Adaptively Secure Blockchain-Aided Decentralized Storage Networks: Formalization and Generic Construction

Xiangyu Su[1⋆], Yuma Tamagawa[1], Mario Larangeira[1,2], Zengpeng Li[3], and Keisuke Tanaka[1]

[1] Institute of Science Tokyo. su.x.4029@m, tamagawa.y.f6c3@m, rebello.m.f72a@m, keisuke@comp.isct.ac.jp.
[2] Input Output, Singapore. mario.larangeira@iohk.io.
[3] Shandong University. zengpengliz@gmail.com.

Abstract. Blockchain-aided decentralized storage networks (DSNs) use public audits to verify outsourced storage and smart contracts to govern payments and collateral. Published audit evidence, however, remains available to servers and may let them reduce local storage while continuing to pass audits. Retrieval must also remain fair across adaptively selected sessions. We formalize DSN protocols across delegation, preservation, and retrieval, accounting for evolving server state and externally available proofs. To support adaptive replication soundness, we reformulate rational replication for proofs of replication (PoRep) as an extractor-based notion. We then give a generic construction combining PoRep, verifiable delay functions, and simulation-extractable zkSNARKs, with encrypted delivery and conditional authorization for retrieval. Our analysis identifies roles for succinctness and simulation-extractability beyond privacy and efficiency. Under the knowledge of strong compression assumption and the stated primitive and timing assumptions, we establish adaptive replication soundness subject to bounds on extraction resources and the cumulative public transcript. We also establish contract unforgeability and retrieval fairness. Controlled experiments illustrate storage offloading through public audit evidence, while a separate Merkle-PLONK prototype measures proving and verification costs through Cardano's Plutus interface.

1 Introduction

Decentralized storage networks (DSNs), *e.g.*, [7,38,33,35,36,24], allow clients to outsource data to untrusted servers. Servers maintain replicas of the delegated data and may demonstrate continued storage through periodic audits during an agreed contract period. DSNs often adopt a blockchain protocol that provides a public ledger for recording storage agreements and audit evidence, with smart contracts enforcing payment and collateral rules.

⋆ This work was supported by Input Output Global.

Proofs of replication (PoRep) [18,19,12] and proofs of space-time (PoST) [31] provide cryptographic foundations for these storage guarantees. However, their use within a DSN introduces a distinction between the server's retained local state and the public information available to it. Audit responses published on the ledger remain accessible after verification. A stateful server may reuse these responses when answering later challenges, adapting its local state as the public transcript grows. Consequently, a storage guarantee for repeated public audits must account for both sources of information, rather than treating each audit in isolation.

Retrieval introduces a separate protocol-level requirement. The client must recover the delegated data, while the server must obtain the collateral release authorized by the exchange. These outcomes depend not only on a proof of correct delivery, but also on which transaction the client authorizes, when the server completes it, and what the ledger accepts. Fair data exchange [37] provides a starting point, but its integration with storage contracts must account for delegation, repeated audits, and other retrieval sessions.

We ask: *how can a blockchain-aided DSN provide formal guarantees for locally retained storage under repeated public audits and for fair data retrieval?*

1.1 Contributions and Technical Overview

We formalize these requirements and provide a generic DSN construction. Our security analysis makes explicit the ledger, timing, and public storage conditions needed beyond the underlying primitive guarantees.

A blockchain-aided DSN formalization (Sec. 3). To our knowledge, no prior work provides a unified DSN formalization that captures adaptive storage guarantees, retrieval fairness, and ledger-enforced authorization and settlement.

We formalize delegation, preservation, and retrieval through explicit transactions and timing conditions. The extended unspent transaction output (EUTxO) model [10] supplies our bookkeeping notation; account-based smart contracts [39] can implement the same authenticated state transitions and atomic value transfers. Our definitions capture *correctness*, requiring successful honest execution; *contract unforgeability*, preventing unauthorized contract establishment; and *adaptive replication soundness*, requiring data recovery and local storage lower bounds across adaptive audits. For retrieval, *adaptive client fairness* links accepted finalization to data recovery despite adaptive session selection and access to proofs from other retrievals, and *server fairness* requires pre-finalization messages to reveal no additional replica information. Together, these notions capture atomicity: either the client can recover the data and the server obtains collateral release, or neither gains its corresponding benefit from the exchange.

Extractor-based replication soundness (Sec. 2.2 and Appendix B). To support adaptive replication soundness, we reformulate rational replication for PoRep [18] as an extractor-based notion. Rational replication compares a prover with a "well-behaved" one that stores separately recoverable replicas. Our notion instead concerns the prover's own retained state. The extractor recovers

replicas and produces residual information and a decoder for the prover's auxiliary output, counting both as storage. This formulation extends naturally to the adaptive setting: extraction and local storage requirements are imposed at successive checkpoints preceding accepted audits. The adaptive definition accounts for the available public transcript and allows the adversary to update its state and access permitted proofs for other delegations.

For the underlying PoRep notion, we establish sufficient conditions from the implicit proofs of retrievable commitment (PoRC) and proofs of space (PoSpace) under the knowledge of strong compression (KoSC) assumption. The extracted representation then yields the well-behaved adversary required by the original rational replication definition.

A generic DSN construction (Sec. 4). We combine PoRep with a verifiable delay function (VDF) [8] to link successive storage audits, and replace public PoRep responses with succinct proofs of their validity using a simulation-extractable zkSNARK [26,16]. Existing DSNs employ zkSNARKs for privacy and efficiency [33,40]; our analysis identifies their additional role in adaptive storage security. Succinctness bounds how much information each published proof adds to the server's external storage, while simulation-extractability ensures that accepted target proofs yield underlying PoRep responses despite access to proofs for other delegations. Together with zero-knowledge, these properties support adaptive replication soundness under the stated timing, extraction resource, and cumulative transcript bounds.

For retrieval, the server supplies an encrypted replica and a proof of correct encryption under a committed ephemeral key. The client conditionally authorizes finalization; an accepted finalization transaction releases the server's collateral and discloses the key needed for recovery. This mechanism adapts that of [37] and draws on the generic adaptor signature construction of [11]. Appendix D further discusses on-demand retrieval and the use of chunked retrieval as in [33].

Controlled experiments and concrete costs (Sec. 5 and Appendix E). We complement the formal analysis with controlled experiments illustrating storage offloading through public audit evidence. Using Merkle and Schnorr implementations alongside the simulation-extractable zkSNARK GM17 [26], we examine reuse of published responses, short nonces chosen by the prover, and subliminal channels in proofs. The results illustrate why zero-knowledge of honestly generated proofs and individual proof succinctness do not replace cumulative transcript accounting.

To assess the cost of the succinct audit layer, we further implement a Merkle-PLONK [21] prototype and measure proving costs across replica sizes and challenge workloads, together with verification costs through the Cardano Plutus interface [34]. These measurements quantify audit layer overhead, not end-to-end DSN scalability.

1.2 Related Work

We review related work on storage proofs, fair exchange, and DSN constructions.

Storage, space, replication, and storage over time. Storage proofs certify data recovery, storage use, or both. Proofs of retrievability [27] allow data extraction from a prover that passes audits with sufficient probability. PoSpace [14] instead lower-bound the storage needed under bounded computation, without requiring storage of particular client data. PoRep combines data recovery with storage guarantees for multiple replicas of designated data. The framework of [18,19] uses computational timing assumptions and defines rational replication by comparing a prover with a well-behaved one. A different construction [12] avoids timing assumptions through randomized, trapdoor-based encodings that are publicly decodable but resist efficient recomputation. PoST [31] captures the tradeoff between computation and storage maintained over time. Proofs of *storage*-time [3] instead formalize continuous data availability through extraction from prover configurations. Our focus is adaptive replication soundness that distinguishes locally retained storage from evolving public audit records and accessible proofs from other delegations.

Fair exchange. Fair exchange protocols aim to make delivery and payment atomic. FairSwap [13] and OptiSwap [15] use smart contracts to resolve disputes in exchanges of digital goods for payment. The protocol of [37] instead uses verifiable encryption under a committed key to link data recovery to payment through key disclosure. Our setting additionally binds adaptively selected retrieval sessions to the delegated instance and its current ledger state.

DSN constructions. Filecoin [33] deploys PoRep and repeated storage proofs within a blockchain-based storage market. On the storage side, FileDES [40] combines encrypted replicas with succinct storage proofs and batched verification. On the retrieval side, PoUDR [29] uses zkSNARKs and batching to support fair exchange between data providers and clients. Our work treats these components jointly, formalizing adaptive local storage guarantees alongside ledger-enforced authorization and retrieval fairness.

2 Preliminaries

Notations. Let $\lambda \in \mathbb{N}$ be the security parameter, $\mathsf{poly}(\cdot)$ and $\mathsf{negl}(\cdot)$ be a polynomial and a negligible function, respectively. PPT is short for probabilistic polynomial time. We use $:=$ for deterministic assignment, $\leftarrow$ for assignment to an algorithm's output, $\leftarrow\!\!\$$ for uniform random sampling, and $\approx_c$ for computational indistinguishability between distribution ensembles. The integer range $[a \,..\, b]$ represents $\{a, a+1, \ldots, b\}$; let $[n] := [1 \,..\, n]$ and $[n]_0 := [0 \,..\, n]$.

We describe the ledger and present the PoRep definitions. Definitions of the remaining building blocks are recalled in Appendix A. Throughout the paper, unless explicitly conditioned otherwise, probabilities and distribution ensembles are taken over the randomness of setup, key generation, and all randomized algorithms invoked by the experiment.

2.1 Secure Blockchain as a Robust Public Ledger

We employ a secure blockchain protocol that implements a robust public ledger; recall the security notions.[1]

Definition 1 (Persistence and Liveness [22]). *Let $k, u \in \mathbb{N}_0$. A blockchain protocol $\Pi_{\mathcal{L}}^{k,u}$ that proceeds in rounds implements a robust public ledger $\mathcal{L}$ if:*

- *k-persistence. In any round, if an honest user reports $\mathcal{L}$ that contains a message m in a block at least k blocks away from the head of $\mathcal{L}$, then m is reported by all honest users in the same position on $\mathcal{L}$ from that round onward.*
- *(k, u)-liveness. If a* valid *message m is given as input to all honest users continuously for u consecutive rounds, then all honest users report this message at least k blocks from the head of $\mathcal{L}$. Message validity is determined by the protocol; the liveness premise requires the message to remain valid while awaiting inclusion.*

Bookkeeping on $\mathcal{L}$. The ledger's bookkeeping semantics are essential to DSN formalization, as they determine how transactions identify and advance the protocol state. For concreteness, we express these semantics using the EUTxO model [10], in which the protocol state is carried by a distinguished output and each accepted non-terminal transition consumes the current state output and creates exactly one successor state output. Our construction is *not restricted to EUTxO*: account-based smart contracts [39] can implement the same authenticated state transitions, validation requirements, and atomic value transfers.

Let $\Pi_{\mathcal{L}}^{k,u}$ be a blockchain protocol that implements a robust public ledger $\mathcal{L}$. Under the EUTxO model, a transaction tx, submitted to $\Pi_{\mathcal{L}}^{k,u}$, consists of an input list I, an output list O, and an optional validity interval vi; we write $\mathsf{tx} \coloneqq (I, O, vi)$. If an input of tx refers to an output of another transaction tx', we call tx' a parent transaction of tx and tx a subsequent transaction of tx'. We present a simplified transaction data structure based on [10, Fig. 3].[2]

- For each $j \in [|\mathsf{tx}.I|]$, if the j-th input spends $\mathsf{tx}'.\mathsf{out}_i$, write $\mathsf{tx}.\mathsf{in}_j \coloneqq (\mathsf{tx}'.\mathsf{out}_i, \rho_j)$,[3] where ρ_j is the *redeemer* supplied to ν_i for validating the expenditure of $\mathsf{tx}'.\mathsf{out}_i$. If ρ_j requires a signature σ_j, write its unsigned form as $\tilde{\rho}_j \coloneqq \rho_j \backslash \{\sigma_j\}$.
- For each $i \in [|\mathsf{tx}'.O|]$, write $\mathsf{tx}'.\mathsf{out}_i \coloneqq (\nu_i, v_i, \delta_i)$, where ν_i is the *validator script* governing expenditure of the output, v_i is the ledger value locked by it, and δ_i is the "*datum*" that contains contract-specific information.
- The validity interval $\mathsf{tx}.vi$ specifies the ledger rounds during which tx is admissible for processing. An omitted interval denotes no additional time restriction.

[1] These notions follow from the underlying blockchain's common prefix, chain growth, and chain quality properties [22].

[2] In [10], the validator and datum are relocated to transaction inputs, with only their hash values retained as references in the outputs to improve memory efficiency. We revert this modification and treat the validator and the datum as components of the outputs for conceptual simplicity.

[3] In [10], the output reference consists of tx''s identifier and the output index i. We write $\mathsf{tx}'.\mathsf{out}_i$ directly to avoid additional notation.

For convenience, we extend the tilde notation to transactions. Let $J_{\mathsf{tx}} \subseteq [|\mathsf{tx}.I|]$ index the input redeemers that contain a signature σ_j. The *base transaction* of tx, obtained by removing these signatures, is denoted by $\tilde{\mathsf{tx}} \coloneqq \mathsf{tx} \setminus \{\sigma_j\}_{j \in J_{\mathsf{tx}}}$.

Following [10, Fig. 6], the validation of tx consists of two subprocesses:

1. Checking the *well-formedness* of tx. The current round must lie within $\mathsf{tx}.vi$; all outputs must have non-negative values; all inputs must refer to unspent outputs; the outputs must preserve the unspent value recorded in $\Pi_{\mathcal{L}}^{k,u}$ (a fundamental requirement inherited from the UTxO model); and no input double spends existing outputs. We abstract these checks by a deterministic algorithm TxVf, with the current ledger state and round left implicit. Thus, tx (*resp.*, $\tilde{\mathsf{tx}}$) is well-formed if and only if $\mathsf{TxVf}(\mathsf{tx}) = 1$ (*resp.*, $\mathsf{TxVf}(\tilde{\mathsf{tx}}) = 1$).
2. Verifying the *eligibility* of each input to spend its referenced output. Consider $\mathsf{tx}'.\mathsf{out}_i = (\nu_i, v_i, \delta_i)$ and $\mathsf{tx}.\mathsf{in}_j = (\mathsf{tx}'.\mathsf{out}_i, \rho_j)$. The input $\mathsf{tx}.\mathsf{in}_j$ is eligible to spend $\mathsf{tx}'.\mathsf{out}_i$ if and only if $\nu_i(\delta_i, \rho_j, \mathsf{tx}) = 1$.

Although both subprocesses inspect tx, they enforce different conditions. The first checks ledger-level well-formedness independently of the spending conditions attached to particular outputs. In the second, tx serves as the transaction context [10, Fig. 4], allowing each validator to inspect the complete input and output lists and validity interval of tx. For a state output, these conditions establish *contract continuity* by requiring: (1) each non-terminal transition to create exactly one successor state output carrying the prescribed validator, value, and datum; and (2) each terminal transition to create no successor state output and transfer the associated value as prescribed. An account-based smart contract [39] enforces the same continuity conditions using a designated storage entry: an accepted non-terminal call atomically replaces the current state with its prescribed successor, whereas an accepted terminal call closes the state and performs the prescribed transfer.

Finally, a transaction tx is *k-confirmed* on $\mathcal{L}$, denoted by $\mathsf{tx} \in_k \mathcal{L}$, if and only if: (1) when tx is processed, $\mathsf{TxVf}(\mathsf{tx}) = 1$; (2) every input of tx is eligible to spend its referenced output; and (3) tx is included in a block at least k blocks from the head of $\mathcal{L}$. We omit k when it is clear from context and write simply $\mathsf{tx} \in \mathcal{L}$. By (k, u)-liveness, a transaction satisfying the liveness premise becomes k-confirmed after being continuously submitted for u consecutive rounds.

2.2 PoRep with Extractor-Based Security

Recall the syntax from [18,19]. A PoRep scheme consists of a tuple of algorithms $\mathsf{PoRep} = (\mathsf{Setup}, \mathsf{Preproc}, \mathsf{TagVf}, \mathsf{Replicate}, \mathsf{Extract}, \mathsf{Poll}, \mathsf{Prove}, \mathsf{Vf})$. All algorithms operate in the random-access machine (RAM) model, and parallel algorithms operate in the parallel RAM (PRAM) model.

- $\mathsf{Setup}(1^\lambda, t)$ takes as input the security parameter $\lambda > 0$ and a time parameter $t > 0$ specifying the challenge-response period, and outputs public parameters pp. All subsequent algorithms implicitly take pp as input.

– $\mathsf{Preproc}(ek, \tilde{D})$ takes as input an optional encoding key ek and data $\tilde{D} \in \{0,1\}^*$ of size $\mathcal{O}(\mathsf{poly}(\lambda))$. It outputs *preprocessed* data D and a corresponding data tag τ_D.
– $\mathsf{TagVf}(D, \tau_D)$ deterministically outputs 1 if D is consistent with τ_D, or 0 otherwise.
– $\mathsf{Replicate}(id, D, \tau_D)$ takes as input a replication identifier id, data D, its tag τ_D. It outputs a replica R and compact auxiliary information aux.
– $\mathsf{Extract}(id, \tau_D, R)$ deterministically outputs D encoded by R consistent with (id, τ_D), or $\perp$ otherwise.

The remaining procedures $(\mathsf{Poll}, \mathsf{Prove}, \mathsf{Vf})$ compose a challenge-response protocol.

– $\mathsf{Poll}(\mathsf{aux})$ takes as input auxiliary information aux and outputs a public challenge ch and an optional verifier secret randomness r.
– $\mathsf{Prove}(id, R, \mathsf{aux}, \mathsf{ch})$ takes as input a replication identifier id, a replica R, auxiliary information aux, and a challenge ch. It outputs a proof π for ch.
– $\mathsf{Vf}(id, \tau_D, \mathsf{aux}, \mathsf{ch}, r, \pi)$ deterministically outputs 1 if both the initial verification of aux and verification of π accept, or 0 otherwise. We overload $\mathsf{Vf}(id, \tau_D, \mathsf{aux})$ to denote initial verification alone, which outputs 1 if aux is valid *w.r.t.* (id, τ_D), or 0 otherwise.

Correctness requires that any honest proof must be accepted within the challenge-response period and the replicated data must be recoverable.[4]

Definition 2 (Correctness). *A PoRep scheme is correct if, for any λ, there exists t_λ s.t. the following holds for any $t \geq t_\lambda$, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, t)$, any data-tag pair (D, τ_D) satisfying $\mathsf{TagVf}(D, \tau_D) = 1$, and any id. Let $(R, \mathsf{aux}) \leftarrow \mathsf{Replicate}(id, D, \tau_D)$, $(\mathsf{ch}, r) \leftarrow \mathsf{Poll}(\mathsf{aux})$, and $\pi \leftarrow \mathsf{Prove}(id, R, \mathsf{aux}, \mathsf{ch})$.*

$$\Pr\begin{bmatrix}\mathsf{Vf}(id, \tau_D, \mathsf{aux}, \mathsf{ch}, r, \pi) = 1 \\ \wedge \mathsf{Extract}(id, \tau_D, R) = D\end{bmatrix} = 1$$

The invocation of Prove *runs in parallel time at most t.*

Extractor-based replication soundness. The security of PoRep requires that any μ-admissible prover, whose probability of simultaneously passing verification in $k \geq 1$ instances under distinct identifiers is at least μ, cannot use substantially less storage than required for k independent replicas, even when all instances have identical preprocessed data as input. A prover may reversibly re-encode its stored state while preserving its size and proving behavior. Hence, the stored state need not itself admit a partition into k separately recoverable replicas [18, Proposition 1]. Rational replication [18, Def. 6] therefore compares an admissible prover with another admissible and well-behaved adversary who stores such k separately recoverable replicas. However, this comparison concerns a replacement adversary's state constructed during preprocessing. To support our later adaptive analysis of the original prover's successive online states, we formulate *replication soundness* directly through an extractor, requiring the recovery

[4] We omit a separate definition of preprocessing correctness, which requires any honest $\mathsf{Preproc}$ outputs to satisfy $\mathsf{TagVf}(D, \tau_D) = 1$.

and storage guarantees to hold except with negligible probability for every μ-admissible adversary.[5] We specify the experiment in Fig. 1, which is used to define admissibility and replication soundness.

