures σ_{i_j} (which are valid, by Correctness of Sig) of $(\texttt{stored}, \mathsf{Commit}(B))$ from at least $(n-t)$ unique storage nodes i_j, which it can bundle into a certificate of retrievability P that satisfies the check of Algorithm 3 by construction, *if* $q \leq (n-t)$.

Soundness. Since $\mathsf{Verify}(P, C) = \top$ by assumption, the client can extract some q unique storage nodes i_j from P in Algorithm 5 l. 2. Of these q storage nodes, at least $(q - t)$ remain honest. Security of Sig implies that they must have previously executed Algorithm 4 l. 11 and hence stored $(h_1, ..., h_k)$ for $C = \mathsf{CRHF}(h_1, ..., h_k)$ in Algorithm 4 l. 10 and their chunks satisfied the consistency check in Algorithm 4 l. 8. Note that by the CRHF property of CRHF, there can be only one set of $(h_1, ..., h_k)$ for C. *As long as* $(q - t) > 0$, the client eventually completes the wait in Algorithm 5 l. 8, and *if* $k \leq (q - t)$, then the client eventually also completes the wait in Algorithm 5 l. 10. Finally, since Code is an MDS (n, k)-code, Algorithm 5 l. 11 succeeds to decode a block B, corresponding to an $L \times k$ matrix $\boldsymbol{U}$. It remains to show that $\mathsf{Commit}(B) = C$, for which (by CRHF) it suffices that $(h_1, ..., h_k) = \mathsf{LVC.Commit}^{\otimes k}(\boldsymbol{U})$. Note the decoder uses that Code is an MDS (n, k)-code and thus any $k \times k$ submatrix of its generator matrix $\boldsymbol{G}$ is invertible, and the relation

$$\underbrace{\left[\boldsymbol{u}_1 \ ... \ \boldsymbol{u}_k\right]}_{=\boldsymbol{U}} \underbrace{\left[\boldsymbol{g}_{i_1} \ ... \ \boldsymbol{g}_{i_k}\right]}_{\triangleq \boldsymbol{G}_{\text{reduced}}} \overset{!}{=} \underbrace{\left[\boldsymbol{c}_{i_1} \ ... \ \boldsymbol{c}_{i_k}\right]}_{\triangleq \boldsymbol{C}_{\text{reduced}}} \iff \boldsymbol{U} = \boldsymbol{C}_{\text{reduced}} \boldsymbol{G}_{\text{reduced}}^{-1}. \tag{8}$$

At the same time, by the checks in Algorithm 5 l. 10,

$$\left[h_1 \ ... \ h_k\right] \boldsymbol{G}_{\text{reduced}} = \left[\mathsf{LVC.Commit}(\boldsymbol{c}_{i_1}) \ ... \ \mathsf{LVC.Commit}(\boldsymbol{c}_{i_k})\right]. \tag{9}$$

By the linear homomorphism of LVC,

$$\mathsf{LVC.Commit}^{\otimes k}(\boldsymbol{U}) = \mathsf{LVC.Commit}^{\otimes k}(\boldsymbol{C}_{\text{reduced}}) \boldsymbol{G}_{\text{reduced}}^{-1} = \left[h_1 \ ... \ h_k\right]. \tag{10}$$

Resilience. From above analysis, we obtain the constraints $q \leq (n - t)$ (for Correctness), $0 < (q - t)$ (for Soundness, to obtain $h_1, ..., h_k$), and $k \leq (q - t)$ (for Soundness, to decode), which the choice of parameters in Section 4 satisfies, and which lead to the resilience bound $t < n/2$. □

6 Evaluation

In this section, we show that the cost of cryptographic computations required for our Semi-AVID-PR scheme is low, and the communication and storage requirements in comparison with AVID [5], AVID-FP [11], AVID-M [24] and ACeD [21] are among the best-of-class (tied with AVID-M) and practically low, while providing superior resilience ($t < n/2$ vs. $t < n/3$) and provable retrievability.

