Information Dispersal with Provable Retrievability for Rollups

Kamilla Nazirkhanova
nazirk@stanford.edu

Joachim Neu
jneu@stanford.edu

David Tse
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 *Semi-AVID-PR* 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 22 MB among 256 storage nodes, up to 85 of which may be adversarial, requires in total ≈ 70 MB of communication and storage, and ≈ 41 s of single-thread runtime (< 3 s on 16 threads) on an AMD Opteron 6378 processor when using the BLS12-381 curve. 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. Finally, we discuss an application of our Semi-AVID-PR scheme to data availability verification schemes based on random sampling.

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 [17, 18] or multi-chain protocols [3, 30] aim at improving the base blockchain protocol, off-chain 'layer 2' solutions such as payment channels [9, 22] and rollups [14, 20] 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' 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.

		Ensuring State Transition Integrity	
		Validity Proofs	*Fraud Proofs*
Data Storage	*Off-Chain*	**Validium:** StarkWare, Matter Labs zkPorter, Loopring	**Plasma:** OMG Network, Polygon, Gluon, Gazelle, LeapDAO
	On-Chain	**Zk-Rollups:** Loopring, Starkware, Matter Labs zkSync, Aztec 2.0, Hermez network, zkTube	**Optimistic Rollups:** Optimism, Offchain Labs Arbitrum, Fuel Network, Cartesi, OMGX

Figure 1: Layer 2 and rollup projects grouped into four categories according to how validity of state transitions and data availability are ensured (fraud/validity proofs vs. data storage on/off chain). Source: https://ethereum.org/en/developers/docs/scaling/, https://twitter.com/vitalikbuterin/status/1267455602764251138

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 [5, 11] 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 condensed 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

KN and JN contributed equally and are listed alphabetically.

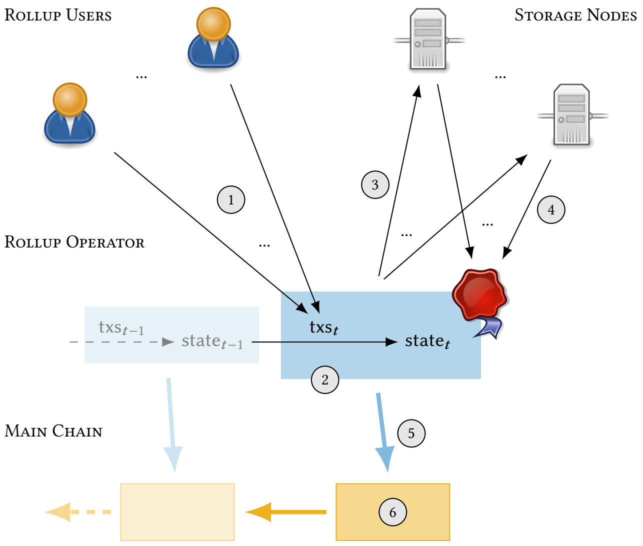


Figure 2: Normal operating mode of a *Validium rollup, i.e.*, a zk-rollup with off-chain storage: ① Rollup operator collects transactions from rollup users. ② Transactions are executed by operator off-chain and new state of rollup is calculated. ③ Transaction and state data is dispersed to storage nodes 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.

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 [12]), 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 [7]) and its successors AVID-FP [13] and AVID-M [29]. 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 [13], informally:

(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 some 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 3). 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 the concept of *Semi-AVID with Provable Retrievability* (Semi-AVID-PR) to capture the requirements in the rollup application. Besides Disperse and Retrieve, a Semi-AVID-PR scheme provides Commit to succinctly and unequivocally identify data blocks and Verify to verify certificates of retrievability.

Definition (Semi-AVID-PR Security (Informal), *cf.* Definition 3.2)**.** *If $f \leq t$ nodes are corrupted, then the Semi-AVID-PR scheme provides:*

(a) ***Commitment-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) ***Availability.*** *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 provide formal game-based definitions of commitment-binding (*cf.* Definition 1) and availability (*cf.* Definition 2).

We propose a construction for a communication- and storage-efficient Semi-AVID-PR scheme with practical computational cost, which is compatible with the established on-chain smart contracts of and thus can be readily adopted for existing Validium rollups, *e.g.*, such as StarkWare's StarkEx. Our construction relies on a collision-resistant hash function, unforgeable signatures, and a deterministic homomorphic vector commitment. We provide a reduction-based security proof. A high-level illustration of Disperse of our scheme is provided in Figure 4. 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

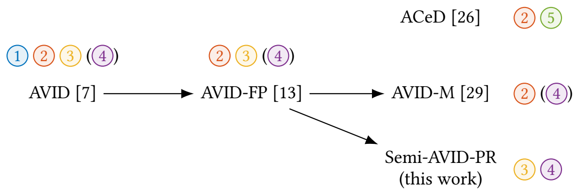


Figure 3: Related protocols and supported properties: ① Retrieval from *any* sufficiently large set of storage nodes ② Comprehensive termination guarantees ③ Retrievability guaranteed ④ Issues certificates of retrievability ⑤ Dispersal verifiable on-chain

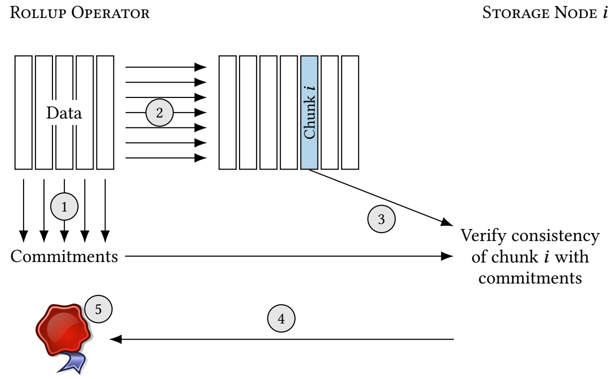


Figure 4: Dispersal in our Semi-AVID-PR scheme. ① Client arranges data in matrix and computes vector commitments of columns. ② Client encodes data row-wise. ③ Commitments and chunk i are sent to storage node i. ④ If chunk i is consistent with commitments, storage node confirms receipt. ⑤ Enough acknowledgements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

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, additional blinding can be used in our scheme to provide privacy against honest-but-curious storage nodes. Finally, the core construction of our scheme can be used to derive a data availability verification scheme based on random sampling with practical computational requirements.

1.4 Related Work

The AVID protocol [7] 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 3, ①), rather than from only some subset, which can be identified from the certificate of retrievability, 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 $O(n|B| + n^3|C|)$, 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. Our Semi-AVID-PR scheme's communication and storage complexity is $O(|B| + n^2|C|)$.

AVID-FP [13], a successor of AVID, brings the communication complexity to $O(|B| + n^3|C|)$ using homomorphic fingerprinting (as in our Semi-AVID-PR scheme) 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 [29], improves the communication complexity to $O(|B| + n^2|C|)$ by reducing the size of fingerprints using Merkle trees [21]. 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 [26], and the dispersal sub-protocol of Dumbo-MVBA [19]. If a block was invalidly encoded during dispersal by a malicious client, then Data Availability Oracles and Dumbo-MVBA's dispersal 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. Semi-AVID-PR issues certificates of retrievability (④) that can be independently verified (*e.g.*, by a smart contract), similar to Dumbo-MVBA's dispersal where the dispersing client produces a 'lock proof' consisting of a quorum of signatures from storage nodes attesting to having received their respective chunks of the dispersed data. AVID, AVID-FP and AVID-M can readily be extended with such a functionality. 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, 31] 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. However, techniques from our Semi-AVID-PR scheme can be used to obtain a data availability check where the consistency of a randomly sampled chunk can be efficiently verified, obviating fraud proofs for invalid encoding [1].

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 prove its security in Section 5. Use of blinding to protect privacy of dispersed data against honest-but-curious storage nodes is discussed in Section 6. An evaluation of computational cost and storage- and communication-efficiency in comparison to other schemes is discussed in Section 7. We close with comments on an application of techniques of our Semi-AVID-PR scheme to data availability sampling in Section 8.

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

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

which this paper makes ample use of.

An efficiently computable hash function $\mathsf{HF} = (\mathsf{Gen}, \mathsf{H})$, see Definition B.1, is *collision resistant* if for any probabilistic poly-time (PPT) adversary $\mathcal{A}$ there exists a negligible function negl(.) such that

$$\Pr\big(\, \mathsf{CFG}_{\mathsf{HF},\mathcal{A}}(\lambda) = \mathtt{true} \,\big) = \mathrm{negl}(\lambda), \quad (2)$$

where $\mathsf{CFG}_{\mathsf{HF},\mathcal{A}}(\lambda)$ is the collision finding game recapitulated in Alg. 13. We use $\mathsf{CRHF}^s(x) \triangleq \mathsf{HF.H}^s(x)$ as a notational shorthand.

A signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, see Definition B.2, is *secure under existential forgery* if for any PPT adversary $\mathcal{A}$ there exists a negligible function negl(.) such that

$$\Pr\Big(\, \mathsf{EFG}_{\mathsf{Sig},\mathcal{A}}(\lambda) = \mathtt{true} \,\Big) = \mathrm{negl}(\lambda), \quad (3)$$

where $\mathsf{EFG}_{\mathsf{Sig},\mathcal{A}}(\lambda)$ is the existential forgery game recapitulated in Alg. 12.

We denote by $[\boldsymbol{x}]_i$ the i-th entry of a vector $\boldsymbol{x}$, by $[\boldsymbol{X}]_i$ the i-th column of a matrix $\boldsymbol{X}$, and by $[n] \triangleq \{1, ..., n\}$.

2.2 Reed-Solomon Codes

A *linear* (n, k)*-code* is a linear mapping $\mathbb{Z}_q^k \longrightarrow \mathbb{Z}_q^n$ with $n \geq k$. It can be represented by a $k \times n$ *generator matrix* $\boldsymbol{G}$, with the encoding operation then $\boldsymbol{c}^\top = \boldsymbol{G}.\mathrm{Encode}(\boldsymbol{u}^\top) \triangleq \boldsymbol{u}^\top \boldsymbol{G}$ 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

$$\boldsymbol{u}^\top \underbrace{\begin{bmatrix} \boldsymbol{g}_{i_1} & ... & \boldsymbol{g}_{i_k} \end{bmatrix}}_{\triangleq \tilde{\boldsymbol{G}}} \overset{!}{=} \underbrace{\begin{bmatrix} c_{i_1} & ... & c_{i_k} \end{bmatrix}}_{\triangleq \tilde{\boldsymbol{c}}^\top} \\ \Longleftrightarrow \ \boldsymbol{u}^\top = \boldsymbol{G}.\mathrm{Decode}(((i_j, c_{i_j}))_{j=1}^{k}) \triangleq \tilde{\boldsymbol{c}}^\top \tilde{\boldsymbol{G}}^{-1}, \quad (4)$$

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

Reed-Solomon codes [25] 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 $\alpha_1, ..., \alpha_n$, such that $\boldsymbol{c}^\top = (U(\alpha_1), ..., U(\alpha_n))^\top$. This corresponds to a generator matrix $\boldsymbol{G}_{\mathrm{RS}}$ with columns $\boldsymbol{g}_{\mathrm{RS},i} = (\alpha_i^0, ..., \alpha_i^{k-1})$.

2.3 Linear Vector Commitment Schemes

A deterministic *vector commitment* (VC) scheme $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{OpenEntry}, \mathsf{VerifyEntry})$ [8, 21] for vectors of length L allows to commit to an element of $\mathbb{Z}_q^L$. Later, the commitment can be compared to the commitment of another vector to check a vector opening, and it can be opened to individual entries of the vector. Ideally, the proof for the opening of an entry of the vector is short and computationally easy to generate and verify. For our purposes it is important that the VC is binding, *i.e.*, if a commitment cannot be opened to values that are inconsistent with the committed vector. Specifically, we call a VC, see Definition B.3, *binding* if for any PPT adversary $\mathcal{A}$ there exists a negligible function negl(.) such that

$$\Pr\big(\, \mathsf{VCBG}_{\mathsf{LVC},\mathcal{A}}(\lambda) = \mathtt{true} \,\big) = \mathrm{negl}(\lambda). \quad (5)$$

where $\mathsf{VCBG}_{\mathsf{LVC},\mathcal{A}}(\lambda)$ is the binding game defined in Alg. 14. We use $\mathsf{VC}(\boldsymbol{v}) \triangleq \mathsf{LVC.Commit}(\boldsymbol{v})$ as a notational shorthand.

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

$$\forall \alpha, \beta \in \mathbb{Z}_q \colon \forall \boldsymbol{v}, \boldsymbol{w} \in \mathbb{Z}_q^L \colon \\ \mathsf{Commit}(\alpha \boldsymbol{v} + \beta \boldsymbol{w}) = \alpha \mathsf{Commit}(\boldsymbol{v}) + \beta \mathsf{Commit}(\boldsymbol{w}). \quad (6)$$

Kate-Zaverucha-Goldberg (KZG) polynomial commitments [15] (here the 'basic' variant $\mathsf{PolyCommit}_{\mathsf{DL}}$ of [15] 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's Commit is linear, consider this. During trusted setup, $\mathsf{KZG.Setup}$ samples $r \xleftarrow{\mathrm{R}} \mathbb{Z}_q$ and computes public parameters $(g^{r^0}, ..., g^{r^{L-1}})$. $\mathsf{KZG.Commit}$ 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

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

$H(x) = g^x$ discussed above, can be obtained from the public parameters as

$$\mathsf{KZG.Commit}(\gamma_0, ..., \gamma_{L-1}) = \mathsf{KZG.Commit}(U) = \prod_{j=0}^{L-1} (g^{r^j})^{\gamma_j}. \quad (7)$$

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* $P_1, ..., P_n$ 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 3.1 (Semi-AVID-PR Syntax). A *Semi-AVID (Asynchronous Verifiable Information Dispersal) Scheme with Provable Retrievability* (Π_{SAVIDPR}) consists of two algorithms, Commit and Verify, and three protocols, Setup, Disperse and Retrieve.