Experiment $\mathrm{RepSound}^{k}_{\mathcal{A},\mathcal{E}}(\mathsf{pp}, z)$:

$(\boldsymbol{id}, \boldsymbol{D}, \boldsymbol{\tau_D}, \eta, \mathbf{aux}) \leftarrow \mathcal{A}_{\mathrm{pre}}(z)$

return $(0,0)$ **if** $(\exists i, j \in [k], i \neq j \wedge id_i = id_j)$

$\quad \vee (\exists i \in [k], \mathsf{TagVf}(D_i, \tau_{D_i}) = 0 \vee \mathsf{Vf}(id_i, \tau_{D_i}, \mathsf{aux}_i) = 0)$

$(R_i, \cdot) \leftarrow \mathsf{Replicate}(id_i, D_i, \tau_{D_i}), \forall i \in [k]$

$(\mathsf{ch}_i, r_i) \leftarrow \mathsf{Poll}(\mathsf{aux}_i), \forall i \in [k]$

$(\boldsymbol{\pi}, \hat{z}) \leftarrow \mathcal{A}_{\mathrm{onl}}(\boldsymbol{id}, \eta, \mathbf{aux}, \mathbf{ch})$

$v_i \coloneqq \mathsf{Vf}(id_i, \tau_{D_i}, \mathsf{aux}_i, \mathsf{ch}_i, r_i, \pi_i), \forall i \in [k];\ v \coloneqq \bigwedge_{i \in [k]} v_i$

$(\boldsymbol{R'}, \omega, \mathsf{Decode}^*) \leftarrow \mathcal{E}^{\mathcal{A}_{\mathrm{onl}}[\eta]}(\boldsymbol{id}, \boldsymbol{\tau_D}, \mathbf{aux})$

$D'_i \coloneqq \mathsf{Extract}(id_i, \tau_{D_i}, R'_i), \forall i \in [k]$

$\ell_i \coloneqq \max\{|R_i|, |R'_i|\}, \forall i \in [k]$

$b \coloneqq (\exists i \in [k] : D'_i \neq D_i) \vee \mathsf{Decode}^*((R'_i)_{i \in [k]}, \omega) \neq \hat{z}$

$\quad \vee |\eta| < (1 - \epsilon)(\sum_{i \in [k]} \ell_i + |\omega| + |\mathsf{Decode}^*|)$

// extraction failure, auxiliary output recovery failure, storage bound violation

return (v, b)

Fig. 1: The replication experiment. Let $\mathcal{A} = (\mathcal{A}_{\mathrm{pre}}, \mathcal{A}_{\mathrm{onl}})$ be a two-stage adversary (*i.e.*, preprocessing and online). All instance vectors have length k. Public parameters are implicit inputs to all algorithms; the static auxiliary input z is supplied only to $\mathcal{A}_{\mathrm{pre}}$ (cf. [18, Fig. 2.3]). The auxiliary output $\hat{z}$ is a deterministic function of the retained state and fixed public inputs, independent of fresh challenges and online randomness. The measure $|\eta|$ counts all data-dependent persistent information available online exactly once, including the separately displayed identifiers and auxiliary information. The honest replicas provide storage-size benchmarks.

Definition 3 (Admissible PoRep Adversary [18]). *An adversary $\mathcal{A} = (\mathcal{A}_{\mathrm{pre}}, \mathcal{A}_{\mathrm{onl}})$ is admissible if $\mathcal{A}_{\mathrm{pre}}$ runs in polynomial time and $\mathcal{A}_{\mathrm{onl}}$ runs in parallel time at most t. It is μ-admissible if, additionally, the following holds for any $\lambda, t > 0$, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, t)$, any polynomially bounded $k \geq 1$, any polynomial-length static auxiliary input z, and any preprocessing output in the support of $\mathcal{A}_{\mathrm{pre}}(z)$.*

$$\Pr[v = 1 | (v, \cdot) \leftarrow \mathrm{RepSound}^{k}_{\mathcal{A},\mathcal{E}}(\mathsf{pp}, z)] \geq \mu$$

The probability is over the experiment randomness for fixed pp, z, and those outputs.

[5] An extractor-based notion of replication soundness without timing assumptions also appears in [12, Def. 5]. Its extraction criterion counts only exact matches to the honestly generated replicas. Our notion follows the time-bounded model of [18, Def. 6], permits other valid replicas that recover the same data, and additionally requires residual information and a decoder for recovering the prover's auxiliary output.

Definition 4 (Replication Soundness). *Let $\epsilon \in [0,1)$ and $\mu \in (0,1]$. A PoRep scheme is (ϵ,μ)-replication sound if, for any μ-admissible adversary $\mathcal{A} = (\mathcal{A}_{\text{pre}}, \mathcal{A}_{\text{onl}})$, there exists a PPT extractor $\mathcal{E}$, with the access specified in Remark 1, s.t. the following holds for any $\lambda, t > 0$, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, t)$, any polynomially bounded $k \geq 1$, and any polynomial-length static auxiliary input z.*

$$\Pr[b = 1 | (\cdot, b) \leftarrow \text{RepSound}^k_{\mathcal{A},\mathcal{E}}(\mathsf{pp}, z)] \leq \mathsf{negl}(\lambda)$$

$\mathcal{E}$ is fixed independently of z and the sampled execution.

Remark 1 (Extractor access and relation to rational replication). The extractor operates in two phases. First, it fixes $\boldsymbol{R}'$ using only rewindable black-box access to $\mathcal{A}_{\text{onl}}[\eta]$, with the identifiers, retained state, and auxiliary information fixed across invocations; it cannot inspect hidden inputs or internal states. It may then inspect the adversary's description and preprocessing execution to produce $(\omega, \mathsf{Decode}^*)$, without modifying $\boldsymbol{R}'$. Neither phase receives the experiment's sampled challenges or verification outcomes. The decoder Decode^* is deterministic and polynomial-time, and receives only the displayed inputs and the implicit public parameters. Its description length includes all embedded instance-dependent information.

Our formulation makes explicit the two extraction procedures used in the proof of [18, Lemma 2]: rewindable PoRC extraction and state-observing extraction of strong compression schemes under the KoSC assumption [18, Assumption 1]. In Appendix B.3, we establish sufficient conditions for our extractor-based replication soundness from PoRC, PoSpace, and KoSC (Thm. 3). We then show how this notion yields rational replication [18, Def. 6] by constructing a well-behaved adversary under the stated admissibility conditions and resource bounds (Thm. 4, Appendix B.4).[6]

From public-coin PoRep to proofs of space-time (PoST). A PoRep is public-coin if its challenge ch is derived from publicly verifiable randomness, in which case $r = \perp$. Public-coin challenges support periodic storage audits following the PoST paradigm [31]. Our DSN uses this approach to verify storage throughout the contract period (Section 4.1), with successive audits linked through their responses. Accordingly, we write $\mathsf{ch} \leftarrow \mathsf{Poll}(\mathsf{pp}, \mathsf{aux}, \mathsf{seed})$, where Poll is deterministic for a fixed public seed seed. We omit r from verification and write $\mathsf{Vf}(\mathsf{pp}, id, \tau_D, \mathsf{aux}, \mathsf{ch}, \pi)$. To derive successive seeds, we instantiate the sequential-work chaining approach of [31, Section 2.2.6] with a VDF [8]. We adopt their sequential timing assumption, which bounds *adversarial* sequential computation per ledger round. The delay is calibrated to allow honest seed generation, audit-response computation, and transaction confirmation within each audit interval. For security, the calibration must also keep each target challenge query fresh relative to the preceding execution prefix and limit subsequent adversarial response computation to parallel time t.

[6] The supporting definitions and assumptions are also in Appendix B.

3 Our Blockchain-Aided DSN Formalization

This section formalizes a blockchain-aided DSN protocol. After setup and client preprocessing, the protocol proceeds through delegation, preservation, and retrieval: the parties establish a storage contract, the server periodically demonstrates storage, and the client recovers the delegated data upon finalization.[7] Using the bookkeeping notation of Section 2.1, we specify transaction inputs, outputs, and validators to describe the required state transitions, authorizations, and value transfers. Account-based realizations may implement these requirements through contract storage and calls, while preserving public records and timing conditions.

3.1 Protocol Phases

Let $\Pi_{\mathcal{L}}$ abbreviate the blockchain protocol $\Pi_{\mathcal{L}}^{k,u}$, and let $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be a secure signature scheme. A DSN protocol built atop $\Pi_{\mathcal{L}}$ is denoted by $\mathsf{DSN}_{\mathcal{L}}$. We consider a client entity $\mathcal{C}$ delegating preprocessed data to one or more servers. The client entity may comprise multiple clients; their internal data and key management are outside our model. The client entity and each server are identified by a signature public key pk and hold the corresponding secret key sk, where $(sk, pk) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$.

Setup and preprocessing. The setup algorithm $\mathsf{Setup}(1^\lambda, t)$ executes once for each DSN instance, either externally or cooperatively by the involved parties. It takes as input the security parameter λ and a time parameter t, and outputs public parameters pp and a trapdoor td. The public parameters include a designated burning address $pk_{\mathcal{B}}$ for forfeited collateral and the audit interval $T(\lambda, t)$, measured in ledger rounds. We write T when the parameters are clear. All subsequent algorithms implicitly take pp as input. The trapdoor td is retained by the security experiment solely for proof simulation and is not provided to protocol participants.

Each client entity uses the PoRep preprocessing interface (Section 2.2) to obtain a data-tag pair $(D, \tau_D) \leftarrow \mathsf{PoRep.Preproc}(ek, \tilde{D})$. The remaining protocol syntax and security definitions take this pair as input and concern D, rather than the original input $\tilde{D}$. Recovering $\tilde{D}$ from D, using the retained encoding key ek when needed, is handled by the preprocessing scheme. This accommodates, *e.g.*, erasure code-based [18] or encryption-based preprocessing [40], without changing the subsequent DSN interfaces.

Delegation. We describe each delegation between the client entity $\mathcal{C}$, with key pair $(sk_{\mathcal{C}}, pk_{\mathcal{C}})$, and a server $\mathcal{S}$, with $(sk_{\mathcal{S}}, pk_{\mathcal{S}})$. The delegation phase comprises two interactive protocols and a public verification algorithm: OffDel establishes the replication outcome; DelVf verifies the resulting contract message; and OnDel records the parties' agreement and locks the associated funds.

[7] Appendix D discusses early, on-demand retrieval and the limitations of extending this mechanism to repeated retrieval sessions, alongside other configuration options and incentive considerations.

Let $\mathsf{md} \coloneqq (id, n, V)$ denote the *contract metadata* agreed upon by both parties before executing OffDel. Here, id identifies the parties and uniquely distinguishes the delegation; $n \geq 1$ specifies the number of preservation audits; and $V \coloneqq \{vi_{\text{con}}, vi^1_{\text{adt}}, \ldots, vi^n_{\text{adt}}, vi_{\text{fin}}\}$ specifies the validity intervals for delegation, the indexed preservation audits, and finalization, based on the audit interval T.

- $\mathsf{OffDel}(\mathsf{md}, \langle \mathcal{C}(D, \tau_D), \mathcal{S}() \rangle)$ takes as input the agreed metadata md and a pre-processed data-tag pair (D, τ_D). On successful completion, both parties obtain the same *contract message* $m_{\text{con}} \coloneqq (\mathsf{md}, \tau_D, \mathsf{aux})$, and $\mathcal{S}$ additionally obtains the replica R, where aux is the contract auxiliary information.
- $\mathsf{DelVf}(m_{\text{con}})$ is a deterministic local verification predicate. It takes as input the contract message $m_{\text{con}} = (\mathsf{md}, \tau_D, \mathsf{aux})$ and outputs 1 if aux is valid for the delegation identified by (md, τ_D), and 0 otherwise.

To fund the contract, $\mathcal{C}$ and $\mathcal{S}$ use their respective unspent outputs $\mathsf{out}_{\text{pay}}$ and $\mathsf{out}_{\text{col}}$ from confirmed transactions $\mathsf{tx}_{\text{pay}}, \mathsf{tx}_{\text{col}} \in \mathcal{L}$. Their values p and c provide the payment and collateral.

- $\mathsf{OnDel}(m_{\text{con}}, \mathsf{tx}_{\text{pay}}, \mathsf{tx}_{\text{col}}, \langle \mathcal{C}(sk_{\mathcal{C}}), \mathcal{S}(sk_{\mathcal{S}}) \rangle)$ takes as input a contract message m_{con} satisfying $\mathsf{DelVf}(m_{\text{con}}) = 1$ and accepted by both parties, and the funding transactions $\mathsf{tx}_{\text{pay}}, \mathsf{tx}_{\text{col}}$. Using the secret keys $sk_{\mathcal{C}}, sk_{\mathcal{S}}$, $\mathcal{C}$ and $\mathcal{S}$ authorize the same base transaction $\tilde{\mathsf{tx}}_{\text{con}}$ and submit tx_{con} with the following components:[8]
  - Inputs $\mathsf{in}^{\text{pay}}_{\text{con}} \coloneqq (\mathsf{out}_{\text{pay}}, \rho^{\text{pay}}_{\text{con}})$ and $\mathsf{in}^{\text{col}}_{\text{con}} \coloneqq (\mathsf{out}_{\text{col}}, \rho^{\text{col}}_{\text{con}})$, authorized by $\mathcal{C}$ and $\mathcal{S}$, respectively.
  - A payment output $\mathsf{out}^{\text{pay}}_{\text{con}} \coloneqq (\nu^{\text{pay}}_{\text{con}}, p, \delta^{\text{pay}}_{\text{con}})$ and a collateral output $\mathsf{out}^{\text{col}}_{\text{con}} \coloneqq (\nu^{\text{col}}_{\text{con}}, c, \delta^{\text{col}}_{\text{con}})$, both recording m_{con} in their datums.
  - The validity interval $vi_{\text{con}} \in V$.

Before authorizing tx_{con}, each party checks that its base $\tilde{\mathsf{tx}}_{\text{con}}$ records the agreed contract message and locks the agreed values under the prescribed validators. The collateral output $\mathsf{out}^{\text{col}}_{\text{con}}$ establishes the initial protocol state. Its validator $\nu^{\text{col}}_{\text{con}}$ accepts the first audit transition only if $\mathsf{AuditVf}^1$ (defined below) accepts, the transaction has validity interval vi^1_{adt}, and it creates exactly one successor state output carrying the same m_{con} and collateral c, with the required successor validator and datum. Successor validators impose corresponding requirements on later audits.

If the current state output remains unspent after the required audit or finalization interval expires, its validator permits spending only with authorization under the designated burning address $pk_{\mathcal{B}}$, whose secret key is unavailable to all parties. The collateral is therefore effectively forfeited. In contrast, the release schedule for the *payment* locked under $\nu^{\text{pay}}_{\text{con}}$ remains configurable: the server $\mathcal{S}$ may receive it upfront or incrementally over the contract period. These choices concern incentives and rational behavior; our security notions impose no particular payment schedule.

Confirmation of tx_{con} marks the completion of delegation. The server retains R, while the client entity no longer needs to retain D locally. The contract

[8] Subscripts and superscripts are inherited component-wise from the enclosing transaction.

message remains publicly available in $\mathsf{tx}_{\mathrm{con}}$. The storage overhead of these public records is discussed in Remark 3.

Preservation. Put $\mathsf{tx}^0_{\mathrm{adt}} \coloneqq \mathsf{tx}_{\mathrm{con}}$ and $\mathsf{out}^0_{\mathrm{adt}} \coloneqq \mathsf{out}^{\mathrm{col}}_{\mathrm{con}}$. The preservation phase consists of the indexed procedures $\{(\mathsf{SeedGen}^i, \mathsf{Audit}^i, \mathsf{AuditVf}^i)\}_{i\in[n]}$. $\mathsf{SeedGen}^i$ chains successive audits through a sequential computation whose input is determined by audit i and the authenticated reference and datum of $\mathsf{out}^{i-1}_{\mathrm{adt}}$, with a delay calibrated to the audit interval T. The helper verification algorithm $\mathsf{AuditVf}^i$ is invoked by the validator of $\mathsf{out}^{i-1}_{\mathrm{adt}}$ to verify the seed and audit response carried by $\mathsf{tx}^i_{\mathrm{adt}}$. We define them inductively for $i \in [n]$.

- $\mathsf{SeedGen}^i(\mathsf{tx}^{i-1}_{\mathrm{adt}}, T)$ takes as input the confirmed transaction $\mathsf{tx}^{i-1}_{\mathrm{adt}} \in \mathcal{L}$ carrying the current state and the audit interval T.[9] It outputs a public seed seed^i for the i-th audit alongside its verification evidence π^i_{seed}.
- $\mathsf{Audit}^i(\mathsf{tx}^{i-1}_{\mathrm{adt}}, R, \mathsf{seed}^i, \pi^i_{\mathsf{seed}})$, run by $\mathcal{S}$, takes as input the confirmed transaction $\mathsf{tx}^{i-1}_{\mathrm{adt}} \in \mathcal{L}$ carrying the current state, the local replica R, and the public seed-proof pair $(\mathsf{seed}^i, \pi^i_{\mathsf{seed}})$. It produces a challenge-proof pair (ch^i, π^i) for the supplied seed and outputs an audit transaction $\mathsf{tx}^i_{\mathrm{adt}}$ for submission to $\Pi_{\mathcal{L}}$ with the following components:
  - Input $\mathsf{in}^i_{\mathrm{adt}} \coloneqq (\mathsf{out}^{i-1}_{\mathrm{adt}}, \rho^i_{\mathrm{adt}})$, where ρ^i_{adt} contains $(\mathsf{seed}^i, \pi^i_{\mathsf{seed}}, \mathsf{ch}^i, \pi^i)$.
  - Exactly one successor state output $\mathsf{out}^i_{\mathrm{adt}} \coloneqq (\nu^i_{\mathrm{adt}}, c, \delta^i_{\mathrm{adt}})$, retaining the collateral c and recording m_{con} in δ^i_{adt}. For $i \in [n-1]_0$, ν^i_{adt} invokes $\mathsf{AuditVf}^{i+1}$ and enforces the contract-continuity requirements for the next audit. For $i = n$, ν^n_{adt} governs the subsequent transitions in the retrieval phase.
  - The validity interval $vi^i_{\mathrm{adt}} \in V$.
- $\mathsf{AuditVf}^i(\mathsf{out}^{i-1}_{\mathrm{adt}}, \mathsf{seed}^i, \pi^i_{\mathsf{seed}}, \mathsf{ch}^i, \pi^i)$ reads m_{con} from $\delta^{i-1}_{\mathrm{adt}}$. The authenticated reference to $\mathsf{out}^{i-1}_{\mathrm{adt}}$ is supplied by the spending input $\mathsf{in}^i_{\mathrm{adt}}$ and is left implicit in the invocation.[10] It deterministically outputs 1 if: (1) π^i_{seed} verifies seed^i for audit i *w.r.t.* this reference, the predecessor datum, and the agreed audit schedule; (2) ch^i is correctly derived from seed^i; and (3) π^i is a valid proof *w.r.t.* $(m_{\mathrm{con}}, \mathsf{ch}^i)$; otherwise, it outputs 0.

For authentication, each ν^{i-1}_{adt} may additionally require a signature from $\mathcal{S}$. Confirmation of the final audit transaction, *i.e.*, $\mathsf{tx}^n_{\mathrm{adt}} \in \mathcal{L}$, marks the end of the preservation phase.

Retrieval. The retrieval phase consists of an interactive protocol $\mathsf{InitRet}$ and three deterministic algorithms $(\mathsf{RetVf}, \mathsf{Finalize}, \mathsf{RetExt})$. During $\mathsf{InitRet}$, the server provides a concealed replica in exchange for the client's conditional authorization and retains private session state. The algorithm $\mathsf{Finalize}$ continues from this state to complete the authorization of the finalization transaction, enabling data recovery and collateral release.

[9] Although $T \in \mathsf{pp}$, we display it explicitly as an input.

[10] The validator ν^{i-1}_{adt} invoking $\mathsf{AuditVf}^i$ receives only the current spending transaction and authenticated information about the consumed output, but not the complete predecessor transaction. Taking the latter as input would require separately supplying and authenticating its contents. We instead define the seed relation using only the predecessor reference and datum available during local validation.

– $\mathsf{InitRet}(\mathsf{tx}^n_{\mathrm{adt}}, \langle \mathcal{C}(sk_{\mathcal{C}}), \mathcal{S}(R)\rangle)$ takes as common input the confirmed transaction $\mathsf{tx}^n_{\mathrm{adt}} \in \mathcal{L}$, with local inputs $sk_{\mathcal{C}}$ for $\mathcal{C}$ and R for $\mathcal{S}$. On successful completion, $\mathcal{C}$ and $\mathcal{S}$ obtain $(C, \mathsf{aux}_{\mathsf{ret}}, \hat{\mathsf{tx}}_{\mathrm{fin}})$, comprising a concealed replica, retrieval auxiliary information, and a *partial* finalization transaction. When needed, we write $\mathsf{aux}_{\mathsf{ret}} \coloneqq (x_{\mathsf{ret}}, \pi_{\mathsf{ret}})$ explicitly, where x_{ret} is an additional public input to the retrieval proof π_{ret}. Additionally, $\mathcal{S}$ retains private session state $\mathsf{st}_{\mathcal{S}}$ for $\mathsf{Finalize}$. $\hat{\mathsf{tx}}_{\mathrm{fin}}$ has the following components:
  - Partial input $\hat{\mathsf{in}}_{\mathrm{fin}} \coloneqq (\mathsf{out}^n_{\mathrm{adt}}, \hat{\rho}_{\mathrm{fin}})$, where $\hat{\rho}_{\mathrm{fin}}$ contains $\mathsf{aux}_{\mathsf{ret}}$ and the client's conditional authorization of the base transaction $\hat{\tilde{\mathsf{tx}}}_{\mathrm{fin}}$.
  - Output $\mathsf{out}_{\mathrm{fin}} = (\nu_{\mathrm{fin}}, c, \delta_{\mathrm{fin}})$, placing the collateral c under $\mathcal{S}$'s control and recording m_{con} in δ_{fin}.[11]
  - The validity interval $vi_{\mathrm{fin}} \in V$.
– $\mathsf{RetVf}(\mathsf{tx}^n_{\mathrm{adt}}, C, \mathsf{aux}_{\mathsf{ret}}, \hat{\mathsf{tx}}_{\mathrm{fin}})$ is a deterministic local verification predicate. It reads m_{con} from $\mathsf{tx}^n_{\mathrm{adt}}$ and outputs 1 if: (1) $(C, \mathsf{aux}_{\mathsf{ret}})$ is valid for m_{con}; (2) $\hat{\mathsf{tx}}_{\mathrm{fin}}$ has the agreed finalization fields and a valid partial redeemer $\hat{\rho}_{\mathrm{fin}}$; and (3) at least u rounds remain before the expiry of vi_{fin}. Otherwise, it outputs 0. The current ledger round is implicit in the invocation.