6.1 Cryptographic Computations

To examine the load caused by cryptographic computations (computing vector commitments) for the client in our Semi-AVID-PR protocol (*i.e.*, the Validium rollup operator), we used the implementation of KZG commitments by [22]. The single-threaded runtime on a consumer-grade laptop computer is plotted for different file and network sizes in Figure 7. The throughput of $\approx 0.95\,\text{MB/s}$ is independent of file and network size and corresponds to $\approx 4{,}800\,\text{tx/s}$ (assuming 200 B as typical transaction size).

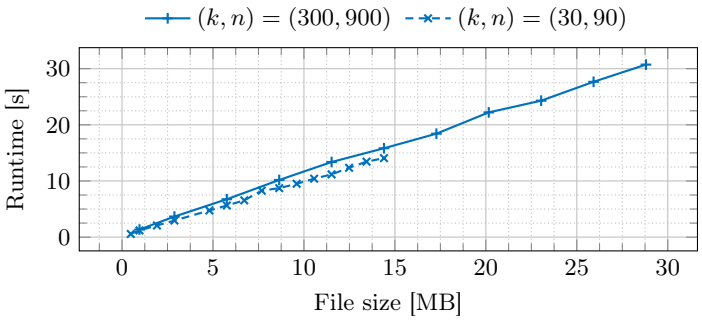

Fig. 7. Single-threaded runtime of cryptographic computations for our protocol (computing vector commitments) for two system sizes on a consumer-grade laptop computer.

Table 1. Communication and storage required to disperse 30 MB among $n = 900$ nodes using different solutions; resilience and whether provable retrievability is supported.

Scheme	Resilience	Communication	Storage	Retrievability
Uncoded (repetition)	$0.49n$	27 GB	27 GB	✔
AVID [5]	$0.33n$	104 GB	116 MB	✔
AVID-FP [11]	$0.33n$	31 GB	125 MB	✔
AVID-M [24]	$0.33n$	116 MB	90 MB	✘
ACeD [21]	$0.33n$	787 MB	787 MB	✘
ACeD [21]	$0.49n$	13 GB	13 GB	✘
Semi-AVID-PR (Sec. 4)	$0.33n$	99 MB	99 MB	✔
Semi-AVID-PR (Sec. 4)	$0.49n$	1.5 GB	1.5 GB	✔

6.2 Communication & Storage

Communication and storage requirements of different data availability solutions are tabulated for a numerical example in Table 1. AVID improves over repetition in that each node only needs to store a chunk rather than the full file. However, nodes still echo chunks to each other, leading to a lot of communication. AVID-FP improves in communication because storage nodes only echo fingerprints rather than full chunks. AVID-M improves over AVID-FP in that it drastically reduces the fingerprint size and hence the communication. ACeD introduces a trade-off of communication and storage with adversarial resilience. In terms of communication and storage, Semi-AVID-PR is among the best-of-class (tied with AVID-M), while providing superior resilience ($t < n/2$ vs. $t < n/3$) and provable retrievability (the lack thereof limits application of AVID-M to Validium rollups). Semi-AVID-PR outperforms ACeD in communication and storage by at least 9×.

The net data throughput of ≈ 0.95 MB/s corresponding to ≈ 4,800 tx/s (*cf.* Section 6.1) entails ≈ 3.4 MB/s communication bandwidth, which is feasible even via consumer-grade Internet connectivity.

Acknowledgment

JN is supported by the Reed-Hodgson Stanford Graduate Fellowship.

