Privacy-Preserving Outsourced Witness Updates for Append-Only RSA Accumulators

Hongzi He[1], Qianhong Wu[1], Bo Qin[2], Hao Gao[1], and Willy Susilo[3]

[1] School of Cyber Science and Technology, Beihang University, Beijing 100191, China
[2] School of Information, Renmin University of China, Beijing 100872, China
bo.qin@ruc.edu.cn
[3] School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia

Abstract. Append-only accumulators are a natural way to realize compact public-state registries, but under high-frequency updates, witness maintenance becomes a severe challenge because each insertion typically invalidates most existing witnesses. This challenge is particularly acute for intermittently online users in anonymous credential systems, who cannot continuously synchronize update information, while directly outsourcing witness updates may make repeated requests linkable. In this paper, we present a privacy-preserving outsourced witness-update protocol for append-only RSA accumulators. The protocol combines witness updates with Linear Integer Secret Sharing (LISS), enabling on-demand, client-stateless witness updates while preserving witness privacy and unlinkability against coalitions of update servers below the threshold, and providing accountability for malicious or malformed server responses. We formalize the system and threat models and analyze the security of the protocol. We further develop server-side optimizations for long catch-up windows and implement the full end-to-end protocol in Rust. Evaluation under multiple threshold settings and offline windows shows that the protocol supports practical one-shot witness updates after long offline periods, with client-side cost remaining independent of the number of missed updates and server-side cost being mainly determined by the catch-up span and the LISS distribution matrix.

Keywords: privacy-preserving protocols · RSA accumulators · Linear Integer Secret Sharing · witness update · unlinkability

1 Introduction

Anonymous credentials are a central primitive for privacy-preserving authentication: a user can prove eligibility without revealing a stable identifier or making repeated authentications linkable [11,3,23,33,31]. A major line of recent work aims to make anonymous credentials easier to deploy by decoupling issuance from verification through a publicly auditable issuance registry. For example, zk-creds [31] shows how public registry state can reduce reliance on issuers to

hold anonymous-credential signing keys or deploy specialized verification infrastructure, while still allowing verifiers to check credential validity against public state. Cryptographic accumulators are a natural way to realize such registries: the public state is a compact digest and each issued element has a compact membership witness [4,12,27,28]. Once validity is tied to a continuously growing append-only public state, however, witness update becomes a central practical challenge.

As new elements are continuously appended to the issuance set, previously issued witnesses generally can no longer be used directly against the latest digest. This is a structural phenomenon rather than a peculiarity of one accumulator family. In classical RSA accumulators and Merkle-tree-based constructions, inserting a new element typically forces many existing witnesses to change. Recent lower bounds further suggest that substantial witness churn is inherent for succinct append-only set commitments [9,30,1]. A natural response is to redesign the authenticated data structure to reduce witness-update frequency. Merkle forests and Merkle Mountain Ranges follow this direction by organizing state into multiple trees and merging them in a structured way, thereby reducing the number of local witness changes over the lifetime of the system [9].

Reducing update frequency, however, does not eliminate client-side participation. To keep a witness usable, a client must still obtain update information, track the evolution of the public structure, and locally propagate its witness to the latest state. In high-frequency issuance environments, this quickly becomes burdensome, especially for intermittently online users that cannot continuously follow the update stream. A different design choice is to delegate witness-update computation to an external service. This removes continuous client-side synchronization, but it introduces a distinct privacy cost: update requests themselves may expose witness material or become linkable over time, allowing the service provider to correlate repeated interactions of the same hidden user. These observations leave a missing point in the design space: outsourced witness update should be on-demand and client-stateless, but should not turn the update service into a privacy bottleneck.

Table 1 summarizes this design space from the client perspective. The gap motivates the following question: can witness updates be outsourced in a way that supports stateless catch-up for intermittently online clients, preserves witness privacy and unlinkability with respect to a threshold-bounded update service, and provides accountable server-side computation?

Our approach. We introduce a threshold-private, unlinkable, and accountable outsourced witness-update protocol for append-only RSA accumulators. At a high level, an intermittently online client obtains an updated witness on demand without synchronizing the missed update stream or maintaining local update history. The system uses a public bulletin board that stores endorsed accumulator digests together with epoch indices. A client retrieves this public state, contacts a qualified set of update servers, and reconstructs an updated witness from verified server responses.

Table 1. Comparison of witness-update approaches

Witness-update approach	**Construction**	**Commitment size**	**Client-side witness-update frequency**	**Witness privacy**	**Unlinkable**
Classical client-updated	Classical accumulators [4,27,28]	$\Theta(1)$	$O(n)$	N/A	N/A
Frequency-optimized client-updated	Merkle Mountain Ranges[9]	$O(\log^2 n)$	$O(\log n / \log \log n)$	N/A	N/A
Generic outsourced update	Classical accumulators [4,27,28]	$\Theta(1)$	On-demand	✗	✗
Ours	**RSA accumulator**[13]	$\Theta(1)$	**On-demand**	✓	✓

Notes. Here n is the number of append operations since the witness was last synchronized. "On-demand" means that the client updates only when needed, rather than following every append-induced witness change. The unlinkability entry for our protocol refers to witness-side linkage under fixed public metadata; public epoch-window metadata is discussed as a deployment limitation in Section 7.

The technical core is the algebraic compatibility between append-only RSA witness update and Linear Integer Secret Sharing (LISS) over the integers [19]. Let W denote a stale witness, and let Δ denote the product of elements inserted since that witness was last synchronized. The updated witness satisfies

$$W' = W^{\Delta} \bmod N.$$

This form allows the same catch-up exponent to be applied directly to blinded witness-share units and later recombined into the updated witness. Concretely, the client encodes W into fresh randomized LISS share units, distributes the resulting blinded powers across update servers, and reconstructs W' from any qualified subset of returned results.

The construction addresses the main security goals of outsourced witness update in a unified way. First, randomized LISS encoding provides threshold privacy: fewer than t update servers learn no additional information about the underlying witness beyond public state and visible protocol metadata. Second, because update servers operate only on fresh randomized powers rather than reusable witness material, repeated requests do not expose a stable witness-side identifier. Third, each update server returns PoKE proofs showing that its output

share units are derived under the intended catch-up exponent; the client verifies these proofs before reconstruction and rejects forged or inconsistent responses.
Contributions. In summary, this paper makes the following contributions.

- We study privacy-preserving outsourced witness update for intermittently online clients in append-only accumulators, and present a threshold-private, unlinkable, and accountable protocol for append-only RSA accumulators. The protocol supports on-demand, client-stateless catch-up without requiring clients to synchronize missed updates or maintain local update history.
- We formalize the system model, threat model, and security goals for outsourced witness update. We provide theorem-level security statements and proof sketches in the main body, and detailed proofs for correctness, threshold witness privacy, unlinkability, accountability, and availability in Appendix B.
- We develop server-side optimizations for long catch-up windows, implement the full protocol in Rust, and evaluate client-side and server-side costs under multiple threshold settings and offline windows. The results show that client-side cost is independent of the number of missed updates, while the main server-side bottlenecks come from catch-up span and the LISS distribution matrix.

2 Preliminaries

2.1 Commitment Schemes

A commitment scheme $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Comm}, \mathsf{Open})$ allows a party to commit to a value or to a collection of values while preserving the standard hiding and binding properties. We denote a commitment to a message or vector x with opening randomness o under key ck by

$$c = \mathsf{Comm}(\mathsf{ck}, x; o).$$

Correctness requires that honestly generated openings verify, hiding requires that commitments reveal no information about the committed value, and binding requires that a commitment cannot be opened to two different values except with negligible probability.

2.2 zk-SNARKs and Commit-and-Prove SNARKs

A zk-SNARK for a relation R consists of setup, proving, and verification algorithms. The setup algorithm generates proving and verification parameters; the proving algorithm takes a statement x and witness w with $(x, w) \in R$ and outputs a proof π; the verification algorithm accepts or rejects π with respect to x. The usual properties are completeness, knowledge soundness, and zero knowledge.

A commit-and-prove SNARK (CP-SNARK) [14] extends this syntax to relations with committed inputs. For a commitment $c_u = \mathsf{Comm}(\mathsf{ck}, u; o_u)$ and an

inner relation $R_{\mathsf{in}}(x; u, \omega)$, the corresponding committed-input relation requires knowledge of an opening (u, o_u) and auxiliary witness ω such that $R_{\mathsf{in}}(x; u, \omega) = 1$. We write this relation as $\widetilde{R}(c_u, x; \omega) = 1$ and use the CP-SNARK procedures $(\mathsf{cp.Setup}, \mathsf{cp.Prove}, \mathsf{cp.Verify})$.