Remark 2 (Partial transaction withholding attack). A malicious client may withhold $\hat{\mathsf{tx}}_{\mathrm{fin}}$ until the honest server has enough time to complete the authorization and submit $\mathsf{tx}_{\mathrm{fin}}$ within vi_{fin}, but fewer than u rounds remain before its expiry. The transaction may therefore be submitted within its validity interval yet fail to be included before that interval expires. Nevertheless, publishing $\mathsf{tx}_{\mathrm{fin}}$, even without confirmation, may enable the client to recover the data, while the server fails to obtain the corresponding collateral release. The third condition in RetVf prevents an honest server from accepting such late partial transactions by requiring a u-round margin before expiry. This margin permits confirmation under ledger liveness, provided that the completed transaction remains valid while awaiting inclusion.

Accordingly, $\mathcal{S}$ invokes RetVf on the retrieval tuple and proceeds to $\mathsf{Finalize}$ only if it outputs 1.

– $\mathsf{Finalize}(\mathsf{st}_{\mathcal{S}}, \hat{\mathsf{tx}}_{\mathrm{fin}})$, run by $\mathcal{S}$, takes as input the private session state $\mathsf{st}_{\mathcal{S}}$ retained from $\mathsf{InitRet}$ and the accepted partial transaction $\hat{\mathsf{tx}}_{\mathrm{fin}}$. It completes the partial redeemer without changing any fields covered by the client's authorization, then outputs $\mathsf{tx}_{\mathrm{fin}}$ for submission to $\Pi_{\mathcal{L}}$.
– $\mathsf{RetExt}(C, \hat{\mathsf{tx}}_{\mathrm{fin}}, \mathsf{tx}_{\mathrm{fin}})$, run by $\mathcal{C}$, takes as input the concealed replica C, the retained partial transaction $\hat{\mathsf{tx}}_{\mathrm{fin}}$, and the confirmed $\mathsf{tx}_{\mathrm{fin}} \in \mathcal{L}$. Using $\hat{\mathsf{tx}}_{\mathrm{fin}}$ and $\mathsf{tx}_{\mathrm{fin}}$, it recovers a replica R from C and reconstructs the preprocessed data D encoded by R. It outputs (R, D) if recovery succeeds and the recovered pair is consistent with the contract message fixed by $\hat{\mathsf{tx}}_{\mathrm{fin}}$; otherwise, it outputs $\perp$.

The validator ν^n_{adt} checks the completed authorization and the agreed finalization fields of $\mathsf{tx}_{\mathrm{fin}}$. Further retrieval policies, including per-chunk retrieval and early retrieval, are discussed in Appendix D.

[11] After $\mathsf{tx}_{\mathrm{fin}}$ is confirmed, $\mathcal{S}$ can spend $\mathsf{out}_{\mathrm{fin}}$ using $sk_{\mathcal{S}}$.

Remark 3 (Honest blockchain storage overhead). In honest executions, the data D, replica R, and concealed replica C are kept off-chain. Protocol transactions carry the contract message m_{con}, verification and authorization data, and the information required for bookkeeping and contract continuity. The resulting storage overhead is measured cumulatively over the contract period. Where supported by the ledger, repeated contract messages and scripts may be represented by references to reduce duplication, provided that validators retain authenticated access to their contents (see Footnote 2).

Correctness. The correctness experiment is given in Fig. 2; it returns 0 if any assertion fails.

$\underline{\text{Experiment } \mathrm{Correct}^{\mathcal{L}}_{\mathsf{PoRep.TagVf}}(\mathsf{pp}, D, \tau_D, \mathsf{md}) :}$
assert $\mathsf{PoRep.TagVf}(D, \tau_D) = 1 \wedge \mathsf{tx}_{\mathrm{pay}} \in \mathcal{L} \wedge \mathsf{tx}_{\mathrm{col}} \in \mathcal{L}$
$(m_{\mathrm{con}}; R) \leftarrow \mathsf{OffDel}(\mathsf{md}, \langle \mathcal{C}(D, \tau_D), \mathcal{S}() \rangle)$
assert $\mathsf{DelVf}(m_{\mathrm{con}}) = 1$
$\mathsf{tx}_{\mathrm{con}} \leftarrow \mathsf{OnDel}(m_{\mathrm{con}}, \mathsf{tx}_{\mathrm{pay}}, \mathsf{tx}_{\mathrm{col}}, \langle \mathcal{C}(sk_{\mathcal{C}}), \mathcal{S}(sk_{\mathcal{S}}) \rangle)$
assert $\mathsf{tx}_{\mathrm{con}} \in \mathcal{L}$ // $\mathsf{tx}^0_{\mathrm{adt}} \coloneqq \mathsf{tx}_{\mathrm{con}}$
for $i \in [n] : (\mathsf{seed}^i, \pi^i_{\mathsf{seed}}) \leftarrow \mathsf{SeedGen}^i(\mathsf{tx}^{i-1}_{\mathrm{adt}}, T)$
$\mathsf{tx}^i_{\mathrm{adt}} \leftarrow \mathsf{Audit}^i(\mathsf{tx}^{i-1}_{\mathrm{adt}}, R, \mathsf{seed}^i, \pi^i_{\mathsf{seed}})$
assert $\mathsf{tx}^i_{\mathrm{adt}} \in \mathcal{L}$
$(C, \mathsf{aux}_{\mathsf{ret}}, \hat{\mathsf{tx}}_{\mathrm{fin}}; \mathsf{st}_{\mathcal{S}}) \leftarrow \mathsf{InitRet}(\mathsf{tx}^n_{\mathrm{adt}}, \langle \mathcal{C}(sk_{\mathcal{C}}), \mathcal{S}(R) \rangle)$
assert $\mathsf{RetVf}(\mathsf{tx}^n_{\mathrm{adt}}, C, \mathsf{aux}_{\mathsf{ret}}, \hat{\mathsf{tx}}_{\mathrm{fin}}) = 1$
$\mathsf{tx}_{\mathrm{fin}} \leftarrow \mathsf{Finalize}(\mathsf{st}_{\mathcal{S}}, \hat{\mathsf{tx}}_{\mathrm{fin}})$
assert $\mathsf{tx}_{\mathrm{fin}} \in \mathcal{L}$; $(R', D') \leftarrow \mathsf{RetExt}(C, \hat{\mathsf{tx}}_{\mathrm{fin}}, \mathsf{tx}_{\mathrm{fin}})$
return $(R' = R) \wedge (D' = D)$

Fig. 2: The correctness experiment for $\mathsf{DSN}_{\mathcal{L}}$ on inputs (D, τ_D, md), using $\mathsf{PoRep.TagVf}$ for data-tag verification. The experiment submits each completed transaction according to the agreed schedule and allows u rounds for k-confirmation before evaluating $\mathsf{tx} \in \mathcal{L}$. The parties' key pairs and funding transactions $\mathsf{tx}_{\mathrm{pay}}, \mathsf{tx}_{\mathrm{col}}$ are implicit; the designated funding outputs are initially unspent and controlled by the corresponding party. In the outputs of OffDel and $\mathsf{InitRet}$, the semicolon separates the common outputs from the server-local output.

Definition 5 (Correctness). *A blockchain-aided DSN protocol is correct if, for any λ, there exists t_λ s.t. the following holds for any $t \geq t_\lambda$, $(\mathsf{pp}, \cdot) \leftarrow \mathsf{Setup}(1^\lambda, t)$, any supported preprocessed data-tag pair (D, τ_D), any contract metadata md whose validity intervals are set according to the audit interval $T(\lambda, t) \in \mathsf{pp}$, with vi_{con} beginning at the experiment's initial ledger round, and any honest parties with correctly generated keys and funding outputs.*

$$\Pr\left[\mathrm{Correct}^{\mathcal{L}}_{\mathsf{PoRep.TagVf}}(\mathsf{pp}, D, \tau_D, \mathsf{md}) = 1\right] = 1$$

3.2 Security Properties

Each security experiment maintains an isolated ledger view $\mathcal{L}_{\text{exp}}$, initialized from a valid snapshot of $\mathcal{L}$. The experiment proceeds in synchronous rounds, with a local round counter initialized to the snapshot's round. All transaction validity intervals and protocol deadlines are evaluated against this counter. All ledger-dependent checks and protocol submissions use $\mathcal{L}_{\text{exp}}$; the experiment neither modifies $\mathcal{L}$ nor depends on its subsequent evolution.

A completed transaction submitted to $\mathcal{L}_{\text{exp}}$ undergoes the validation checks specified in Section 2.1, using the current experimental view and round. If accepted, it is recorded in $\mathcal{L}_{\text{exp}}$, its consumed outputs are marked as spent, and its newly created outputs are marked as unspent. Otherwise, the view remains unchanged. We write $\mathsf{tx} \in \mathcal{L}_{\text{exp}}$ for acceptance into this experimental view, not k-confirmation on the underlying ledger. Producing or disclosing a transaction does not itself submit it. Any confirmation delays and liveness guarantees used by an experiment are specified separately.

Since parties may use their signing keys to sign messages other than DSN transactions, we grant $\mathcal{A}$ access to the following chosen-message signing oracle.

– $\mathcal{O}_{\mathsf{Sign}}^{sk_{\mathcal{C}}, sk_{\mathcal{S}}, Q_{\mathsf{Sign}}}(\mathcal{H}, m)$. On input an honest-party indicator $\mathcal{H} \in \{\mathcal{C}, \mathcal{S}\}$ and an arbitrary message m chosen by $\mathcal{A}$, the oracle computes $\sigma \leftarrow \mathsf{SIG.Sign}(sk_{\mathcal{H}}, m)$, records $(\mathcal{H}, m)$ in Q_{Sign}, and returns σ.

Security of delegation: contract unforgeability. Contract unforgeability ensures that a malicious party cannot produce a valid contract transaction that spends its honest counterparty's funding output unless that counterparty has authorized the same base transaction. The experiment is given in Fig. 3.

$\underline{\text{Experiment } \text{ConUF}_{\mathcal{A}}^{\mathcal{L}_{\text{exp}}}(\mathsf{pp}, pk_{\mathcal{C}}, pk_{\mathcal{S}}, \mathsf{tx}_{\text{pay}}, \mathsf{tx}_{\text{col}}) :}$
return 0 **if** $\mathsf{tx}_{\text{pay}} \notin \mathcal{L}_{\text{exp}} \vee \mathsf{tx}_{\text{col}} \notin \mathcal{L}_{\text{exp}}$
$Q_{\mathsf{Sign}} \coloneqq \emptyset$
submit $\mathsf{tx}^*_{\text{con}} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}^{sk_{\mathcal{C}}, sk_{\mathcal{S}}, Q_{\mathsf{Sign}}}}(\mathsf{pp}, pk_{\mathcal{C}}, pk_{\mathcal{S}}, \mathsf{tx}_{\text{pay}}, \mathsf{tx}_{\text{col}})$
$b_1 \coloneqq (\mathsf{out}^*_{\text{pay}} = \mathsf{out}_{\text{pay}}) \wedge ((\mathcal{C}, \tilde{\mathsf{tx}}^*_{\text{con}}) \notin Q_{\mathsf{Sign}})$
$b_2 \coloneqq (\mathsf{out}^*_{\text{col}} = \mathsf{out}_{\text{col}}) \wedge ((\mathcal{S}, \tilde{\mathsf{tx}}^*_{\text{con}}) \notin Q_{\mathsf{Sign}})$
return $(\mathsf{tx}^*_{\text{con}} \in \mathcal{L}_{\text{exp}}) \wedge (b_1 \vee b_2)$

Fig. 3: The contract unforgeability experiment. The designated funding outputs $\mathsf{out}_{\text{pay}}$, $\mathsf{out}_{\text{col}}$ are initially unspent and each requires its owner's signature on the base contract transaction $\tilde{\mathsf{tx}}^*_{\text{con}}$. The candidate $\mathsf{tx}^*_{\text{con}}$ is parsed according to OnDel, with $\mathsf{out}^*_{\text{pay}}, \mathsf{out}^*_{\text{col}}$ denoting its payment and collateral input references, respectively.

Definition 6 (Contract Unforgeability). *A blockchain-aided DSN protocol satisfies contract unforgeability if, for any PPT adversary $\mathcal{A}$ with access to $\mathcal{O}_{\mathsf{Sign}}$, the following holds for any $\lambda, t > 0$, $(\mathsf{pp}, \cdot) \leftarrow \mathsf{Setup}(1^\lambda, t)$, key pairs $(sk_{\mathcal{C}}, pk_{\mathcal{C}})$ and $(sk_{\mathcal{S}}, pk_{\mathcal{S}})$ independently generated by $\mathsf{SIG.KGen}(1^\lambda)$, and any funding transactions $\mathsf{tx}_{\text{pay}}, \mathsf{tx}_{\text{col}} \in \mathcal{L}$ satisfying the conditions in Fig. 3.*

$$\Pr\left[\text{ConUF}_{\mathcal{A}}^{\mathcal{L}_{\text{exp}}}(\mathsf{pp}, pk_{\mathcal{C}}, pk_{\mathcal{S}}, \mathsf{tx}_{\text{pay}}, \mathsf{tx}_{\text{col}}) = 1\right] \leq \mathsf{negl}(\lambda)$$

Repeated audits on the ledger: adaptive replication soundness. Repeated audits in the preservation phase produce a growing public transaction transcript that a malicious server may reuse as external storage while adapting its retained local state. We extend the replication soundness notion in Def. 4 to account for this evolution, requiring data recovery and storage lower bounds at the checkpoints preceding accepted audits.[12]

A direct extension of the baseline's k-instance formulation would need to track a shared retained state across instances at potentially different audit stages: advancing one instance does not necessarily advance the others. To simplify the storage accounting, we follow a single target delegation and measure the permitted shrinkage along its own audit sequence. The following oracles allow the adversary to establish the target delegation with an honest client on data of its choice and obtain simulated audit proofs for other instances. These instances may concern identical data under distinct identifiers.

- $\mathcal{O}^{sk_{\mathcal{C}}}_{\mathsf{CliDel}}(D, \tau_D, \mathsf{md})$. On input a preprocessed data-tag pair (D, τ_D) and agreed contract metadata md, the oracle returns $\perp$ if $\mathsf{PoRep.TagVf}(D, \tau_D) = 0$. Otherwise, it executes the delegation phase with $\mathcal{A}$ acting as the server:
$$(m_{\mathrm{con}}; \cdot) \leftarrow \mathsf{OffDel}(\mathsf{md}, \langle \mathcal{C}(D, \tau_D), \mathcal{A}() \rangle),$$
$$\mathsf{tx}_{\mathrm{con}} \leftarrow \mathsf{OnDel}(m_{\mathrm{con}}, \mathsf{tx}_{\mathrm{pay}}, \mathsf{tx}_{\mathrm{col}}, \langle \mathcal{C}(sk_{\mathcal{C}}), \mathcal{A}(sk_{\mathcal{A}}) \rangle),$$
and **submit** $\mathsf{tx}_{\mathrm{con}}$. The funding transactions and the adversary's local key are supplied implicitly. The oracle returns $\mathsf{tx}_{\mathrm{con}}$ on successful completion, or $\perp$ otherwise.
- $\mathcal{O}^{\mathsf{td}, id, Q_{\mathsf{SimAudit}}}_{\mathsf{SimAudit}}(m^*_{\mathrm{con}})$, where id identifies the target delegation established via $\mathcal{O}_{\mathsf{CliDel}}$. On input a contract message $m^*_{\mathrm{con}} = (\mathsf{md}^*, \tau^*_D, \mathsf{aux}^*)$ chosen by $\mathcal{A}$, the oracle parses $\mathsf{md}^* = (id^*, n^*, V^*)$ and returns $\perp$ if $\mathsf{DelVf}(m^*_{\mathrm{con}}) = 0 \vee id^* = id$. Otherwise, it initializes an auxiliary contract transaction carrying m^*_{con} and its initial state output. For each $j \in [n^*]$, it evaluates the seed derivation underlying $\mathsf{SeedGen}^j$ using the preceding auxiliary state output's reference and datum, obtaining the seed and its verification evidence. It then derives the corresponding challenge and uses td to simulate the proof π^{j*} so that $\mathsf{AuditVf}^j$ accepts. The oracle forms the corresponding auxiliary audit transaction, whose state output supplies the predecessor for the next audit. It returns $\{\pi^{j*}\}_{j \in [n^*]}$ and records $(m^*_{\mathrm{con}}, \{\pi^{j*}\}_{j \in [n^*]})$ in Q_{SimAudit}.

The auxiliary transactions remain internal to $\mathcal{O}_{\mathsf{SimAudit}}$. This oracle models externally available audit proofs. Its internal computation is idealized: it neither advances the experimental round counter nor counts toward the target online invocation's time budget.

The adaptive replication soundness experiment is given in Fig. 4. Checkpoint i is the point immediately before $\mathcal{A}^{i+1}_{\mathrm{onl}}$, following successful delegation and i accepted target audits.

[12] The auxiliary input z can encode prior execution history, but does not by itself capture access to an evolving public transcript that remains available online without being counted as local storage.

Experiment $\mathrm{AdaptRepSound}^{\mathcal{L}_{\mathrm{exp}}}_{\mathsf{PoRep},\mathcal{A},\mathcal{E}}(\mathsf{pp},\mathsf{td},z)$:

$(D,\eta^0,\mathsf{tx}_{\mathrm{con}}) \leftarrow \mathcal{A}_{\mathrm{pre}}^{\mathcal{O}^{sk_{\mathcal{C}}}_{\mathsf{CliDel}}}(z)$

return 0 **if** $\mathsf{tx}_{\mathrm{con}} \notin \mathcal{L}_{\mathrm{exp}}$

parse $m_{\mathrm{con}} = (\mathsf{md},\tau_D,\mathsf{aux})$ from $\delta^{\mathrm{col}}_{\mathrm{con}}$, $\mathsf{md} = (id, n, V)$

$(R,\cdot) \leftarrow \mathsf{PoRep.Replicate}(id, D, \tau_D)$

$\mathsf{tx}^0_{\mathrm{adt}} \coloneqq \mathsf{tx}_{\mathrm{con}}$; $\mathsf{out}^0_{\mathrm{adt}} \coloneqq \mathsf{out}^{\mathrm{col}}_{\mathrm{con}}$; $l \coloneqq 0$; $Q_{\mathsf{SimAudit}} \coloneqq \emptyset$

for $i \in [n]$:

  $(\eta^i, \mathsf{tx}^i_{\mathrm{adt}}, \hat{z}^{i-1}) \leftarrow \mathcal{A}^{i,\mathcal{O}^{\mathsf{td},id,Q_{\mathsf{SimAudit}}}_{\mathsf{SimAudit}}}_{\mathrm{onl}}(id, \eta^{i-1}, \mathsf{aux}, \mathsf{tr}^{i-1})$

  submit $\mathsf{tx}^i_{\mathrm{adt}}$; **parse** $\mathsf{in}^i_{\mathrm{adt}} = (\mathsf{out}', \cdot)$

  $v_i \coloneqq \mathsf{tx}^i_{\mathrm{adt}} \in \mathcal{L}_{\mathrm{exp}} \wedge \mathsf{out}' = \mathsf{out}^{i-1}_{\mathrm{adt}}$

  break if $v_i = 0$; $l \coloneqq i$

return 0 **if** $l = 0$

$(\{R'^i, \omega^i\}_{i\in[l-1]_0}, \mathsf{Decode}^*)$

  $\leftarrow \mathcal{E}^{\{\mathcal{A}^{i+1}_{\mathrm{onl}}[\eta^i]\}_{i\in[l-1]_0}}(id, \tau_D, \mathsf{aux}, \{\mathsf{tr}^i\}_{i\in[l-1]_0})$

$D'^i \coloneqq \mathsf{PoRep.Extract}(id, \tau_D, R'^i), \forall i \in [l-1]_0$

$\ell_i \coloneqq \max\{(1-\epsilon')^i|R|, |R'^i|\}, \forall i \in [l-1]_0$

return $\big(\exists i \in [l-1]_0 : (D'^i \neq D)$

  $\vee\ \mathsf{Decode}^*(R'^i, \omega^i, \mathsf{tr}^i) \neq \hat{z}^i$

  $\vee\ |\eta^i| < (1-\epsilon)(\ell_i + |\omega^i| + |\mathsf{Decode}^*|)\big)$

  $\vee\ (\exists i \in [l-1] : |R'^i| < (1-\epsilon')|R'^{(i-1)}|)$

// extraction failure, auxiliary output recovery failure, storage bound violation

// excessive replica shrinkage

Fig. 4: The adaptive replication soundness experiment. Unchanged conventions follow Fig. 1. The preprocessing output must match a successful delegation established through $\mathcal{O}_{\mathsf{CliDel}}$. At each reached checkpoint i, tr^i contains $\{\mathsf{tx}^j_{\mathrm{adt}}\}_{j\in[i]_0}$ and the simulated responses recorded in Q_{SimAudit} before audit $i+1$. The adversary's external storage is restricted to tr^i. Extractor access follows Remark 1 separately at each checkpoint, except that replica extraction may use non-black-box access to $\mathcal{A}^{i+1}_{\mathrm{onl}}[\eta^i]$, with its retained state, public inputs, and tr^i fixed across invocations. Rewinding neither modifies the recorded execution nor permits later states or transcripts to be used for an earlier checkpoint. The measure $|\eta^i|$ counts all *locally* retained data-dependent information exactly once, including the separately displayed identifier and auxiliary information unless already supplied through tr^i.