References

1. Al-Bassam, M., Sonnino, A., Buterin, V.: Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities. CoRR **abs/1809.09044** (2018), http://arxiv.org/abs/1809.09044
2. Bagaria, V.K., Kannan, S., Tse, D., Fanti, G.C., Viswanath, P.: Prism: Deconstructing the blockchain to approach physical limits. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019. pp. 585–602. ACM (2019). https://doi.org/10.1145/3319535.3363213
3. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptol. ePrint Arch. p. 46 (2018), http://eprint.iacr.org/2018/046
4. Boneh, D., Shoup, V.: A Graduate Course in Applied Cryptography. https://cryptobook.us, version 0.5 (2021)
5. Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3724, pp. 503–504. Springer (2005). https://doi.org/10.1007/11561927_42
6. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7778, pp. 55–72. Springer (2013). https://doi.org/10.1007/978-3-642-36362-7_5
7. Decker, C., Wattenhofer, R.: A fast and scalable payment network with Bitcoin duplex micropayment channels. In: Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9212, pp. 3–18. Springer (2015)
8. Feist, D., Khovratovich, D.: Fast amortized Kate proofs https://github.com/khovratovich/Kate/blob/master/Kate_amortized.pdf
9. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 626–645. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_37
10. Gluchowski, A.: zkRollup vs. Validium (2020), https://medium.com/matter-labs/zkrollup-vs-validium-starkex-5614e38bc263
11. Hendricks, J., Ganger, G.R., Reiter, M.K.: Verifying distributed erasure-coded data. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007. pp. 139–146. ACM (2007). https://doi.org/10.1145/1281100.1281122

12. Kalodner, H.A., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018. pp. 1353–1370. USENIX Association (2018), https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner
13. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6477, pp. 177–194. Springer (2010). https://doi.org/10.1007/978-3-642-17373-8_11
14. 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 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. pp. 583–598. IEEE Computer Society (2018). https://doi.org/10.1109/SP.2018.000-5
15. Li, S., Yu, M., Yang, C., Avestimehr, A.S., Kannan, S., Viswanath, P.: PolyShard: Coded sharding achieves linearly scaling efficiency and security simultaneously. In: IEEE International Symposium on Information Theory, ISIT 2020, Los Angeles, CA, USA, June 21-26, 2020. pp. 203–208. IEEE (2020). https://doi.org/10.1109/ISIT44484.2020.9174305
16. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings. Lecture Notes in Computer Science, vol. 293, pp. 369–378. Springer (1987). https://doi.org/10.1007/3-540-48184-2_32
17. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., McCorry, P.: Sprites and state channels: Payment networks that go faster than Lightning. In: Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11598, pp. 508–526. Springer (2019). https://doi.org/10.1007/978-3-030-32101-7_30
18. Mitra, D., Tauz, L., Dolecek, L.: Concentrated stopping set design for Coded Merkle Tree: Improving security against data availability attacks in blockchain systems. In: IEEE Information Theory Workshop, ITW 2020, Riva del Garda, Italy, April 11-15, 2021. pp. 1–5. IEEE (2020). https://doi.org/10.1109/ITW46852.2021.9457630, https://doi.org/10.1109/ITW46852.2021.9457630
19. Mitra, D., Tauz, L., Dolecek, L.: Overcoming data availability attacks in blockchain systems: LDPC code design for Coded Merkle Tree. CoRR **abs/2108.13332** (2021), https://arxiv.org/abs/2108.13332
20. Reed, I., Solomon, G.: Polynomial codes over certain finite fields. Journal of The Society for Industrial and Applied Mathematics **8**, 300–304 (1960)
21. Sheng, P., Xue, B., Kannan, S., Viswanath, P.: ACeD: Scalable data availability oracle. CoRR **abs/2011.00102** (2020), https://arxiv.org/abs/2011.00102
22. Tomescu, A.: How to compute all pointproofs. IACR Cryptol. ePrint Arch. p. 1516 (2020), https://eprint.iacr.org/2020/1516
23. Tomescu, A., Abraham, I., Buterin, V., Drake, J., Feist, D., Khovratovich, D.: Aggregatable subvector commitments for stateless cryptocurrencies. In: SCN. Lecture Notes in Computer Science, vol. 12238, pp. 45–64. Springer (2020)