- Setup: $1^\lambda \mapsto (\mathsf{pp}, \mathsf{sp}_1, ..., \mathsf{sp}_n)$: 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 of these inputs is subsequently omitted for simplicity of notation.
- Commit: $B \mapsto C$: The algorithm Commit takes as input a block B of data, and returns a commitment C to the data.
- Disperse: $B \mapsto P$: 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 $\bot$ or a *certificate of retrievability* P for commitment $C = \mathsf{Commit}(B)$ to the client.
- Verify: $(P, C) \mapsto b \in \{\mathtt{true}, \mathtt{false}\}$: The algorithm Verify takes as input a certificate of retrievability P and a commitment C, and returns `true` or `false`, depending on whether the certificate is considered valid.
- Retrieve: $(P, C) \mapsto B$: 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 $\bot$ or a block B of data to the client.

Algorithm 1 Commitment-binding game (CBG) against Semi-AVID-PR scheme Π_{SAVIDPR} = (Setup, Commit, Disperse, Verify, Retrieve)

```
$(\mathsf{pp}, \mathsf{sp}_1, ..., \mathsf{sp}_n) \leftarrow \mathsf{Setup}(1^\lambda)$   ▷ Run setup for all parties
$(B, B') \leftarrow \mathcal{A}_{\text{CBG}}(\mathsf{pp}, \mathsf{sp}_1, ..., \mathsf{sp}_n)$   ▷ $\mathcal{A}$ can simulate any party
return $B \neq B' \wedge \mathsf{Commit}(B) = \mathsf{Commit}(B')$
```

Algorithm 2 Availability game (AvG) with resilience t against Semi-AVID-PR scheme Π_{SAVIDPR} = (Setup, Commit, Disperse, Verify, Retrieve)

```
$C \leftarrow \emptyset$   ▷ Bookkeeping of corrupted parties
$\forall i \in [n] : P_i \leftarrow \mathsf{new}\,\Pi_{\text{SAVIDPR}}(\emptyset)$   ▷ Instantiate $P_i$ as $\Pi_{\text{SAVIDPR}}$ with blank state
$\mathsf{pp} \leftarrow \mathsf{Setup}^{P_1,...,P_n}(1^\lambda)$   ▷ Run setup among all parties
function $O^{\text{corrupt}}(i)$   ▷ Oracle for $\mathcal{A}$ to corrupt parties
    assert $i \notin C$
    $C \leftarrow C \cup \{i\}$   ▷ Mark party as corrupted
    return $P_i$   ▷ Hand $P_i$'s state to $\mathcal{A}$
function $O^{\text{interact}}(i, m)$   ▷ Oracle for $\mathcal{A}$ to interact with parties
    assert $i \notin C$
    return $P_i(m)$   ▷ Execute $P_i$ on input $m$, return output to $\mathcal{A}$
$\left(P, C, \left(O_i^{\text{node}}(.)\right)_{i \in C}\right) \leftarrow \mathcal{A}_{\text{AvG}}^{O^{\text{corrupt}}(.), O^{\text{interact}}(.)}(\mathsf{pp})$   ▷ $\mathcal{A}$ returns certificate of retrievability $P$, commitment $C$, and oracle access to corrupted nodes for retrieval
$\hat{B} \leftarrow \mathsf{Retrieve}^{P_1,...,P_n}\left[O_i^{\text{node}}(.)/\textsc{Query}(i, .)\right]_{i \in C}(P, C)$   ▷ During retrieval, interact with corrupted nodes through oracles
return $|C| \leq t$   ▷ $\mathcal{A}$ wins iff: while
        $\wedge\ \mathsf{Verify}(P, C) = \mathtt{true}$
        $\wedge\ \mathsf{Commit}(\hat{B}) \neq C$
    corrupting no more than t parties, $\mathcal{A}$ produces a valid certificate of retrievability $P$ for $C$ such that retrieval does not return a file matching $C$
```

Definition 3.2 (Semi-AVID-PR Security). A Semi-AVID-PR scheme Π_{SAVIDPR} is *secure with resilience t* if for all executions with $f \leq t$:

(1) **Commitment-Binding.** Commit of Π_{SAVIDPR} implements a binding deterministic commitment to a block B of data. More formally, Π_{SAVIDPR} is *commitment-binding* if for any PPT adversary $\mathcal{A}$ there exists a negligible function negl(.) such that

$$\Pr\left(\mathsf{CBG}_{\Pi_{\text{SAVIDPR}},\mathcal{A}}(\lambda) = \mathtt{true}\right) = \mathrm{negl}(\lambda), \quad (8)$$

where $\mathsf{CBG}_{\Pi_{\text{SAVIDPR}},\mathcal{A}}(\lambda)$ is the commitment-binding game defined in Alg. 1.

(2) **Correctness.** If an honest client invokes Disperse with a block B of data, then eventually it outputs a certificate of retrievability P with the property that $\mathsf{Verify}(P, \mathsf{Commit}(B)) = \mathtt{true}$.

(3) **Availability.** For a certificate of retrievability P and a commitment C, if $\mathsf{Verify}(P, C) = \mathtt{true}$, then if an honest client invokes Retrieve with P and C, then eventually it outputs a block B of data such that $\mathsf{Commit}(B) = C$. More formally, Π_{SAVIDPR} provides *availability* if for any PPT adversary $\mathcal{A}$ there exists a negligible function negl(.) such that

$$\Pr\left(\mathsf{AvG}_{\Pi_{\text{SAVIDPR}},\mathcal{A}}(\lambda, t) = \mathtt{true}\right) = \mathrm{negl}(\lambda), \quad (9)$$

where $\mathsf{AvG}_{\Pi_{\text{SAVIDPR}},\mathcal{A}}(\lambda, t)$ is the availability game defined in Alg. 2.

Algorithm 3 $\Pi^{\bigstar}.\mathsf{Setup}(1^{\lambda})$

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

Algorithm 4 $\Pi^{\bigstar}.\mathsf{Commit}(B)$

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

Algorithm 5 $\Pi^{\bigstar}.\mathsf{Verify}(P, C)$

```
$\hat{q} \leftarrow \left|\left\{ i \,\middle|\, \exists (i \mapsto \sigma) \in P\colon \mathsf{Sig.Verify}(\mathsf{pk}_i, (\mathsf{ack}, C), \sigma) = \mathtt{true} \right\}\right|$
if $\hat{q} \geq q$ return true
return false
```

A few remarks are due on this formulation. Unlike earlier formulations of AVID [7, 13, 29], 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 Commitment-Binding property of the commitment scheme and can be proven to a third party using the certificate of retrievability. This matches the Validium rollup application, where on the one hand retrievability of content matching a certain commitment needs to be verifiable on-chain, and on the other hand validity of the block content is proved and verified with respect to the commitment. This can also 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 [7], our Correctness property takes the place of the Termination and Agreement properties, and our Availability 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 $\Pi^{\bigstar}$ 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^s, 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 and erasure-correction coding as demonstrated in Section 7. 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 Section 6).

Algorithm 6 $\Pi^{\bigstar}.\mathsf{Disperse}(B)$

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

Algorithm 7 $\Pi^{\bigstar}.\mathsf{Retrieve}(P, C)$

```
At the client:
Extract from $P$ any $q$ unique $\left\{ i \,\middle|\, \exists (i \mapsto \sigma) \in P\colon \mathsf{Sig.Verify}(\mathsf{pk}_i, (\mathsf{ack}, C), \sigma) = \mathtt{true} \right\}$
Send $(\mathtt{load}, C)$ to all storage nodes $i$
At storage node $i$ upon receiving $(\mathtt{load}, C)$:
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 $C = \mathsf{CRHF}^s(h_1 \| ... \| h_k)$
$\hat{\boldsymbol{h}} \leftarrow \mathsf{Code.Encode}(h_1, ..., h_k)$
Discarding any $i$ with $[\hat{\boldsymbol{h}}]_i \neq \mathsf{VC}(\boldsymbol{c}_i)$, wait for $k$ remaining unique $\{i_j\}_{j=1}^{k}$
return $\mathsf{Code.Decode}(((i_j, \boldsymbol{c}_{i_j}))_{j=1}^{k})$
```

Our construction $\Pi^{\bigstar}$ is provided in Algs. 3 to 7. See also Figure 5 for an illustration of the Disperse protocol (*cf.* Figure 4). Our approach is related to AVID-FP [13] in that we also use the linear homomorphism between the LVC and the erasure-correcting code. More specifically, during Disperse, the input file B is arranged as an $L \times k$ matrix $\boldsymbol{U}$ (using $\mathrm{AsMatrix}_{L\times k}$) and a commitment h_i is taken per column $\boldsymbol{u}_i$. Vectorization of the k column commitments of $\boldsymbol{U}$ is denoted as $\mathsf{VC}^{\otimes k}(\boldsymbol{U})$. 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

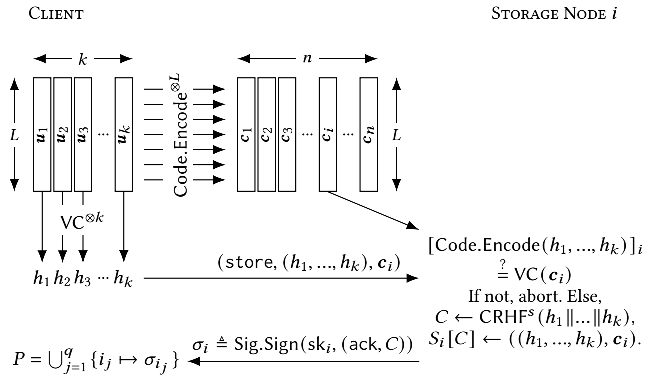

Client
Storage Node i
k
n
L
$\boldsymbol{u}_1$ $\boldsymbol{u}_2$ $\boldsymbol{u}_3$ ⋯ $\boldsymbol{u}_k$
$\mathsf{Code.Encode}^{\otimes L}$
$\boldsymbol{c}_1$ $\boldsymbol{c}_2$ $\boldsymbol{c}_3$ ⋯ $\boldsymbol{c}_i$ ⋯ $\boldsymbol{c}_n$
$\mathsf{VC}^{\otimes k}$
$h_1\, h_2\, h_3 \cdots h_k$
$(\mathsf{store}, (h_1, ..., h_k), \boldsymbol{c}_i)$
$[\mathsf{Code.Encode}(h_1, ..., h_k)]_i \overset{?}{=} \mathsf{VC}(\boldsymbol{c}_i)$
If not, abort. Else,
$C \leftarrow \mathsf{CRHF}^s(h_1\|...\|h_k)$,
$S_i[C] \leftarrow ((h_1, ..., h_k), \boldsymbol{c}_i)$.
$\sigma_i \triangleq \mathsf{Sig.Sign}(\mathsf{sk}_i, (\mathsf{ack}, C))$
$P = \bigcup_{j=1}^{q} \{i_j \mapsto \sigma_{i_j}\}$

Figure 5: Disperse **protocol of our Semi-AVID-PR construction $\Pi^\bigstar$ (*cf.* Figure 4). Client arranges data in $L \times k$ matrix $\boldsymbol{U}$, computes commitments $h_1, ..., h_k$ column-wise and $L \times n$ coded matrix C row-wise. Commitments and i-th column $\boldsymbol{c}_i$ of 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}.**

from AVID-FP in that AVID-FP still performs a round of AVID (for the 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 3).

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 3), a trusted party performs the setup of the LVC and the HF, 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 4) is computed by arranging B as an $L \times k$ matrix $\boldsymbol{U}$, then computing the commitments h_i as $\mathsf{VC}(\boldsymbol{u}_i)$ for each of the k columns $\boldsymbol{u}_i$ of $\boldsymbol{U}$, and finally $\mathsf{CRHF}^s(h_1\|...\|h_k)$ is the commitment.

To Disperse a block B (Algorithm 6, Figures 4, 5), 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)

$$[\mathsf{Code.Encode}(h_1, ..., h_k)]_i \overset{?}{=} \mathsf{VC}(\boldsymbol{c}_i), \qquad (10)$$

before computing the file's commitment $C \triangleq \mathsf{CRHF}^s(h_1\|...\|h_k)$ and storing commitments and chunk indexed by C. The storage node then acknowledges receipt of the chunk by sending a signature on (ack, 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 5), one counts whether P contains valid signatures on (ack, 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 (ack, C). The client then queries 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 $C = \mathsf{CRHF}^s(h_1\|...\|h_k)$. Then, the client discards any chunks that do not satisfy the homomorphic property (10). 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), $\boldsymbol{U}$ can be augmented by the client with a column and a row drawn uniformly a random, to blind the encoded chunks. Details in Section 6.

5 SECURITY PROOF

We first provide a proof sketch to convey the relevant high level intuition, before providing a formal reduction-based proof of Commitment-Binding and Availability in Lemmas 5.2 and 5.3.