Definition 7 (Admissible DSN Adversary). *A PPT adversary $\mathcal{A} = (\mathcal{A}_{\mathrm{pre}}, \mathcal{A}_{\mathrm{onl}})$ is admissible if it chooses a polynomially bounded $n \geq 1$ and each online invocation completes within $T(\lambda, t)$ ledger rounds. It is μ-admissible if $\Pr[v_{i+1} = 1] \geq \mu$ for any $\lambda, t > 0$, $(\mathsf{pp},\mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, t)$, any polynomial-length static auxiliary input z, and every execution prefix in the support of the experiment that reaches*

a checkpoint $i \in [n-1]_0$. The probability is over the experiment randomness for that fixed execution prefix.

Definition 8 (Adaptive Replication Soundness). *Let $\epsilon, \epsilon' \in [0,1)$ and $\mu \in (0,1]$. A blockchain-aided DSN protocol satisfies $(\epsilon, \epsilon', \mu)$-adaptive replication soundness if, for any μ-admissible adversary $\mathcal{A} = (\mathcal{A}_{\mathrm{pre}}, \mathcal{A}_{\mathrm{onl}})$, there exists a PPT extractor $\mathcal{E}$, with the access specified in Fig. 4, s.t. the following holds for any $\lambda, t > 0$, $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, t)$, and any polynomial-length static auxiliary input z.*

$$\Pr\left[\mathrm{AdaptRepSound}^{\mathcal{L}_{\mathrm{exp}}}_{\mathsf{PoRep},\mathcal{A},\mathcal{E}}(\mathsf{pp}, \mathsf{td}, z) = 1\right] \leq \mathsf{negl}(\lambda)$$

$\mathcal{E}$ is fixed independently of z and the sampled execution.

Against malicious servers in retrieval: adaptive client fairness. Adaptive client fairness requires that a malicious server cannot produce a valid finalization transaction releasing its collateral without the client's authorization. It further requires that, if retrieval artifacts have been accepted by RetVf, such finalization enables the client to recover the delegated data. The adversary may establish delegations on data of its choice via $\mathcal{O}_{\mathsf{CliDel}}$. We define two additional oracles: $\mathcal{O}_{\mathsf{SimAux}}$ provides simulated retrieval proofs, and $\mathcal{O}_{\mathsf{CliInitRet}}$ models adaptive interactions with the honest client to obtain conditional authorizations for established delegations.

- $\mathcal{O}^{\mathsf{td}, Q_{\mathsf{SimAux}}}_{\mathsf{SimAux}}(m_{\mathrm{con}}, C, x_{\mathsf{ret}})$. On input a statement $(m_{\mathrm{con}}, C, x_{\mathsf{ret}})$ chosen by $\mathcal{A}$, the oracle returns $\perp$ if $\mathsf{DelVf}(m_{\mathrm{con}}) = 0$. Otherwise, it uses td to produce a simulated retrieval proof π_{ret} for this statement. It returns $\mathsf{aux}_{\mathsf{ret}} := (x_{\mathsf{ret}}, \pi_{\mathsf{ret}})$ and records $(m_{\mathrm{con}}, C, \mathsf{aux}_{\mathsf{ret}})$ in Q_{SimAux}.
- $\mathcal{O}^{sk_{\mathcal{C}}, Q_{\mathsf{CliInitRet}}}_{\mathsf{CliInitRet}}(\mathsf{tx}^n_{\mathrm{adt}})$. On input $\mathsf{tx}^n_{\mathrm{adt}}$, the oracle returns $\perp$ unless it is recorded in $\mathcal{L}_{\mathrm{exp}}$ as the final audit transaction of a delegation established through $\mathcal{O}_{\mathsf{CliDel}}$, and its collateral state output is unspent. Otherwise, it executes $\mathsf{InitRet}$ with $\mathcal{A}$ acting as the server:

  $$(C, \mathsf{aux}_{\mathsf{ret}}, \hat{\mathsf{tx}}_{\mathrm{fin}}; \cdot) \leftarrow \mathsf{InitRet}(\mathsf{tx}^n_{\mathrm{adt}}, \langle \mathcal{C}(sk_{\mathcal{C}}), \mathcal{A}() \rangle),$$

  Whenever the client releases the conditional authorization in $\hat{\rho}_{\mathrm{fin}}$, the oracle evaluates:

  $$b_{\mathsf{ret}} := \mathsf{RetVf}(\mathsf{tx}^n_{\mathrm{adt}}, C, \mathsf{aux}_{\mathsf{ret}}, \hat{\mathsf{tx}}_{\mathrm{fin}})$$

  at that time and records $(\mathsf{out}^n_{\mathrm{adt}}, C, \mathsf{aux}_{\mathsf{ret}}, \hat{\mathsf{tx}}_{\mathrm{fin}}, b_{\mathsf{ret}})$ in $Q_{\mathsf{CliInitRet}}$ from the client's local view, even if the interaction subsequently aborts.

The adaptive client fairness experiment is given in Fig. 5.

Definition 9 (Adaptive Client Fairness). *A blockchain-aided DSN protocol satisfies adaptive client fairness if, for any PPT adversary $\mathcal{A}$ with the oracle access specified in Fig. 5, the following holds for any $\lambda, t > 0$, $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, t)$, and any $(sk_{\mathcal{C}}, pk_{\mathcal{C}}) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$.*

$$\Pr\left[\mathrm{AdaptCliFair}^{\mathcal{L}_{\mathrm{exp}}}_{\mathcal{A}}(\mathsf{pp}, \mathsf{td}) = 1\right] \leq \mathsf{negl}(\lambda)$$

Against malicious clients in retrieval: server fairness. A malicious client may receive the server's retrieval messages and then refuse to authorize collateral

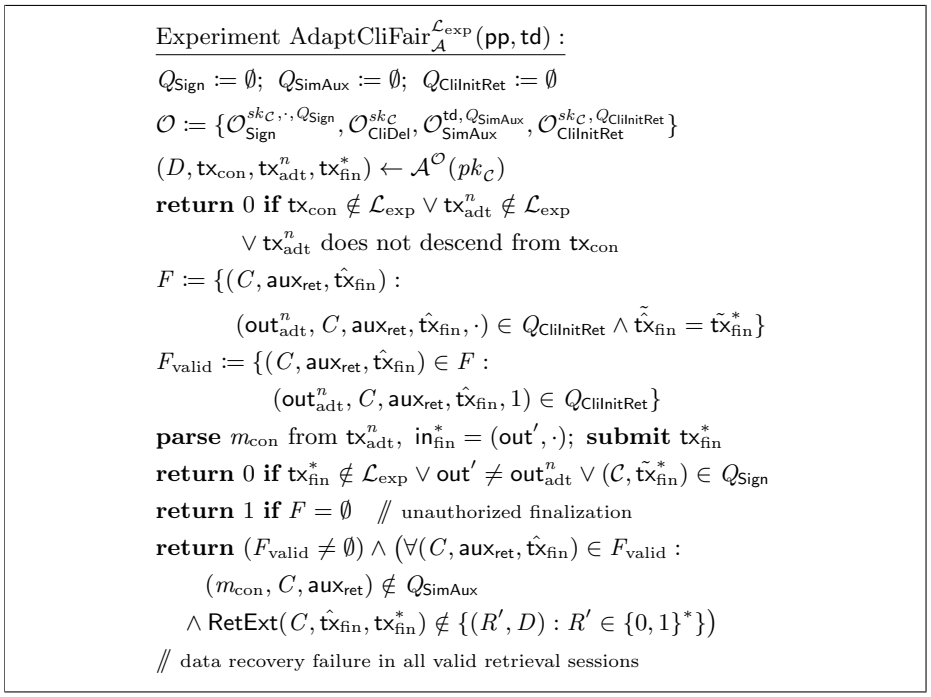

Fig. 5: The adaptive client fairness experiment. The selected $(D, \mathsf{tx}_{\mathrm{con}})$ must match a successful delegation established through $\mathcal{O}_{\mathsf{CliDel}}$. For each delegation, the adversary submits $(\mathsf{tx}_{\mathrm{adt}}^{i})_{i\in[n]}$ to $\mathcal{L}_{\mathrm{exp}}$ in order before invoking $\mathcal{O}_{\mathsf{CliInitRet}}$ on $\mathsf{tx}_{\mathrm{adt}}^{n}$. Descent refers to the n prescribed audit transitions rooted at $\mathsf{out}_{\mathrm{con}}^{\mathrm{col}}$.

release. Server fairness requires that these messages reveal no additional information about the delegated data before finalization. We formalize this requirement through simulation in the experiment of Fig. 6.

Definition 10 (Server Fairness). *A blockchain-aided DSN protocol satisfies server fairness if there exists a PPT simulator* Sim *s.t., for any PPT adversary* $\mathcal{A}$ *choosing a polynomially bounded* n *and any* $\lambda, t > 0$*, with* $(\mathsf{pp},\mathsf{td}) \leftarrow \mathsf{Setup}(1^{\lambda}, t)$ *and* $(sk_{\mathcal{S}}, pk_{\mathcal{S}}) \leftarrow \mathsf{SIG.KGen}(1^{\lambda})$*, the following holds for the output ensembles of Fig. 6.*

$$\{\mathsf{View}_0\}_{\lambda} \approx_c \{\mathsf{View}_1\}_{\lambda}$$

Remark 4 (Scope of server fairness). A client may retain the delegated data or obtain it from another server; cryptography alone cannot prevent it from aborting the target retrieval session on that basis. Our guarantee therefore concerns only the additional information disclosed by the target retrieval session before finalization. Moreover, we do not provide a CCA-style decryption oracle. The honest server generates the concealed replica using fresh session-specific secret state and provides no interface for recovering data from client-chosen concealed replicas using that state.

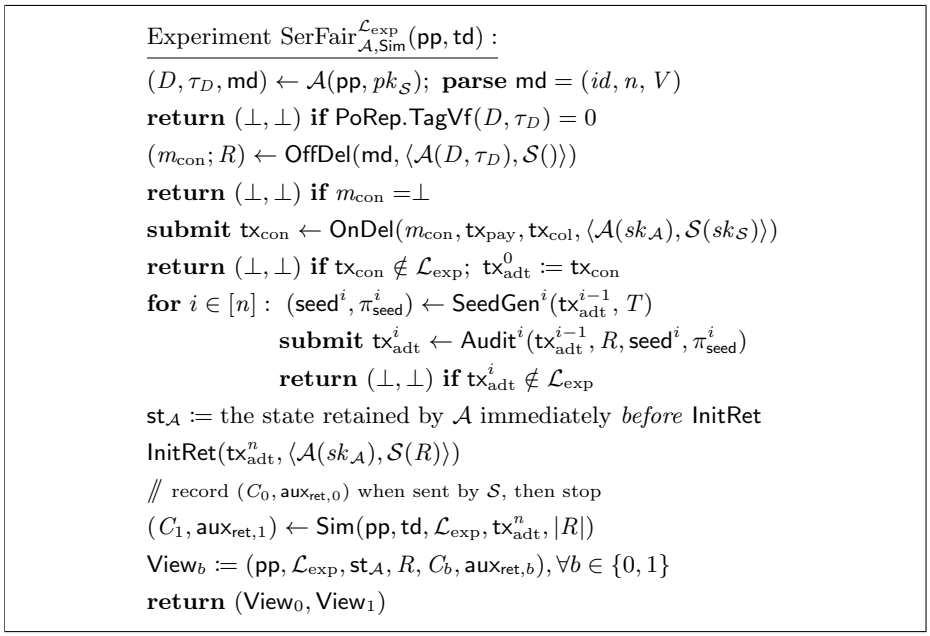

Fig. 6: The server fairness experiment. The parties' local keys and funding transactions are implicit; delegation and preservation follow the agreed schedule. $\mathsf{st}_{\mathcal{A}}$ includes the adversary's local state and messages from delegation and preservation.

4 Generic Construction and Security Analysis

This section presents our generic DSN construction and analyzes its security. We state the properties required of each building block without fixing concrete instantiations.

Building blocks and instantiations. We use a public-coin PoRep scheme PoRep whose replication soundness is derived from its implicit PoRC and PoSpace (Appendix B.2) under the KoSC assumption.[13] We instantiate PoRep.Poll as a domain-separated hash function $\mathsf{H}_{\mathsf{Poll}}$ into the PoRep challenge space and model it as a random oracle. For preservation, we derive successive audit seeds using a VDF scheme $\mathsf{VDF} = (\mathsf{Setup},\mathsf{Eval},\mathsf{Vf})$, following the PoST approach and timing calibration described in Section 2.2. For retrieval, we combine symmetric encryption $\mathsf{SE} = (\mathsf{KGen},\mathsf{Enc},\mathsf{Dec})$, commitment scheme $\mathsf{COM} = (\mathsf{Setup},\mathsf{Commit},\mathsf{Vf})$, and signature scheme $\mathsf{SIG} = (\mathsf{KGen},\mathsf{Sign},\mathsf{Vf})$ to implement conditional authorization. The mechanism follows the encryption-then-adaptor-signature approach of [37] and is inspired by the generic adaptor signature construction of [11]. Storage and retrieval proofs use zkSNARKs Π_{PoRep} and Π_{ret}, respectively. Both have the interface $(\mathsf{Setup},\mathsf{Prove},\mathsf{Vf})$ and satisfy simulation-extractability.

[13] Concrete PoRep can be constructed using graph labeling [18,19].

4.1 Protocol Construction

We describe each protocol phase together with its transactions and validator checks. Throughout, we implicitly parse $m_{\text{con}} = (\mathsf{md}, \tau_D, \mathsf{aux})$, $\mathsf{md} = (id, n, V)$, and $\mathsf{aux} = (\tau_R, \mathsf{seed}_{\mathsf{aux}}, \pi_{\mathsf{aux}})$. On parsing failure, each procedure returns its designated failure value, either $\perp$ or 0. Submitting a transaction implicitly returns it as the procedure's output.

Setup and preprocessing. They are specified in Fig. 7. The relations $\mathcal{R}_{\mathsf{PoRep}}$ and $\mathcal{R}_{\mathsf{ret}}$ are defined in the delegation and retrieval phases, respectively. The designated burning address $pk_{\mathcal{B}}$ and audit interval $T(\lambda, t)$ are as in the syntax.

$\underline{\mathsf{Setup}(1^\lambda, t):}$

$\mathsf{pp}_{\mathsf{PoRep}} \leftarrow \mathsf{PoRep.Setup}(1^\lambda, t)$

$T \coloneqq T(\lambda, t);\ T_{\mathsf{VDF}} \coloneqq T_{\mathsf{VDF}}(T)$

$\mathsf{pp}_{\mathsf{VDF}} \coloneqq (ek_{\mathsf{VDF}}, vk_{\mathsf{VDF}}) \leftarrow \mathsf{VDF.Setup}(1^\lambda, T_{\mathsf{VDF}})$

$ck \leftarrow \mathsf{COM.Setup}(1^\lambda)$

$(\mathsf{crs}_{\mathsf{PoRep}}, \mathsf{td}_{\mathsf{PoRep}}) \leftarrow \Pi_{\mathsf{PoRep}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathsf{PoRep}})$

$(\mathsf{crs}_{\mathsf{ret}}, \mathsf{td}_{\mathsf{ret}}) \leftarrow \Pi_{\mathsf{ret}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathsf{ret}})$

$\mathsf{pp} \coloneqq (\mathsf{pp}_{\mathsf{PoRep}}, \mathsf{pp}_{\mathsf{VDF}}, ck, \mathsf{crs}_{\mathsf{PoRep}}, \mathsf{crs}_{\mathsf{ret}}, pk_{\mathcal{B}}, T, T_{\mathsf{VDF}})$

return $(\mathsf{pp}, \mathsf{td} \coloneqq (\mathsf{td}_{\mathsf{PoRep}}, \mathsf{td}_{\mathsf{ret}}))$

$\underline{\text{Client preprocessing } (ek, \tilde{D}):}$

return $(D, \tau_D) \leftarrow \mathsf{PoRep.Preproc}(ek, \tilde{D})$

Fig. 7: Setup and client preprocessing. The VDF delay parameter T_{VDF} is calibrated to the audit interval T in ledger rounds. pp is distributed to protocol participants, while td is retained by the security experiment. The client retains ek when required to recover $\tilde{D}$ from D.

Delegation. The delegation phase is specified in Fig. 8, with the funding validators of tx_{pay} and tx_{col} defined by:

$$\nu_{\text{pay}}(\delta_{\text{pay}}, \rho^{\text{pay}}_{\text{con}}, \mathsf{tx}_{\text{con}}) \coloneqq \mathsf{SIG.Vf}(pk_{\mathcal{C}}, \tilde{\mathsf{tx}}_{\text{con}}, \rho^{\text{pay}}_{\text{con}}),$$
$$\nu_{\text{col}}(\delta_{\text{col}}, \rho^{\text{col}}_{\text{con}}, \mathsf{tx}_{\text{con}}) \coloneqq \mathsf{SIG.Vf}(pk_{\mathcal{S}}, \tilde{\mathsf{tx}}_{\text{con}}, \rho^{\text{col}}_{\text{con}}).$$

During OffDel, the server generates the replica and uses Π_{PoRep} to compress the initial PoRep response into a *succinct* proof of its validity. We use τ_R to denote the auxiliary information returned by $\mathsf{PoRep.Replicate}$ and reserve aux for the contract auxiliary information. A statement $x = (id, \tau_D, \tau_R, \mathsf{ch})$ and witness $w = \pi_{\mathsf{PoRep}}$ satisfy $\mathcal{R}_{\mathsf{PoRep}}$, *i.e.*, $\mathcal{R}_{\mathsf{PoRep}}(x, w) = 1$ if and only if:

$$\mathsf{PoRep.Vf}(id, \tau_D, \tau_R, \mathsf{ch}, \pi_{\mathsf{PoRep}}) = 1.$$

The same relation is used for preservation audits. During OnDel, the parties lock the payment and collateral under $\nu^{\text{pay}}_{\text{con}}$ and $\nu^{\text{col}}_{\text{con}}$, respectively. The former implements the agreed payment policy. We set $\nu^{\text{col}}_{\text{con}} \coloneqq \nu^0_{\text{adt}}$, the initial collateral validator, whose checks are defined in the preservation phase below.

For the first preservation audit, the input to $\mathsf{SeedGen}^1$ must depend on the client's authorization of tx_{con} and remain unavailable to the server before that

authorization is released. If the client were to sign first, a malicious server could complete the transaction privately and begin VDF evaluation while delaying submission, gaining additional computation time. Under the prescribed signing order, if the client withholds its signature, delegation does not complete and the collateral is not locked under the contract.

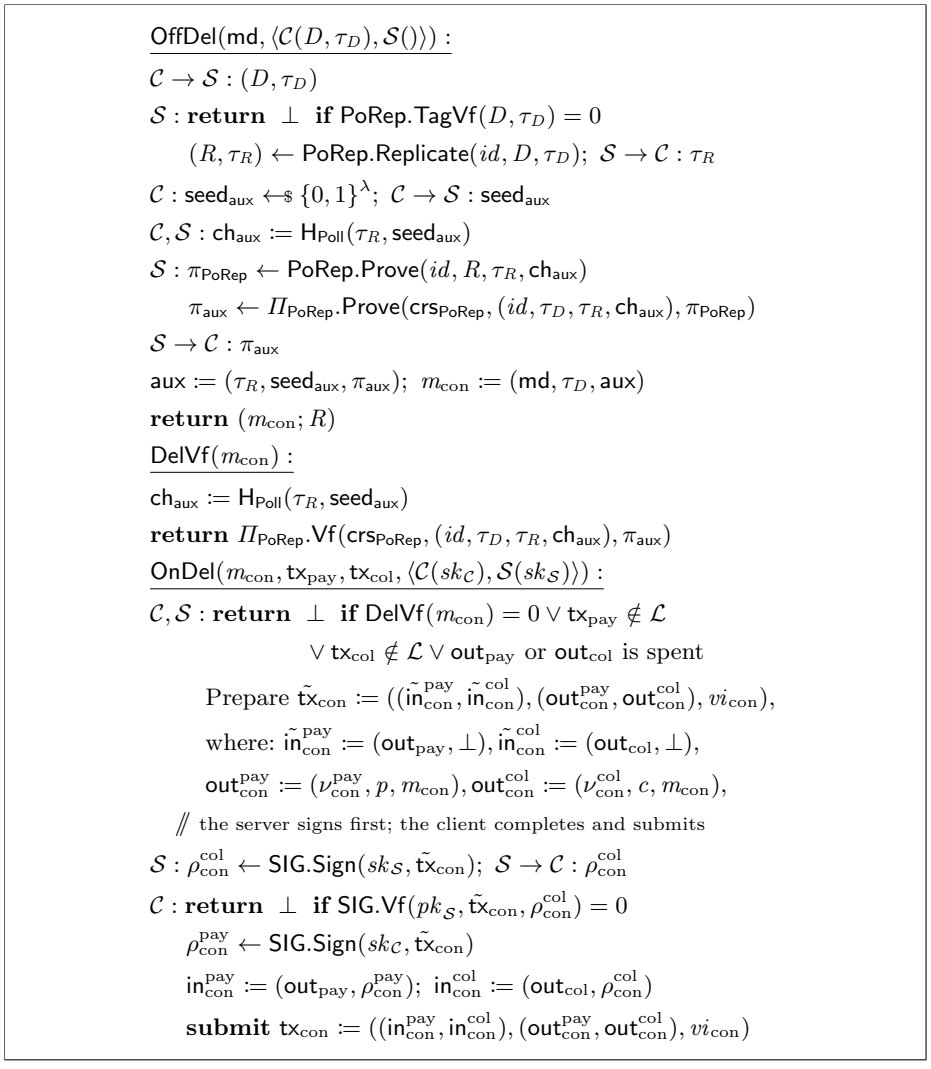

Fig. 8: The delegation phase. The initial challenge-response exchange augments the output τ_R to form the contract auxiliary information aux.