24. Yang, L., Park, S.J., Alizadeh, M., Kannan, S., Tse, D.: DispersedLedger: High-throughput Byzantine consensus on variable bandwidth networks (2022)
25. Yu, H., Nikolic, I., Hou, R., Saxena, P.: OHIE: Blockchain scaling made simple. In: 2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020. pp. 90–105. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00008
26. Yu, M., Sahraei, S., Li, S., Avestimehr, S., Kannan, S., Viswanath, P.: Coded Merkle Tree: Solving data availability attacks in blockchains. In: Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers. Lecture Notes in Computer Science, vol. 12059, pp. 114–134. Springer (2020). https://doi.org/10.1007/978-3-030-51280-4_8

A Application to Data Availability Sampling

In common blockchain designs all nodes have to download the full blockchain and validate all included transactions (*e.g.*, check that accounts have sufficient balances, no funds are created out of thin air, etc.). However, if a node does not have enough bandwidth, storage, or computational resources to do so, it can instead participate as a so called *light node*.[2] We assume that every block consists of a header comprised of meta data and a body comprised of a list of transactions. The header includes a commitment to the block content, binding the two together. Full nodes process block headers and content, while light nodes only process block headers. If an invalid transaction was added to a block, this block would be rejected by full nodes but a header of this block can be accepted by a light node, since the light node cannot inspect the block content and verify transaction validity. To prevent light nodes from accepting (the header of) an invalid block, full nodes can produce an invalid transaction fraud proof (*i.e.*, a succinct string of the evidence necessary to verify relative to the block header that the block indeed contains an invalid transaction). To take full nodes' ability to issue invalid transaction fraud proofs, a malicious block producer can perform a data availability attack and withhold parts of the block content, including the invalid transaction. Full nodes would now temporarily reject the block (until its content becomes fully available), but light nodes would not notice the missing content since they do not attempt to download the block content anyway. In this setting, the absence of an invalid transaction fraud proof can thus mean two things: either that the block is alright, or that full nodes were not able to verify the block due to missing data. To rule out the possibility of data unavailability (so that finally lack of fraud proof implies the block is valid), various data availability sampling schemes for light nodes were introduced.

Data availability schemes using Reed-Solomon codes were proposed in [1]. In a naive scheme, a block producer encodes a list of transactions, consisting of k chunks, with a $(2k, k)$ Reed-Solomon code. Once a light node receives a header

[2] Light nodes also occur in the context of sharding, where each node is assigned to a shard and behaves in-shard (*i.e.*, towards their assigned shard) as a full node and out-of-shard (*i.e.*, towards other shards) as a light node.

of the block, it randomly queries a few chunks of the encoded block content. The block is accepted only if the queried chunks are available. For a block to be widely accepted by light nodes, most of the light nodes' queried chunks have to be available. Quickly, light nodes' queries cover more than 50% of coded chunks of the block, so that any remaining missing chunks can be recovered using the Reed-Solomon code. It is therefore no longer possible to trick light nodes into accepting a block while withholding a chunk in an attempt to prevent full nodes from generating an invalid transaction fraud proof. The main drawback of this solution is that a malicious block producer could invalidly encode the block. Decoding would then not consistently recover the original chunks' data, even if nominally enough chunks are available. Again, full nodes would be able to detect invalid encoding, but light nodes would not. And again, full nodes could issue a fraud proof to prevent light nodes from accepting an invalidly encoded block. However, the amount of evidence needed to prove invalid encoding in this scheme is as big as the block content itself – defying the idea of light nodes downloading less than the full block content. For example, an invalid encoding fraud proof consists of the full original block data, which the light node can verify with respect to the block header, re-encode, and then check that some of the 'encoded' chunks received in response to data availability queries do not match the properly encoded chunks. Subsequent works [1,26,18,19] on data availability schemes of this flavor have thus focussed on reducing the size of invalid encoding fraud proofs, but drawbacks remain (*e.g.*, additional complexity, timing assumptions).