Theorem 5.1. *The Semi-AVID-PR construction* $\Pi^\bigstar$ *of Section 4 is* secure with resilience t *for any* $t < n/2$ *as defined in Section 3, assuming* HF *is collision resistant,* Sig *is existentially unforgeable, and* LVC *is binding.*

Proof sketch. **Commitment-Binding.** $\Pi^\bigstar$ is deterministic because VC and CRHF^s are. For binding, assume for contradiction that $\Pi^\bigstar$ was not commitment-binding, *i.e.*, there was an adversary $\mathcal{A}$ that can produce blocks $B \neq B'$ such that $\mathsf{Commit}(B) = \mathsf{Commit}(B')$ with non-negligible probability. Since $B \neq B'$, for their respective representations as $L \times k$ matrices, $\boldsymbol{U} \neq \boldsymbol{U'}$. Either $\boldsymbol{h} \triangleq \mathsf{VC}^{\otimes k}(\boldsymbol{U}) \neq \mathsf{VC}^{\otimes k}(\boldsymbol{U'}) \triangleq \boldsymbol{h'}$ but $\mathsf{CRHF}^s(\boldsymbol{h}) = \mathsf{CRHF}^s(\boldsymbol{h'})$, a collision in CRHF^s, which can happen with negligible probability only by assumption, or $\boldsymbol{h} = \boldsymbol{h'}$, so that for some i, $\mathsf{VC}([\boldsymbol{U}]_i) = \mathsf{VC}([\boldsymbol{U'}]_i)$ but $[\boldsymbol{U}]_i \neq [\boldsymbol{U'}]_i$, so $([\boldsymbol{U}]_i, [\boldsymbol{U'}]_i)$ is a pair that breaks the binding property of VC, which can happen with negligible probability only by assumption. Thus, $\Pi^\bigstar$ is commitment-binding.

Lemma 5.2 below establishes Commitment-Binding formally.

Correctness. Since the client is honest and LVC is linearly homomorphic, the consistency check in Algorithm 6 l. 8 passes at all honest storage nodes. So the client receives signatures σ_{i_j} (which are valid, by correctness of Sig) of $(\mathsf{ack}, \mathsf{Commit}(B))$ 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 5 by construction, *if* $q \leq (n-t)$.

Availability. Since $\mathsf{Verify}(P, C) = \mathtt{true}$ by assumption, the client can extract some q unique storage nodes i_j from P in Algorithm 7 l. 2. Of these q storage nodes, at least $(q-t)$ remain honest. Security of Sig implies that they must have previously executed Algorithm 6 l. 11 and hence stored $(h_1, ..., h_k)$ for $C =$

Algorithm 8 $\mathcal{A}_{\text{CFG}\leftarrow\text{CBG}}(s)$ constructed from $\mathcal{A}_{\text{CBG}}$

```
$\mathsf{pp}_{\mathsf{LVC}} \leftarrow \mathsf{LVC.Setup}(1^\lambda)$    ▷ Setup $\Pi^\bigstar$ (cf. Alg. 3) ...
$\forall i \in [n] : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$
$\mathsf{pp}_{\mathsf{HF}} \leftarrow s$    ▷ ... except use $s$ for $\mathsf{pp}_{\mathsf{HF}}$
$\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{LVC}}, \mathsf{pp}_{\mathsf{HF}}, \mathsf{pk}_1, ..., \mathsf{pk}_n)$
$(B, B') \leftarrow \mathcal{A}_{\text{CBG}}(\mathsf{pp}, \mathsf{sp}_1, ..., \mathsf{sp}_n)$
$\boldsymbol{U} \leftarrow \text{AsMatrix}_{L\times k}(B)$
$\boldsymbol{U}' \leftarrow \text{AsMatrix}_{L\times k}(B')$
$(h_1, ..., h_k) \leftarrow \mathsf{VC}^{\otimes k}(\boldsymbol{U})$
$(h'_1, ..., h'_k) \leftarrow \mathsf{VC}^{\otimes k}(\boldsymbol{U}')$
if $h_1\|...\|h_k \neq h'_1\|...\|h'_k$
        $\wedge\ \mathsf{CRHF}^s(h_1\|...\|h_k) = \mathsf{CRHF}^s(h'_1\|...\|h'_k)$
    return $(h_1\|...\|h_k, h'_1\|...\|h'_k)$    ▷ Collision in $\mathsf{HF.H}$
else
    abort    ▷ No collision identified
```

Algorithm 9 $\mathcal{A}_{\text{VCBG}\leftarrow\text{CBG}}(\mathsf{pp})$ constructed from $\mathcal{A}_{\text{CBG}}$

```
$\mathsf{pp}_{\mathsf{HF}} \leftarrow \mathsf{HF.Gen}(1^\lambda)$    ▷ Setup $\Pi^\bigstar$ (cf. Alg. 3) ...
$\forall i \in [n] : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$
$\mathsf{pp}_{\mathsf{LVC}} \leftarrow \mathsf{pp}$    ▷ ... except use $\mathsf{pp}$ for $\mathsf{pp}_{\mathsf{LVC}}$
$\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{LVC}}, \mathsf{pp}_{\mathsf{HF}}, \mathsf{pk}_1, ..., \mathsf{pk}_n)$
$(B, B') \leftarrow \mathcal{A}_{\text{CBG}}(\mathsf{pp}, \mathsf{sp}_1, ..., \mathsf{sp}_n)$
$\boldsymbol{U} \leftarrow \text{AsMatrix}_{L\times k}(B)$
$\boldsymbol{U}' \leftarrow \text{AsMatrix}_{L\times k}(B')$
if $\exists i \in [k] : [\boldsymbol{U}]_i \neq [\boldsymbol{U}']_i \wedge \mathsf{VC}([\boldsymbol{U}]_i) = \mathsf{VC}([\boldsymbol{U}']_i)$
    return $([\boldsymbol{U}]_i, [\boldsymbol{U}']_i)$
else
    abort
```

$\mathsf{CRHF}^s(h_1, ..., h_k)$ in Algorithm 6 l. 10 and their chunks satisfied the consistency check in Algorithm 6 l. 8. Note that by collision resistance of CRHF^s, 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 7 l. 8, and *if* $k \leq (q - t)$, then the client eventually also completes the wait in Algorithm 7 l. 10. Finally, since Code is an MDS (n, k)-code, Algorithm 7 l. 11 succeeds to decode a block B, corresponding to an $L \times k$ matrix $\boldsymbol{U}$. It remains to show that $\text{Commit}(B) = C$, for which (by CRHF^s) it suffices that $(h_1, ..., h_k) = \mathsf{VC}^{\otimes k}(\boldsymbol{U})$. 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

$$\underbrace{\begin{bmatrix}\boldsymbol{u}_1 & \dots & \boldsymbol{u}_k\end{bmatrix}}_{=\boldsymbol{U}} \underbrace{\begin{bmatrix}\boldsymbol{g}_{i_1} & \dots & \boldsymbol{g}_{i_k}\end{bmatrix}}_{\triangleq \tilde{\boldsymbol{G}}} \overset{!}{=} \underbrace{\begin{bmatrix}\boldsymbol{c}_{i_1} & \dots & \boldsymbol{c}_{i_k}\end{bmatrix}}_{\triangleq \tilde{\boldsymbol{C}}} \iff \boldsymbol{U} = \tilde{\boldsymbol{C}}\tilde{\boldsymbol{G}}^{-1}. \quad (11)$$

At the same time, by the checks in Algorithm 7 l. 10,

$$\begin{bmatrix}h_1 & \dots & h_k\end{bmatrix} \tilde{\boldsymbol{G}} = \begin{bmatrix}\mathsf{VC}(\boldsymbol{c}_{i_1}) & \dots & \mathsf{VC}(\boldsymbol{c}_{i_k})\end{bmatrix}. \quad (12)$$

By the linear homomorphism of LVC,

$$\mathsf{VC}^{\otimes k}(\boldsymbol{U}) = \mathsf{VC}^{\otimes k}(\tilde{\boldsymbol{C}})\tilde{\boldsymbol{G}}^{-1} = \begin{bmatrix}h_1 & \dots & h_k\end{bmatrix}. \quad (13)$$

Thus, as long as the retrieving client receives enough chunks from honest storage nodes such that Retrieve outputs any block $\hat{B}$ rather than $\perp$, $\text{Commit}(\hat{B}) = C$ for the commitment C associated with the certificate of retrievability P. This insight is crucial for the subsequent formal proof of Availability.

Lemma 5.3 below establishes Availability formally.

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

We proceed to give formal reduction-based proofs of the Commitment-Binding and Availability properties of $\Pi^\bigstar$.

Lemma 5.2. *If* HF *is collision resistant and* LVC *is binding, then* $\Pi^\bigstar$ *is commitment-binding,* i.e., *for all PPT adversaries* $\mathcal{A}_{\text{CBG}}$ *there exists a negligible function* $\text{negl}(.)$ *such that*

$$\Pr\Big(\mathsf{CBG}_{\Pi^\bigstar, \mathcal{A}_{\text{CBG}}}(\lambda) = \mathtt{true}\Big) \leq \text{negl}(\lambda). \quad (14)$$

Proof. Let $\mathcal{A}_{\text{CBG}}$ be an arbitrary PPT CBG adversary. We construct from it the adversaries $\mathcal{A}_{\text{CFG}\leftarrow\text{CBG}}$ against CFG and $\mathcal{A}_{\text{VCBG}\leftarrow\text{CBG}}$ against VCBG. The adversary $\mathcal{A}_{\text{CFG}\leftarrow\text{CBG}}$ is detailed in Alg. 8. It receives a challenge s and runs $\mathsf{LVC.Setup}(1^\lambda)$ and $\mathsf{Sig.KeyGen}(1^\lambda)$ to produce the remaining public parameters pp and the secret parameters $(\mathsf{sp}_1, ..., \mathsf{sp}_n)$ for $\mathcal{A}_{\text{CBG}}$. After $\mathcal{A}_{\text{CBG}}$ outputs a pair (B, B'), $\mathcal{A}_{\text{CFG}\leftarrow\text{CBG}}$ computes $\boldsymbol{U} \leftarrow \text{AsMatrix}_{L\times k}(B)$ and $\boldsymbol{U}' \leftarrow \text{AsMatrix}_{L\times k}(B)$. Next, it computes $(h_1, ..., h_k) \leftarrow \mathsf{VC}^{\otimes k}(\boldsymbol{U})$ and $(h'_1, ..., h'_k) \leftarrow \mathsf{VC}^{\otimes k}(\boldsymbol{U}')$ to check whether $h_1\|...\|h_k$ and $h'_1\|...\|h'_k$ are a collision of CRHF^s. If so, it outputs $(h_1\|...\|h_k, h'_1\|...\|h'_k)$; else it aborts.

The adversary $\mathcal{A}_{\text{VCBG}\leftarrow\text{CBG}}$ is detailed in Alg. 9. It receives the challenge $\mathsf{pp}_{\mathsf{LVC}}$ and runs $\mathsf{HF.Gen}(1^\lambda)$ and $\mathsf{Sig.KeyGen}(1^\lambda)$ to produce the remaining public parameters pp and the secret parameters $(\mathsf{sp}_1, ..., \mathsf{sp}_n)$ for $\mathcal{A}_{\text{CBG}}$. After $\mathcal{A}_{\text{CBG}}$ outputs a pair (B, B'), $\mathcal{A}_{\text{VCBG}\leftarrow\text{CBG}}$ computes $\boldsymbol{U} \leftarrow \text{AsMatrix}_{L\times k}(B)$ and $\boldsymbol{U}' \leftarrow \text{AsMatrix}_{L\times k}(B')$. Next, it checks if there exists $i \in [k]$ such that the column $[\boldsymbol{U}]_i \neq [\boldsymbol{U}']_i$ but $\mathsf{VC}([\boldsymbol{U}]_i) = \mathsf{VC}([\boldsymbol{U}']_i)$. If so, it outputs the collision $([\boldsymbol{U}]_i, [\boldsymbol{U}']_i)$; else it aborts.

The adversaries $\mathcal{A}_{\text{CFG}\leftarrow\text{CBG}}$ and $\mathcal{A}_{\text{VCBG}\leftarrow\text{CBG}}$ run in polynomial time. The input of the adversary $\mathcal{A}_{\text{CBG}}$ when run as a subroutine of $\mathcal{A}_{\text{CFG}\leftarrow\text{CBG}}$ or $\mathcal{A}_{\text{VCBG}\leftarrow\text{CBG}}$ is distributed identically to the input of the adversary $\mathcal{A}_{\text{CBG}}$ when run in CBG.