2.3 Append-only RSA Accumulators and Witness Update

Let $\mathbb{G} \subseteq \mathbb{Z}_N^*$ be a group of unknown order with generator $g \in \mathbb{G}$, and let $\mathsf{pp} = (N, g)$ denote the public parameters. For a set S of prime representatives, the RSA accumulator digest is

$$\mathsf{acc} := g^{\prod_{x \in S} x} \bmod N.$$

A membership witness for $x \in S$ is

$$W_x := g^{\prod_{y \in S \setminus \{x\}} y} \bmod N, \qquad W_x^x \equiv \mathsf{acc} \pmod N.$$

Appending a batch U defines the public update factor

$$\Delta_U := \prod_{u \in U} u.$$

The updated digest is

$$\mathsf{acc}' = \mathsf{acc}^{\Delta_U} \bmod N,$$

and an existing witness updates as

$$W_x' = W_x^{\Delta_U} \bmod N.$$

Equivalently, the insertion and witness-update interfaces can be written as

$$\mathsf{Ins}(\mathsf{pp}, \mathsf{acc}, U) \to \mathsf{acc}', \qquad \mathsf{MemWitUp}(\mathsf{pp}, W_x, \Delta_U) \to W_x'.$$

For a sequence of appended batches over an epoch window (e_0, e), the catch-up factor is

$$\Delta_{e_0 \to e} := \prod_{u=e_0+1}^{e} \Delta_u,$$

so that

$$\mathsf{acc}_e = \mathsf{acc}_{e_0}^{\Delta_{e_0 \to e}} \bmod N, \qquad W_{x,e} = W_{x,e_0}^{\Delta_{e_0 \to e}} \bmod N.$$

2.4 Linear Integer Secret Sharing

A Linear Integer Secret Sharing (LISS) scheme [19] is a linear secret-sharing scheme over the integers, commonly specified through an integer span program (ISP). For the threshold access structure $\Gamma_{t,n}$, where any set of at least t parties is qualified, a LISS scheme provides

$$\mathsf{LISS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Coeff}, \mathsf{Combine}).$$

The setup algorithm outputs a public ISP/LISS description $\mathsf{isp} = (M, \psi, \boldsymbol{\varepsilon})$, where $M \in \mathbb{Z}^{d\times e}$ is the distribution matrix, $\psi : [d] \to [n]$ assigns rows to parties, and $\boldsymbol{\varepsilon} = (1, 0, \ldots, 0)^\top$ is the target vector. Party P_j is associated with

$$T_j := \{i \in [d] : \psi(i) = j\}.$$

To share a secret s, the sharing algorithm samples $\boldsymbol{\rho} = (s, \rho_2, \ldots, \rho_e)^\top$ and computes

$$\boldsymbol{\sigma} = M\boldsymbol{\rho} \in \mathbb{Z}^d.$$

Party P_j receives the share units indexed by T_j. For a qualified set $A \in \Gamma_{t,n}$, let $I_A := \bigcup_{P_j\in A} T_j$. The coefficient algorithm outputs public reconstruction coefficients $\{\lambda_i\}_{i\in I_A}$ satisfying

$$\sum_{i\in I_A} \lambda_i M[i] = \boldsymbol{\varepsilon}^\top.$$

Combining the corresponding share units reconstructs the secret:

$$\sum_{i\in I_A} \lambda_i \sigma_i = s.$$

Correctness requires reconstruction for every qualified set, while privacy requires that every forbidden set learns only statistically negligible information about the secret.

2.5 Proofs of Knowledge of Exponent

Let $\mathbb{G} \subseteq \mathbb{Z}_N^*$ be a group of unknown order. A proof of knowledge of exponent (PoKE) is an argument of knowledge for the relation

$$R_{\mathsf{PoKE}}(A, B; \alpha) = 1 \iff A^\alpha \equiv B \pmod N.$$

A PoKE scheme is specified by PPT algorithms

$$\mathsf{PoKE} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}).$$

The setup algorithm outputs public parameters; the proving algorithm takes group elements $A, B \in \mathbb{G}$ and a witness exponent α satisfying $A^\alpha \equiv B \pmod N$ and outputs a proof π; the verification algorithm checks the proof with respect to (A, B). In the unknown-order generic group model, the scheme satisfies completeness and knowledge soundness. We instantiate PoKE using the construction in [8].

3 System Model

The system involves an Accumulator Manager, an Auditor, a Bulletin Board, Update Servers, and Clients. The Manager batches append operations and prepares

epoch-update data. The Auditor verifies each state transition before the endorsed digest is posted to the append-only Bulletin Board. Update Servers store verified update data and answer outsourced witness-update requests. Clients are intermittently online parties that hold stale witnesses and update them only when needed. Conceptually, the public-state maintenance logic and the witness-update logic are separated: the former makes epoch updates auditable, while the latter lets clients obtain fresh witnesses without revealing the representative or unblinded witness to update servers.

3.1 Entities and Public State

Accumulator Manager. The Accumulator Manager maintains the append-only accumulator state across epochs. For each epoch it chooses the inserted batch, computes the updated digest and update factor, and produces the proof material that binds the public digest transition to the server-side update factor. The Manager is not trusted for correctness: an epoch becomes usable only after the Auditor accepts the corresponding proof and the Bulletin Board records the endorsed state.

Auditor and Bulletin Board. The Auditor verifies the epoch-update proof submitted by the Manager. Upon acceptance it endorses the new epoch record and publishes it to the Bulletin Board. The Bulletin Board is an append-only, epoch-indexed public log containing endorsed accumulator digests and commitments to inserted batches. It is the source of public state for both clients and update servers: clients use it to identify the current target digest, and servers use it to check that Manager-provided update messages correspond to endorsed epochs. In practice, this role can be instantiated by a smart contract or any replicated append-only log with public verifiability.

Update Servers. Update Servers are distributed computational entities that support outsourced witness update. A server stores only epoch-update data that has passed local verification against the Bulletin Board, and it uses this verified data to assemble catch-up exponents for client requests. It receives only blinded witness-share units, not the representative x or the unblinded stale witness. A server response is therefore a proof-bearing share update, not a direct updated witness.

Clients. A client holds a representative x, a stale witness W_{old}, and the epoch index e_0 at which that witness is valid. The client need not store per-epoch update messages. When it needs a fresh witness, it reads the latest endorsed digest, creates a fresh LISS sharing instance for the exponent 1, sends server-specific blinded share units to update servers, verifies the returned PoKEs, and reconstructs W_{new} from a qualified set of valid responses.

3.2 Threat Model and Trust Assumptions

We consider a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ and adopt the standard Dolev–Yao network model [21], where $\mathcal{A}$ can eavesdrop, intercept, modify, and inject network messages. Honest parties communicate over authenticated channels. We do not consider linkability arising from network-layer identifiers such as IP addresses; this issue is orthogonal to witness-update privacy and can be mitigated by anonymous-communication mechanisms such as Tor [20]. We further assume that the unknown-order group satisfies the Strong RSA assumption, that CP-SNARKs are sound, and that PoKE satisfies completeness and knowledge soundness in the unknown-order generic group model.

Trust Model and Corruptions. The adversary may corrupt the following entities:

- **Update Servers:** $\mathcal{A}$ may non-adaptively corrupt any subset of fewer than t update servers and obtain their internal state and local protocol transcripts. Corrupted servers may deviate arbitrarily, e.g., by returning malformed share updates, invalid proofs, or no response. Each witness-update execution is assumed to have at least t honest and responsive update servers.
- **Accumulator Manager:** The Accumulator Manager is potentially malicious. It may attempt to equivocate on accumulator state, disseminate malformed epoch updates, or collude with corrupted update servers. Pure denial-of-service by the Manager is outside our scope.
- **Auditor and Bulletin Board:** The Auditor and append-only Bulletin Board are modeled as a trusted immutable public-state service. This service verifies epoch-update proofs and maintains a globally consistent append-only log of endorsed epoch records. In practice, this public-state service can be implemented using a blockchain ledger, with a smart contract verifying epoch-update proofs.
- **Clients:** We focus on honest clients executing the protocol as specified. Client-initiated denial-of-service and resource-exhaustion behavior are outside the scope.

3.3 Security Goals

Under the above threat model, the outsourced witness-update protocol is designed to provide:

- **Correctness.** If the client obtains correct responses from at least t honest and responsive update servers, it reconstructs a valid updated witness for the latest accumulator digest.
- **Availability.** If at least t honest and responsive update servers remain available, an honest client can complete witness update despite the remaining servers being malicious or offline.
- **Threshold Privacy.** Any coalition of fewer than t corrupted update servers learns no additional information about the hidden witness or accumulated element beyond public state and visible protocol metadata.

- **Unlinkability.** Any coalition of fewer than t corrupted update servers cannot determine whether two witness-update executions originate from the same hidden representative, except through public metadata.
- **Accountability.** Malformed or malicious server-side update responses can be detected and attributed to the corresponding responding server.

4 Protocol Construction

4.1 Overview of the Protocol

The protocol combines append-only RSA witness update with LISS-based blinding. The client shares the exponent 1 under the public ISP/LISS description and sends each contacted server only the blinded witness-share units assigned to it. A server assembles the catch-up exponent for the requested epoch window, updates its blinded units, and attaches PoKE proofs. The client verifies the proofs, keeps any qualified set of valid responses, and reconstructs the refreshed witness. As a result, the client does not maintain or process epoch-level update material locally and interacts with update servers only when it needs to refresh a stale witness.