A different approach is to eliminate invalid encoding fraud proofs by making it impossible for block producers to invalidly encode data. Such schemes can be achieved using polynomial commitment schemes such as KZG [13]. Treated as evaluations of a polynomial at agreed-upon locations, the k chunks of the block content uniquely determine a polynomial of degree $k-1$. A commitment to this polynomial is included in the block header. To ensure data availability, light nodes query for evaluations of this polynomial at random locations. Consistency of every query response with the polynomial committed to in the block header can be verified using the polynomial commitment scheme by providing an evaluation witness. Even with few queries each, light nodes together will soon have queried evaluations at at least k distinct locations. If the block producer withholds any of these evaluations, light nodes will not accept the block. But once evaluations at k distinct locations are available, the polynomial, and thus the block content, can be reconstructed. Any invalid transaction becomes visible and full nodes can generate corresponding fraud proofs. Schemes of this flavor however require to compute an evaluation witness for each query, which despite recent algorithmic improvements is still computationally heavy [8,22,23].

The Semi-AVID-PR scheme, described in Section 4 and illustrated in Figure 6, can be seen as combining 'the best of both worlds' in that it does not require invalid encoding fraud proofs, but sampled chunks can be verified efficiently. We assume that a block contains an alternating sequence of transactions and commitments to resulting intermediary chain states (this is used for invalid transaction fraud proofs as in [1]). As illustrated in Figure 6, the block

producer arranges the block content $\boldsymbol{U}$ as a matrix of size $L \times k$, where k and L are system parameters. It commits to each of the columns $\boldsymbol{u}_1, ..., \boldsymbol{u}_k$ of that matrix using the linear vector commitment scheme defined in Section 2.3 (to obtain commitments $h_1, ..., h_k$), and encodes the matrix $\boldsymbol{U}$ row-wise using a (n, k) Reed-Solomon code to obtain chunks $\boldsymbol{c}_1, ..., \boldsymbol{c}_n$ of a coded matrix $\boldsymbol{C}$. A final commitment to the full block content is computed as $C \triangleq \mathsf{CRHF}(h_1 \| ... \| h_k)$ and used on-chain in the block's header to uniquely reference the block content. Full nodes receive the full block content, recompute the column commitments and their hash, and compare it with C to verify the block content. Light nodes receive only C from the block header. Prior to accepting a new block header, a light node samples random coded chunks $\boldsymbol{c}_i$. The response to each query is accompanied by purported column commitments $h_1, ..., h_k$. Every light node can verify the column commitments by locally recomputing their hash and comparing it with the commitment C in the block header. Subsequently, the light node verifies the downloaded chunk $\boldsymbol{c}_i$ using the linear homomorphic property of the vector commitment scheme and the column commitments $h_1, ..., h_k$.

To employ the invalid transaction fraud proofs of [1], it remains to show how a full node can open any entry of $\boldsymbol{U}$ to a light node in a verifiable manner. For this purpose, an opening witness for value $y = [\boldsymbol{u}_j]_i$ at position (i, j) consists of:

- Value $y = [\boldsymbol{u}_j]_i$ and coordinates (i, j).
- Commitments $h_1, ..., h_k$ to columns $\boldsymbol{u}_1, ..., \boldsymbol{u}_k$.
- A witness w for the opening of y at the i-th position in $\boldsymbol{u}_j$ with respect to the vector commitment h_j.

The light client first verifies the commitments $h_1, ..., h_k$ by comparing their hash to the commitment C in the block header. The client then verifies the opening of the value y at position i in $\boldsymbol{u}_j$ with respect to the vector commitment h_j.