Preservation. Put $\mathsf{tx}^0_{\mathrm{adt}} := \mathsf{tx}_{\mathrm{con}}$ and $\mathsf{out}^0_{\mathrm{adt}} := \mathsf{out}^{\mathrm{col}}_{\mathrm{con}}$. The preservation phase is specified in Fig. 9. Each audit derives its seed by evaluating the VDF on the preceding state output's reference $\mathsf{out}^{i-1}_{\mathrm{adt}}$. This reference must bind the client's

authorization for the first audit and the preceding audit response thereafter. The server authenticates each audit transaction by signing its base.

For each $i \in [n]$, $\nu_{\mathrm{adt}}^{i-1}(m_{\mathrm{con}}, \rho, \mathsf{tx})$ has two branches:

– *Audit branch.* The expected redeemer is $\rho = (\mathsf{seed}^i, \pi_{\mathsf{seed}}^i, \mathsf{ch}^i, \pi^i, \sigma^i)$. The branch accepts if $\mathsf{tx}.vi = vi_{\mathrm{adt}}^i$, tx creates exactly one successor state output $(\nu_{\mathrm{adt}}^i, c, m_{\mathrm{con}})$, and:

$$\mathsf{AuditVf}^i(\mathsf{out}_{\mathrm{adt}}^{i-1}, \mathsf{seed}^i, \pi_{\mathsf{seed}}^i, \mathsf{ch}^i, \pi^i) = 1$$
$$\wedge \mathsf{SIG.Vf}(pk_{\mathcal{S}}, \tilde{\mathsf{tx}}, \sigma^i) = 1.$$

– *Timeout branch.* The expected redeemer is a signature under $pk_{\mathcal{B}}$. The branch accepts if tx creates no successor state output, its validity interval starts after vi_{adt}^i ends, and $\mathsf{SIG.Vf}(pk_{\mathcal{B}}, \tilde{\mathsf{tx}}, \rho) = 1$.

Each branch rejects redeemers that do not have the required form. The validator returns 1 if either branch accepts, or 0 otherwise. Here, $\tilde{\mathsf{tx}}$ is the base transaction obtained from tx by replacing its signature component with $\perp$. Since the secret key for $pk_{\mathcal{B}}$ is unavailable, expired collateral remains locked. The terminal validator ν_{adt}^n is specified in the retrieval phase below.

$/\!/$ parse $\mathsf{out}_{\mathrm{adt}}^{i-1} = (\nu_{\mathrm{adt}}^{i-1}, c, m_{\mathrm{con}})$ from $\mathsf{tx}_{\mathrm{adt}}^{i-1}$

$\underline{\mathsf{SeedGen}^i(\mathsf{tx}_{\mathrm{adt}}^{i-1}, T):}$

return $\perp$ **if** $\mathsf{tx}_{\mathrm{adt}}^{i-1} \notin \mathcal{L}$

return $(\mathsf{seed}^i, \pi_{\mathsf{seed}}^i) \leftarrow \mathsf{VDF.Eval}(ek_{\mathsf{VDF}}, \mathsf{out}_{\mathrm{adt}}^{i-1})$

$\underline{\mathsf{Audit}^i(\mathsf{tx}_{\mathrm{adt}}^{i-1}, R, \mathsf{seed}^i, \pi_{\mathsf{seed}}^i):}$

return $\perp$ **if** $\mathsf{tx}_{\mathrm{adt}}^{i-1} \notin \mathcal{L} \vee \mathsf{out}_{\mathrm{adt}}^{i-1}$ is spent

$\mathsf{ch}^i \coloneqq \mathsf{H}_{\mathsf{Poll}}(\tau_R, \mathsf{seed}^i)$

$\pi_{\mathsf{PoRep}} \leftarrow \mathsf{PoRep.Prove}(id, R, \tau_R, \mathsf{ch}^i)$

$\pi^i \leftarrow \Pi_{\mathsf{PoRep}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{PoRep}}, (id, \tau_D, \tau_R, \mathsf{ch}^i), \pi_{\mathsf{PoRep}})$

$\tilde{\rho}_{\mathrm{adt}}^i \coloneqq (\mathsf{seed}^i, \pi_{\mathsf{seed}}^i, \mathsf{ch}^i, \pi^i, \perp)$; $\mathsf{out}_{\mathrm{adt}}^i \coloneqq (\nu_{\mathrm{adt}}^i, c, m_{\mathrm{con}})$

$\tilde{\mathsf{tx}}_{\mathrm{adt}}^i \coloneqq ((\mathsf{out}_{\mathrm{adt}}^{i-1}, \tilde{\rho}_{\mathrm{adt}}^i), \mathsf{out}_{\mathrm{adt}}^i, vi_{\mathrm{adt}}^i)$

$\sigma^i \leftarrow \mathsf{SIG.Sign}(sk_{\mathcal{S}}, \tilde{\mathsf{tx}}_{\mathrm{adt}}^i)$

$\rho_{\mathrm{adt}}^i \coloneqq (\mathsf{seed}^i, \pi_{\mathsf{seed}}^i, \mathsf{ch}^i, \pi^i, \sigma^i)$

submit $\mathsf{tx}_{\mathrm{adt}}^i \coloneqq ((\mathsf{out}_{\mathrm{adt}}^{i-1}, \rho_{\mathrm{adt}}^i), \mathsf{out}_{\mathrm{adt}}^i, vi_{\mathrm{adt}}^i)$

$\underline{\mathsf{AuditVf}^i(\mathsf{out}_{\mathrm{adt}}^{i-1}, \mathsf{seed}^i, \pi_{\mathsf{seed}}^i, \mathsf{ch}^i, \pi^i):}$

return $\mathsf{VDF.Vf}(vk_{\mathsf{VDF}}, \mathsf{out}_{\mathrm{adt}}^{i-1}, \mathsf{seed}^i, \pi_{\mathsf{seed}}^i) = 1$
$\wedge\ (\mathsf{ch}^i = \mathsf{H}_{\mathsf{Poll}}(\tau_R, \mathsf{seed}^i))$
$\wedge\ \Pi_{\mathsf{PoRep}}.\mathsf{Vf}(\mathsf{crs}_{\mathsf{PoRep}}, (id, \tau_D, \tau_R, \mathsf{ch}^i), \pi^i) = 1$

Fig. 9: The preservation phase. The server's signing key $sk_{\mathcal{S}}$ is implicit.

Retrieval. The retrieval phase is specified in Fig. 10. The server supplies a ciphertext C, a commitment x_{ret} to a fresh key k_{ret}, and a proof that C encrypts a replica recovering data consistent with τ_D. After verifying the proof, the client

signs a partial finalization transaction. The server completes the transaction by opening x_{ret}, revealing the encryption key and enabling collateral release.

For any statement $x = (m_{\mathrm{con}}, C, x_{\mathsf{ret}})$ and witness $w = (R, k_{\mathsf{ret}}, o_{\mathsf{ret}})$, the retrieval relation $\mathcal{R}_{\mathsf{ret}}$ underlying Π_{ret} satisfies $\mathcal{R}_{\mathsf{ret}}(x, w) = 1$ if and only if:

$$\begin{aligned} &D = \mathsf{PoRep.Extract}(id, \tau_D, R) \neq \perp \\ \wedge &\mathsf{PoRep.TagVf}(D, \tau_D) = 1 \\ \wedge &\mathsf{COM.Vf}(ck, x_{\mathsf{ret}}, k_{\mathsf{ret}}, o_{\mathsf{ret}}) = 1 \\ \wedge &R = \mathsf{SE.Dec}(k_{\mathsf{ret}}, C) \neq \perp . \end{aligned}$$

Here, D is a private intermediate value computed within the relation. The first two conditions ensure that R recovers data consistent with τ_D.

The terminal $\nu^n_{\mathrm{adt}}(m_{\mathrm{con}}, \rho, \mathsf{tx})$ has two branches:

– *Finalization branch.* The branch expects $\rho = (\mathsf{aux}_{\mathsf{ret}}, (\sigma_{\mathcal{C}}, (k_{\mathsf{ret}}, o_{\mathsf{ret}})))$, where $\mathsf{aux}_{\mathsf{ret}} = (x_{\mathsf{ret}}, \pi_{\mathsf{ret}})$. It accepts if $\mathsf{tx}.vi = vi_{\mathrm{fin}}$, tx creates the collateral-release output $(\nu_{\mathrm{fin}}, c, m_{\mathrm{con}})$ and no successor state output, and:
$$\mathsf{SIG.Vf}(pk_{\mathcal{C}}, \tilde{\mathsf{tx}}, \sigma_{\mathcal{C}}) = 1 \wedge \mathsf{COM.Vf}(ck, x_{\mathsf{ret}}, k_{\mathsf{ret}}, o_{\mathsf{ret}}) = 1.$$
The collateral-release output is controlled by the server:
$$\nu_{\mathrm{fin}}(\delta_{\mathrm{fin}}, \rho, \mathsf{tx}) \coloneqq \mathsf{SIG.Vf}(pk_{\mathcal{S}}, \tilde{\mathsf{tx}}, \rho).$$
– *Timeout branch.* This branch is identical to the preservation timeout branch, with vi_{fin} replacing vi^i_{adt}.

Both reject redeemers that do not have the required form. The validator returns 1 if either accepts, or 0 otherwise.

4.2 Security Analysis

We establish the following security guarantees and provide the proof in Appendix C.

Theorem 1. *Let $\Pi_{\mathcal{L}}$ be a blockchain protocol satisfying k-persistence and (k, u)-liveness. Let $\mathsf{H}_{\mathsf{Poll}}$ implementing* PoRep.Poll *be a hash function modeled as a random oracle. Assume that the underlying primitives are correct, the proof systems are complete, and the prescribed schedule and timing calibration are satisfied. Then the constructed DSN protocol $\mathsf{DSN}_{\mathcal{L}}$ satisfies:*

– *Correctness.*
– *Contract unforgeability, if* SIG *is EUF-CMA secure.*
– *Adaptive replication soundness, if* SIG *is EUF-CMA secure; the replication soundness of* PoRep *follows from its implicit PoRC and PoSpace under KoSC;* VDF *satisfies soundness and sequentiality and has succinct proof size; Π_{PoRep} satisfies zero-knowledge, succinctness, and simulation-extractability with constant-factor parallel extraction overhead relative to the online prover; and the public transcript satisfies $|\mathsf{tr}^i| = o(N)$ uniformly over all reached checkpoints, except with negligible probability, where N is the replica length. The proof gives an explicit bound on $|\mathsf{tr}^i|$ and specifies the achievable $(\epsilon, \epsilon', \mu)$.*
– *Adaptive client fairness, if* SIG *is EUF-CMA secure,* COM *is binding, the implicit PoRC of* PoRep *is binding, and Π_{ret} satisfies simulation-extractability.*

– *Server fairness, if* SE *is IND-CPA secure,* COM *is hiding, and* Π_{ret} *satisfies zero-knowledge.*

5 Implementation and Experimental Results

We assume that replica generation and the underlying PoSpace component operate as specified, and examine whether a DSN prover can exploit public PoRep audit evidence to reduce its local replica storage while continuing to pass subsequent audits. Under the common block-sampling workload specified below, we compare three configurations: Merkle openings as used in vanilla PoRep [18, Sec. 3.3.1], a toy non-succinct zero-knowledge layer based on vector Schnorr proofs, and a succinct zero-knowledge layer using GM17 [26]. These configurations provide controlled comparisons of concrete offloading mechanisms and do not constitute attacks on deployed protocols.

Appendix E further complements these experiments with a deployment-oriented evaluation of a PLONK [21] audit layer. It measures proving costs across replica sizes and challenge workloads, together with verification costs through the Cardano Plutus interface [34].

Setup. In this evaluation, we represent the replica R by $N = 100$ random blocks of $B = 512$ bytes each, totaling 50 KiB. Each run consists of $n = 100$ preservation audits. For each audit $i \in [n]$, we sample a set I_i of five distinct block indices uniformly without replacement. The replica tag τ_R is instantiated as a Merkle root authenticating the replica blocks. We instantiate the audit challenge as $\mathsf{ch}^i = (i, I_i)$, where i is the index of the corresponding audit transaction $\mathsf{tx}^i_{\mathrm{adt}}$. Schnorr responses and GM17 proofs are bound to (τ_R, ch^i). Hence, repeating a block selection in a later preservation audit still changes the proof statement.

We implement vector Schnorr using ark-ed-on-bn254 for Edwards-curve arithmetic and GM17 over BN254 using ark-gm17 0.3.0 [2]. For Schnorr, each 512-byte block is encoded injectively into 17 scalar chunks of at most 31 bytes each;[14] its Merkle tree authenticates the vector of scalar commitments for each block. We use Poseidon [25] for Merkle hashing in all configurations.

A public archive models the accepted audit evidence available through the public transaction transcript tr^i. After i accepted audits, we measure the serialized size of the server's retained state η^i, including remaining replica data, retained nonces, and any auxiliary decoding information. Public parameters and evidence are accounted for separately. Each claimed storage reduction must support exact recovery of the experimental replica and a subsequent preservation audit with a new index.

Experiments. We evaluate the following mechanisms, using full retention as the reference configuration.

Merkle openings without zero-knowledge. The prover publishes each requested block and its authentication path, then removes the block from local storage. Whenever that block is requested again, the prover retrieves it from the archive and supplies an opening for the current audit.

[14] Each 31-byte chunk fits within the curve's scalar field without modular reduction.

Although the audit index changes, the archived block and path remain valid evidence about the fixed commitment. Openings are reusable independently: an opening obtained in one selected set remains useful when the same index appears in a different set. Only exposed blocks are removed, since sibling hashes do not reveal sibling block contents.

Vector Schnorr with short retained nonces. For audit i, the prover publishes authenticated vector Schnorr responses:

$$s^i_{j,l} = r^i_{j,l} + c^i_{j,l} x_{j,l} \pmod{q},$$

where $x_{j,l}$ is the scalar encoding of chunk l of block $j \in I_i$, $r^i_{j,l}$ is the *prover-chosen nonce*, $c^i_{j,l}$ is the nonzero Fiat-Shamir challenge derived from the audit statement (τ_R, ch^i) and the corresponding Schnorr commitments, and q is the prime subgroup order. After publishing an accepted response, the prover retains the chunk nonces in place of the challenged block. It can subsequently reconstruct each chunk from the archived response as:

$$x_{j,l} = (s^i_{j,l} - r^i_{j,l})(c^i_{j,l})^{-1} \pmod{q}.$$

The recovered blocks allow it to generate new responses for later audits. Binding proofs to the audit index does not prevent this recovery mechanism.

We use 8-bit nonces per chunk in the main comparison, totaling 17 nonce bytes per discarded block, and vary the nonce width in a separate sweep. This strategy deliberately departs from Schnorr's prescribed nonce distribution; the resulting transcripts need not preserve zero-knowledge.

Succinct GM17 proofs. The prover publishes one GM17 proof covering the blocks indexed by I_i. Its public statement is (τ_R, ch^i); the challenged blocks and their Merkle authentication paths remain private. The nonce-based recovery procedure above requires a public scalar response for each replica chunk. The GM17-based configuration generates no such responses; retaining its proving randomness therefore does not supply the inputs required by that procedure.

General subliminal-channel offloading. This method applies to any randomized algorithm whose randomness the prover can control, provided the valid public outputs offer sufficient variability for encoding. The prover varies the randomness until a designated b-bit value computed from a valid output matches the next b replica bits. Only the selected output is published. After acceptance, the prover discards the encoded bits, which can be recovered from the archived output and combined with the retained replica suffix. We evaluate this method using GM17 proofs, with the statement fixed to (τ_R, ch^i) for each audit i. For $b \in \{1, 2, 4, 8\}$, we encode replica bits in the low b bits of the proof hash and count the candidate proofs examined per audit.

Results and evaluation. All runs completed the 100 preservation audits, recovered the replica exactly, and passed a subsequent audit with a new index.

Fig. 11(a) shows substantial offloading through public Merkle openings and short-nonce Schnorr responses. After 100 audits, these strategies retain an average of 450.8 and 2139.75 bytes, reducing serialized local storage by 99.12% and 95.82%, respectively. Schnorr savings decrease as retained nonces grow (Fig. 11(b)). At 248 and 250 bits per chunk, retained state exceeds full retention by 3.1% and 6.4%, respectively. Fig. 11(c) reports the search cost of GM17

subliminal encoding. For $b \in \{1, 2, 4, 8\}$, the encoder examines an average of 2.059, 3.876, 15.305, and 255.463 candidate proofs per audit, respectively. These means are close to the 2^b expectation for independent uniform hash prefixes, and all searches find the requested prefix. For fixed b, the number of replica bits offloaded grows linearly with the number of accepted audits, at b bits per audit.

6 Conclusion

We answer the question posed in the introduction by treating public audit evidence as part of the storage model and binding retrieval to ledger-enforced authorization. Our extractor-based PoRep formulation supports storage guarantees across evolving prover states, while conditional finalization links data recovery to collateral release. Under the stated cryptographic and resource assumptions, the construction provides adaptive replication soundness and retrieval fairness. The analysis identifies complementary roles for succinctness and simulation-extractability, while the controlled experiments demonstrate why cumulative transcript accounting remains necessary. Our PLONK prototype quantifies the cost of the audit layer; implementation and evaluation of the complete DSN remain future work.