During initialization, the RSA-accumulator parameters are generated without a single trusted trapdoor holder, e.g., using a distributed setup [10]. The (t, n) threshold policy is compiled into a public ISP/LISS description via the Benaloh–Leichter technique [5]. This public description determines both how blinded share units are assigned to update servers and how the updated witness is reconstructed from a qualified server subset.

The protocol has two layers. The *accumulator-management layer* evolves the append-only public state, verifies every epoch transition before endorsement, and disseminates verified update factors to the update servers. The *witness-update layer* handles on-demand client catch-up: the client blinds a stale witness into LISS share units, servers update the assigned units under the catch-up exponent, and the client reconstructs the refreshed witness after verifying server accountability proofs.

A complete outsourced update execution has four phases, shown in Figure 1. First, the client reads the endorsed source and target digests from the Bulletin Board and fixes the epoch window (e_0, e). Second, it generates a fresh LISS sharing of the exponent 1 and derives server-specific blinded request packages. Third, each server locally assembles the verified catch-up factor for the requested window and updates only the share units assigned to it. Finally, the client verifies row-indexed PoKEs, discards invalid items, and reconstructs from any qualified set of accepted rows. The separation between public state retrieval, randomized request generation, server-side catch-up, and client-side reconstruction is important: public state is auditable, while all witness-dependent group elements are freshly randomized per execution.

The workflow has two different kinds of information flow. The public-state flow is between the client, the Bulletin Board, and the update servers: the client learns the target epoch and digest, while servers use the same public log to verify

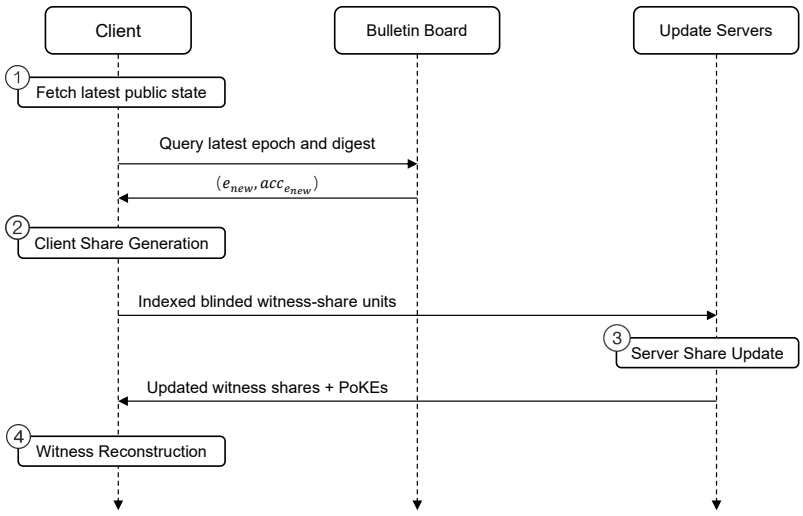

Fig. 1. Workflow of one outsourced witness-update execution.

that their stored update messages correspond to endorsed epochs. The witness-dependent flow is separate: the client sends only indexed blinded witness-share units, each server returns updated share units together with row-bound PoKEs, and the final witness appears only after local reconstruction by the client. This separation is what allows the server side to perform the expensive catch-up exponentiation without learning the representative x or the unblinded witness.

4.2 Accumulator Management

For epoch e, the Accumulator Manager inserts the batch U_e, computes $\Delta_e = \prod_{u \in U_e} u$ and $\mathsf{acc}_e = \mathsf{acc}_{e-1}^{\Delta_e} \bmod N$, and prepares proofs binding the public digest transition and the server-side update factor to the same committed batch. We assume that a client already holds a valid membership witness for its representative; witness issuance is an external preprocessing step and is not part of the outsourced update protocol. The detailed epoch-update procedure is given in Appendix A.

Audit and Publish. The Auditor receives $(\mathsf{acc}_e, c_{U_e}, \pi^{\mathsf{acc}}_{\mathsf{update},e})$ together with the prior endorsed digest acc_{e-1}. It verifies the state-transition proof on the public statement $(e, \mathsf{acc}_{e-1}, \mathsf{acc}_e, c_{U_e})$. If verification succeeds, the Auditor endorses the new digest and publishes the epoch record $(e, \mathsf{acc}_e, c_{U_e})$ on the Bulletin Board; otherwise, the epoch update is rejected and no new public state is finalized.

Server Verification and Storage. After receiving $(\mathsf{updmsg}_e, c_{U_e}, \pi^{\Delta}_{\mathsf{update},e})$ from the Manager, each update server waits until the corresponding endorsed epoch record

is available on the Bulletin Board. It then verifies $\pi^{\Delta}_{\mathsf{update},e}$ for the statement (e, c_{U_e}, Δ_e), where $\mathsf{updmsg}_e = (e, \Delta_e)$, and checks that c_{U_e} matches the commitment in the endorsed epoch record. Only if both checks succeed is updmsg_e stored and used in subsequent witness updates.

4.3 Witness Update

To update a stale witness at epoch e_0, the client fetches the latest endorsed digest acc_e, contacts a server set B with $|B| \geq t$, and creates fresh blinded requests using the public ISP/LISS description. Update servers maintain locally verified epoch update messages and, upon request, assemble the catch-up update over the requested epoch window. The client neither stores nor processes epoch-level update material locally. If fewer than t valid responses are obtained, the client can contact additional servers under the same sharing instance until a qualified subset is available or an application-defined timeout is reached.

Request Metadata. Each request contains the stale epoch e_0, the target epoch e, and the server-specific blinded request package. The epoch window is public metadata: it is needed by the server to select the verified update factors and by the client to verify that the returned PoKEs are bound to the intended public digest transition. The representative x and the unblinded witness W_{old} are never sent to the update servers. Repeated requests use independent sharing randomness, so even if the same stale witness is updated in two executions, the server-side group elements are fresh randomized powers.

The client may contact exactly t servers when it expects all of them to be responsive, or more than t servers when it wants to tolerate delays and invalid responses. Contacting additional servers does not change the reconstruction equation: the client ultimately chooses a qualified accepted subset A and uses only the rows in I_A. It also does not weaken threshold privacy against coalitions of fewer than t corrupted servers, because every such coalition still sees an unqualified set of LISS rows generated with fresh randomness.

Response Semantics. The response of a server is not merely an updated group element. It is an indexed set of updated share units together with proofs that tie those units to the public transition from acc_{e_0} to acc_e. The row index i is included in the hash defining the folded PoKE challenge, so a malicious server cannot safely swap responses between rows or reuse a proof for a different share unit. This indexing is also what makes rejected items attributable: every row belongs to a unique server according to the public row-assignment map ψ.

Client Share Generation. The client fixes the LISS secret to $s := 1$. Thus, for every qualified set A, the reconstruction coefficients satisfy $\sum_{i \in I_A} \lambda_i \sigma_i = 1$, which makes the final reconstruction recover the updated witness itself rather than a non-trivial power of it. The client samples $\boldsymbol{\rho} = (1, \rho_2, \ldots, \rho_e)^\top$, computes $\boldsymbol{\sigma} = M\boldsymbol{\rho}$, and sends server j the indexed blinded elements $\widetilde{W}_{j,i} = W_{\text{old}}^{\sigma_i} \bmod N$

Algorithm 1 Client Share Generation

Input: Stale witness W_{old}; public ISP/LISS description $\mathsf{isp} = (M, \psi, \boldsymbol{\varepsilon})$ with $M \in \mathbb{Z}^{d\times e}$ and $\boldsymbol{\varepsilon} \in \mathbb{Z}^e$; chosen server set B with $|B| \geq t$; RSA modulus N.

Output: Blinded request sets $\{\widetilde{\mathsf{req}}_j\}_{j\in B}$, where $\widetilde{\mathsf{req}}_j = \{(i, \widetilde{W}_{j,i})\}_{i\in T_j}$ and $T_j = \psi^{-1}(j)$.

Sample fresh randomness and form a distribution vector $\boldsymbol{\rho} \leftarrow (1, \rho_2, \ldots, \rho_e)^\top \in \mathbb{Z}^e$
Compute share units $\boldsymbol{\sigma} \leftarrow M \cdot \boldsymbol{\rho} \in \mathbb{Z}^d$
for each server $j \in B$ **do**
    $T_j \leftarrow \{\, i \in \{1, \ldots, d\} \;:\; \psi(i) = j \,\}$
    Initialize $\widetilde{\mathsf{req}}_j \leftarrow \emptyset$
    for each row index $i \in T_j$ **do**
        Compute blinded element $\widetilde{W}_{j,i} \leftarrow W_{\text{old}}^{\sigma_i} \bmod N$
        $\widetilde{\mathsf{req}}_j \leftarrow \widetilde{\mathsf{req}}_j \cup \{(i, \widetilde{W}_{j,i})\}$
    end for
end for
return $\{\widetilde{\mathsf{req}}_j\}_{j\in B}$

for $i \in T_j$. By LISS privacy, any coalition of fewer than t servers sees only randomized powers of the stale witness, whereas any qualified set remains sufficient for reconstruction. Algorithm 1 gives the exact procedure.