Since in Semi-AVID-PR valid encoding can be verified by light nodes using the homomorphic property of linear vector commitments, invalid encoding fraud proofs are not needed needed. At the same time, verifying a chunk requires only to compute a vector commitment (to $\boldsymbol{c}_i$) and a linear combination of the vector commitments $h_1, ..., h_k$, which is lightweight to compute. Performance is discussed in more detail in Section 6.

B Privacy

Semi-AVID-PR can be extended to hide the dispersed data from single honest-but-curious storage nodes. For this purpose, with a slight abuse of notation, let $\tilde{\boldsymbol{U}}$ denote the $(L-1) \times (k-1)$ matrix of information to be dispersed (with columns $\tilde{\boldsymbol{u}}_i$). The dispersing client augments it first with a blinding column $\boldsymbol{b} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{L-1}$ to the right of $\tilde{\boldsymbol{U}}$ and then with a blinding row $\boldsymbol{s}^\top \xleftarrow{\mathrm{R}} \mathbb{Z}_q^k$ to the bottom of both $\tilde{\boldsymbol{U}}$ and $\boldsymbol{b}$, to obtain the $L \times k$ matrix $\boldsymbol{U}$,

$$\boldsymbol{U} \triangleq \begin{bmatrix} \tilde{\boldsymbol{U}} & \boldsymbol{b} \\ \boldsymbol{s}^\top & \end{bmatrix}. \tag{11}$$

The coded matrix $\boldsymbol{C}$ (with columns $\boldsymbol{c}_i$) and the column commitments $(h_1, ..., h_k)$ continue to be computed as detailed in Figure 6 and Section 4. Thus, storage node m receives $(\boldsymbol{c}_m, h_1, ..., h_k)$ as part of the protocol (see Figure 5), and we show that the distribution of $(\boldsymbol{c}_m, h_1, ..., h_k)$ induced by the randomness in the blinding $\boldsymbol{b}$ and $\boldsymbol{s}^\top$ is independent of $\tilde{\boldsymbol{U}}$, so that storage node m learns nothing about the dispersed information.

Assume that storage node m could even compute $\log_g(.)$ in $\mathbb{G}$, and hence knows the secret r sampled during trusted setup of the KZG polynomial commitment scheme (see Section 2.3), as well as $\log_g(h_i)$ for the column commitments h_i. Furthermore, assume a Reed-Solomon code (see Section 2.2) is used as part of Semi-AVID-PR so that the column $\boldsymbol{g}_{\mathrm{RS},m} = (\alpha_m^0, ..., \alpha_m^{k-1})$ corresponds to storage node m in the code's generator matrix $\boldsymbol{G}_{\mathrm{RS}}$. Then, the data obtained by storage node m are related to the unknowns by the following equations:

$$\begin{bmatrix} [\boldsymbol{c}_m]_1 \\ \vdots \\ [\boldsymbol{c}_m]_{L-1} \\ \hline [\boldsymbol{c}_m]_L \\ \hline \log_g(h_1) \\ \vdots \\ \log_g(h_{k-1}) \\ \hline \log_g(h_k) \end{bmatrix} = \underbrace{\left[\begin{array}{cccccc|ccc|cccc} \alpha_m^0 & ... & \alpha_m^{k-2} & & & & \alpha_m^{k-1} & 0 & ... & & & & \\ & \ddots & & & & & & \ddots & & & & & \\ & & & \alpha_m^0 & ... & \alpha_m^{k-2} & ... & 0 & \alpha_m^{k-1} & & & & \\ \hline & & & & & & & & & \alpha_m^0 & ... & \alpha_m^{k-2} & \alpha_m^{k-1} \\ \hline r^0 & 0 & ... & & r^{L-2} & 0 & ... & & & r^{L-1} & 0 & ... & \\ & \ddots & & \ddots & & \ddots & & & & & \ddots & & \\ ... & 0 & r^0 & & ... & 0 & r^{L-2} & & & ... & 0 & r^{L-1} & \\ \hline & & & & & & r^0 & ... & r^{L-2} & & & & r^{L-1} \end{array}\right]}_{\triangleq \boldsymbol{M} \in \mathbb{Z}_q^{(L+k)\times(Lk+(L-1)+(k-1)+1)}} \begin{bmatrix} [\tilde{\boldsymbol{u}}_1]_1 \\ \vdots \\ [\tilde{\boldsymbol{u}}_{k-1}]_1 \\ \vdots \\ [\tilde{\boldsymbol{u}}_1]_{L-1} \\ \vdots \\ [\tilde{\boldsymbol{u}}_{k-1}]_{L-1} \\ \hline [\boldsymbol{b}]_1 \\ \vdots \\ [\boldsymbol{b}]_{L-1} \\ \hline [\boldsymbol{s}^\top]_1 \\ \vdots \\ [\boldsymbol{s}^\top]_{k-1} \\ \hline [\boldsymbol{s}^\top]_k \end{bmatrix} \tag{12}$$