References

1. Abdolmaleki, B., Campanelli, M., Dao, Q., Khoshakhlagh, H.: On the simulation-extractability of proof-carrying data. In: Bai, S., Persichetti, E. (eds.) PKC 2026, Part III. LNCS, vol. 16553, pp. 269–302. Springer, Cham (May 2026). https://doi.org/10.1007/978-3-032-26737-5_9
2. arkworks contributors: arkworks zkSNARK ecosystem (2022), https://arkworks.rs
3. Ateniese, G., Chen, L., Etemad, M., Tang, Q.: Proof of storage-time: Efficiently checking continuous data availability. In: NDSS 2020. The Internet Society (Feb 2020). https://doi.org/10.14722/ndss.2020.24427
4. Bellés-Muñoz, M., Isabel, M., Muñoz-Tapia, J.L., Rubio, A., Melé, J.B.: Circom: A circuit description language for building zero-knowledge applications. IEEE Trans. Dependable Secur. Comput. **20**(6), 4733–4751 (2023). https://doi.org/10.1109/TDSC.2022.3232813, https://doi.org/10.1109/TDSC.2022.3232813
5. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 276–294. Springer, Berlin, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44381-1_16
6. Benarroch, D., Campanelli, M., Fiore, D., Kim, J., Lee, J., Oh, H., Querol, A.: Proposal: Commit-and-prove zero-knowledge proof systems and extensions (2024), https://docs.zkproof.org/pages/standards/accepted-workshop4/proposal-commit.pdf
7. Benet, J.: IPFS - content addressed, versioned, p2p file system (2014), https://arxiv.org/abs/1407.3561
8. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 757–788. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_25
9. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo infinite: Proof-carrying data from additive polynomial commitments. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS, vol. 12825, pp. 649–680. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_23
10. Chakravarty, M.M.T., Chapman, J., MacKenzie, K., Melkonian, O., Jones, M.P., Wadler, P.: The extended UTXO model. In: Bernhard, M., Bracciali, A., Camp, L.J., Matsuo, S., Maurushat, A., Rønne, P.B., Sala, M. (eds.) FC 2020 Workshops. LNCS, vol. 12063, pp. 525–539. Springer, Cham (Feb 2020). https://doi.org/10.1007/978-3-030-54455-3_37
11. Dai, W., Okamoto, T., Yamamoto, G.: Stronger security and generic constructions for adaptor signatures. In: Isobe, T., Sarkar, S. (eds.) INDOCRYPT 2022. LNCS, vol. 13774, pp. 52–77. Springer, Cham (Dec 2022). https://doi.org/10.1007/978-3-031-22912-1_3
12. Damgård, I., Ganesh, C., Orlandi, C.: Proofs of replicated storage without timing assumptions. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 355–380. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26948-7_13
13. Dziembowski, S., Eckey, L., Faust, S.: FairSwap: How to fairly exchange digital goods. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 967–984. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243857

14. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 585–605. Springer, Berlin, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-48000-7_29
15. Eckey, L., Faust, S., Schlosser, B.: OptiSwap: Fast optimistic fair exchange. In: Sun, H.M., Shieh, S.P., Gu, G., Ateniese, G. (eds.) ASIACCS 20. pp. 543–557. ACM Press (Oct 2020). https://doi.org/10.1145/3320269.3384749
16. Faonio, A., Fiore, D., Russo, L.: Real-world universal zkSNARKs are non-malleable. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024. pp. 3138–3151. ACM Press (Oct 2024). https://doi.org/10.1145/3658644.3690351
17. Feige, U., Shamir, A.: Zero knowledge proofs of knowledge in two rounds. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 526–544. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34805-0_46
18. Fisch, B.: PoReps: Proofs of space on useful data. Cryptology ePrint Archive, Report 2018/678 (2018), https://eprint.iacr.org/2018/678
19. Fisch, B.: Tight proofs of space and replication. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 324–348. Springer, Cham (May 2019). https://doi.org/10.1007/978-3-030-17656-3_12
20. Fisch, B., Silas, S.: Weak compression and (in)security of rational proofs of storage. Cryptology ePrint Archive, Report 2018/514 (2018), https://eprint.iacr.org/2018/514
21. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019), https://eprint.iacr.org/2019/953
22. Garay, J.A., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: Analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 281–310. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6_10
23. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing **17**(2), 281–308 (Apr 1988)
24. Goren, G., Hariri, A., Hartley, T.D.R., Kappiyoor, R., Spiegelman, A., Zmick, D.: Shelby: Decentralized storage designed to serve (2025), https://arxiv.org/abs/2506.19233
25. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for zero-knowledge proof systems. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021. pp. 519–535. USENIX Association (Aug 2021), https://www.usenix.org/conference/usenixsecurity21/presentation/grassi
26. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 581–612. Springer, Cham (Aug 2017). https://doi.org/10.1007/978-3-319-63715-0_20
27. Juels, A., Kaliski, Jr., B.S.: Pors: proofs of retrievability for large files. In: Ning, P., De Capitani di Vimercati, S., Syverson, P.F. (eds.) ACM CCS 2007. pp. 584–597. ACM Press (Oct 2007). https://doi.org/10.1145/1315245.1315317
28. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC. pp. 723–732. ACM Press (May 1992). https://doi.org/10.1145/129712.129782

29. Li, Z., Zheng, S., Luo, J., Xin, Z., Yuan, D., Gao, S., Yang, S., Xiao, B., Liu, X.: PoUDR: Proof of unified data retrieval in decentralized storage networks. Cryptology ePrint Archive, Report 2024/1544 (2024), https://eprint.iacr.org/2024/1544
30. Micali, S.: CS proofs (extended abstracts). In: 35th FOCS. pp. 436–453. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365746
31. Moran, T., Orlov, I.: Simple proofs of space-time and rational proofs of storage. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 381–409. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26948-7_14
32. Pass, R., Rosen, A.: New and improved constructions of non-malleable cryptographic protocols. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 533–542. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060670
33. Protocol Labs: Filecoin: A decentralized storage network (2017), https://filecoin.io/filecoin.pdf
34. Querejeta-Azurmendi, I.: Plutus support for pairings over BLS12-381 (2022), https://cips.cardano.org/cip/CIP-0381
35. Storj Labs, Inc.: Storj: A decentralized cloud storage network framework (2018), https://static.storj.io/storjv3.pdf
36. Swarm team: Swarm: Storage and communication infrastructure for a self-sovereign digital society (2021), https://www.ethswarm.org/swarm-whitepaper.pdf
37. Tas, E.N., Seres, I.A., Zhang, Y., Melczer, M., Kelkar, M., Bonneau, J., Nikolaenko, V.: Atomic and fair data exchange via blockchain. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024. pp. 3227–3241. ACM Press (Oct 2024). https://doi.org/10.1145/3658644.3690248
38. Vorick, D., Champine, L.: Sia: Simple decentralized storage (2014)
39. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger (2014)
40. Xu, M., Zhang, J., Guo, H., Cheng, X., Yu, D., Hu, Q., Li, Y., Wu, Y.: FileDES: A secure, scalable and succinct decentralized encrypted storage network. Cryptology ePrint Archive, Report 2024/182 (2024), https://eprint.iacr.org/2024/182

A Auxiliary Preliminaries

This section supplements Sec. 2 with definitions of digital signatures, symmetric encryption, commitment schemes, non-interactive arguments, and verifiable delay functions.

Digital signatures. A digital signature scheme consists of a tuple of algorithms $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$.

- $\mathsf{KGen}(1^\lambda)$ takes as input the security parameter λ and outputs a secret-public key pair (sk, pk).
- $\mathsf{Sign}(sk, m)$ takes as input a secret key sk and a message m. It outputs a signature σ on m under sk.
- $\mathsf{Vf}(pk, m, \sigma)$ deterministically outputs 1 if σ is valid, or 0 otherwise.

We recall correctness and existential unforgeability under adaptive chosen message attacks (EUF-CMA) [23].

Definition 11 (Correctness). *A signature scheme is correct if the following holds for any λ and message m.*

$$\Pr\left[\mathsf{Vf}(pk, m, \sigma) = 1 \middle| \begin{array}{c}(sk, pk) \leftarrow \mathsf{KGen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sign}(sk, m)\end{array}\right] = 1$$

Definition 12 (EUF-CMA). *A signature scheme is EUF-CMA secure if, for any PPT adversary $\mathcal{A}$ with access to a signing oracle $\mathcal{O}_{\mathsf{Sign}}^{sk, Q_{\mathsf{Sign}}}$, the following probability is $\mathsf{negl}(\lambda)$ for any λ and $(sk, pk) \leftarrow \mathsf{KGen}(1^\lambda)$.*

$$\Pr\left[\begin{array}{c}\mathsf{Vf}(pk, m^*, \sigma^*) = 1 \\ \wedge m^* \notin Q_{\mathsf{Sign}}\end{array}\middle| \begin{array}{c}Q_{\mathsf{Sign}} \coloneqq \emptyset \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(pk)\end{array}\right],$$

where, on input m, the oracle $\mathcal{O}_{\mathsf{Sign}}$ returns $\sigma \leftarrow \mathsf{Sign}(sk, m)$ and records m in a queried set Q_{Sign}.

Symmetric encryption. A symmetric encryption scheme consists of a tuple of algorithms $\mathsf{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$.

- $\mathsf{KGen}(1^\lambda)$ takes as input the security parameter λ and outputs a secret key k.
- $\mathsf{Enc}(k, m)$ takes as input a secret key k and a message $m \in \{0,1\}^*$. It outputs a ciphertext ct.
- $\mathsf{Dec}(k, ct)$ deterministically outputs $m \in \{0,1\}^*$ or $\perp$.

Correctness and the indistinguishability under chosen plaintext attacks (IND-CPA) [**?**] are recalled as follows.

Definition 13 (Correctness). *A symmetric encryption is correct if the following holds for any λ and $m \in \{0,1\}^*$.*

$$\Pr\left[\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m | k \leftarrow \mathsf{KGen}(1^\lambda)\right] = 1$$

Definition 14 (IND-CPA). *A symmetric encryption scheme is IND-CPA secure if, for any PPT adversary $\mathcal{A}$ with access to an encryption oracle $\mathcal{O}_{\mathsf{Enc}}^k$, the following advantage is $\mathsf{negl}(\lambda)$ for any λ and $k \leftarrow \mathsf{KGen}(1^\lambda)$.*

$$\left|\Pr\left[b' = b \middle| \begin{array}{c}(m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}^k}(1^\lambda) \\ s.t.\ |m_0| = |m_1| \\ b \leftarrow\!\!\$\ \{0,1\};\ ct_b \leftarrow \mathsf{Enc}(k, m_b) \\ b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Enc}}^k}(\mathsf{st}, ct_b)\end{array}\right] - \frac{1}{2}\right|,$$

where, on input m, the oracle $\mathcal{O}_{\mathsf{Enc}}^k$ returns $ct \leftarrow \mathsf{Enc}(k, m)$.

Commitment scheme. For a (non-interactive) commitment scheme $\mathsf{COM} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Vf})$, we adopt the formulation from [6].

- $\mathsf{Setup}(1^\lambda)$ takes as input the security parameter λ and outputs a public commitment key ck, which defines an input space $\mathcal{M}$.
- $\mathsf{Commit}(ck, m)$ takes as input a key ck and a value $m \in \mathcal{M}$. It outputs a commitment c and an opening o.
- $\mathsf{Vf}(ck, c, m, o)$ deterministically outputs 1 if the input tuple is valid, or 0 otherwise.

Correctness, binding, and hiding are recalled as follows.

Definition 15 (Correctness). *A commitment scheme is correct if the following holds for any λ and $m \in \mathcal{M}$.*

$$\Pr\left[\mathsf{Vf}(ck, c, m, o) = 1 \,\middle|\, \begin{matrix} ck \leftarrow \mathsf{Setup}(1^\lambda) \\ (c, o) \leftarrow \mathsf{Commit}(ck, m) \end{matrix}\right] = 1$$

Definition 16 (Binding). *A commitment scheme is (computational) binding if, for any PPT adversary $\mathcal{A}$, the following probability is $\mathsf{negl}(\lambda)$ for any λ.*

$$\Pr\left[\begin{matrix} \mathsf{Vf}(ck, c, m, o) = 1 \wedge \\ \mathsf{Vf}(ck, c, m', o') = 1 \wedge \\ m \neq m' \end{matrix} \,\middle|\, \begin{matrix} ck \leftarrow \mathsf{Setup}(1^\lambda) \\ (c, m, o, m', o') \leftarrow \mathcal{A}(ck) \end{matrix}\right]$$

Definition 17 (Hiding). *A commitment scheme is (perfect) hiding if the following ensembles are identical for any λ, $ck \leftarrow \mathsf{Setup}(1^\lambda)$, and any $m_0, m_1 \in \mathcal{M}$.*

$$\mathcal{D}_0 \coloneqq \{c_0 : (c_0, \cdot) \leftarrow \mathsf{Commit}(ck, m_0)\} \text{ and}$$
$$\mathcal{D}_1 \coloneqq \{c_1 : (c_1, \cdot) \leftarrow \mathsf{Commit}(ck, m_1)\}$$

Non-interactive arguments. Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be an NP relation, parameterized by the security parameter λ. Denote the language defined by $\mathcal{R}$ as $L_\mathcal{R} \coloneqq \{x \in \{0,1\}^* : \exists w \in \{0,1\}^{\mathsf{poly}(|x|)} \text{ s.t. } (x, w) \in \mathcal{R}\}$. A non-interactive argument protocol for $\mathcal{R}$ consists of a tuple of algorithms $\Pi_\mathcal{R} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$.

- $\mathsf{Setup}(1^\lambda, \mathcal{R})$ takes as input λ and a relation $\mathcal{R}$. It outputs a common reference string crs and a trapdoor td.
- $\mathsf{Prove}(\mathsf{crs}, x, w)$ takes as input crs, an instance x, and a witness w. It outputs an argument π.
- $\mathsf{Vf}(\mathsf{crs}, x, \pi)$ deterministically outputs 1 if π is valid, or 0 otherwise.

We recall completeness, zero-knowledge, knowledge soundness [17], and preprocessing succinctness [28,30], which together characterize a zkSNARK. Additionally, our construction requires simulation-extractability [32].

Definition 18 (Completeness). *A non-interactive argument protocol for a relation $\mathcal{R}$ satisfies completeness if the following holds for any λ, $x \in L_\mathcal{R}$, and w s.t. $(x, w) \in \mathcal{R}$.*

$$\Pr\left[\mathsf{Vf}(\mathsf{crs}, x, \pi) = 1 \,\middle|\, \begin{matrix} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{matrix}\right] = 1$$

Definition 19 (Zero-Knowledge). *A non-interactive argument for $\mathcal{R}$ is statistically zero-knowledge if there exists a PPT simulator Sim s.t. the following ensembles have negligible statistical distance for any λ and $(x, w) \in \mathcal{R}$.*

$$\mathcal{D}_0 \coloneqq \{(\mathsf{crs}, x, \pi_0) : \pi_0 \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\} \text{ and}$$
$$\mathcal{D}_1 \coloneqq \{(\mathsf{crs}, x, \pi_1) : \pi_1 \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathsf{td}, x)\}$$

$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$ is sampled in each distribution.

Definition 20 (Knowledge Soundness). *A non-interactive argument for $\mathcal{R}$ is an argument of knowledge if, for any* PPT *adversary $\mathcal{A}$, there exists a PPT*

extractor $\mathcal{E}^{\mathcal{A}}$ *s.t. the following probability is* $\mathsf{negl}(\lambda)$ *for any* λ*.*

$$\Pr\left[\begin{array}{c}\mathsf{Vf}(\mathsf{crs},x,\pi)=1\\ \wedge(x,w)\notin\mathcal{R}\end{array}\middle|\begin{array}{c}(\mathsf{crs},\mathsf{td})\leftarrow\mathsf{Setup}(1^\lambda,\mathcal{R})\\ (x,\pi)\leftarrow\mathcal{A}(\mathsf{crs})\\ w\leftarrow\mathcal{E}^{\mathcal{A}}(\mathsf{crs},x,\pi)\end{array}\right]$$

Definition 21 (Simulation-Extractability). *A non-interactive argument for* $\mathcal{R}$*, with a PPT simulator* Sim*, satisfies simu-lation-extractability if, for any PPT adversary* $\mathcal{A}$*, there exists a PPT extractor* $\mathcal{E}^{\mathcal{A}}$ *s.t. the following probability is* $\mathsf{negl}(\lambda)$ *for any* λ *and* $(\mathsf{crs},\mathsf{td})\leftarrow\mathsf{Setup}(1^\lambda,\mathcal{R})$*.*

The adversary has access to a proof-simulation oracle $\mathcal{O}^{\mathsf{td}}_{\mathsf{Sim}}$*. On input* x*, the oracle computes* $\pi\leftarrow\mathsf{Sim}(\mathsf{crs},\mathsf{td},x)$*, records* (x,π) *in* Q_{Sim}*, and returns* π*.*

$$\Pr\left[\begin{array}{c}\mathsf{Vf}(\mathsf{crs},x,\pi)=1\\ \wedge(x,w)\notin\mathcal{R}\\ \wedge(x,\pi)\notin Q_{\mathsf{Sim}}\end{array}\middle|\begin{array}{c}Q_{\mathsf{Sim}}:=\emptyset\\ (x,\pi)\leftarrow\mathcal{A}^{\mathcal{O}_{\mathsf{Sim}}}(\mathsf{crs})\\ w\leftarrow\mathcal{E}^{\mathcal{A}}(\mathsf{crs},x,\pi,Q_{\mathsf{Sim}})\end{array}\right]$$

Definition 22 (Preprocessing Succinctness). *A non-inter-active argument protocol satisfies preprocessing succinctness if there exist fixed polynomials* p,q *s.t., for any* λ*, relation* $\mathcal{R}$*, and* $(\mathsf{crs},\mathsf{td})\leftarrow\mathsf{Setup}(1^\lambda,\mathcal{R})$*: the verifier* $\mathsf{Vf}(\mathsf{crs},x,\pi)$ *runs in time* $q(\lambda,|x|)$ *and rejects whenever* $|\pi|>p(\lambda)$*; and, for every* $(x,w)\in\mathcal{R}$*, the proof* $\pi\leftarrow\mathsf{Prove}(\mathsf{crs},x,w)$ *satisfies* $|\pi|\leq p(\lambda)$*. No additional succinctness requirement is imposed on* Setup*.*

Verifiable delay functions. A verifiable delay function [8] consists of a tuple of algorithms $\mathsf{VDF}=(\mathsf{Setup},\mathsf{Eval},\mathsf{Vf})$.

- $\mathsf{Setup}(1^\lambda,T)$ takes as input the security parameter λ and a delay parameter $T>0$ (subexponential in λ). It outputs public parameters $\mathsf{pp}:=(ek,vk)$, comprising an evaluation key ek and a verification key vk. It runs in time $\mathsf{poly}(\lambda)$; pp specify an efficiently sampleable input space $\mathcal{X}$ and an output space $\mathcal{Y}$.
- $\mathsf{Eval}(ek,x)$ takes as input an evaluation key ek and $x\in\mathcal{X}$, and outputs (y,π), where $y\in\mathcal{Y}$ is determined by (ek,x) and π is a potentially empty proof. It runs in parallel time T using $\mathsf{poly}(\lambda,\log T)$ processors.
- $\mathsf{Vf}(vk,x,y,\pi)$ deterministically outputs 1 if π is valid, or 0 otherwise. It runs in time $\mathsf{poly}(\lambda,\log T)$.

We recall correctness, soundness, and sequentiality.

Definition 23 (Correctness). *A VDF is correct if the following holds for any* $\lambda,T>0$ *and* $x\in\mathcal{X}$*.*

$$\Pr\left[\mathsf{Vf}(vk,x,y,\pi)=1\middle|\begin{array}{c}(ek,vk)\leftarrow\mathsf{Setup}(1^\lambda,T)\\ (y,\pi)\leftarrow\mathsf{Eval}(ek,x)\end{array}\right]=1$$

Definition 24 (Soundness). *A VDF is sound if, for any adversary* $\mathcal{A}$ *running in time* $\mathsf{poly}(\lambda,T)$*, the following probability is* $\mathsf{negl}(\lambda)$ *for any* $\lambda,T>0$*.*

$$\Pr\left[\begin{array}{c}x\in\mathcal{X}\\ \wedge\mathsf{Vf}(vk,x,y',\pi')=1\\ \wedge y'\neq y\end{array}\middle|\begin{array}{c}\mathsf{pp}\leftarrow\mathsf{Setup}(1^\lambda,T)\\ (x,y',\pi')\leftarrow\mathcal{A}(\mathsf{pp},T)\\ (y,\cdot)\leftarrow\mathsf{Eval}(ek,x)\end{array}\right]$$

Definition 25 (Sequentiality). *For functions p and δ with $\delta(T) < T$, a VDF is (p, δ)-sequential if, for any adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ s.t. $\mathcal{A}_0$ runs in total time $\mathsf{poly}(\lambda, T)$ and $\mathcal{A}_1$ runs in parallel time at most $\delta(T)$ using at most $p(T)$ processors, the following probability is $\mathsf{negl}(\lambda)$ for any $\lambda, T > 0$.*

$$\Pr\left[y' = y \;\middle|\; \begin{array}{c} \mathsf{pp} := (ek, vk) \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \mathsf{st} \leftarrow \mathcal{A}_0(\mathsf{pp}, T);\ x \leftarrow\!\!\$\ \mathcal{X};\ y' \leftarrow \mathcal{A}_1(\mathsf{st}, x) \\ (y, \cdot) \leftarrow \mathsf{Eval}(ek, x) \end{array}\right]$$

If this holds for every polynomial p, the VDF is δ-sequential.

B Replication Soundness of PoRep

We first recall the KoSC assumption and the implicit PoRC and PoSpace schemes induced by PoRep [18]. We then give sufficient conditions for replication soundness and establish its relation to the rational replication notion of [18].

B.1 Strong Compression and Incompressibility

A compression scheme encodes its input into a shorter representation from which the input can be recovered. In a strong compression scheme [20], the encoder produces $\omega \leftarrow \mathsf{Encode}^*(s, x)$, and the decoder Decode^* receives both s and ω. The seed s is retained separately and excluded from the compressed length $|\omega|$. The scheme is ξ-lossy if $\Pr[\mathsf{Decode}^*(s, \mathsf{Encode}^*(s, x)) = x] > 1 - \xi$, where the probability is over the sampled input and seed, and the algorithms' randomness. For the following definition of time-bounded incompressibility, we use the strong compression and seed sampling conventions of [18, Defs. 5 and 12].

Definition 26 (Time-Bounded Incompressibility [18]). *A source $\mathcal{S}$ is (t_1, t_2)-incompressible to b bits if, for any constant $\xi \in (0, 1)$, no ξ-lossy strong compression scheme compresses $\mathcal{S}$ to $b - \log(1/\xi)$ bits with its encoder and decoder running in parallel time at most t_1 and t_2, respectively.*

Assumption 2 (Knowledge of Strong Compression [18]). *Let $\mathcal{S}$ and $\mathcal{X}$ be samplable sources over $\{0,1\}^N$ s.t. $\mathcal{S}$ is (t_1, t_2)-incompressible to $(1 - \epsilon_0)N$ bits for some $\epsilon_0 \in (0, 1)$. For any compression scheme $(\mathsf{Encode}, \mathsf{Decode})$ that compresses $\mathcal{S} \times \mathcal{X}$ to M bits, with its encoder running in parallel time at most t_1 or its decoder running in parallel time at most t_2, there exists a strong compression scheme $(\mathsf{Encode}^*, \mathsf{Decode}^*)$ that compresses $\mathcal{X}$ to $M - (1 - \epsilon_0)N$ bits using seeds from $\mathcal{S}$. Its encoder and decoder run in parallel time at most t_1 and t_2, respectively. Moreover, for any algorithm $\mathcal{A}$ producing $(\mathsf{Encode}, \mathsf{Decode})$, there exists an efficient extractor that observes the internal state of $\mathcal{A}$ and outputs $(\mathsf{Encode}^*, \mathsf{Decode}^*)$.*

B.2 Implicit PoRC and PoSpace Schemes

We recall the schemes induced by the public-coin PoRep interface. Public parameters are implicit.