Server Share Update. For a request window (e_0, e), server j assembles $\Delta_{e_0\to e} = \prod_{u=e_0+1}^{e} \Delta_u$ from verified update messages. For each blinded unit it computes $W'_{j,i} = \widetilde{W}_{j,i}^{\Delta_{e_0\to e}} \bmod N$. To make the result accountable, the server uses a Fiat–Shamir challenge [22] to fold the share-update relation $\widetilde{W}_{j,i}^{\Delta_{e_0\to e}} = W'_{j,i}$ and the public digest-transition relation $\mathsf{acc}_{e_0}^{\Delta_{e_0\to e}} = \mathsf{acc}_e$ into one PoKE statement. The returned response set therefore contains both the updated witness-share units and the associated correctness proofs. Algorithm 2 specifies the response computation.

Client Reconstruction and Verification. After collecting replies, the client verifies the PoKEs and keeps only valid updated share units. It selects a qualified valid responder set A, derives $\{\lambda_i\}_{i\in I_A}$, parses the accepted response sets into a unified indexed set $\mathcal{R}_A = \{(i, W'_i)\}_{i\in I_A}$, and reconstructs

$$W_{\text{new}} = \prod_{i\in I_A} (W'_i)^{\lambda_i} = W_{\text{old}}^{\Delta_{e_0\to e}\cdot\sum_{i\in I_A}\lambda_i\sigma_i} \equiv W_{\text{old}}^{\Delta_{e_0\to e}} \pmod N.$$

The final consistency check $W_{\text{new}}^{x} \equiv \mathsf{acc}_e \pmod N$ prevents the client from accepting a malformed reconstructed witness even if some accepted responses are inconsistent with the client's hidden representative. Algorithm 3 gives the exact reconstruction routine.

This final check is deliberately kept on the client side. Servers do not learn x and therefore cannot verify membership of the hidden representative directly.

Algorithm 2 Server Share Update

Input: Public RSA-accumulator parameters $\mathsf{pp} = (N, g)$; blinded request set $\widetilde{\mathsf{req}}_j = \{(i, \widetilde{W}_{j,i})\}_{i \in T_j}$; stale epoch index e_0; target epoch index e; endorsed digests $(\mathsf{acc}_{e_0}, \mathsf{acc}_e)$; stored verified update messages $\{\mathsf{updmsg}_u\}_{u=e_0+1}^{e}$ where $\mathsf{updmsg}_u = (u, \Delta_u)$.

Output: Response set $\mathsf{resp}_j = \{(i, W'_{j,i}, \pi_{j,i})\}_{i \in T_j}$.

$\Delta_{e_0 \to e} \leftarrow 1$
for $u \leftarrow e_0 + 1$ **to** e **do**
  Parse updmsg_u as (u, Δ_u)
  $\Delta_{e_0 \to e} \leftarrow \Delta_{e_0 \to e} \cdot \Delta_u$
end for
Initialize $\mathsf{resp}_j \leftarrow \emptyset$
for each $(i, \widetilde{W}_{j,i}) \in \widetilde{\mathsf{req}}_j$ **do**
  $W'_{j,i} \leftarrow \widetilde{W}_{j,i}^{\Delta_{e_0 \to e}} \bmod N$
  $r_{j,i} \leftarrow \mathsf{H}(i, e_0, e, \mathsf{acc}_{e_0}, \mathsf{acc}_e, \widetilde{W}_{j,i}, W'_{j,i})$
  $X_{j,i} \leftarrow \widetilde{W}_{j,i} \cdot \mathsf{acc}_{e_0}^{r_{j,i}} \bmod N$
  $Y_{j,i} \leftarrow W'_{j,i} \cdot \mathsf{acc}_e^{r_{j,i}} \bmod N$
  $\pi_{j,i} \leftarrow \mathsf{PoKE.Prove}(\mathsf{pp}, X_{j,i}, Y_{j,i}; \Delta_{e_0 \to e})$
  $\mathsf{resp}_j \leftarrow \mathsf{resp}_j \cup \{(i, W'_{j,i}, \pi_{j,i})\}$
end for
return resp_j

The client, however, can check the standard RSA-accumulator membership equation after reconstruction. This gives a last line of defense against any mismatch between accepted row updates, public epoch metadata, and the client's local witness state. If the check fails, the client rejects and can retry with a different qualified subset or a new sharing instance.

4.4 Optimizations for Catch-Up Exponent Assembly

Algorithm 2 requires assembling $\Delta_{e_0 \to e}$ over long windows. For short windows this cost is modest, but after many missed epochs it becomes a dominant server-side component. We use two server-side optimizations that preserve the exact exponent.

Bucket Pre-computation. The server fixes a bucket size k and partitions epochs into contiguous buckets. For each finalized bucket, it pre-computes and stores the product of the epoch update factors in that bucket, while still retaining per-epoch factors for boundary intervals. For a query window (e_0, e), the server decomposes $[e_0 + 1, e]$ into cached full buckets plus residual boundary epochs. If the window length is $L = e - e_0$, the number of online operands drops from L to $O(L/k + k)$.

Divide-and-Conquer Multiplication. The selected bucket products and residual epoch factors are multiplied using a balanced product tree. A sequential left-to-right product repeatedly multiplies a large intermediate value by a small

Algorithm 3 Client Reconstruction and Final Check

Input: Public parameters $\mathsf{pp} = (N, g)$; public ISP/LISS description $\mathsf{isp} = (M, \psi, \varepsilon)$; client representative x; current epoch digest acc_e; a qualified threshold set A of *verified* responders with $|A| \geq t$; verified response sets $\{\mathsf{resp}_j\}_{j \in A}$ after PoKE checking, where $\mathsf{resp}_j = \{(i, W'_{j,i})\}_{i \in T_j}$.

Output: Updated witness W_{new} or $\perp$.

$I_A \leftarrow \bigcup_{j \in A} \{\, i \in \{1, \ldots, d\} \,:\, \psi(i) = j \,\}$ ▷ row indices owned by servers in A
Compute reconstruction coefficients $\{\lambda_i\}_{i \in I_A} \leftarrow \mathsf{LISS.Coeff}(\mathsf{isp}, A)$
Parse $\{\mathsf{resp}_j\}_{j \in A}$ into a unified set $\mathcal{R}_A = \{(i, W'_i)\}_{i \in I_A}$
$W_{\text{new}} \leftarrow \prod_{i \in I_A} (W'_i)^{\lambda_i} \bmod N$
if $W_{\text{new}}^{x} \not\equiv \mathsf{acc}_e \pmod N$ **then**
    return $\perp$
end if
return W_{new}

operand, which is poorly matched to large-integer multiplication routines. The balanced tree keeps operand sizes more comparable across recursive levels and avoids this systematic long-times-short pattern.

5 Security Overview

This section states the main guarantees and gives proof sketches. Detailed definitions and full proofs are deferred to Appendix B. The adversary is probabilistic polynomial time and may non-adaptively corrupt fewer than t update servers before the fresh sharing randomness of an execution is sampled. Privacy is argued in the unknown-order generic group model [18]; accountability uses the random-oracle model for the Fiat–Shamir challenge in the folded PoKE statement. Public protocol metadata, including the epoch window, public digests, public ISP/LISS description, and contacted server set, is not hidden by the cryptographic protocol; its deployment implications are discussed in Section 7.

Theorem 1 (Correctness). *Let W_{old} be a valid witness for representative x at epoch e_0, and let $\Delta_{e_0 \to e}$ be the update factor induced by the verified epoch updates over $(e_0, e]$. If the client reconstructs from a qualified set of accepted response items, Algorithm 3 outputs*

$$W_{\mathsf{new}} = W_{\mathsf{old}}^{\Delta_{e_0 \to e}} \pmod N,$$

and W_{new} is a valid witness for x at epoch e.

Proof sketch. Every accepted row update is bound by the folded PoKE to the same exponent that maps acc_{e_0} to acc_e. For a qualified responder set A, LISS reconstruction gives $\sum_{i \in I_A} \lambda_i \sigma_i = 1$ because the shared secret is fixed to 1. Multiplying the accepted updated share units with coefficients λ_i therefore yields $W_{\mathsf{old}}^{\Delta_{e_0 \to e}}$. The final membership equation follows from $W_{\mathsf{old}}^{x} = \mathsf{acc}_{e_0}$ and $\mathsf{acc}_e = \mathsf{acc}_{e_0}^{\Delta_{e_0 \to e}}$. The complete proof is given in Appendix B.1.