Denote by $[\boldsymbol{M}]_i^\top$ the i-th row of $\boldsymbol{M}$. Observe that

$$[\boldsymbol{M}]_{L+k}^\top = \sum_{i=1}^{L} r^{i-1} \alpha_m^{-(k-1)} [\boldsymbol{M}]_i^\top - \sum_{i=1}^{k-1} \alpha_m^{-(k-i)} [\boldsymbol{M}]_{L+i}^\top. \tag{13}$$

Thus, the last row of $\boldsymbol{M}$ is redundant. Striking the last row,

$$\begin{bmatrix} [\boldsymbol{c}_m]_1 \\ \vdots \\ [\boldsymbol{c}_m]_{L-1} \\ \hline [\boldsymbol{c}_m]_L \\ \hline \log_g(h_1) \\ \vdots \\ \log_g(h_{k-1}) \end{bmatrix} = \left[\begin{array}{cccccc} \alpha_m^0 & ... & \alpha_m^{k-2} & & & \\ & \ddots & & & & \\ & & & \alpha_m^0 & ... & \alpha_m^{k-2} \\ \hline & & & & & \\ \hline r^0 & 0 & ... & r^{L-2} & 0 & ... \\ & \ddots & & \ddots & & \ddots \\ ... & 0 & r^0 & ... & 0 & r^{L-2} \end{array}\right] \begin{bmatrix} [\tilde{\boldsymbol{u}}_1]_1 \\ \vdots \\ [\tilde{\boldsymbol{u}}_{k-1}]_1 \\ \vdots \\ [\tilde{\boldsymbol{u}}_1]_{L-1} \\ \vdots \\ [\tilde{\boldsymbol{u}}_{k-1}]_{L-1} \end{bmatrix}$$

$$+\underbrace{\left[\begin{array}{ccc|ccc|c} \alpha_m^{k-1} & 0 & \dots & & & & \\ & \ddots & & & & & \\ \dots & 0 & \alpha_m^{k-1} & & & & \\ \hline & & & \alpha_m^0 & \dots & \alpha_m^{k-2} & \alpha_m^{k-1} \\ \hline & & & r^{L-1} & 0 & \dots & \\ & & & & \ddots & & \\ & & & \dots & 0 & r^{L-1} & \end{array}\right]}_{\triangleq \boldsymbol{M}' \in \mathbb{Z}_q^{(L+k-1)\times(L-1+k)}} \left[\begin{array}{c} [\boldsymbol{b}]_1 \\ \vdots \\ [\boldsymbol{b}]_{L-1} \\ \hline [\boldsymbol{s}^\top]_1 \\ \vdots \\ [\boldsymbol{s}^\top]_{k-1} \\ \hline [\boldsymbol{s}^\top]_k \end{array}\right]. \tag{14}$$

Observe that $\boldsymbol{M}'$ is full-rank. Thus, the randomness of $\boldsymbol{b}$ and $\boldsymbol{s}^\top$ renders the distribution of $(\boldsymbol{c}_m, h_1, ..., h_{k-1})$ uniform while h_k is a function of $(\boldsymbol{c}_m, h_1, ..., h_{k-1})$, both independent of the dispersed information $\tilde{\boldsymbol{U}}$.