For the subsequent arguments we define the following events:

$$E_{\text{CB}} \triangleq \{\mathsf{CBG}_{\Pi^\bigstar, \mathcal{A}_{\text{CBG}}}(\lambda) = \mathtt{true}\} \quad (15)$$

$$E_{\text{CF}} \triangleq \{\mathsf{CFG}_{\mathsf{HF}, \mathcal{A}_{\text{CFG}\leftarrow\text{CBG}}}(\lambda) = \mathtt{true}\} \quad (16)$$

$$E_{\text{VCB}} \triangleq \{\mathsf{VCBG}_{\mathsf{LVC}, \mathcal{A}_{\text{VCBG}\leftarrow\text{CBG}}}(\lambda) = \mathtt{true}\} \quad (17)$$

$$E \triangleq \{\mathsf{VC}^{\otimes k}(\boldsymbol{U}) \neq \mathsf{VC}^{\otimes k}(\boldsymbol{U}') \wedge \boldsymbol{U} \neq \boldsymbol{U}'\} \quad (18)$$

$$U \triangleq E_{\text{CF}} \vee E_{\text{VCB}} \quad (19)$$

Suppose E_{CB} holds and $\mathcal{A}_{\text{CBG}}$ outputs (B, B') such that $\text{Commit}(B) = \text{Commit}(B')$ but $B \neq B'$. Hence, $\mathsf{CRHF}^s(h_1\|...\|h_k) = \mathsf{CRHF}^s(h'_1\|...\|h'_k)$, where $\boldsymbol{U} \leftarrow \text{AsMatrix}_{L\times k}(B)$, $\boldsymbol{U}' \leftarrow \text{AsMatrix}_{L\times k}(B')$, $(h_1, ..., h_k) \leftarrow \mathsf{VC}^{\otimes k}(\boldsymbol{U})$ and $(h'_1, ..., h'_k) \leftarrow \mathsf{VC}^{\otimes k}(\boldsymbol{U}')$. We consider two cases. If E holds, then $(h_1, ..., h_k) \neq (h_1, ..., h_k)$. Thus, in the event of E, $(h_1\|...\|h_k, h'_1\|...\|h'_k)$ is a collision of CRHF^s, so the event E_{CF} holds. In the case of $\neg E$, $(h_1, ..., h_k) = (h'_1, ..., h'_k)$. Thus, there exists

$i \in [k]$ such that $[\boldsymbol{U}]_i \neq [\boldsymbol{U'}]_i$ but $\mathsf{VC}([\boldsymbol{U}]_i) = \mathsf{VC}([\boldsymbol{U'}]_i)$. So under $\neg E$, the event E_{VCB} holds.

Observe that if E holds, then E_{CF} holds. Hence, $E \subseteq E_{\mathrm{CF}}$ and $\Pr(\neg E_{\mathrm{CF}} \wedge E) = 0$. Similarly, if $\neg E$ holds, then E_{VCB} holds. Hence, $\neg E \subseteq E_{\mathrm{VCB}}$ and $\Pr(\neg E_{\mathrm{VCB}} \wedge \neg E) = 0$.

We can now bound the probability of E_{CB}:

$$
\begin{aligned}
& \Pr(E_{\mathrm{CB}}) \\
\overset{\text{(a)}}{=}\; & \Pr(E_{\mathrm{CB}} \mid U)\Pr(U) + \Pr(E_{\mathrm{CB}} \mid \neg U)\Pr(\neg U) && (20)\\
\overset{\text{(b)}}{\leq}\; & \Pr(U) + \Pr(E_{\mathrm{CB}} \wedge \neg U) && (21)\\
\overset{\text{(c)}}{\leq}\; & \Pr(U) + \Pr(E_{\mathrm{CB}} \wedge \neg U \wedge E) + \Pr(E_{\mathrm{CB}} \wedge \neg U \wedge \neg E) && (22)\\
\overset{\text{(d)}}{\leq}\; & \Pr(U) + \Pr(\neg E_{\mathrm{CF}} \wedge E) + \Pr(\neg E_{\mathrm{VCB}} \wedge \neg E) && (23)\\
\overset{\text{(e)}}{\leq}\; & \Pr(U) && (24)\\
\overset{\text{(f)}}{\leq}\; & \Pr(E_{\mathrm{CF}}) + \Pr(E_{\mathrm{VCB}}) && (25)
\end{aligned}
$$

where (a) uses the law of total probability (TP) to introduce U; (b) uses $\Pr(E_{\mathrm{CB}} \mid U) \leq 1$; (c) uses TP to introduce E; (d) uses

$$\Pr(E_{\mathrm{CB}} \wedge \neg E_{\mathrm{CF}} \wedge \neg E_{\mathrm{VCB}} \wedge E) \leq \Pr(\neg E_{\mathrm{CF}} \wedge E) \qquad (26)$$

and

$$\Pr(E_{\mathrm{CB}} \wedge \neg E_{\mathrm{CF}} \wedge \neg E_{\mathrm{VCB}} \wedge \neg E) \leq \Pr(\neg E_{\mathrm{VCB}} \wedge \neg E); \qquad (27)$$

(e) uses $\Pr(\neg E_{\mathrm{CF}} \wedge E) = 0$ and $\Pr(\neg E_{\mathrm{VCB}} \wedge \neg E) = 0$; (f) uses a union bound.

Since by assumption HF is collision resistant and LVC is binding, there exist $\mathrm{negl}_1(.), \mathrm{negl}_2(.)$ such that $\Pr(E_{\mathrm{CF}}) \leq \mathrm{negl}_1(\lambda)$ and $\Pr(E_{\mathrm{VCB}}) \leq \mathrm{negl}_2(\lambda)$. Thus,

$$\Pr(E_{\mathrm{CB}}) \leq \mathrm{negl}_1(\lambda) + \mathrm{negl}_2(\lambda) \leq \mathrm{negl}(\lambda). \qquad (28)$$

Hence, $\Pi^{\bigstar}$ is commitment-binding. □

Lemma 5.3. *If* HF *is collision resistant,* Sig *is secure against existential forgery, and* $t < n/2$*, then* $\Pi^{\bigstar}$ *provides availability,* i.e., *for all PPT adversaries* $\mathcal{A}_{\mathsf{AvG}}$ *there exists a negligible function* $\mathrm{negl}(.)$ *such that*

$$\Pr\Big(\mathsf{AvG}_{\Pi^{\bigstar},\mathcal{A}_{\mathsf{AvG}}}(\lambda, t) = \mathtt{true}\Big) \leq \mathrm{negl}(\lambda). \qquad (29)$$

Proof. First, we modify the availability game AvG (Alg. 2) to obtain AvG′ (Alg. 15) in which initially the index I of a storage node is sampled uniformly at random, and subsequently the game is aborted if the adversary $\mathcal{A}_{\mathsf{AvG'}}$ attempts to corrupt I. This modification will subsequently streamline the reduction of availability of $\Pi^{\bigstar}$ to security of Sig against existential forgery.

We reduce availability of $\Pi^{\bigstar}$ to availability′ of $\Pi^{\bigstar}$, *i.e.*, if for all PPT $\mathcal{A}_{\mathsf{AvG'}}$ there exists $\mathrm{negl}(.)$ such that

$$\Pr\Big(\mathsf{AvG}'_{\Pi^{\bigstar},\mathcal{A}_{\mathsf{AvG'}}}(\lambda, t) = \mathtt{true}\Big) \leq \mathrm{negl}(\lambda), \qquad (30)$$

then for all PPT $\mathcal{A}_{\mathsf{AvG}}$ there exists $\mathrm{negl}(.)$ such that

$$\Pr\Big(\mathsf{AvG}_{\Pi^{\bigstar},\mathcal{A}_{\mathsf{AvG}}}(\lambda, t) = \mathtt{true}\Big) \leq \mathrm{negl}(\lambda). \qquad (31)$$

To this end, pick any AvG adversary $\mathcal{A}_{\mathsf{AvG}}$. Note that $\mathcal{A}_{\mathsf{AvG'}} \triangleq \mathcal{A}_{\mathsf{AvG}}$ is an AvG′ adversary. Define the events:

$$
\begin{aligned}
E_{\mathrm{A}} &\triangleq \{\mathsf{AvG}_{\Pi^{\bigstar},\mathcal{A}_{\mathsf{AvG}}}(\lambda, t) = \mathtt{true}\} && (32)\\
E_{\mathrm{A'}} &\triangleq \{\mathsf{AvG}'_{\Pi^{\bigstar},\mathcal{A}_{\mathsf{AvG'}}}(\lambda, t) = \mathtt{true}\} && (33)
\end{aligned}
$$

Algorithm 10 $\mathcal{A}_{\mathsf{EFG}\leftarrow\mathsf{AvG'}}(\mathsf{pk})$ constructed from $\mathcal{A}_{\mathsf{AvG'}}$

```
$I \xleftarrow{R} [n]$    ▷ Choose random party $I$ to emulate using $O^{\mathrm{sign}}(.)$
$C \leftarrow \emptyset$    ▷ Bookkeeping of corrupted parties $C$
$\forall i \in [n] : S_i \leftarrow \emptyset$    ▷ Blank state for each party $P_i$
$\mathsf{pp}_{\mathsf{LVC}} \leftarrow \mathsf{LVC.Setup}(1^\lambda)$    ▷ Setup $\Pi^{\bigstar}$ (cf. Alg. 3) ...
$\mathsf{pp}_{\mathsf{HF}} \leftarrow \mathsf{HF.Gen}(1^\lambda)$
$\forall i \in [n] \setminus \{I\} : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$
$\mathsf{pk}_I \leftarrow \mathsf{pk}$    ▷ ... except use pk for party $I$'s $\mathsf{pk}_I$
$\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{LVC}}, \mathsf{pp}_{\mathsf{HF}}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$
function $O^{\mathrm{corrupt}}(i)$
    assert $i \notin C$
    if $i \neq I$
        $C \leftarrow C \cup \{i\}$
        return $(\mathsf{sk}_i, S_i)$
    else
        abort    ▷ Cannot hand over $P_I$ since $\mathsf{sk}_I$ is unknown
function $O^{\mathrm{interact}}(i, m)$
    assert $i \notin C$
    if $i \neq I$
        return $\Pi^{\bigstar,\mathsf{sk}_i,S_i}(m)$    ▷ Execute $P_i$ on input $m$ and state $\mathsf{sk}_i, S_i$, and return output to $\mathcal{A}$
    else
        return $\Pi^{\bigstar,S_i}[O^{\mathrm{sign}}(.)/\mathsf{Sig.Sign}(\mathsf{sk}_i,.)](m)$    ▷ Execute $P_I$ on input $m$ and state $S_i$, substituting $O^{\mathrm{sign}}(.)$ for invocations of $\mathsf{Sig.Sign}(\mathsf{sk}_i,.)$, and return output to $\mathcal{A}$
$\left(P, C, \left(O_i^{\mathrm{node}}(.)\right)_{i \in C}\right) \leftarrow \mathcal{A}_{\mathsf{AvG'}}^{O^{\mathrm{corrupt}}(.), O^{\mathrm{interact}}(.)}(\mathsf{pp})$
$\hat{B} \leftarrow \mathsf{Retrieve}^{P_1,\ldots,P_n}\left[O_i^{\mathrm{node}}(.)/\textsc{Query}(i,.)\right]_{i \in C}(P, C)$    ▷ During retrieval, interact with corrupted nodes through oracles
if $\exists \sigma : (I \mapsto \sigma) \in P \wedge S_I[C] = \emptyset$
    return $((\mathsf{ack}, C), \sigma)$    ▷ Forgery for $\mathsf{pk}_I = \mathsf{pk}$
else
    abort    ▷ No forgery for $\mathsf{pk}_I = \mathsf{pk}$ identified
```

Obviously, $E_{\mathrm{A}} \wedge \{I \notin C\} \subseteq E_{\mathrm{A'}}$. Furthermore, E_{A} implies $|C| \leq t < n/2$, so that $\Pr(I \notin C \mid E_{\mathrm{A}}) \geq 1/2$. Thus, by availability′ of $\Pi^{\bigstar}$,

$$
\begin{aligned}
\Pr(E_{\mathrm{A'}}) &\geq \Pr(E_{\mathrm{A}} \wedge \{I \notin C\}) && (34)\\
&\geq \Pr(\{I \notin C\} \mid E_{\mathrm{A}})\Pr(E_{\mathrm{A}}) \geq \frac{1}{2}\Pr(E_{\mathrm{A}}) && (35)\\
\Pr(E_{\mathrm{A}}) &\leq 2\Pr(E_{\mathrm{A'}}) \leq \mathrm{negl}(\lambda). && (36)
\end{aligned}
$$

We proceed with the reduction of availability′ of $\Pi^{\bigstar}$ to collision resistance of HF and security of Sig against existential forgery. Let $\mathcal{A}_{\mathsf{AvG'}}$ be an arbitrary PPT AvG′ adversary. We construct from it the adversaries $\mathcal{A}_{\mathsf{EFG}\leftarrow\mathsf{AvG'}}$ for the EFG and $\mathcal{A}_{\mathsf{CFG}\leftarrow\mathsf{AvG'}}$ for the CFG as detailed in Algs. 10 and 11, respectively. $\mathcal{A}_{\mathsf{EFG}\leftarrow\mathsf{AvG'}}$ emulates the AvG′ challenger (Alg. 15), except it does not generate a signature public/secret key pair for node I, but instead uses the input challenge pk, and it attempts to forge a signature for I. It uses the signature oracle $O^{\mathrm{sign}}(.)$ provided in the EFG to produce signatures for node I whenever $\Pi^{\bigstar}$ requires to do so. $\mathcal{A}_{\mathsf{CFG}\leftarrow\mathsf{AvG'}}$ emulates the AvG′ challenger (Alg. 15), except it uses the input challenge s for HF's key in the public parameters of $\Pi^{\bigstar}$. Throughout the protocol execution, for both dispersal and retrieval operations, the reduction adversary keeps track of any $(h_1, \ldots, h_k)$ that may present