Theorem 2 (Availability). *If at least t update servers are honest and responsive, then an honest client can complete witness update despite the remaining servers being malicious or offline.*

Proof sketch. The honest responsive servers provide a qualified set of correctly formed row updates. Malicious responses can be discarded after failed verification, and offline servers are not needed once a qualified accepted set is available. Correctness then gives successful reconstruction. The complete proof is given in Appendix B.2.

Theorem 3 (Threshold witness privacy). *For every non-adaptively corrupted server set $\mathcal{C}$ with $|\mathcal{C}| < t$, the protocol satisfies threshold witness privacy in the unknown-order generic group model, conditioned on fixed public metadata.*

Proof sketch. A corrupted coalition sees only generic encodings of $\{W^{\sigma_i}\}_{i \in I_C}$ for a forbidden set of LISS rows. The key linear fact is that a forbidden row set cannot form a nonzero constant exponent from its observed share units; any nontrivial hidden-base-dependent element remains masked by fresh LISS randomness. In the generic group model, the adversary observes only random encodings of such elements and cannot distinguish which valid stale witness was used, except with negligible collision and LISS statistical error. The detailed proof is given in Appendix B.3.

Theorem 4 (Unlinkability modulo public metadata). *Fix any non-adaptively corrupted server set $\mathcal{C}$ with $|\mathcal{C}| < t$, and fix two public metadata profiles $\mu^{(1)}$ and $\mu^{(2)}$. For $k \in \{1, 2\}$ and any valid stale witness W consistent with $\mu^{(k)}$, let $\mathsf{View}_{\mathcal{C}}^{(k)}(W)$ denote the corrupted-server view of execution k when the hidden witness is W. Suppose the two executions use fresh independent sharing randomness. Then for any two witness pairs $(W_0^{(1)}, W_0^{(2)})$ and $(W_1^{(1)}, W_1^{(2)})$ such that, for each k, both $W_0^{(k)}$ and $W_1^{(k)}$ are consistent with the same metadata profile $\mu^{(k)}$, the joint views induced by the two pairs are indistinguishable in the unknown-order generic group model.*

Proof sketch. Apply threshold witness privacy independently to the two executions. A hybrid argument first replaces the hidden witness in the first execution and then in the second. Fresh sharing randomness ensures that the two replacements do not interact. Thus witness-dependent group elements do not provide a reusable linkage handle. The detailed proof is given in Appendix B.4.

Theorem 5 (Accountability). *Assume that $(\mathsf{acc}_{e_0}, \mathsf{acc}_e)$ is an audited public digest transition, the underlying PoKE is a sound argument of knowledge, and H is modeled as a random oracle. Except with negligible probability, any response item accepted by the client is consistent with a common exponent for the returned share update and the public digest transition. Rejected response items are attributable to the designated responding server.*

Proof sketch. For each row, the client recomputes the Fiat–Shamir challenge over the row index, epoch window, public digests, input share, and output share, then verifies a folded PoKE tying the share-update relation to the digest-transition relation. A malformed item can pass only by breaking the knowledge soundness of PoKE or by making the random-oracle challenge satisfy a nontrivial algebraic relation for the malformed tuple. The row index and authenticated response package identify the responsible server. The detailed proof is given in Appendix B.5.

6 Implementation and Evaluation

6.1 Implementation

We implement the outsourced witness-update pipeline in Rust using Arkworks [16]. The prototype reuses the duplex codebase for the membership-proof component [25], follows its MiMC-7/LegoGroth16-over-BN254 choices [2,24], and uses a 2048-bit RSA group with approximately 256-bit prime representatives. We implement the LISS/ISP layer, blinded request generation, threshold reconstruction, and per-share PoKE generation and verification. Big-integer operations use `rug`, and intra-server parallelism uses `rayon`. Benchmarks run on an Alibaba Cloud ECS `c9i.8xlarge` instance with 32 vCPUs, 64 GiB memory, and Ubuntu Server 24.04 LTS.

6.2 Evaluation

Evaluation Setup and Methodology We simulate intermittently online clients that miss multiple epochs and then update a stale witness in one shot. One epoch corresponds to insertion of 20 new elements into the global accumulator. In the main runtime experiments, the catch-up span ranges from 100 to 5,000 epochs, corresponding to 2,000 to 100,000 appended elements. To study the impact of threshold settings, we evaluate four configurations: $(t = 2, n = 3)$, $(t = 3, n = 5)$, $(t = 4, n = 7)$, and $(t = 5, n = 9)$, corresponding to $n = 2f + 1$ and $t = f + 1$.

Each reported data point is averaged over 10 independent runs. For each run, the client uniformly samples a fresh valid t-subset of update servers and issues a complete witness-update request to that subset. Reported online latencies exclude offline pre-computation. Unless otherwise stated, bucket products are precomputed offline and reused during online updates with bucket size 20. Online server measurements include catch-up exponent assembly, share update, and PoKE generation. Logical servers are executed serially on one machine to isolate per-server cryptographic computation; we do not measure WAN latency, bandwidth constraints, or partially synchronous network effects.

Share-Load Characterization The current threshold-access ISP/LISS instantiation induces non-uniform share loads. For a valid t-subset A, let its total share

Table 2. Share-load statistics

(t, n)	Avg.	Min	Max	Max/Min
$(2, 3)$	1.67	3	4	1.33
$(3, 5)$	3.80	7	16	2.29
$(4, 7)$	9.86	15	62	4.13
$(5, 9)$	27.89	31	236	7.61
$(6, 11)$	83.91	63	892	14.16
$(7, 13)$	263.92	127	3368	26.52

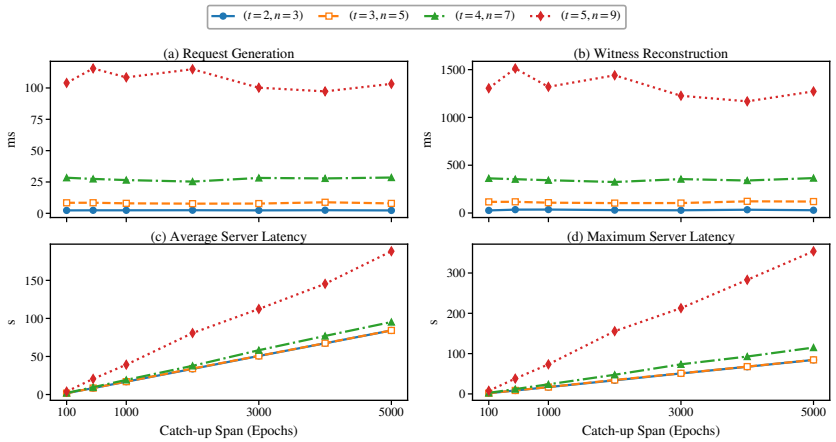

Fig. 2. Client-side and server-side performance for outsourced witness update.

load be $\sum_{j \in A} |T_j|$. Table 2 reports the average number of share units assigned to a single server, the minimum and maximum total loads over all valid t-subsets, and their ratio.

The average per-server workload and subset-level skew grow rapidly with the threshold. In particular, Avg. increases from 1.67 at $(2, 3)$ to 27.89 at $(5, 9)$, while the Max/Min ratio rises from 1.33 to 7.61. This skew explains why the maximum server latency becomes a more informative end-to-end metric at larger thresholds.

Client-Side Performance A central objective of the protocol is to relieve intermittently online clients from continuously monitoring update information. Figure 2(a) and Figure 2(b) show that request generation and witness reconstruction are essentially independent of the catch-up span. This matches the protocol design: the client does not process missed epoch updates and only performs work determined by the threshold configuration and the selected share load. Even at $(t = 5, n = 9)$, the client-side latency remains about 1.5 s.

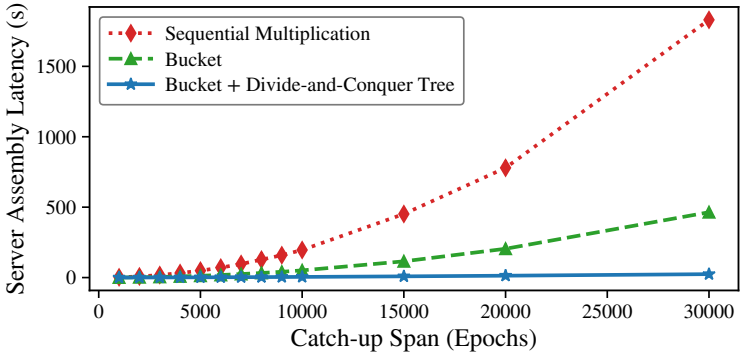

Fig. 3. Ablation study on update-factor assembly latency.

Under a fixed threshold setting, small fluctuations remain because each request is issued to a randomly selected valid t-subset. Different valid subsets induce different aggregate share loads, which affects client-side exponentiations and PoKE verifications. This effect is more visible during reconstruction, since that phase verifies server responses before combining the accepted updated share units.