Implicit PoRC. Following [18, Section 2.1.2], we assume that the data tag τ_D binds D under $\mathsf{PoRep.TagVf}$, and that the auxiliary information aux returned by $\mathsf{PoRep.Replicate}$ binds R under the opening procedure below. The algorithms are the following wrappers around PoRep; we implicitly parse $c = (id, \tau_D, \mathsf{aux})$ and $o = R$ whenever these inputs are supplied.

- $\mathsf{PoRC.Setup}(1^\lambda)$ runs $\mathsf{PoRep.Setup}(1^\lambda, t)$, with t fixed by the underlying PoRep instance.
- $\mathsf{PoRC.Commit}(id, D, \tau_D)$ returns $\perp$ if $\mathsf{PoRep.TagVf}(D, \tau_D) = 0$. Otherwise, it computes $(R, \mathsf{aux}) \leftarrow \mathsf{PoRep.Replicate}(id, D, \tau_D)$, sets $c \coloneqq (id, \tau_D, \mathsf{aux})$ and $o \coloneqq R$, and returns (c, o).
- $\mathsf{PoRC.Open}(c, o)$ computes $D \leftarrow \mathsf{PoRep.Extract}(id, \tau_D, R)$ and returns $\perp$ if $D = \perp$, $\mathsf{PoRep.TagVf}(D, \tau_D) = 0$, or $\mathsf{PoRep.Vf}(id, \tau_D, \mathsf{aux}) = 0$. Otherwise, it returns (D, R).
- $\mathsf{PoRC.Prove}(c, o, \mathsf{ch})$ returns $\pi \leftarrow \mathsf{PoRep.Prove}(id, R, \mathsf{aux}, \mathsf{ch})$, where ch is sampled from $\mathsf{PoRep.Poll}(\mathsf{aux})$.
- $\mathsf{PoRC.Vf}(c, \mathsf{ch}, \pi)$ returns $\mathsf{PoRep.Vf}(id, \tau_D, \mathsf{aux}, \mathsf{ch}, \pi)$.

Including id in c is a notational specialization of that of [18, Section 2.1.2]; it avoids carrying id again inside each proof. For our implicit PoRC, the message is $m = (D, R)$.

Definition 27 (PoRC Binding [18]). *A PoRC is binding if, for any PPT adversary $\mathcal{A}$, the following probability is $\mathsf{negl}(\lambda)$ for any λ.*

$$\Pr\left[\begin{array}{c|c} m \neq \perp \wedge m' \neq \perp & \mathsf{pp} \leftarrow \mathsf{PoRC.Setup}(1^\lambda) \\ \wedge m \neq m' & (c, o, o') \leftarrow \mathcal{A}(\mathsf{pp}) \\ & m \coloneqq \mathsf{PoRC.Open}(c, o) \\ & m' \coloneqq \mathsf{PoRC.Open}(c, o') \end{array}\right]$$

We recall the PoRC soundness experiment in Fig. 12.

Definition 28 (Admissible PoRC Adversary [18]). *Let $\mu \in (0, 1]$. An adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is μ-admissible on m if the following holds for any λ, $\mathsf{pp} \leftarrow \mathsf{PoRC.Setup}(1^\lambda)$, and any (c, o, st) in the support of $\mathcal{A}_1(\mathsf{pp}, m)$.*

$$\Pr[\mathrm{PoRCSound}_{\mathcal{A}}(\mathsf{pp}, m) = 1] \geq \mu$$

The probability is over the experiment randomness for fixed pp, m, and the first-stage output.

Definition 29 (PoRC Soundness [18]). *A PoRC scheme is μ-sound if it is binding and there exists an extractor $\mathcal{E}$, running in time $\mathsf{poly}(\lambda, |m|, 1/\mu)$, s.t. the following probability is $\mathsf{negl}(\lambda)$ for any λ, $\mathsf{pp} \leftarrow \mathsf{PoRC.Setup}(1^\lambda)$, any message m, and any μ-admissible adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ on m.*

$$\Pr\left[m_{\mathcal{E}} \neq m \middle| \begin{array}{c} (c, o, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp}, m) \\ m_{\mathcal{E}} \leftarrow \mathcal{E}^{\mathcal{A}_2[\mathsf{st}]}(\mathsf{pp}, c) \end{array}\right].$$

The extractor has rewindable black-box access to $\mathcal{A}_2[\mathsf{st}]$: it may issue fresh challenges and rewind the prover's randomness, while the commitment and retained state remain fixed. It receives neither m nor o and cannot inspect the prover's retained state or hidden inputs.

The scheme is PRAM strongly μ-sound *if, additionally, the extractor has parallel running time at most a constant factor c times that of* $\mathcal{A}_2$*. The preprocessing time of* $\mathcal{A}_1$ *is excluded from the extractor's running time.*

Implicit PoSpace. The implicit PoSpace consists of setup, initialization, and execution, defined as follows.

- $\mathsf{PoSpace.Setup}$: Run $\mathsf{PoRep.Setup}(1^\lambda, t)$;
- $\mathsf{PoSpace.Init}$: Prover runs $(R, \mathsf{aux}) \leftarrow \mathsf{PoRep.Replicate}(id, D, \tau_D)$, publishes $(id, \tau_D, \mathsf{aux})$, and retains R.
- $\mathsf{PoSpace.Execution}$: The verifier challenges with $\mathsf{ch} \leftarrow \mathsf{PoRep.Poll}(\mathsf{aux})$, the prover responds with $\pi \leftarrow \mathsf{PoRep.Prove}(id, R, \mathsf{aux}, \mathsf{ch})$, and the verifier runs $\mathsf{PoRep.Vf}(id, \tau_D, \mathsf{aux}, \mathsf{ch}, \pi)$.

We use parallel PoSpace soundness as described in [18, Appendix A.7], without reproducing the underlying graph-labeling game. For multiple replication instances, we require the corresponding k-fold composition to satisfy the stated parallel soundness bound. The identifiers are distinct, but the data inputs need not be distinct or independent.

B.3 Sufficient Conditions

The following theorem follows the extraction and compression argument of [18, Lemma 2].

Theorem 3 (Sufficient Conditions for Replication Soundness). *Fix* $0 < \epsilon_0 < \epsilon < 1$ *and* $\mu \in (0, 1]$ *with* $1/\mu = \mathsf{poly}(\lambda)$*. A PoRep construction with setup parameters* (λ, t) *and replicas of common length* $N \geq 1$ *is* (ϵ, μ)*-replication sound if the following hold for some constant* $c \geq 1$ *and any polynomially bounded* $k \geq 1$*.*

1. *It is correct.*
2. *Its implicit PoRC is PRAM strongly* μ*-sound, with extractor blowup at most* c*.*
3. k*-fold composition of its implicit PoSpace under distinct identifiers is parallel* $((1-\epsilon_0)kN, (c+1)t, \mathsf{negl}(\lambda))$*-sound.*[15]
4. *For every* μ*-admissible adversary, we assume KoSC applies to the encoding of* $(\boldsymbol{R}', \hat{z})$ *constructed in the proof, with a uniform polynomial-time realization of the KoSC extractor. The joint encoding is decodable in parallel time at most* ct*. The decoder* Decode^* *that recovers* $\hat{z}$ *from* $\boldsymbol{R}'$ *and* ω *is deterministic.*
   The overhead consists of the joint encoding's excess over $|\eta|$ *and any additional retained information needed for honest proving or decoding, beyond the replicas and KoSC's compressed auxiliary output. Its total length is at most* $(\epsilon - \epsilon_0)kN$ *except with negligible probability. All residual decoding information, including the decoder description and required auxiliary or replay information, is included in* ω *or* Decode^* *and counted once.*

[15] We write $\mathsf{negl}(\lambda)$ explicitly for the negligible PoSpace soundness error required in [18, Appendix A.7], reserving μ for admissibility.

Proof. Let $\mathcal{A} = (\mathcal{A}_{\text{pre}}, \mathcal{A}_{\text{onl}})$ be a μ-admissible PoRep adversary. For every fixed first stage output, each instance accepts with probability at least μ. PoRep correctness and the initial checks (in Fig. 1) make an honestly generated replica a valid PoRC opening. To obtain $\boldsymbol{R}'$, the replication soundness extractor $\mathcal{E}$ runs the PoRC extractors from condition (2) in parallel within time ct. Since k is polynomially bounded, PoRC soundness ensures that $R'_i = R_i$ for every $i \in [k]$ except with negligible probability. Each extracted replica has length N and recovers the corresponding D_i.

Following [18, Lemma 2], the retained state η jointly encodes $\boldsymbol{R}'$ and $\hat{z}$. The replicas are recovered by the PoRC extractors, and $\hat{z}$ by running $\mathcal{A}_{\text{onl}}$. Since $\hat{z}$ is determined by the retained state and fixed public inputs, this recovers the same auxiliary output regardless of the fresh challenges and online randomness. These computations run in parallel within time ct, with the required encoding and decoding information accounted for by condition (4).

To separate the storage for the replicas from that for $\hat{z}$, we establish their time-bounded incompressibility using condition (3). Suppose, for contradiction, that a prover could retain at most $(1-\epsilon_0)kN$ bits and recover all k replicas within parallel time ct with non-negligible probability. After recovery, it could generate honest proofs in another t time, using the matching auxiliary information published during PoSpace initialization and available as public instance input. By correctness, these proofs would be accepted whenever recovery succeeds. The prover would therefore pass verification with non-negligible probability within parallel time $(c+1)t$, contradicting condition (3). Hence, the replica source is time-bounded incompressible to $(1-\epsilon_0)kN$ bits when decoding is restricted to parallel time ct.

Together with condition (4), this incompressibility bound allows $\mathcal{E}$ to invoke the KoSC extractor and obtain a strong compression scheme $(\mathsf{Encode}^*, \mathsf{Decode}^*)$ for $\hat{z}$, using the extracted replicas as side information. It computes $\omega \leftarrow \mathsf{Encode}^*(\boldsymbol{R}', \hat{z})$ without changing the replicas. The decoder Decode^* is deterministic and satisfies:

$$\mathsf{Decode}^*((R'_i)_{i\in[k]}, \omega) = \hat{z}.$$

KoSC guarantees a saving of $(1-\epsilon_0)kN$ bits. Adding the charged encoding and decoding overhead gives:

$$\begin{aligned} |\omega| + |\mathsf{Decode}^*| &\le |\eta| - (1-\epsilon_0)kN + (\epsilon - \epsilon_0)kN \\ &= |\eta| - (1-\epsilon)kN. \end{aligned}$$

Therefore,

$$\begin{aligned} |\eta| &\ge (1-\epsilon)kN + |\omega| + |\mathsf{Decode}^*| \\ &\ge (1-\epsilon)(kN + |\omega| + |\mathsf{Decode}^*|). \end{aligned}$$

Since $\sum_{i\in[k]} \max\{|R_i|, |R'_i|\} = kN$, this is the required storage inequality.

$\mathcal{E}$ is uniform, polynomial-time, and fixed independently of z and the sampled execution. Both recovery requirements and the storage bound hold except with negligible probability, proving (ϵ, μ)-replication soundness.

B.4 Relation to Rational Replication

We recall rational replication [18, Def. 6], using our admissibility and auxiliary output conventions and allowing negligible exceptions in the storage comparison.

A retained state is a k-replication of $\boldsymbol{D} = (D_i)_{i\in[k]}$ if it contains k disjoint separately decodable components, the i-th of which recovers D_i [18, Def. 2]. An admissible adversary retaining such a state is *well-behaved*. All required decoding information is counted as storage unless it belongs to the fixed public parameters.

Definition 30 (Rational Replication [18]). *Let $\epsilon \in [0,1)$ and $\mu \in (0,1]$. A PoRep scheme satisfies ϵ-rational replication with admissibility threshold μ if, for any μ-admissible adversary $\mathcal{A}$, there exists a well-behaved admissible adversary $\mathcal{A}'$ s.t. the following hold for any $\lambda, t > 0$, polynomially bounded $k \geq 1$, and polynomial-length static auxiliary input z. Sample $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, t)$. The adversary $\mathcal{A}'$ preserves the data vector $\boldsymbol{D}$ chosen by $\mathcal{A}_{\mathrm{pre}}$.*

– *Their retained states satisfy:*

$$|\eta'| \leq \frac{|\eta|}{1-\epsilon} \tag{B.1}$$

except with negligible probability over setup and the preprocessing executions.[16]

– *Let $\hat{z}, \hat{z}'$ denote the respective auxiliary outputs and $\hat{b}, \hat{b}'$ the joint verification bits. Then:*

$$\{\hat{z}'\}_\lambda \approx_c \{\hat{z}\}_\lambda,$$

and, for any auxiliary output value u for which both conditional probabilities are defined:

$$\Pr[\hat{b}' = 1 | \hat{z}' = u] \geq \Pr[\hat{b} = 1 | \hat{z} = u].$$

The extractor constructed in Thm. 3 provides the representation needed by the well-behaved adversary. The following theorem makes this construction explicit.

Theorem 4 (Relation to Rational Replication). *Under the conditions of Thm. 3, the construction satisfies ϵ-rational replication with admissibility threshold μ, provided that honest proving and residual decoding together run in parallel time at most t.*

Proof. Let $\mathcal{A} = (\mathcal{A}_{\mathrm{pre}}, \mathcal{A}_{\mathrm{onl}})$ be a μ-admissible adversary. We construct a well-behaved admissible adversary $\mathcal{A}' = (\mathcal{A}'_{\mathrm{pre}}, \mathcal{A}'_{\mathrm{onl}})$.

The algorithm $\mathcal{A}'_{\mathrm{pre}}$ first runs $\mathcal{A}_{\mathrm{pre}}$ and then the extractor $\mathcal{E}$ constructed in Thm. 3, obtaining $(\boldsymbol{R}', \omega, \mathsf{Decode}^*)$. It also generates honest replicas for the same identifiers and data-tag pairs, with matching auxiliary information. It computes $\hat{z}$ by running $\mathcal{A}_{\mathrm{onl}}$ once; by our auxiliary output convention, this value is independent of the challenges and online randomness. The algorithm checks that the extracted replicas equal the honest replicas and that $\mathsf{Decode}^*(\boldsymbol{R}', \omega) = \hat{z}$. If both checks pass, it retains $(\boldsymbol{R}', \omega, \mathsf{Decode}^*)$; otherwise, it retains the honest replicas and $\hat{z}$ directly. In either case, the replicas and their decoding information

[16] The displayed inequality in [18, Def. 6] has the opposite direction; we use the direction in its accompanying explanation and proof of Lemma 2.

form k disjoint separately decodable components. Preprocessing takes polynomial time. Except with negligible probability, both checks pass and the overhead bound in condition (4) holds. On these executions, the complete retained state satisfies:

$$|\eta'| \leq |\eta| + \epsilon k N \leq \frac{|\eta|}{1-\epsilon},$$

where the last inequality follows from $|\eta| \geq (1-\epsilon)kN$. Thus, the retained states satisfy the comparison required by rational replication except with negligible probability.

The online algorithm $\mathcal{A}'_{\mathrm{onl}}$ answers each challenge honestly using the corresponding replica and its matching auxiliary information. It recovers $\hat{z}$ using Decode^*, or returns the directly retained value on a fallback execution. Since these computations fit within parallel time t by hypothesis, $\mathcal{A}'$ is also admissible. Correctness gives $\hat{b}' = 1$ on every execution, and the construction ensures $\hat{z}' = \hat{z}$. Hence, their auxiliary output distributions are identical, and whenever the conditional probabilities are defined:

$$\Pr[\hat{b}' = 1 \mid \hat{z}' = u] = 1 \geq \Pr[\hat{b} = 1 \mid \hat{z} = u].$$

C Proof of Thm. 1

Proof. We establish each property in turn.

Correctness. It follows from the correctness of the underlying primitives, completeness of the SNARK protocols, and the prescribed schedule. All honest delegation, audit, and retrieval checks succeed. Ledger liveness ensures confirmation within the corresponding validity intervals, while persistence preserves the confirmed transactions. Finalization reveals the correct encryption key; hence, RetExt recovers the original replica and, by PoRep correctness, the delegated data.

Contract unforgeability. It follows from the EUF-CMA security of SIG. An accepted contract transaction spending an honest party's funding output must contain that party's valid signature on its base transaction. The winning condition requires that this base was not submitted to the corresponding signing oracle; hence, the signature constitutes a forgery. Applying this argument to the client and server gives negligible winning probability.

Adaptive replication soundness. Fix constants $0 < \bar{\epsilon} < \epsilon < 1$ and $0 < \bar{\mu} < \mu \leq 1$ *s.t.* both $1/\bar{\mu}$ and $1/(\mu - \bar{\mu})$ are polynomially bounded in λ. Let PoRep be $(\bar{\epsilon}, \bar{\mu})$-replication sound under the conditions of Thm. 3, with replica length N. Let $\mathcal{A}$ be a μ-admissible DSN adversary.

The sublinear transcript qualification implies that, for sufficiently large λ:

$$|\mathsf{tr}^i| \leq (\epsilon - \bar{\epsilon})N \tag{C.1}$$

at every reached checkpoint, except with negligible probability. This bound captures the intended regime in which cumulative public data remains small relative to the replica.[17]

Fix a reached checkpoint i and its execution prefix. The retained state η^i, public inputs, and transcript tr^i remain fixed during extraction. The VDF timing calibration ensures that the next target hash input is fresh relative to this prefix. The extractor replays $\mathcal{A}^{i+1}_{\mathrm{onl}}[\eta^i]$ and answers the fresh target query to $\mathsf{H}_{\mathsf{Poll}}$ with the challenge requested by the PoRC extractor. For uniformly sampled challenges, this preserves the random oracle distribution. Previous oracle responses remain available through tr^i, while subsequent calls to $\mathcal{O}_{\mathsf{SimAudit}}$ are replayed consistently under its prescribed interface, with its internal computation uncharged. The recorded execution is unchanged. An accepted target audit contains a valid Π_{PoRep} proof. Since $\mathcal{O}_{\mathsf{SimAudit}}$ rejects the target identifier, this proof was not supplied by the oracle. Simulation-extractability yields a valid underlying PoRep response, except with negligible probability. The Π_{PoRep} extractor supplies these responses to the PoRC extractor. The former may use non-black-box access, while the latter retains its rewindable black-box interface. Under the stated extractor restriction, their composition runs within parallel time ct, measured *w.r.t.* the experiment's time accounting, where c includes the SNARK extraction overhead. Conditions (3) and (4) of Thm. 3 are required for this c value.

DSN admissibility gives acceptance probability at least μ. After accounting for negligible simulation and extraction errors, the underlying PoRep responses are accepted with probability at least $\bar{\mu}$, except for a negligible set of execution prefixes. PoRC soundness therefore gives $R'^i = R$ except with negligible probability. Hence, $|R'^i| = N$ and, by PoRep correctness, $\mathsf{PoRep.Extract}(id, \tau_D, R'^i) = D$. We then apply the PoSpace-KoSC argument from Thm. 3, using conditions (3) and (4) and charging $|\eta^i| + |\mathsf{tr}^i|$ as the available storage. With R'^i fixed, the extractor obtains residual information ω^i and a deterministic decoder satisfying:

$$\mathsf{Decode}^*(R'^i, \omega^i, \mathsf{tr}^i) = \hat{z}^i.$$

A single decoder serves all checkpoints, with checkpoint-specific decoding information included in ω^i. The decoder description and all required encoding and replay information are charged as required by condition (4).

The resulting storage bound is:

$$|\eta^i| + |\mathsf{tr}^i| \geq (1 - \bar{\epsilon})(N + |\omega^i| + |\mathsf{Decode}^*|).$$

[17] For example, if each transaction and simulated proof has size $\mathcal{O}\big(\lambda^a(\log N)^b\big)$ for fixed constants a, b, then:

$$|\mathsf{tr}^i| = \mathcal{O}\Big((i + 1 + q_i)\lambda^a(\log N)^b\Big),$$

where $i + 1$ accounts for the delegation transaction and subsequent audits, and q_i counts the available simulated proofs. Choosing the contract length and external-proof budget so that this upper bound is $o(N)$ uniformly over the checkpoints ensures the theorem's transcript assumption.

Using Eq. C.1, we obtain:

$$\begin{aligned}|\eta^i| &\geq (1-\epsilon)N + (1-\bar{\epsilon})(|\omega^i| + |\mathsf{Decode}^*|) \\ &\geq (1-\epsilon)(N + |\omega^i| + |\mathsf{Decode}^*|).\end{aligned}$$

Since every extracted replica has length N, we have $\ell_i = N$, and the shrinkage condition holds for every $\epsilon' \in [0, 1)$.

The extractor applies this construction at each checkpoint preceding an accepted audit, without using later states or transcripts. Under the stated realization and extraction bounds, it is uniform and polynomial-time, and fixed independently of z and the sampled execution. Combining the transcript bound with a union bound over the polynomially many checkpoints, all recovery, storage, and shrinkage requirements hold except with negligible probability. Therefore, the construction satisfies $(\epsilon, \epsilon', \mu)$-adaptive replication soundness for any $\epsilon' \in [0, 1)$.

Adaptive client fairness. Consider the two winning conditions. An accepted $\mathsf{tx}^*_{\mathrm{fin}}$ must contain the client's signature on $\tilde{\mathsf{tx}}^*_{\mathrm{fin}}$. If $F = \emptyset$ and $(\mathcal{C}, \tilde{\mathsf{tx}}^*_{\mathrm{fin}}) \notin Q_{\mathsf{Sign}}$, this signature constitutes an EUF-CMA forgery.