Algorithm 11 $\mathcal{A}_{\mathsf{CFG}\leftarrow\mathsf{AvG}'}(s)$ constructed from $\mathcal{A}_{\mathsf{AvG}'}$

```
$I \xleftarrow{R} [n]$
$C, \mathcal{H} \leftarrow \emptyset, \emptyset$        ▷ Bookkeeping of corrupted parties $C$ and image/preimage pairs $\mathcal{H}$ for HF.H
$\forall i \in [n] : S_i \leftarrow \emptyset$        ▷ Blank state for each party $P_i$
$\mathsf{pp}_{\mathsf{LVC}} \leftarrow \mathsf{LVC.Setup}(1^\lambda)$        ▷ Setup $\Pi^\bigstar$ (cf. Alg. 3) ...
$\forall i \in [n] : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$
$\mathsf{pp}_{\mathsf{HF}} \leftarrow s$        ▷ ... except use $s$ for $\mathsf{pp}_{\mathsf{HF}}$
$\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{LVC}}, \mathsf{pp}_{\mathsf{HF}}, \mathsf{pk}_1, ..., \mathsf{pk}_n)$
function $\mathcal{O}^{\text{corrupt}}(i)$
    assert $i \notin C$
    if $i \neq I$
        $C \leftarrow C \cup \{i\}$
        return $(\mathsf{sk}_i, S_i)$
    else
        abort
function $\mathcal{O}^{\text{interact}}(i, m)$
    assert $i \notin C$
    if $m$ parses as $(\mathtt{store}, (h_1, ..., h_k), \boldsymbol{c})$
        $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\mathsf{CRHF}^s(h_1\|...\|h_k) \mapsto h_1\|...\|h_k)\}$        ▷ Record image/preimage pair for HF.H
    return $\Pi^{\bigstar,\mathsf{sk}_i,S_i}(m)$        ▷ Execute $P_i$ on input $m$ and state $\mathsf{sk}_i, S_i$, and return output to $\mathcal{A}$
$\left(P, C, \left(\mathcal{O}_i^{\text{node}}(.)\right)_{i \in C}\right) \leftarrow \mathcal{A}_{\mathsf{AvG}'}^{\mathcal{O}^{\text{corrupt}}(.),\mathcal{O}^{\text{interact}}(.)}(\mathsf{pp})$
function $\tilde{\mathcal{O}}_i^{\text{node}}(m)$
    $r \leftarrow \mathcal{O}_i^{\text{node}}(m)$
    if $r$ parses as $(i, (h_1, ..., h_k), \boldsymbol{c}_i)$
        $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\mathsf{CRHF}^s(h_1\|...\|h_k) \mapsto h_1\|...\|h_k)\}$        ▷ Record image/preimage pair for HF.H
    return $r$
$\mathsf{Retrieve}^{P_1,...,P_n}\left[\tilde{\mathcal{O}}_i^{\text{node}}(.)/\textsc{Query}(i, .)\right]_{i \in C}(P, C)$        ▷ During retrieval, interact with corrupted nodes through wrapped oracles
if $\exists x, x' : (C \mapsto x) \in \mathcal{H} \wedge (C \mapsto x') \in \mathcal{H} \wedge x \neq x'$
    return $(x, x')$        ▷ Collision in HF.H
else
    abort        ▷ No collision identified
```

colliding inputs for CRHF^s. Clearly, the adversaries $\mathcal{A}_{\mathsf{EFG}\leftarrow\mathsf{AvG}'}$ and $\mathcal{A}_{\mathsf{CFG}\leftarrow\mathsf{AvG}'}$ run in time polynomial in the security parameter λ. Furthermore, the input pp of $\mathcal{A}_{\mathsf{AvG}'}$ and its interactions through the oracles $\mathcal{O}^{\text{corrupt}}(.)$ and $\mathcal{O}^{\text{interact}}(.)$ are distributed identically when run by the challenger of AvG' and when run as a subroutine of $\mathcal{A}_{\mathsf{EFG}\leftarrow\mathsf{AvG}'}$ or $\mathcal{A}_{\mathsf{CFG}\leftarrow\mathsf{AvG}'}$ invoked by the challenger of EFG or CFG, respectively.

For the subsequent arguments we define the following events:

$$E_{\mathrm{E}} \triangleq \{\mathsf{EFG}_{\mathsf{Sig},\mathcal{A}_{\mathsf{EFG}\leftarrow\mathsf{AvG}'}}(\lambda) = \mathtt{true}\} \tag{37}$$

$$E_{\mathrm{C}} \triangleq \{\mathsf{CFG}_{\mathsf{HF},\mathcal{A}_{\mathsf{CFG}\leftarrow\mathsf{AvG}'}}(\lambda) = \mathtt{true}\} \tag{38}$$

$$E \triangleq \{(\exists\sigma : (I \mapsto \sigma) \in P) \wedge S_I[C] = \emptyset\} \tag{39}$$

The following facts will be useful. Observe that if $E \wedge E_{\mathrm{A}'}$ holds, then $\mathcal{A}_{\mathsf{EFG}\leftarrow\mathsf{AvG}'}$ wins the EFG, *i.e.*, E_{E} holds. So $E \wedge E_{\mathrm{A}'} \subseteq E_{\mathrm{E}}$. Thus, $\Pr(E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}} \wedge \neg E_{\mathrm{E}} \wedge E) = 0$. Inverting the implication, $\neg E_{\mathrm{E}} \subseteq \neg E \vee \neg E_{\mathrm{A}'}$. Thus, $\Pr(E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}} \wedge \neg E_{\mathrm{E}} \wedge \neg E) \leq \Pr(E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}} \wedge \neg E)$, where we have used a union bound and $E_{\mathrm{A}'} \wedge \neg E_{\mathrm{A}'} = \emptyset$.

Finally, suppose $\mathcal{A}_{\mathsf{AvG}'}$ as a subroutine of $\mathcal{A}_{\mathsf{EFG}\leftarrow\mathsf{AvG}'}$ behaves such that it would win the corresponding AvG'. Furthermore, suppose no collision is identified by $\mathcal{A}_{\mathsf{CFG}\leftarrow\mathsf{AvG}'}$, so $\neg E_{\mathrm{C}}$. Then, given that $|C| \leq t < q$, $\mathsf{Verify}(P, C) = \mathtt{true}$, the only way in which $\mathsf{Commit}(\hat{B}) \neq C$ could be true is if there is at least one node that is part of P, has not been corrupted, and has not previously stored a chunk associated with C. This is because, as was argued earlier in the proof sketch, if the retrieving client receives chunks from at least k honest storage nodes, it decodes a block that matches the expected commitment C. Hence, it must be the case that the client does not receive sufficiently many valid chunks. But since P contains $q > t$ valid signatures, but at most $t \geq |C|$ storage nodes are corrupted, and $k \leq (q - t)$ by design, there must be an honest storage node whose signature on $(\mathtt{ack}, C)$ is in P, yet the node does not respond to the retrieving client's query because it has never stored a chunk associated with C (and hence not signed $(\mathtt{ack}, C)$). Thus, $\Pr(E \mid E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}}) \geq \frac{1}{n}$.

Now we can bound, with $x \triangleq \Pr(E_{\mathrm{C}}) + \Pr(E_{\mathrm{E}})$,

$$\begin{align} \Pr(E_{\mathrm{A}'}) &\overset{(a)}{=} \Pr(E_{\mathrm{A}'} \wedge E_{\mathrm{C}}) + \Pr(E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}}) \tag{40} \\ &\overset{(b)}{\leq} \Pr(E_{\mathrm{C}}) + \Pr(E_{\mathrm{E}}) + \Pr(E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}} \wedge \neg E_{\mathrm{E}}) \tag{41} \\ &\overset{(c)}{\leq} x + \Pr(E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}} \wedge \neg E_{\mathrm{E}} \wedge \neg E) \tag{42} \\ &\overset{(d)}{\leq} x + \Pr(E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}} \wedge \neg E) \tag{43} \\ &= x + \Pr(\neg E \mid E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}}) \Pr(E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}}) \tag{44} \\ &\overset{(e)}{\leq} x + \frac{n-1}{n} \Pr(E_{\mathrm{A}'}) \tag{45} \\ \Pr(E_{\mathrm{A}'}) &\leq nx = n \Pr(E_{\mathrm{C}}) + n \Pr(E_{\mathrm{E}}) \tag{46} \end{align}$$

where (a) uses the law of total probability (TP) to introduce E_{C}; (b) uses TP to introduce E_{E}, $E_{\mathrm{A}'} \wedge E_{\mathrm{C}} \subseteq E_{\mathrm{C}}$, $E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}} \wedge E_{\mathrm{E}} \subseteq E_{\mathrm{E}}$; (c) uses TP to introduce E, $\Pr(E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}} \wedge \neg E_{\mathrm{E}} \wedge E) = 0$; (d) uses $\Pr(E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}} \wedge \neg E_{\mathrm{E}} \wedge \neg E) \leq \Pr(E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}} \wedge \neg E)$; (e) uses $E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}} \subseteq E_{\mathrm{A}'}$, $\Pr(E \mid E_{\mathrm{A}'} \wedge \neg E_{\mathrm{C}}) \geq \frac{1}{n}$.

Since by assumption HF is collision resistant and Sig is secure against existential forgery, there exist $\mathrm{negl}_1(.), \mathrm{negl}_2(.)$ such that $\Pr(E_{\mathrm{C}}) \leq \mathrm{negl}_1(\lambda)$ and $\Pr(E_{\mathrm{E}}) \leq \mathrm{negl}_2(\lambda)$. Furthermore, n is a constant independent of the security parameter λ. Thus,

$$\Pr(E_{\mathrm{A}'}) \leq n\,\mathrm{negl}_1(\lambda) + n\,\mathrm{negl}_2(\lambda) \leq \mathrm{negl}(\lambda). \tag{47}$$

Hence, $\Pi^\bigstar$ provides availability′, to which availability of $\Pi^\bigstar$ was reduced in the first part of the proof. □

6 PRIVACY

Our Semi-AVID-PR scheme $\Pi^\bigstar$ can be extended to hide the dispersed data from non-colluding honest-but-curious storage nodes, *i.e.*, formally, for each storage node, the distribution of the dispersed information is independent of the data received by the storage node. 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 $\boldsymbol{b} \xleftarrow{R} \mathbb{Z}_q^{L-1}$ to the right of $\tilde{\boldsymbol{U}}$ and then with a blinding row $\boldsymbol{s} \xleftarrow{R} \mathbb{Z}_q^k$

to the bottom of both $\tilde{\boldsymbol{U}}$ and $\boldsymbol{b}$, to obtain the $L \times k$ matrix $\boldsymbol{U}$,

$$\boldsymbol{U} \triangleq \begin{bmatrix} \tilde{\boldsymbol{U}} & \boldsymbol{b} \\ - \, \boldsymbol{s}^\top - \end{bmatrix}. \tag{48}$$

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 5 and Section 4. Thus, storage node m receives $(\boldsymbol{c}_m, h_1, ..., h_k)$ as part of the protocol (see Alg. 6).

Theorem 6.1. *The distribution of* $(\boldsymbol{c}_m, h_1, ..., h_k)$ *induced by the randomness in the blinding* $\boldsymbol{b}$ *and* $\boldsymbol{s}$ *is independent of* $\tilde{\boldsymbol{U}}$, *so that storage node* m *learns nothing about the dispersed information.*

Proof. 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 $\Pi^\bigstar$ 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 node m is related to the unknowns by the following equations:

$$\left[\begin{array}{c} [\boldsymbol{c}_m]_1 \\ \vdots \\ [\boldsymbol{c}_m]_{L-1} \\ \hline [\boldsymbol{c}_m]_L \\ \hline \log_g(h_1) \\ \vdots \\ \log_g(h_{k-1}) \\ \hline \log_g(h_k) \end{array}\right] = \underbrace{\left[\begin{array}{cccccc|ccc|ccc|c} \alpha_m^0 & \ldots & \alpha_m^{k-2} & & & & \alpha_m^{k-1} & 0 & \ldots & & & & \\ & & \ddots & & & & & \ddots & & & & & \\ & & & \alpha_m^0 & \ldots & \alpha_m^{k-2} & \ldots & 0 & \alpha_m^{k-1} & & & & \\ \hline & & & & & & & & & \alpha_m^0 & \ldots & \alpha_m^{k-2} & \alpha_m^{k-1} \\ \hline r^0 & 0 & \ldots & r^{L-2} & 0 & \ldots & & & & r^{L-1} & 0 & \ldots & \\ & \ddots & & \ddots & & \ddots & & & & & \ddots & & \\ \ldots & 0 & r^0 & \ldots & 0 & r^{L-2} & & & & \ldots & 0 & r^{L-1} & \\ \hline & & & & & & r^0 & \ldots & r^{L-2} & & & & r^{L-1} \end{array}\right]}_{\triangleq \boldsymbol{M} \in \mathbb{Z}_q^{(L+k)\times(Lk+(L-1)+(k-1)+1)}} \left[\begin{array}{c} [\tilde{\boldsymbol{u}}_1]_1 \\ \vdots \\ [\tilde{\boldsymbol{u}}_{k-1}]_1 \\ \vdots \\ [\tilde{\boldsymbol{u}}_1]_{L-1} \\ \vdots \\ [\tilde{\boldsymbol{u}}_{k-1}]_{L-1} \\ \hline \boldsymbol{b} \\ \hline \boldsymbol{s} \end{array}\right] \tag{49}$$