Server-Side Scalability and Bottleneck Analysis Figure 2(c) and Figure 2(d) report average and maximum server latencies. Server latency grows with the catch-up span because each server assembles and applies the catch-up exponent. The maximum server latency is the relevant end-to-end bottleneck, since reconstruction waits for all servers in the chosen qualified set. In the largest tested setting, $(t = 5, n = 9)$ with 5,000 missed epochs, the maximum server latency is about 355.1 s, supporting one-shot updates after long offline windows but also exposing the cost of high-threshold LISS share loads.

At lower thresholds, especially $(2, 3)$ and $(3, 5)$, the average and maximum server curves remain close because per-server share loads are small and can be handled effectively with intra-server parallelism. At higher thresholds, the ISP/LISS instantiation creates more shares and a more uneven assignment. The slowest selected server therefore dominates the completion time and widens the gap between the average and maximum latency curves.

Ablation Study: Update-Factor Assembly Figure 3 isolates the cost of assembling $\Delta_{e_0 \to e}$ over 1,000 to 30,000 epochs. It compares *Sequential Multiplication*, *Bucket*, and *Bucket + Divide-and-Conquer Tree*. At 30,000 epochs, the measured assembly latencies are about 1830.0 s, 464.0 s, and 24.3 s, respectively. Thus bucket pre-computation and balanced product-tree multiplication give about 3.9× and 75× speedups over sequential multiplication. This confirms that the optimizations in Section 4.4 are essential for long catch-up windows.

Extension and Communication Analysis Relative to client-side maintenance, the client no longer downloads or processes missed epoch updates; its online work depends on the selected LISS share load rather than the catch-up span. Relative to single-server delegation, the threshold design incurs additional share-update and PoKE costs to avoid exposing the witness to any single update service.

Let A be a qualified responding subset and let $L_A = \sum_{j \in A} |T_j|$ denote its aggregate share load. The server-side work is proportional to L_A, since each share unit requires one update and one proof-bearing response. For server j, the request sent by the client contains $|T_j|$ blinded RSA-group elements, while the response returned by the server contains $|T_j|$ updated group elements and $|T_j|$ PoKE proofs. With our 2048-bit RSA modulus, one group element occupies 256 bytes and one PoKE occupies approximately 800 bytes. Excluding compact row indices and fixed epoch metadata, the client-to-server request and server-to-client response for server j are therefore approximately $256|T_j|$ and $1056|T_j|$ bytes, respectively. Replacing $|T_j|$ with L_A gives the aggregate payloads for a qualified subset. Hence, the Min and Max share loads in Table 2 characterize communication growth across threshold settings. Communication is independent of the catch-up span and is determined by the LISS share load.

Our single-machine measurements isolate per-server cryptographic computation and exclude WAN latency, bandwidth variation, and distributed stragglers. In a distributed deployment, each contacted server additionally incurs one request–response exchange and the corresponding payload transmission time. These network effects may be more noticeable for shorter windows and lower threshold settings; for the longer catch-up windows evaluated here, server computation is expected to remain the dominant component. A geographically distributed evaluation is left for future work.

7 Discussion

Protocol choices. The algorithms deliberately use LISS to share the fixed exponent 1, rather than sharing the witness or the representative. This choice is what makes the algebra line up with append-only RSA updates: servers update randomized powers $W_{\text{old}}^{\sigma_i}$ using the same catch-up exponent, and the client combines the resulting powers with LISS reconstruction coefficients to recover $W_{\text{old}}^{\Delta_{e_0 \to e}}$. The fixed secret 1 therefore keeps all witness-dependent values within freshly randomized powers while preserving the exact reconstruction equation used in Algorithm 3.

Proof binding. The folded PoKE in Algorithm 2 is also part of the design, not only an implementation convenience. A proof for $\widetilde{W}_{j,i}^{\Delta} = W'_{j,i}$ alone would show that a share unit was updated by some exponent, but it would not by itself bind that exponent to the public transition from acc_{e_0} to acc_e. By folding the share-update relation with the digest-transition relation under a Fiat–Shamir challenge that includes the row index and epoch window, the response becomes tied to the

requested public state. This is important because the client reconstructs from multiple independently returned rows: row-level binding prevents a malicious server from mixing stale, fresh, or cross-row responses while still passing a local proof check.

Metadata limitations. The protocol hides witness material and prevents update servers from observing a stable identifier in the randomized request elements, but it does not hide the public epoch window $(e_{\text{in}}, e_{\text{out}})$. This metadata enables a natural forward-correlation attempt. After observing a request (e_0, e_1), a server may regard a later request (e_1, e_2) as a possible continuation, since a client that refreshed its witness to epoch e_1 may later become stale again and issue precisely such a request.

Our target deployment is a high-frequency append-only registry serving intermittently online clients, where continuously synchronizing epoch updates is precisely the burden that motivates outsourced catch-up. In this setting, the metadata-compatibility relation is generally many-to-many: multiple clients may share the same stale or target epoch, and a request (e_1, e_2) may therefore be compatible with several earlier requests ending at e_1. Repeated observations consequently induce a branching compatibility structure, in which each newly observed request may extend multiple metadata-compatible predecessor chains rather than a unique path associated with one hidden representative.

Under a heuristic branching model, if each observation leaves, on average, $b > 1$ plausible predecessor branches, then after k observations the number of metadata-compatible chains grows on the order of b^k. This estimate is illustrative and assumes that no auxiliary information beyond the epoch windows is available. When request activity is sparse or additional side information is available, the effective branching factor may be smaller and the anonymity set may shrink accordingly. Thus, public epoch metadata may reduce the effective anonymity set, but in the intended high-frequency setting it generally leaves multiple plausible linkage chains rather than isolating a unique chain associated with a particular hidden representative. Meanwhile, the freshly randomized LISS request elements do not expose a stable value that can be reused to link different executions.

8 Related Work

8.1 Accumulators

Cryptographic accumulators provide succinct commitments to dynamic sets with compact membership witnesses. RSA accumulators trace back to one-way accumulators [4] and dynamic revocation-oriented constructions [12], and have been extended to universal accumulators, batching, scalable verification, and modular zero-knowledge membership proofs [26,8,29,13,6]. Bilinear accumulators give a pairing-based alternative [28,34], while Merkle-tree approaches rely on collision-resistant hashing and logarithmic proofs [27,17].

8.2 Accumulator Witness Update Frequency

Bonneau *et al.* show that for classical accumulator families, including Merkle, RSA, and bilinear accumulators, a single insertion typically forces updates to essentially all existing witnesses, and they analyze Merkle Mountain Ranges as an append-only alternative that reduces witness-update events through structured subtree merges [9]. Recent lower bounds further study inherent limits of short accumulators and vector commitments: for additive accumulators and updatable vector commitments with constant-size digests, one addition can invalidate nearly all existing proofs on average [1]. Related lower bounds for revocable or stateless proof systems clarify the limits of witness maintenance [30,15]. From a systems perspective, zk-creds likewise highlights the synchronization burden of dynamic set membership under frequent state evolution [31].

8.3 Threshold and Secret Sharing

Threshold cryptography and secret sharing distribute trust and support robust multi-party computation. Shamir and Blakley introduced threshold secret sharing [32,7], and generalized secret sharing captures authorization policies beyond simple t-out-of-n thresholds [5]. Damgård and Thorbek's LISS enables integer secret sharing and distributed exponentiation under arbitrary access structures [19]; our protocol uses this integer-linear structure to make RSA witness update compatible with threshold-private outsourced computation.

9 Conclusion

We presented a privacy-preserving outsourced witness-update protocol for append-only RSA accumulators. The construction combines LISS-based witness blinding with PoKE-based accountability, so intermittently online clients can refresh stale witnesses on demand without processing missed updates locally. The implementation confirms the intended asymmetry: client work is independent of the offline window, while the remaining cost lies in server catch-up exponent assembly and LISS share load.

References

1. Abusalah, H., Anthoine, G., Avitabile, G., Giunta, E.: Lower bounding update frequency in short accumulators and vector commitments. Cryptology ePrint Archive, Paper 2025/1558 (2025), https://eprint.iacr.org/2025/1558
2. Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 191–219. Springer (2016)
3. Baldimtsi, F., Lysyanskaya, A.: Anonymous credentials light. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 1087–1098 (2013)