C Calculations for Table 1

Table 1 shows communication and storage required to disperse a file of size $|F| = 30\,\text{MB}$ among $n = 900$ storage nodes using different schemes. We provide the corresponding calculations here. We denote communication and storage costs as C and S, respectively. We assume the size of a hash is $H = 32\,\text{B}$. Given adversarial resilience t, we choose $k \triangleq n - 2t$.

- Uncoded (repetition) scheme: $C = S = n|F|$
- AVID: $C = \left(\frac{|F|}{k} + nH\right)(n + n^2)$ and $S = n\left(\frac{|F|}{k} + nH\right)$
- AVID-FP: $C = n\left(\frac{|F|}{k} + (n+k)H\right) + n^2(n+k)H$ and $S = n\left(\frac{|F|}{k} + (n+k)H\right)$
- AVID-M: $C = n\left(\frac{|F|}{k} + (1+\log_2 n)H\right) + n^2 H$ and $S = n\left(\frac{|F|}{k} + (1+\log_2 n)H\right)$
- ACeD: $C = S = t'H + \frac{|F|}{nr\lambda} + \frac{(2q-1)|F|H}{nrc\lambda}\log_{qr}\frac{|F|}{ct'r}$
  Parameters: $t' = 16, r = 0.25, q = 8, c = 48\,\text{kB}, \eta = 0.875$, $\lambda = \frac{1-2t/n}{\ln\left(\frac{1}{1-\eta}\right)}$.
  As illustrated in Figure 8, the communication and storage cost of ACeD can be decreased slightly by increasing the base layer symbol size c, at the expense of an increased invalid encoding fraud proof size.
- Semi-AVID-PR: $C = n\left(\frac{|F|}{k} + kH\right) + nH$ and $S = n\left(\frac{|F|}{k} + kH\right)$

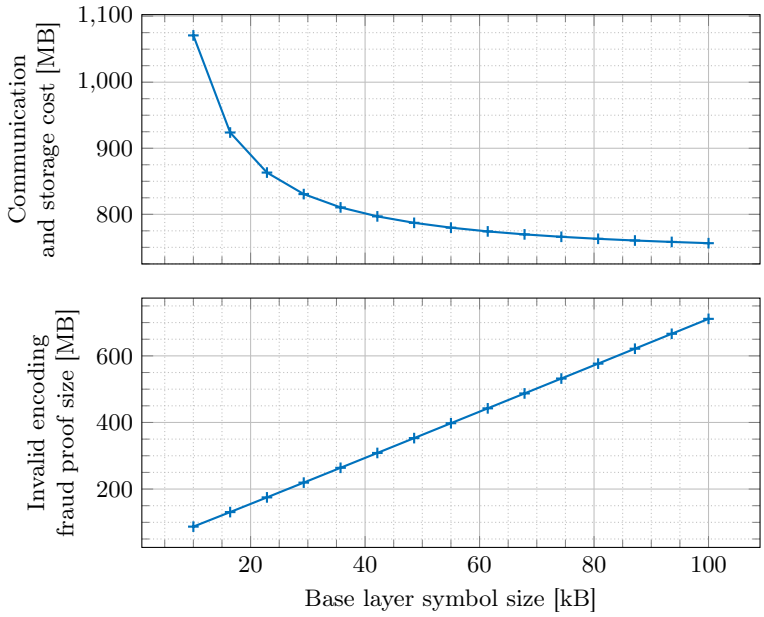

Fig. 8. Communication and storage cost (top) and invalid encoding fraud proof size (bottom) as a function of the base layer symbol size c, when dispersing a file of size 30 MB among 900 nodes using ACeD [21] with resilience $t = 0.33n$.