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

$$[\boldsymbol{M}]_{L+k}^\top = \sum_{i=1}^{L} r^{i-1} \alpha_m^{-(k-1)} [\boldsymbol{M}]_i^\top - \sum_{i=1}^{k-1} \alpha_m^{-(k-i)} [\boldsymbol{M}]_{L+i}^\top. \tag{50}$$

Thus, the last equation of the system is redundant. Striking it,

$$\left[\begin{array}{c} [\boldsymbol{c}_m]_1 \\ \vdots \\ [\boldsymbol{c}_m]_{L-1} \\ \hline [\boldsymbol{c}_m]_L \\ \hline \log_g(h_1) \\ \vdots \\ \log_g(h_{k-1}) \end{array}\right] = \left[\begin{array}{cccccc} \alpha_m^0 & \cdots & \alpha_m^{k-2} & & & \\ & & \ddots & & & \\ & & & \alpha_m^0 & \cdots & \alpha_m^{k-2} \\ \hline & & & & & \\ \hline r^0 & 0 & \cdots & r^{L-2} & 0 & \cdots \\ & \ddots & & \ddots & & \ddots \\ \cdots & 0 & r^0 & \cdots & 0 & r^{L-2} \end{array}\right] \left[\begin{array}{c} [\tilde{\boldsymbol{u}}_1]_1 \\ \vdots \\ [\tilde{\boldsymbol{u}}_{k-1}]_1 \\ \vdots \\ [\tilde{\boldsymbol{u}}_1]_{L-1} \\ \vdots \\ [\tilde{\boldsymbol{u}}_{k-1}]_{L-1} \end{array}\right]$$
$$+ \underbrace{\left[\begin{array}{ccc|ccc|c} \alpha_m^{k-1} & 0 & \cdots & & & & \\ & \ddots & & & & & \\ \cdots & 0 & \alpha_m^{k-1} & & & & \\ \hline & & & \alpha_m^0 & \cdots & \alpha_m^{k-2} & \alpha_m^{k-1} \\ \hline & & & r^{L-1} & 0 & \cdots & \\ & & & & \ddots & & \\ & & & \cdots & 0 & r^{L-1} & \end{array}\right]}_{\triangleq \boldsymbol{M}' \in \mathbb{Z}_q^{(L+k-1)\times(L-1+k)}} \left[\begin{array}{c} [\boldsymbol{b}]_1 \\ \vdots \\ [\boldsymbol{b}]_{L-1} \\ \hline [\boldsymbol{s}]_1 \\ \vdots \\ [\boldsymbol{s}]_{k-1} \\ \hline [\boldsymbol{s}]_k \end{array}\right]. \tag{51}$$

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

$$\Pr\Big(\tilde{\boldsymbol{U}} = \boldsymbol{y} \;\Big|\; (\boldsymbol{c}_m, h_1, ..., h_k) = \boldsymbol{x}\Big) = \Pr\Big(\tilde{\boldsymbol{U}} = \boldsymbol{y}\Big). \tag{52}$$

□

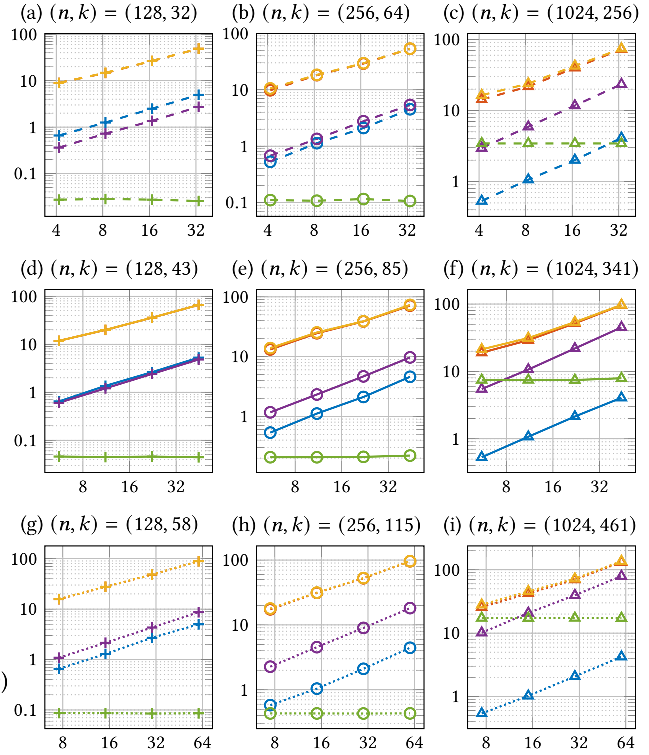


Figure 6: Single-thread runtime (ordinate, in seconds) of different steps of $\Pi^\bigstar$ on AMD Opteron 6378 processor for varying file sizes (abscissa, in 10^6 bytes; for varying L) in BLS12-381 curve. Rows: code rates $k/n \approx 0.25, 0.33, 0.45$ (- - - , ——, ·········). Columns: system sizes $n = 128, 256, 1024$ (+, ○, △). Steps of $\Pi^\bigstar$: Disperse: **Reed-Solomon (RS) encoding (🟦), computing vector commitments (🟧).** Retrieve: **Verifying downloaded chunks (🟨), RS decoding $k \times k$ matrix inversion (🟩), RS decoding matrix-matrix product (🟪). (Aggregated: Figure 9. Note that 🟨 lies on top of 🟧 in some plots.)**

7 EVALUATION

In this section, we show that the computational cost required for our Semi-AVID-PR scheme $\Pi^\bigstar$ is low, and the communication and storage requirements in comparison with AVID [7], AVID-FP [13], AVID-M [29] and ACeD [26] are among the best-of-class (tied with AVID-M) and practically low, while providing superior resilience (up to $t < n/2$ vs. $t < n/3$) and provable retrievability.

7.1 Computation

To evaluate the computational requirements imposed by computation and verification of vector commitments and encoding and decoding of the erasure-correcting code during Disperse (*i.e.*, computational burden to the Validium rollup operator) and Retrieve (*i.e.*, computational burden to the Validium rollup user, in case of malicious operator) of our Semi-AVID-PR scheme $\Pi^\bigstar$, we implemented a prototype in the Rust programming language using libraries from

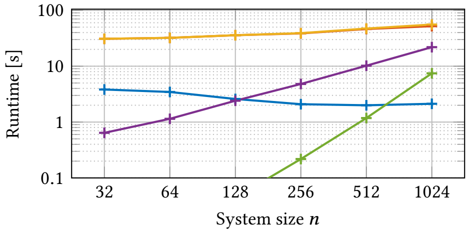


Figure 7: Single-thread runtime of steps of $\Pi^\bigstar$ on AMD Opteron 6378 processor for varying system size n in BLS12-381 curve. Fixed code rate $k/n \approx 0.33$, and file size ≈ 22 MB. Disperse: **RS encoding (🟦), computing vector commitments (🟧).** Retrieve: **Verifying downloaded chunks (🟨), RS decoding $k \times k$ matrix inversion (🟩), RS decoding matrix-matrix product (🟪). (*cf.* Figure 6)**

arkworks [2]. We make the source code available on Github.[2] We used KZG commitments [15] on the BLS12-381 curve [4, 6] as vector commitments, and Reed-Solomon codes [25] over the underlying scalar prime field. Reed-Solomon (RS) encoding was implemented using the fast Fourier transform (FFT). The consistency check of chunks performed by storage nodes in Alg. 6 line 8 was implemented naively, by computing a vector commitment of the received chunk and comparing it with the corresponding 'encoded' combination (computed using k exponentiations and multiplications in the group) of the column commitments $h_1, ..., h_k$. Note that during dispersal, these consistency checks are naturally parallelized across storage nodes, but have to be computed independently.

For retrieval, on the other hand, the computation of the coded chunks' commitments (Alg. 6 line 9) was sped up using the FFT. RS decoding was naive (generic for any MDS code): After downloading k valid chunks from distinct storage nodes, the retrieving client first inverts the corresponding $k \times k$ sub-matrix of the code's generator matrix $\boldsymbol{G}$, and then obtains the matrix $\boldsymbol{U}$ of uncoded chunks from the downloaded sub-matrix of the coded chunks $\boldsymbol{C}$ by way of a matrix-matrix product with the inverse. This way, the cubic complexity (in k) of naive matrix inversion via Gaussian elimination is amortized over the decoding of L rows (each with quadratic complexity in k; in the regime of interest $k \leq L$). Note that this naive approach to RS decoding is permissible as our experiments indicate that Retrieve's runtime is bottlenecked by another step (verifying downloaded chunks), and during normal operation (when the rollup operator is honest) only Disperse is invoked. Note that a systematic erasure-correcting code can be used to further speed up decoding in the realistic scenario where few storage nodes are corrupted (as long as the privacy techniques of Section 6 are not used).

Figure 6 shows the single-thread runtime (ordinate) on an AMD Opteron 6378 processor of the different steps of $\Pi^\bigstar$ for varying file sizes (abscissa), system sizes (columns), and code rates (rows). Figure 9 shows the measurements aggregated on the level of Disperse and Retrieve, respectively. The plots reveal a minor slowdown of Disperse with increasing system size and code rate. The runtime of Disperse is dominated by computing the vector commitments. The runtime of Retrieve is dominated by verifying downloaded chunks.

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
Repetition	$441 = 0.49n$	27 GB	27 GB	✔
AVID [7]	$297 = 0.33n$	104 GB	116 MB	✔
AVID-FP [13]	$297 = 0.33n$	31 GB	125 MB	✔
AVID-M [29]	$297 = 0.33n$	116 MB	90 MB	✘
ACeD [26]	$297 = 0.33n$	787 MB	787 MB	✘
ACeD [26]	$441 = 0.49n$	13 GB	13 GB	✘
This work ($\Pi^\bigstar$)	$297 = 0.33n$	99 MB	99 MB	✔
This work ($\Pi^\bigstar$)	$441 = 0.49n$	1.5 GB	1.5 GB	✔

Recall that both bottlenecking steps have been optimized using the FFT. Naive RS decoding does not introduce a performance bottleneck, but becomes relevant for large systems ($n = 1024$). Fixing file size to ≈ 22 MB and code rate to $k/n \approx 0.33$ while varying system size n (*cf.* Figure 7), corroborates the earlier observations.

Concretely, the client computation for dispersing a file of 22 MB among 256 storage nodes, up to 85 of which may be adversarial, requires ≈ 41 s of single-thread runtime on an AMD Opteron 6378 processor when using the BLS12-381 curve. The corresponding retrieval takes ≈ 44 s of single-thread runtime. The dispersal throughput of ≈ 0.54 MB/s corresponds to ≈ 2,700 tx/s (assuming 200 B transaction size). It should be noted that we report single-thread runtime on a seven year old processor here. The workload is embarrassingly parallel and hence wall-clock time reduces trivially with an increasing number of parallel workers. With 16 threads, the Validium operator can complete dispersal and Validium users can complete retrieval in less than 3 s, respectively.

7.2 Communication & Storage

Communication and storage required for different data availability solutions are tabulated for a numerical example in Table 1. The calculations are provided in Appendix A. Note the $t < n/2$ resilience upper bound for any scheme that simultaneously provides *availability* and *correctness* (*cf.* Definition 3.2). To see this, suppose the cooperation of q storage nodes is necessary and sufficient to complete a dispersal. Correctness requires $q \leq n - t$ (else adversarial storage nodes can 'block' dispersal), and availability requires $q > t$ (else adversarial storage nodes can 'forge' a dispersal). Combining the two conditions yields $t < n/2$, which is achieved by the naive repetition (full replication) scheme, at high communication and storage cost. Our $\Pi^\bigstar$ recovers the same trade-off when parameterized for resilience close to $n/2$; but our scheme can also be parameterized for lower resilience, in which case it achieves considerably lower communication and storage, whereas the repetition scheme does not allow for such parameterization. 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 allows for a trade-off of communication and storage with adversarial resilience.

[2] Source code: https://github.com/tse-group/semiavidpr-experiments

In terms of communication and storage, our Semi-AVID-PR scheme $\Pi^\bigstar$ (Sec. 4) 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). Our Semi-AVID-PR scheme outperforms ACeD in communication and storage by at least 7×. The net data throughput of $\approx 0.54\,\mathrm{MB/s}$ for $(n, k) = (256, 85)$ corresponding to $\approx$ 2,700 tx/s (*cf.* Section 7.1) entails $\approx 1.7\,\mathrm{MB/s}$ communication bandwidth usage, which is feasible even via consumer-grade Internet connectivity. Finally, it should be noted that the VID-based schemes in Table 1 have resilience t at most $t < n/3$. In that regime, $\Pi^\bigstar$ matches or exceeds the communication- and storage-efficiency of VID-based schemes. However, like ACeD, $\Pi^\bigstar$ also supports higher resilience up to $t < n/2$. In this regime, the overhead from erasure coding increases, as for ACeD, but still outperforms ACeD.

8 APPLICATION TO DATA AVAILABILITY SAMPLING

In common blockchain designs every block consists of a meta data header and transaction content. Nodes download the full chain and validate all transactions. However, a resource-limited node can instead participate as a *light node*.[3] Then, it only processes block headers. If a block contained an invalid transaction, it would be rejected by full nodes but its header would be accepted by a light node unable to inspect the block content and verify transaction validity. To prevent this, full nodes can produce an *invalid transaction fraud proof* [1]. To take full nodes' ability to issue such fraud proofs, a malicious block producer can withhold parts of the block content. Full nodes would then reject the block until its content is fully available, but light nodes would not notice the missing content. The absence of an invalid transaction fraud proof can thus mean two things: either the block is valid, or full nodes are unable to verify the block due to missing data. To rule out the second possibility, data availability sampling schemes for light nodes were introduced.