4. Benaloh, J., De Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Workshop on the Theory and Application of Cryptographic Techniques. pp. 274–285. Springer (1993)
5. Benaloh, J., Leichter, J.: Generalized secret sharing and monotone functions. In: Conference on the Theory and Application of Cryptography. pp. 27–35. Springer (1988)
6. Benarroch, D., Campanelli, M., Fiore, D., Gurkan, K., Kolonelos, D.: Zero-knowledge proofs for set membership: Efficient, succinct, modular. Designs, Codes and Cryptography **91**(11), 3457–3525 (2023)
7. Blakley, G.R.: Safeguarding cryptographic keys. In: 1979 International workshop on managing requirements knowledge. pp. 313–318. IEEE Computer Society (1979)
8. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: Annual International Cryptology Conference. pp. 561–586. Springer (2019)
9. Bonneau, J., Chen, J., Christ, M., Karantaidou, I.: Merkle mountain ranges are optimal: On witness update frequency for cryptographic accumulators. In: Annual International Cryptology Conference. pp. 170–202. Springer (2025)
10. Burkhardt, J., Damgård, I., Frederiksen, T.K., Ghosh, S., Orlandi, C.: Improved distributed rsa key generation using the miller-rabin test. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 2501–2515 (2023)
11. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: International conference on the theory and applications of cryptographic techniques. pp. 93–118. Springer (2001)
12. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Annual International Cryptology Conference. pp. 61–76. Springer (2002)
13. Campanelli, M., Fiore, D., Han, S., Kim, J., Kolonelos, D., Oh, H.: Succinct zero-knowledge batch proofs for set accumulators. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 455–469 (2022)
14. Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2075–2092 (2019)
15. Christ, M., Bonneau, J.: Limits on revocable proof systems, with implications for stateless blockchains. In: International Conference on Financial Cryptography and Data Security. pp. 54–71. Springer (2023)
16. arkworks contributors: `arkworks` zksnark ecosystem (2022), https://arkworks.rs
17. Dahlberg, R., Pulls, T., Peeters, R.: Efficient sparse merkle trees: Caching strategies and secure (non-) membership proofs. In: Nordic Conference on Secure IT Systems. pp. 199–215. Springer (2016)
18. Damgård, I., Koprowski, M.: Generic lower bounds for root extraction and signature schemes in general groups. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 256–271. Springer (2002)
19. Damgård, I., Thorbek, R.: Linear integer secret sharing and distributed exponentiation. In: International Workshop on Public Key Cryptography. pp. 75–90. Springer (2006)
20. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: 13th USENIX Security Symposium (USENIX Security 04). pp. 303–320 (2004)

21. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–208 (1983)
22. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the theory and application of cryptographic techniques. pp. 186–194. Springer (1986)
23. Garman, C., Green, M., Miers, I.: Decentralized anonymous credentials. Cryptology ePrint Archive, Paper 2013/622 (2013), available at https://eprint.iacr.org/2013/622
24. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 305–326. Springer (2016)
25. Han, S., Yoon, G., Oh, H., Kim, J.: DUPLEX: Scalable zero-knowledge lookup arguments over RSA group. In: Proceedings of the 20th ACM Asia Conference on Computer and Communications Security. pp. 72–86 (2025)
26. Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: International Conference on Applied Cryptography and Network Security. pp. 253–269. Springer (2007)
27. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Conference on the Theory and Application of Cryptographic Techniques. pp. 369–378 (1987)
28. Nguyen, L.: Accumulators from bilinear pairings and applications. In: Cryptographers' Track at the RSA Conference. pp. 275–292. Springer (2005)
29. Ozdemir, A., Wahby, R., Whitehat, B., Boneh, D.: Scaling verifiable computation using efficient set accumulators. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 2075–2092 (2020)
30. Qi, W.: Tight lower bound on witness update frequency in additive positive accumulators. Cryptology ePrint Archive (2025)
31. Rosenberg, M., White, J., Garman, C., Miers, I.: zk-creds: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 790–808. IEEE (2023)
32. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
33. Sonnino, A., et al.: Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers (2019)
34. Srinivasan, S., Karantaidou, I., Baldimtsi, F., Papamanthou, C.: Batching, aggregation, and zero-knowledge proofs in bilinear accumulators. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2719–2733 (2022)

A Epoch Update

In epoch e, the Manager commits to the epoch batch U_e, derives $\Delta_e := \prod_{u \in U_e} u$, and updates the digest from acc_{e-1} to acc_e. It outputs $\pi_{\mathsf{update},e} := (\pi^{\mathsf{acc}}_{\mathsf{update},e}, \pi^{\Delta}_{\mathsf{update},e})$, where the two CP-SNARK proofs certify insertion correctness and product correctness for the same committed batch. Thus, the public digest transition and the server-side update factor are bound to the same epoch update. Algorithm 4 gives the procedure.

Algorithm 4 Epoch Update

Input: Commitment key ck, proof parameters $\mathsf{crs}_{\mathsf{Ins}}$, $\mathsf{crs}_{\mathsf{Prod}}$, public parameters pp, epoch index e, prior digest acc_{e-1}, and epoch batch U_e.

Output: Tuple $(\mathsf{acc}_e, c_{U_e}, \pi_{\mathsf{update},e}, \mathsf{updmsg}_e)$.

1: Sample opening randomness o_e and commit $c_{U_e} \leftarrow \mathsf{Comm}(\mathsf{ck}, U_e; o_e)$
2: Compute $\Delta_e \leftarrow \prod_{u \in U_e} u$
3: Set $\mathsf{updmsg}_e \leftarrow (e, \Delta_e)$
4: Update digest $\mathsf{acc}_e \leftarrow \mathsf{Ins}(\mathsf{pp}, \mathsf{acc}_{e-1}, U_e)$
5: $\pi^{\mathsf{acc}}_{\mathsf{update},e} \leftarrow \mathsf{cp.ProveIns}(\mathsf{crs}_{\mathsf{Ins}}, e, \mathsf{acc}_{e-1}, \mathsf{acc}_e, c_{U_e}; U_e, o_e)$
6: $\pi^{\Delta}_{\mathsf{update},e} \leftarrow \mathsf{cp.ProveProd}(\mathsf{crs}_{\mathsf{Prod}}, e, c_{U_e}, \Delta_e; U_e, o_e)$
7: Set $\pi_{\mathsf{update},e} \leftarrow (\pi^{\mathsf{acc}}_{\mathsf{update},e}, \pi^{\Delta}_{\mathsf{update},e})$
8: **return** $(\mathsf{acc}_e, c_{U_e}, \pi_{\mathsf{update},e}, \mathsf{updmsg}_e)$

B Detailed Security Proofs

For the LISS privacy arguments below, we use the standard statistical LISS instantiation [19]: the non-secret coordinates of the distribution vector are sampled from intervals whose widths are chosen as a function of the security parameter λ. All statistical errors below are negligible in λ.

Definition 1 (Unknown-order generic group model). *Let $GGen$ be a generator for groups of unknown order. In the unknown-order generic group model, the challenger samples a group $G \xleftarrow{\$} GGen(\lambda)$ and represents each element of G by a uniformly random encoding under an injective map unknown to the adversary. A generic adversary does not learn $|G|$ and may access group elements only through the following oracles:*

- *$\mathcal{O}_{\mathsf{rand}}$, which returns the encoding of a uniformly random element of G;*
- *$\mathcal{O}_{\mathsf{op}}$, which on input encodings of two previously obtained elements returns the encoding of their product or quotient.*

Definition 2 (Threshold witness privacy). *Fix a contacted-server set $\mathcal{B} \subseteq [n]$ and any non-adaptively corrupted server set $\mathcal{C} \subseteq \mathcal{B}$ with $|\mathcal{C}| < t$. Let μ denote the fixed public protocol metadata of the execution, including the epoch window (e_0, e), public digests $(\mathsf{acc}_{e_0}, \mathsf{acc}_e)$, the public ISP/LISS description isp, and the contacted-server set $\mathcal{B}$. Let*

$$I_{\mathcal{C}} := \bigcup_{j \in \mathcal{C}} T_j.$$

The adversary chooses two valid stale witnesses W_0, W_1 that are both consistent with μ, meaning that there exist accumulated prime representatives x_0, x_1 such that

$$W_0^{x_0} \equiv \mathsf{acc}_{e_0} \pmod N \qquad \text{and} \qquad W_1^{x_1} \equiv \mathsf{acc}_{e_0} \pmod N.$$

The challenger samples $b \leftarrow \{0,1\}$, samples a fresh LISS distribution vector $\boldsymbol{\rho} = (1, \rho_2, \ldots, \rho_e)^\top$, sets $\boldsymbol{\sigma} = M\boldsymbol{\rho}$, and returns

$$(\mu, \{\xi(W_b^{\sigma_i})\}_{i \in I_{\mathcal{C}}}),$$

where ξ is the random encoding map in the unknown-order generic group model. The adversary outputs a guess b'. Its advantage is

$$\mathsf{Adv}^{\mathsf{priv}}_{\mathcal{A},\mathcal{C}}(\lambda) := \left|\Pr[b' = b] - \frac{1}{2}\right|.$$

The protocol satisfies threshold witness privacy if this advantage is negligible for every PPT adversary $\mathcal{A}$, every contacted-server set $\mathcal{B} \subseteq [n]$, and every $\mathcal{C} \subseteq \mathcal{B}$ with $|\mathcal{C}| < t$.

This experiment gives the corrupted servers their incoming witness-dependent request elements. Their honest response elements, folded-PoKE bases, and any additional values they can compute from the request are obtained from these encodings using known integers, public digests, and generic group operations. Thus allowing the adversary arbitrary generic-group operations on the displayed encodings captures the full witness-dependent part of the corrupted-server view; server-local randomness and adversarially chosen outgoing messages are independent of the hidden witness except through that view.