Otherwise, a winning execution has $F_{\mathrm{valid}} \neq \emptyset$. Fix $(C, \mathsf{aux}_{\mathsf{ret}}, \hat{\mathsf{tx}}_{\mathrm{fin}}) \in F_{\mathrm{valid}}$. Its recorded RetVf result is 1, and the winning condition requires $(m_{\mathrm{con}}, C, \mathsf{aux}_{\mathsf{ret}}) \notin Q_{\mathsf{SimAux}}$. Simulation-extractability provides a witness satisfying $\mathcal{R}_{\mathsf{ret}}$. Since $\tilde{\hat{\mathsf{tx}}}_{\mathrm{fin}} = \tilde{\mathsf{tx}}^*_{\mathrm{fin}}$, both transactions contain the same key commitment x_{ret}. Binding of COM ensures that the key disclosed in $\mathsf{tx}^*_{\mathrm{fin}}$ equals the witness key; then, decrypting C recovers the witness replica. Correctness and binding of the implicit PoRC ensure that this replica decodes to the delegated data. Therefore, $\mathsf{RetExt}(C, \hat{\mathsf{tx}}_{\mathrm{fin}}, \mathsf{tx}^*_{\mathrm{fin}})$ succeeds, contradicting the winning condition that recovery fails for all tuples in F_{valid}. As the number of interactions is polynomially bounded, the overall winning probability is negligible.

Server fairness. Given $(\mathsf{pp}, \mathsf{td}, \mathcal{L}_{\mathrm{exp}}, \mathsf{tx}^n_{\mathrm{adt}}, |R|)$, the simulator Sim samples $k_{\mathsf{ret}} \leftarrow \mathsf{SE.KGen}(1^\lambda)$, and computes $C \leftarrow \mathsf{SE.Enc}(k_{\mathsf{ret}}, 0^{|R|})$ and $(x_{\mathsf{ret}}, \cdot) \leftarrow \mathsf{COM.Commit}(ck, 0^{|k_{\mathsf{ret}}|})$. It simulates π_{ret} for $(m_{\mathrm{con}}, C, x_{\mathsf{ret}})$ using $\mathsf{td}_{\mathsf{ret}}$ and returns $(C, \mathsf{aux}_{\mathsf{ret}})$, where $\mathsf{aux}_{\mathsf{ret}} = (x_{\mathsf{ret}}, \pi_{\mathsf{ret}})$.

Starting from View_0, we make three replacements, each preserving computational indistinguishability. First, replace π_{ret} by a simulated proof, using zero-knowledge of Π_{ret}. Next, replace x_{ret} by a commitment to $0^{|k_{\mathsf{ret}}|}$, using hiding of COM and continuing to simulate π_{ret}. Finally, replace $C \leftarrow \mathsf{SE.Enc}(k_{\mathsf{ret}}, R)$ by $C \leftarrow \mathsf{SE.Enc}(k_{\mathsf{ret}}, 0^{|R|})$. IND-CPA security of SE justifies this last replacement, since x_{ret} no longer commits to k_{ret}. The resulting distribution is View_1. The argument remains valid with $\mathsf{st}_{\mathcal{A}}$ and R included in both views, since k_{ret} is freshly and independently generated. Hence, $\{\mathsf{View}_0\}_\lambda \approx_c \{\mathsf{View}_1\}_\lambda$.

D Further Discussion

This section discusses protocol configurations and extensions, followed by the incentive considerations beyond our cryptographic fairness guarantees.

D.1 Protocol Configurations and Extensions

Here, we consider three aspects: (1) erasure code-based preprocessing and partial PoRC extraction; (2) recursive aggregation of preservation audits; and (3) retrieval policies and configurations.

Preprocessing and partial PoRC extraction. Full extraction of every committed block is stronger than necessary for recovering the original file. Following [18], $\mathsf{PoRep.Preproc}$ may encode $\tilde{D}$ using an erasure code, while the implicit PoRC guarantees extraction of a sufficiently large subset of authenticated blocks of D. If these blocks meet the code's recovery threshold, the client can reconstruct $\tilde{D}$, using the retained encoding key ek when required. The delegation interface continues to take (D, τ_D), while recovery of $\tilde{D}$ remains part of the preprocessing scheme.

This observation concerns file recovery, rather than full-replica extraction. Our sufficient conditions use the latter for simplicity. Supporting partial PoRC extraction requires accounting for the missing replica blocks and decoding costs in the extraction guarantee and PoSpace-KoSC argument, as in [18, Lemma 2]; erasure code-based preprocessing alone does not establish reconstruction of the complete replica within the required extraction time.

Scalability optimization. The default preservation phase publishes one audit transaction per iteration. Recursive zkSNARKs [5,9] offer an alternative: the server maintains a succinct proof of the audit chain alongside the VDF chain and publishes it only at designated checkpoints. Each recursive step verifies the preceding proof, the current VDF transition and challenge derivation, and the corresponding PoRep response, while preserving the contract identifier, audit index, and state linkage. A checkpoint transaction consumes the current on-chain state output and creates the successor state certified by the aggregate proof.

This variant changes the preservation schedule and validators. The recursive proof system must satisfy the appropriate simulation-extractability guarantee [1], together with the extraction time and challenge freshness requirements used in our analysis. Aggregation reduces the volume of audit evidence published on chain, helping satisfy the cumulative transcript bound.

Retrieval policy and configurations. The main construction couples final retrieval with collateral release. Extending it to early, on-demand retrieval raises a limitation: after obtaining the data through an earlier session or from another server, the client may refuse to authorize a later finalization, leaving an honest server exposed to collateral loss.

For on-demand transfers, Filecoin's retrieval-market design [33] interleaves data transfer with incremental payments through off-chain payment channel vouchers, limiting exposure within each transfer and payment interval. Our conditional authorization mechanism could similarly be applied to separately encrypted chunks. However, directly executing an on-chain exchange for every chunk would increase transaction overhead; a channel-based realization requires a separate design and analysis. Chunking does not itself resolve the collateral-release problem above.

Batch retrieval partially mitigates this problem when a client delegates multiple replicas of identical data to the same server. The parties may jointly finalize the corresponding contracts through a single base transaction $\tilde{\mathsf{tx}}_{\mathrm{fin}}$ that references their unspent state outputs as inputs, subject to each contract's finalization conditions. A completed batch releases the included collateral together, reducing opportunities for the client to obtain the data from one replica and then refuse separate finalizations for the others.

As a minor variant, the server may encrypt D obtained through $\mathsf{PoRep.Extract}$ rather than R, with corresponding adjustments to the retrieval relation and adaptive client fairness definition.

D.2 Incentive model

Our fairness notions constrain exchange outcomes but do not force either party to participate or complete the exchange. An aborted exchange may leave the client without the requested data and the server without its collateral. Burning forfeited collateral removes a direct financial transfer to the client, rather than ensuring cooperation. Timeout rules must account for what the ledger can establish. Failure to finalize does not identify the responsible party, since either party may withhold a required off-chain message. Unconditionally releasing payment or collateral to the server may reward server-side withholding, whereas unconditional forfeiture may penalize an honest server for client-side abort.

Bilateral collateral may discourage aborts under an explicit economic model. For example, a client deposit refunded upon successful finalization may motivate participation even after the client has obtained the data, provided that the expected refund exceeds the remaining participation costs and any benefit from aborting. The expected loss of server collateral must likewise outweigh the benefit of withholding or abandoning the exchange. These are conditional incentive requirements, not a claim that cooperation is always rational. A complete model must specify the parties' utilities and consistent payment, deposit, timeout, and fault-evidence rules. Designing and analyzing additional incentive and dispute resolution mechanisms is outside our scope.

E PLONK Audit Layer Costs

Using the block size of Sec. 5, we benchmark a Merkle-PLONK [21] audit layer implemented in Circom and snarkjs [4] over BLS12-381, with Plutus V3 verification [34]. We vary the number of challenged blocks $k \in \{1, 2, 5, 10, 20\}$ at $N = 100$ and $N \in \{100, 1000, 10000\}$ at $k = 5$. Variable measurements are means over ten independent trials per setting, using four workers on an Apple M3 with 16 GiB RAM. Proof timings exclude replica and authentication path preparation; Plutus costs use the epoch-659 protocol parameter snapshot.

Table 1 reports baseline costs. Mean proving time grows from 16.36 to 279.55 s as k increases from 1 to 20 (Fig. 13). At $k = 5$, increasing N yields means of 67.86, 63.47, and 127.93 s; the largest setting doubles the proving domain

Tab. 1: Baseline costs at $N = 100$ and $k = 5$. Warm verification averages five subsequent checks per proof. Plutus costs are execution units

Measurement	Value
Proof/proof redeemer	624/697.9 B
Validator, verification key included	5249 B
Proving key/time	335.84 MiB/67.86 s
Native verification (warm)	22.76 ms
Plutus CPU/memory	$3.709 \times 10^9/1.010 \times 10^6$
Tx size (embedded/reference)	6409.4/1193.4 B

from 2^{17} to 2^{18}. Compact proofs remain 624 bytes across all settings. Maximum embedded-script and reference-script transaction sizes are 7493 and 1223 bytes, below the 16384-byte limit. CPU and memory costs remain below 40.4% and 12.1% of their respective transaction budgets. The largest embedded-script and reference-script transactions occupy 7493 and 1223 bytes, below the snapshot's 16384-byte limit. Maximum CPU and memory costs remain below 40.4% and 12.1% of the respective transaction budgets.

The on-chain measurements use synthetic, verifier-only transactions; the reference-script variant assumes a separately published script. The study excludes PoSpace, VDF execution, other DSN contract logic, and network and confirmation latency, measuring audit layer costs rather than end-to-end DSN scalability.

$\underline{\mathsf{InitRet}(\mathsf{tx}^n_{\mathrm{adt}}, \langle \mathcal{C}(sk_{\mathcal{C}}), \mathcal{S}(R)\rangle):}$

return $\perp$ **if** $\mathsf{tx}^n_{\mathrm{adt}} \notin \mathcal{L} \vee \mathsf{out}^n_{\mathrm{adt}}$ is spent

$\mathcal{S}: k_{\mathsf{ret}} \leftarrow \mathsf{SE.KGen}(1^\lambda);\ C \leftarrow \mathsf{SE.Enc}(k_{\mathsf{ret}}, R)$

$\quad (x_{\mathsf{ret}}, o_{\mathsf{ret}}) \leftarrow \mathsf{COM.Commit}(ck, k_{\mathsf{ret}})$

$\pi_{\mathsf{ret}} \leftarrow \Pi_{\mathsf{ret}}.\mathsf{Prove}(\mathsf{crs}_{\mathsf{ret}}, (m_{\mathrm{con}}, C, x_{\mathsf{ret}}), (R, k_{\mathsf{ret}}, o_{\mathsf{ret}}))$

$\quad \mathsf{aux}_{\mathsf{ret}} \coloneqq (x_{\mathsf{ret}}, \pi_{\mathsf{ret}});\ \mathcal{S} \to \mathcal{C}: (C, \mathsf{aux}_{\mathsf{ret}})$

$\mathcal{C}:$ **return** $\perp$ **if** $\Pi_{\mathsf{ret}}.\mathsf{Vf}(\mathsf{crs}_{\mathsf{ret}}, (m_{\mathrm{con}}, C, x_{\mathsf{ret}}), \pi_{\mathsf{ret}}) = 0$

$\quad \tilde{\rho}_{\mathrm{fin}} \coloneqq (\mathsf{aux}_{\mathsf{ret}}, (\perp, \perp));\ \mathsf{out}_{\mathrm{fin}} \coloneqq (\nu_{\mathrm{fin}}, c, m_{\mathrm{con}})$

$\quad \tilde{\mathsf{tx}}_{\mathrm{fin}} \coloneqq ((\mathsf{out}^n_{\mathrm{adt}}, \tilde{\rho}_{\mathrm{fin}}), \mathsf{out}_{\mathrm{fin}}, vi_{\mathrm{fin}})$

$\quad \sigma_{\mathcal{C}} \leftarrow \mathsf{SIG.Sign}(sk_{\mathcal{C}}, \tilde{\mathsf{tx}}_{\mathrm{fin}})$

$\quad \hat{\rho}_{\mathrm{fin}} \coloneqq (\mathsf{aux}_{\mathsf{ret}}, (\sigma_{\mathcal{C}}, \perp))$

$\mathcal{C} \to \mathcal{S}: \hat{\mathsf{tx}}_{\mathrm{fin}} \coloneqq ((\mathsf{out}^n_{\mathrm{adt}}, \hat{\rho}_{\mathrm{fin}}), \mathsf{out}_{\mathrm{fin}}, vi_{\mathrm{fin}})$

$\mathcal{S}: \mathsf{st}_{\mathcal{S}} \coloneqq (\mathsf{tx}^n_{\mathrm{adt}}, k_{\mathsf{ret}}, o_{\mathsf{ret}}, C, \mathsf{aux}_{\mathsf{ret}})$

return $(C, \mathsf{aux}_{\mathsf{ret}}, \hat{\mathsf{tx}}_{\mathrm{fin}}; \mathsf{st}_{\mathcal{S}})$

$\underline{\mathsf{RetVf}(\mathsf{tx}^n_{\mathrm{adt}}, C, \mathsf{aux}_{\mathsf{ret}}, \hat{\mathsf{tx}}_{\mathrm{fin}}):}$

return 0 **if** $\mathsf{tx}^n_{\mathrm{adt}} \notin \mathcal{L} \vee \mathsf{out}^n_{\mathrm{adt}}$ is spent

parse $\mathsf{aux}_{\mathsf{ret}} = (x_{\mathsf{ret}}, \pi_{\mathsf{ret}}),\ \hat{\rho}_{\mathrm{fin}} = (\cdot, (\sigma_{\mathcal{C}}, \perp))$ from $\hat{\mathsf{tx}}_{\mathrm{fin}}$

$\tilde{\mathsf{tx}}_{\mathrm{fin}} \coloneqq ((\mathsf{out}^n_{\mathrm{adt}}, (\mathsf{aux}_{\mathsf{ret}}, (\perp, \perp))), (\nu_{\mathrm{fin}}, c, m_{\mathrm{con}}), vi_{\mathrm{fin}})$

return $(\tilde{\hat{\mathsf{tx}}}_{\mathrm{fin}} = \tilde{\mathsf{tx}}_{\mathrm{fin}}) \wedge (r \in vi_{\mathrm{fin}}) \wedge (r + u \leq \max(vi_{\mathrm{fin}}))$

$\quad \wedge\ \mathsf{SIG.Vf}(pk_{\mathcal{C}}, \tilde{\mathsf{tx}}_{\mathrm{fin}}, \sigma_{\mathcal{C}}) = 1$

$\quad \wedge\ \Pi_{\mathsf{ret}}.\mathsf{Vf}(\mathsf{crs}_{\mathsf{ret}}, (m_{\mathrm{con}}, C, x_{\mathsf{ret}}), \pi_{\mathsf{ret}}) = 1$

$\underline{\mathsf{Finalize}(\mathsf{st}_{\mathcal{S}}, \hat{\mathsf{tx}}_{\mathrm{fin}}):}$

parse $\mathsf{st}_{\mathcal{S}} = (\mathsf{tx}^n_{\mathrm{adt}}, k_{\mathsf{ret}}, o_{\mathsf{ret}}, C, \mathsf{aux}_{\mathsf{ret}}),\ \mathsf{aux}_{\mathsf{ret}} = (x_{\mathsf{ret}}, \pi_{\mathsf{ret}})$

return $\perp$ **if** $\mathsf{RetVf}(\mathsf{tx}^n_{\mathrm{adt}}, C, \mathsf{aux}_{\mathsf{ret}}, \hat{\mathsf{tx}}_{\mathrm{fin}}) = 0$

$\quad \vee\ \mathsf{COM.Vf}(ck, x_{\mathsf{ret}}, k_{\mathsf{ret}}, o_{\mathsf{ret}}) = 0$

parse $\hat{\mathsf{tx}}_{\mathrm{fin}} = ((\mathsf{out}^n_{\mathrm{adt}}, (\mathsf{aux}_{\mathsf{ret}}, (\sigma_{\mathcal{C}}, \perp))), \mathsf{out}_{\mathrm{fin}}, vi_{\mathrm{fin}})$

$\rho_{\mathrm{fin}} \coloneqq (\mathsf{aux}_{\mathsf{ret}}, (\sigma_{\mathcal{C}}, (k_{\mathsf{ret}}, o_{\mathsf{ret}})))$

submit $\mathsf{tx}_{\mathrm{fin}} \coloneqq ((\mathsf{out}^n_{\mathrm{adt}}, \rho_{\mathrm{fin}}), \mathsf{out}_{\mathrm{fin}}, vi_{\mathrm{fin}})$

$\underline{\mathsf{RetExt}(C, \hat{\mathsf{tx}}_{\mathrm{fin}}, \mathsf{tx}_{\mathrm{fin}}):}$

return $\perp$ **if** $\mathsf{tx}_{\mathrm{fin}} \notin \mathcal{L} \vee \tilde{\hat{\mathsf{tx}}}_{\mathrm{fin}} \neq \tilde{\mathsf{tx}}_{\mathrm{fin}}$

parse $\rho_{\mathrm{fin}} = ((x_{\mathsf{ret}}, \pi_{\mathsf{ret}}), (\cdot, (k_{\mathsf{ret}}, o_{\mathsf{ret}})))$ from $\mathsf{tx}_{\mathrm{fin}}$

return $\perp$ **if** $\mathsf{COM.Vf}(ck, x_{\mathsf{ret}}, k_{\mathsf{ret}}, o_{\mathsf{ret}}) = 0$

$R \leftarrow \mathsf{SE.Dec}(k_{\mathsf{ret}}, C);$ **return** $\perp$ **if** $R = \perp$

$D \leftarrow \mathsf{PoRep.Extract}(id, \tau_D, R);$ **return** $\perp$ **if** $D = \perp$

return $\perp$ **if** $\mathsf{PoRep.TagVf}(D, \tau_D) = 0$

return (R, D)

Fig. 10: The retrieval phase. The current ledger round is denoted by r. For finalization transactions, forming the base replaces both the client's signature and the key-opening pair by $\perp$, *i.e.*, $\tilde{\rho}_{\mathrm{fin}} \coloneqq (\mathsf{aux}_{\mathsf{ret}}, (\perp, \perp))$.

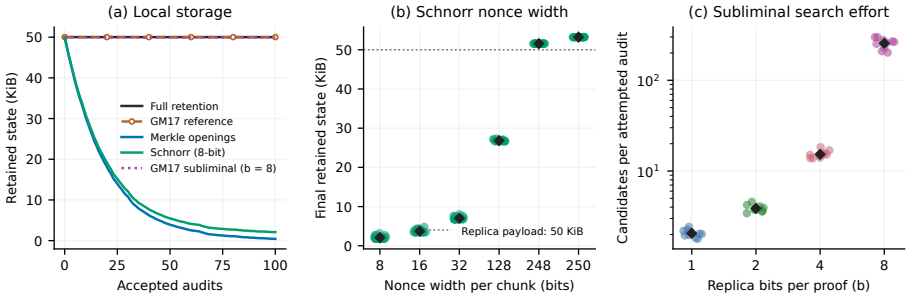

Fig. 11: Storage offloading and GM17 subliminal search cost. (a) Serialized retained state over audits. (b) Final retained state versus Schnorr nonce width. (c) Candidate proofs examined per audit. Merkle and Schnorr configurations each use 20 independent runs; GM17 uses 10 runs with replicas and challenge schedules matching the corresponding Merkle and Schnorr runs. Curves in (a) and diamonds in (b) and (c) show means. Colored points show individual runs' final retained states in (b) and mean candidate counts per audit in (c).

Experiment $\mathrm{PoRCSound}_{\mathcal{A}}(\mathsf{pp}, m)$:

$(c, o, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp}, m)$; $m' \leftarrow \mathsf{PoRC.Open}(c, o)$

return 0 **if** $m' \neq m$

parse $c = (id, \tau_D, \mathsf{aux})$

$\mathsf{ch} \leftarrow \mathsf{PoRep.Poll}(\mathsf{aux})$; $\pi \leftarrow \mathcal{A}_2(\mathsf{st}, c, \mathsf{ch})$

return $\mathsf{PoRC.Vf}(c, \mathsf{ch}, \pi)$

Fig. 12: The PoRC soundness experiment of [18, Appendix A.4].

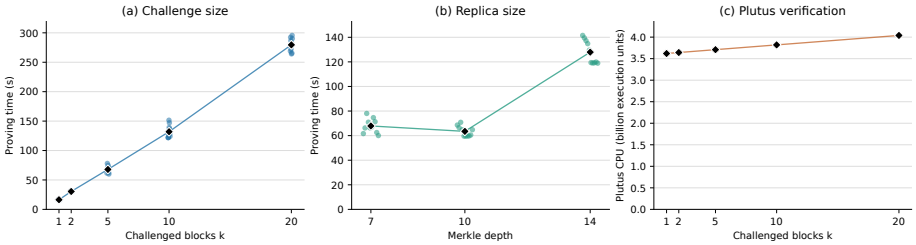

Fig. 13: Proving time versus (a) challenged positions and (b) Merkle depth; (c) Plutus CPU cost versus challenged positions. Diamonds denote means; colored points denote individual trials.