Data availability schemes using Reed-Solomon codes were proposed in [1], where the block producer encodes the k chunks block content with a $(2k, k)$ Reed-Solomon (RS) code. Light nodes randomly query 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 and any remaining missing chunks can be recovered using the RS code. It is therefore no longer possible to trick light nodes into accepting a block while withholding data to prevent invalid transaction fraud proofs. However, a malicious block producer could invalidly encode the block. Decoding would then not consistently recover the original chunks' data. Full nodes can detect invalid encoding and issue a fraud proof for light nodes. But, the size of such proofs in this scheme is commensurate to the block content size—defying the idea of light nodes downloading less than the full block. Subsequent works [1, 23, 24, 31] focussed on reducing the fraud proof size, but drawbacks remain (*e.g.*, complexity, timing assumptions).

A different approach is to make it impossible for block producers to invalidly encode data. Such schemes can be achieved using polynomial commitments [15], where the block is interpreted as a low-degree polynomial, the commitment to which is included in the block header and gets opened at locations randomly sampled by light clients. This effectively enforces valid RS encoding. Schemes of this flavor however require to compute an evaluation witness for each query, which despite recent algorithmic improvements is still computationally heavy [10, 27, 28].

Algorithm $\Pi^\bigstar$.Commit of our Semi-AVID-PR scheme is suitable for the application at hand. It can commit to a block B such that: (a) The commitment can be opened to *chunks*, but only of a valid RS encoding of B. Computing and verifying these openings is practically efficient. *This enables data availability sampling.* (b) The commitment can be opened to *entries* of the original block B. The openings are short and can be produced and verified practically efficiently. *This enables the invalid transaction fraud proofs of [1].*

Let $\boldsymbol{U} \equiv \mathrm{AsMatrix}_{L\times k}(B)$ with columns $\boldsymbol{u}_1, ..., \boldsymbol{u}_k$. An opening $(\boldsymbol{c}_i, i, (h_1, ..., h_k))$ to chunk i of an RS encoding of B is computed as:

$$\boldsymbol{c}_i \leftarrow [\mathsf{Code.Encode}^{\otimes L}(\boldsymbol{U})]_i \quad (h_1, ..., h_k) \leftarrow \mathsf{VC}^{\otimes k}(\boldsymbol{U}) \quad (53)$$

An opening $(\boldsymbol{c}, i, (h_1, ..., h_k))$ to chunk i of an RS encoding of a block with commitment C is verified as:

$$C \overset{?}{=} \mathsf{CRHF}^s(h_1 \| ... \| h_k) \wedge [\mathsf{Code.Encode}(h_1, ..., h_k)]_i \overset{?}{=} \mathsf{VC}(\boldsymbol{c}) \quad (54)$$

An opening $([\boldsymbol{u}_j]_i, i, j, (h_1, ..., h_k), w)$ to the entry at (i, j) of the matrix $\boldsymbol{U}$ corresponding to B is computed as:

$$(h_1, ..., h_k) \leftarrow \mathsf{VC}^{\otimes k}(\boldsymbol{U}) \quad w \leftarrow \mathsf{VC.OpenEntry}(\mathsf{pp}, \boldsymbol{u}_j, i) \quad (55)$$

An opening $(y, i, j, (h_1, ..., h_k), w)$ to entry $[\boldsymbol{u}_j]_i$ of a block with commitment C is verified as:

$$C \overset{?}{=} \mathsf{CRHF}^s(h_1 \| ... \| h_k) \wedge \mathsf{VC.VerifyEntry}(\mathsf{pp}, h_j, i, y, w) \overset{?}{=} \mathtt{true} \quad (56)$$

For more details on the application of $\Pi^\bigstar$ to data availability sampling see Appendix E.

ACKNOWLEDGMENT

The authors thank Mohammad Ali Maddah-Ali and Dionysis Zindros for fruitful discussions. KN is supported by a gift from IOG (Input-Output Global) and a grant from the National Science Foundation Center for Science of Information. JN is supported by the Reed-Hodgson Stanford Graduate Fellowship.

REFERENCES

[1] Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, and Ismail Khoffi. 2021. Fraud and Data Availability Proofs: Detecting Invalid Blocks in Light Clients. In *Financial Cryptography (2) (Lecture Notes in Computer Science, Vol. 12675)*. Springer, 279–298.

[2] arkworks contributors. 2022. `arkworks` *zkSNARK ecosystem.* https://arkworks.rs

[3] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia C. Fanti, and Pramod Viswanath. 2019. 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*. ACM, 585–602. https://doi.org/10.1145/3319535.3363213

[4] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. 2002. Constructing Elliptic Curves with Prescribed Embedding Degrees. In *SCN (Lecture Notes in Computer Science, Vol. 2576)*. Springer, 257–267.

[5] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.* (2018), 46. http://eprint.iacr.org/2018/046

[6] Sean Bowe. 2017. BLS12-381: New zk-SNARK Elliptic Curve Construction. (2017). https://electriccoin.co/blog/new-snark-curve/

[3] Light nodes also occur in the context of sharding, where each node is assigned to a shard and behaves in-shard as a full node and out-of-shard as a light node.

[7] Christian Cachin and Stefano Tessaro. 2005. 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)*. Springer, 503–504. https://doi.org/10.1007/11561927_42

[8] Dario Catalano and Dario Fiore. 2013. 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)*. Springer, 55–72. https://doi.org/10.1007/978-3-642-36362-7_5

[9] Christian Decker and Roger Wattenhofer. 2015. 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)*. Springer, 3–18.

[10] Dankrad Feist and Dmitry Khovratovich. [n.d.]. Fast Amortized Kate Proofs. ([n. d.]). https://github.com/khovratovich/Kate/blob/master/Kate_amortized.pdf

[11] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. 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)*. Springer, 626–645. https://doi.org/10.1007/978-3-642-38348-9_37

[12] Alex Gluchowski. 2020. zkRollup vs. Validium. (2020). https://medium.com/matter-labs/zkrollup-vs-validium-starkex-5614e38bc263

[13] James Hendricks, Gregory R. Ganger, and Michael K. Reiter. 2007. 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*. ACM, 139–146. https://doi.org/10.1145/1281100.1281122

[14] Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. 2018. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. USENIX Association, 1353–1370. https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner

[15] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. 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)*. Springer, 177–194. https://doi.org/10.1007/978-3-642-17373-8_11

[16] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography, Second Edition*. CRC Press.

[17] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. 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*. IEEE Computer Society, 583–598. https://doi.org/10.1109/SP.2018.000-5

[18] Songze Li, Mingchao Yu, Chien-Sheng Yang, Amir Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. 2020. 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*. IEEE, 203–208. https://doi.org/10.1109/ISIT44484.2020.9174305

[19] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited. In *PODC*. ACM, 129–138.

[20] Patrick McCorry, Chris Buckland, Bennet Yee, and Dawn Song. 2021. SoK: Validating Bridges as a Scaling Solution for Blockchains. *IACR Cryptol. ePrint Arch.* (2021), 1589.

[21] Ralph C. Merkle. 1987. 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)*. Springer, 369–378. https://doi.org/10.1007/3-540-48184-2_32

[22] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. 2019. 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)*. Springer, 508–526. https://doi.org/10.1007/978-3-030-32101-7_30

[23] Debarnab Mitra, Lev Tauz, and Lara Dolecek. 2020. 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*. IEEE, 1–5. https://doi.org/10.1109/ITW46852.2021.9457630

[24] Debarnab Mitra, Lev Tauz, and Lara Dolecek. 2021. Overcoming Data Availability Attacks in Blockchain Systems: LDPC Code Design for Coded Merkle Tree. *CoRR* abs/2108.13332 (2021). arXiv:2108.13332 https://arxiv.org/abs/2108.13332

[25] I. Reed and G. Solomon. 1960. Polynomial Codes Over Certain Finite Fields. *Journal of The Society for Industrial and Applied Mathematics* 8 (1960), 300–304.

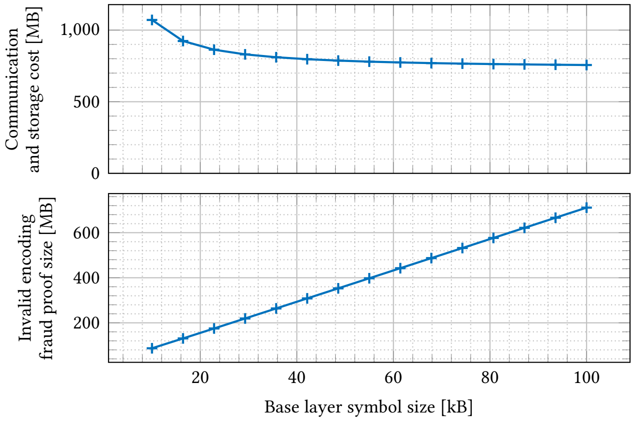


Figure 8: Communication and storage cost (top) and invalid encoding fraud proof size (bottom) as a function of the base layer symbol size c, when dispersing a file of size 30 MB among 900 nodes using ACeD [26] with resilience $t = 0.33n$.

[26] Peiyao Sheng, Bowen Xue, Sreeram Kannan, and Pramod Viswanath. 2021. ACeD: Scalable Data Availability Oracle. In *Financial Cryptography (2) (Lecture Notes in Computer Science, Vol. 12675)*. Springer, 299–318.

[27] Alin Tomescu. 2020. How to compute all Pointproofs. *IACR Cryptol. ePrint Arch.* (2020), 1516. https://eprint.iacr.org/2020/1516

[28] Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. 2020. Aggregatable Subvector Commitments for Stateless Cryptocurrencies. In *SCN (Lecture Notes in Computer Science, Vol. 12238)*. Springer, 45–64.

[29] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. 2022. DispersedLedger: High-Throughput Byzantine Consensus on Variable Bandwidth Networks. (2022).

[30] Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. 2020. OHIE: Blockchain Scaling Made Simple. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 90–105. https://doi.org/10.1109/SP40000.2020.00008

[31] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. 2020. 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)*. Springer, 114–134. https://doi.org/10.1007/978-3-030-51280-4_8

A 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 or signature is $H = 32\,\text{B}$. Given adversarial resilience t, we choose $k \triangleq n - 2t$.

- Repetition (uncoded) scheme:

$$C = S = n|F| \tag{57}$$

- AVID:

$$C = \left(\frac{|F|}{k} + nH\right)\left(n + n^2\right) \tag{58}$$

$$S = n\left(\frac{|F|}{k} + nH\right) \tag{59}$$

- AVID-FP:

$$C = n\left(\frac{|F|}{k} + (n+k)H\right) + n^2(n+k)H \tag{60}$$

Algorithm 12 Existential forgery game (EFG) against $\mathsf{Sig} = (\mathsf{Sig.KeyGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$

```
$\mathcal{M} \leftarrow \emptyset$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$
function $O^{\mathsf{sign}}(\mathrm{m})$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{m\}$
    return $\mathsf{Sig.Sign}(\mathsf{sk}, m)$
$(m, \sigma) \leftarrow \mathcal{A}_{\mathsf{EFG}}^{O^{\mathsf{sign}}(.)}(\mathsf{pk})$
return $m \notin \mathcal{M} \land \mathsf{Sig.Verify}(\mathsf{pk}, m, \sigma) = \mathtt{true}$
```

Algorithm 13 Collision finding game (CFG) against $\mathsf{HF} = (\mathsf{HF.Gen}, \mathsf{HF.H})$

```
$s \leftarrow \mathsf{HF.Gen}(1^\lambda)$
$(x, x') \leftarrow \mathcal{A}_{\mathsf{CFG}}(s)$
return $x \neq x' \land \mathsf{CRHF}^s(x) = \mathsf{CRHF}^s(x')$
```

Algorithm 14 Binding game (VCBG) against $\mathsf{LVC} = (\mathsf{LVC.Setup}, \mathsf{LVC.Commit})$

```
$\mathsf{pp} \leftarrow \mathsf{LVC.Setup}(1^\lambda)$
$(\boldsymbol{v}, \boldsymbol{v}') \leftarrow \mathcal{A}_{\mathsf{VCBG}}(\mathsf{pp})$
return $\boldsymbol{v} \neq \boldsymbol{v}' \land \mathsf{LVC.Commit}(\boldsymbol{v}) = \mathsf{LVC.Commit}(\boldsymbol{v}')$
```

$$S = n\left(\frac{|F|}{k} + (n+k)H\right) \tag{61}$$

- AVID-M:

$$C = n\left(\frac{|F|}{k} + (1 + \log_2 n)H\right) + n^2 H \tag{62}$$

$$S = n\left(\frac{|F|}{k} + (1 + \log_2 n)H\right) \tag{63}$$

- ACeD:

$$C = S = n\left(t'H + \frac{|F|}{nr\lambda} + \frac{(2q-1)|F|H}{nrc\lambda}\log_{qr}\frac{|F|}{ct'r}\right) \tag{64}$$

  Parameters:

$$t' = 16 \qquad r = 0.25 \qquad q = 8 \qquad d = 8 \tag{65}$$

$$c = 48\,\mathrm{kB} \qquad \eta = 0.875 \qquad \lambda = \frac{1 - 2t/n}{\ln\left(\frac{1}{1-\eta}\right)} \tag{66}$$

  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:

$$C = n\left(\frac{|F|}{k} + kH + H\right) \tag{67}$$

$$S = n\left(\frac{|F|}{k} + kH\right) \tag{68}$$

B PRELIMINARIES

Definition B.1 (Hash Function). A hash function is a pair of PPT algorithms $(\mathsf{Gen}, \mathsf{H})$ [16], such that:

- $\mathsf{Gen}\colon 1^\lambda \mapsto s$: takes as input a security parameter λ and outputs a randomly sampled key s,
- $\mathsf{H}\colon (s, x) \mapsto h$: takes as input a key s and a string $x \in \{0,1\}^*$ and outputs a string $h \in \{0,1\}^\lambda$.

Definition B.2 (Digital Signature Scheme). A digital signature scheme $\mathsf{Sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ [16] consists of three PPT algorithms, such that:

- $\mathsf{KeyGen}\colon 1^\lambda \mapsto (\mathsf{pk}, \mathsf{sk})$: takes as input a security parameter λ and outputs a public key pk and a secret key sk,
- $\mathsf{Sign}\colon (\mathsf{sk}, m) \mapsto \sigma$: takes as input a secret key sk and a message m and outputs a signature σ,
- $\mathsf{Verify}\colon (\mathsf{pk}, m, \sigma) \mapsto b \in \{\mathtt{true}, \mathtt{false}\}$: takes as input a public key pk, a message m and a signature σ and outputs a boolean b indicating whether the signature is valid.

Definition B.3 (Deterministic Vector Commitment Scheme). A deterministic vector commitment scheme $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{OpenEntry}, \mathsf{VerifyEntry})$ [8, 15] consists of four PPT algorithms, such that:

- $\mathsf{Setup}\colon 1^\lambda \mapsto \mathsf{pp}$: takes as input a security parameter λ and outputs some public parameters pp,
- $\mathsf{Commit}\colon (\mathsf{pp}, \boldsymbol{v}) \mapsto C$: takes as input the public parameters pp and a vector $\boldsymbol{v}$ and outputs a commitment C,
- $\mathsf{OpenEntry}\colon (\mathsf{pp}, \boldsymbol{v}, i) \mapsto \pi_i$: takes as input the public parameters pp, a vector $\boldsymbol{v}$, a position i and returns a proof π_i attesting to the fact that $[\boldsymbol{v}]_i$ is the i-th entry of $\boldsymbol{v}$,
- $\mathsf{VerifyEntry}\colon (\mathsf{pp}, C, i, y, \pi) \mapsto b \in \{\mathtt{true}, \mathtt{false}\}$: takes as input the public parameters pp, a commitment C, a position i, a value y, and an opening proof π, and returns a boolean b indicating whether π is a proof attesting to the fact that C is a commitment to a vector $\boldsymbol{v}$ such that $[\boldsymbol{v}]_i = y$.

C ADDITIONAL EVALUATION PLOTS

The runtime measurements for different steps of $\Pi^\bigstar$ shown in Figure 6 are aggregated for Disperse and Retrieve in Figure 9.

Figures 6 and 9 use curve BLS12-381. Corresponding plots for curve BN254 are provided in Figures 10 and 11. The plots show a slight speedup, due to faster operations in BN254.

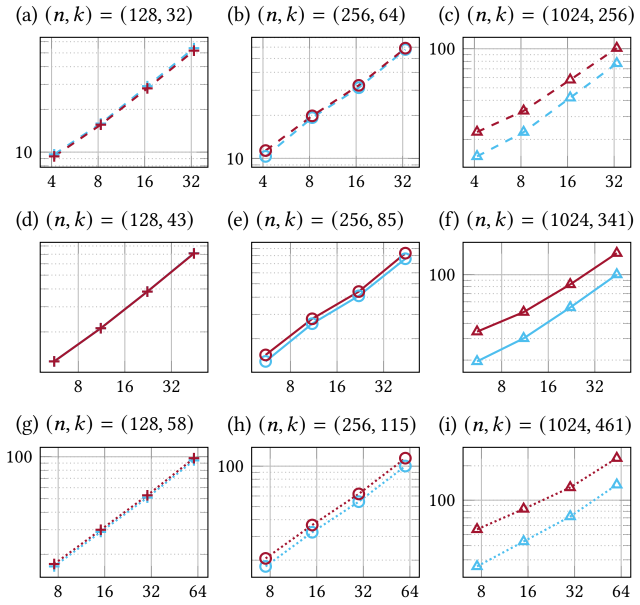
D PROOF DETAILS

We modify the availability game AvG (Alg. 2) to obtain AvG′ (Alg. 15) in which initially the index I of a storage node is sampled uniformly at random, and subsequently the game is aborted if the adversary $\mathcal{A}_{\mathsf{AvG}'}$ attempts to corrupt I.

E DETAILS ON 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*.[4] 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

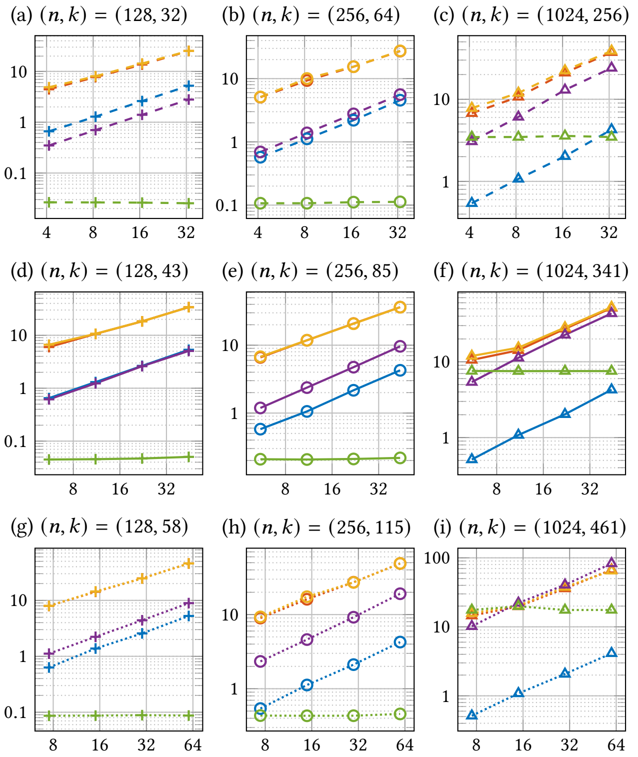
[4] Light nodes also occur in the context of sharding, where each node is assigned to a shard and behaves in-shard (*i.e.*, towards their assigned shard) as a full node and out-of-shard (*i.e.*, towards other shards) as a light node.


(a) $(n, k) = (128, 32)$ (b) $(n, k) = (256, 64)$ (c) $(n, k) = (1024, 256)$

(d) $(n, k) = (128, 43)$ (e) $(n, k) = (256, 85)$ (f) $(n, k) = (1024, 341)$

(g) $(n, k) = (128, 58)$ (h) $(n, k) = (256, 115)$ (i) $(n, k) = (1024, 461)$

Figure 9: Single-thread runtime (ordinate, in seconds) of Disperse (■) and Retrieve (■) on AMD Opteron 6378 processor for varying file sizes (abscissa, in 10^6 bytes; for varying L) in BLS12-381 curve. Rows: code rates $k/n \approx 0.25, 0.33, 0.45$ (- - - , ——, ·········). Columns: system sizes $n = 128, 256, 1024$ (+, ○, △). (Disaggregated: Figure 6)


(a) $(n, k) = (128, 32)$ (b) $(n, k) = (256, 64)$ (c) $(n, k) = (1024, 256)$

(d) $(n, k) = (128, 43)$ (e) $(n, k) = (256, 85)$ (f) $(n, k) = (1024, 341)$

(g) $(n, k) = (128, 58)$ (h) $(n, k) = (256, 115)$ (i) $(n, k) = (1024, 461)$

Figure 10: Single-thread runtime (ordinate, in seconds) of different steps of $\Pi^{\bigstar}$ on AMD Opteron 6378 processor for varying file sizes (abscissa, in 10^6 bytes; for varying L) in BN254 curve. Rows: code rates $k/n \approx 0.25, 0.33, 0.45$ (- - - , ——, ·········). Columns: system sizes $n = 128, 256, 1024$ (+, ○, △). Steps of $\Pi^{\bigstar}$: Disperse: Reed-Solomon (RS) encoding (■), computing vector commitments (■). Retrieve: Verifying downloaded chunks (■), RS decoding $k \times k$ matrix inversion (■), RS decoding matrix-matrix product (■). (Aggregated: Figure 11. Note that ■ lies on top of ■ in some plots.)

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 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

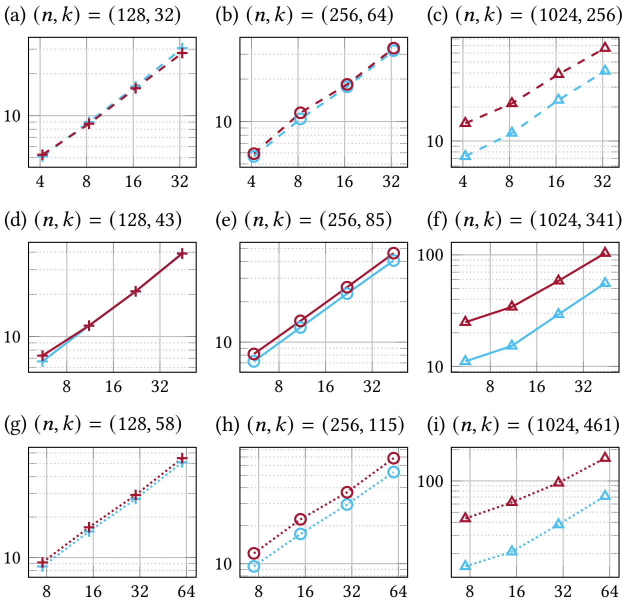


Figure 11: Single-thread runtime (ordinate, in seconds) of Disperse (🟦) **and** Retrieve (🟥) **on AMD Opteron 6378 processor for varying file sizes (abscissa, in 10^6 bytes; for varying L) in BN254 curve. Rows: code rates $k/n \approx 0.25, 0.33, 0.45$ (- - - , ——, ·········). Columns: system sizes $n = 128, 256, 1024$ (+, ○, △). (Disaggregated: Figure 10)**

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, 23, 24, 31] 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 [15]. 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 [10, 27, 28].

Algorithm 15 Modified availability game (AvG′) with resilience t against Semi-AVID-PR scheme Π_{SAVIDPR} = (Setup, Commit, Disperse, Verify, Retrieve)

```
$I \xleftarrow{R} [n]$
$C \leftarrow \emptyset$    ▷ Bookkeeping of corrupted parties
$\forall i \in [n] : P_i \leftarrow$ new $\Pi_{\mathsf{SAVIDPR}}(\emptyset)$ ▷ Instantiate $P_i$ as $\Pi_{\mathsf{SAVIDPR}}$ with blank state
$\mathsf{pp} \leftarrow \mathsf{Setup}^{P_1,\ldots,P_n}(1^\lambda)$    ▷ Run setup among all parties
function $O^{\mathrm{corrupt}}(i)$    ▷ Oracle for $\mathcal{A}$ to corrupt parties
    assert $i \notin C$
    if $i \neq I$
        $C \leftarrow C \cup \{i\}$    ▷ Mark party as corrupted
        return $P_i$    ▷ Hand $P_i$'s state to $\mathcal{A}$
    else
        abort
function $O^{\mathrm{interact}}(i, m)$ ▷ Oracle for $\mathcal{A}$ to interact with parties
    assert $i \notin C$
    return $P_i(m)$  ▷ Execute $P_i$ on input $m$, return output to $\mathcal{A}$
$\left(P, C, \left(O_i^{\mathrm{node}}(.)\right)_{i\in C}\right) \leftarrow \mathcal{A}_{\mathrm{AvG'}}^{O^{\mathrm{corrupt}}(.),O^{\mathrm{interact}}(.)}(\mathsf{pp})$    ▷ $\mathcal{A}$ returns certificate of retrievability $P$, commitment $C$, and oracle access to corrupted nodes for retrieval
$\hat{B} \leftarrow \mathsf{Retrieve}^{P_1,\ldots,P_n}\left[O_i^{\mathrm{node}}(.)/\textsc{Query}(i,.)\right]_{i\in C}(P, C)$    ▷ During retrieval, interact with corrupted nodes through oracles
return $|C| \le t$    ▷ $\mathcal{A}$ wins iff: while corrupting no more than $t$ parties, $\mathcal{A}$ produces a valid certificate of retrievability $P$ for $C$ such that retrieval does not return a file matching $C$
    $\wedge\ \mathsf{Verify}(P, C) = \mathsf{true}$
    $\wedge\ \mathsf{Commit}(\hat{B}) \neq C$
```

The Semi-AVID-PR scheme, described in Section 4 and illustrated in Figure 5, 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 5, 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 $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ of that matrix using the linear vector commitment scheme defined in Section 2.3 (to obtain commitments $h_1, \ldots, h_k$), and encodes the matrix $\boldsymbol{U}$ row-wise using a (n, k) Reed-Solomon code to obtain chunks $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n$ of a coded matrix $\boldsymbol{C}$. A final commitment to the full block content is computed as $C \triangleq \mathsf{CRHF}^s(h_1 \| \ldots \| h_k)$ 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 $h_1, \ldots, h_k$. 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 7.