B.1 Correctness

Proof (Proof of Theorem 1). Let $I_A := \bigcup_{j \in A} T_j$. For every accepted row index $i \in I_A$, folded PoKE verification binds the returned share update to the same exponent that maps acc_{e_0} to acc_e. Except with negligible error from PoKE knowledge soundness and the random-oracle binding of the folded statement,

$$W_i' = (W_{\mathsf{old}}^{\sigma_i})^{\Delta_{e_0 \to e}} \pmod N.$$

Since A is qualified, the LISS reconstruction coefficients satisfy

$$\sum_{i \in I_A} \lambda_i M[i] = \boldsymbol{\varepsilon}^\top.$$

Because the client shares the fixed secret 1, this gives

$$\sum_{i \in I_A} \lambda_i \sigma_i = 1.$$

Interpreting negative reconstruction coefficients via inverses in $\mathbb{Z}_N^*$, the reconstruction step gives

$$\prod_{i \in I_A} (W_i')^{\lambda_i} = W_{\mathsf{old}}^{\Delta_{e_0 \to e} \cdot \sum_{i \in I_A} \lambda_i \sigma_i} = W_{\mathsf{old}}^{\Delta_{e_0 \to e}} \pmod N.$$

Finally, $W_{\mathsf{old}}^x \equiv \mathsf{acc}_{e_0} \pmod N$ and $\mathsf{acc}_e \equiv \mathsf{acc}_{e_0}^{\Delta_{e_0 \to e}} \pmod N$, so

$$W_{\mathsf{new}}^x \equiv (W_{\mathsf{old}}^x)^{\Delta_{e_0 \to e}} \equiv \mathsf{acc}_e \pmod N.$$

Thus the final membership check accepts a valid witness for epoch e.

B.2 Availability

Proof (Proof of Theorem 2). The honest responsive servers provide a qualified set of correctly formed response items. By Theorem 1, any such qualified accepted set reconstructs the updated witness. Malicious responses can be ignored after failed verification, and offline servers are not needed once a qualified accepted set is available.

B.3 Threshold Privacy

Lemma 1 (No nonzero constant from forbidden rows). *Let $\boldsymbol{\sigma} = M\boldsymbol{\rho}$ with $\boldsymbol{\rho} = (1, \rho_2, \ldots, \rho_e)^\top$, and let $\mathcal{C} \subseteq [n]$ satisfy $|\mathcal{C}| < t$. For any coefficients $\{c_i\}_{i \in I_\mathcal{C}}$, if the linear combination*

$$L(\boldsymbol{\sigma}) = \sum_{i \in I_\mathcal{C}} c_i \sigma_i$$

is independent of the random coordinates $\rho_2, \ldots, \rho_e$, then $L(\boldsymbol{\sigma}) = 0$.

Proof. Let $m_i = M[i]$ denote row i of M, so that $\sigma_i = \langle m_i, \boldsymbol{\rho} \rangle$. Set

$$v := \sum_{i \in I_\mathcal{C}} c_i m_i.$$

If $L(\boldsymbol{\sigma}) = \langle v, \boldsymbol{\rho} \rangle$ is independent of $\rho_2, \ldots, \rho_e$, then v has zero entries in coordinates $2, \ldots, e$, hence $v = \beta \boldsymbol{\varepsilon}^\top$ for some integer β.

Since $\mathcal{C}$ is forbidden, the ISP characterization gives a sweeping vector $\boldsymbol{\kappa}$ such that

$$M_{I_\mathcal{C}} \boldsymbol{\kappa} = 0 \qquad \text{and} \qquad \kappa_1 = 1.$$

Therefore,

$$0 = \left\langle \sum_{i \in I_\mathcal{C}} c_i m_i, \boldsymbol{\kappa} \right\rangle = \langle \beta \boldsymbol{\varepsilon}^\top, \boldsymbol{\kappa} \rangle = \beta \kappa_1 = \beta.$$

Thus the only constant exponent obtainable from the corrupted rows is 0.

Proof (Proof of Theorem 3). In the privacy game, the adversary receives the public metadata and the generic encodings of $\{W_b^{\sigma_i}\}_{i \in I_\mathcal{C}}$. In an actual execution, corrupted servers may also compute and observe updated row units $(W_b^{\sigma_i})^{\Delta_{e_0 \to e}}$ and proof inputs derived from them. These values add only known, challenge-independent scalar multiples of the same row exponents, because $\Delta_{e_0 \to e}$ and the public digest transition are fixed by the metadata. Thus it suffices to analyze the generic closure of the received hidden-base encodings under integer linear combinations of the corrupted rows.

A generic adversary can only derive new encodings by applying group operations to encodings it already has. Hence every hidden-base-dependent element it derives has the form

$$W_b^{L(\boldsymbol{\sigma})}$$

for an integer linear form L over rows in $I_{\mathcal{C}}$, possibly multiplied by public constants or public digest terms that are identical in both challenge branches. Products, quotients, and exponentiation by public integers merely change the coefficients in L. Multiplication by a fixed public digest term corresponds, in challenge branch b, to adding a fixed offset cx_b to the randomized exponent expression. For every nonconstant $L(\boldsymbol{\sigma})$, the fresh LISS randomness statistically masks this offset, so the shifted exponent distributions remain indistinguishable up to negligible statistical error.

Consider any such exponent expression $L(\boldsymbol{\sigma})$. By Lemma 1, the only case in which $L(\boldsymbol{\sigma})$ is independent of the fresh random coordinates is $L(\boldsymbol{\sigma}) = 0$, in which case the hidden-base-dependent part is the identity. Otherwise, the exponent retains at least one fresh LISS random coordinate. The LISS sampling range includes statistical slack, so these nonconstant linear forms reveal only statistically negligible information about the fixed first coordinate.

The generic group adversary does not observe exponents directly; it observes only random encodings of the resulting group elements. For $L(\boldsymbol{\sigma}) = 0$, the hidden-base contribution is the identity in both branches. For every nonzero hidden-base-dependent expression, the unresolved LISS randomness masks the exponent, and the distributions of the resulting generic encodings and their equality pattern are indistinguishable between the two challenge branches, except with negligible probability from generic-encoding collisions and LISS statistical error. All non-witness-dependent values, including public digests, row indices, epoch metadata, update factors, and Fiat–Shamir inputs that contain only public values or already analyzed encodings, are identical or generically simulatable in both branches. Therefore the corrupted coalition's full generic transcript is indistinguishable for $b = 0$ and $b = 1$, and $\mathsf{Adv}^{\mathsf{priv}}_{\mathcal{A},\mathcal{C}}(\lambda)$ is negligible.

B.4 Unlinkability

Proof (Proof of Theorem 4). Let $\mathsf{View}^{(k)}_{\mathcal{C}}(W)$ denote the corrupted-server view of execution k when the hidden witness is W under the fixed metadata profile of that execution. Define hybrids

$$H_0 := \left(\mathsf{View}^{(1)}_{\mathcal{C}}(W_0^{(1)}), \mathsf{View}^{(2)}_{\mathcal{C}}(W_0^{(2)})\right),$$

$$H_1 := \left(\mathsf{View}^{(1)}_{\mathcal{C}}(W_1^{(1)}), \mathsf{View}^{(2)}_{\mathcal{C}}(W_0^{(2)})\right),$$

and

$$H_2 := \left(\mathsf{View}^{(1)}_{\mathcal{C}}(W_1^{(1)}), \mathsf{View}^{(2)}_{\mathcal{C}}(W_1^{(2)})\right).$$

By Theorem 3, replacing the hidden witness in the first execution under its fixed metadata profile is indistinguishable, so H_0 and H_1 are indistinguishable. Since the second execution uses independent sharing randomness, this replacement does not affect the second execution's distribution. Applying Theorem 3 again to the second execution shows that H_1 and H_2 are indistinguishable. The claim follows by transitivity.

The claim is intentionally conditioned on the public metadata profile of each execution. It rules out linkage through witness-dependent group elements in the corrupted-server transcripts; it does not assert network-layer or timing privacy.

B.5 Accountability

Proof (Proof of Theorem 5). Fix a returned item $(i, W'_{j,i}, \pi_{j,i}) \in \mathsf{resp}_j$. The client recomputes

$$r_{j,i} := \mathsf{H}(i, e_0, e, \mathsf{acc}_{e_0}, \mathsf{acc}_e, \widetilde{W}_{j,i}, W'_{j,i})$$

and forms

$$X_{j,i} := \widetilde{W}_{j,i} \cdot \mathsf{acc}_{e_0}^{r_{j,i}} \pmod N, \qquad Y_{j,i} := W'_{j,i} \cdot \mathsf{acc}_e^{r_{j,i}} \pmod N.$$

It accepts only if $\mathsf{PoKE.Verify}(\mathsf{pp}, X_{j,i}, Y_{j,i}, \pi_{j,i}) = 1$. By the knowledge soundness of PoKE, any PPT server that causes verification to accept with non-negligible probability admits an extractor that outputs an exponent α such that

$$Y_{j,i} \equiv X_{j,i}^{\alpha} \pmod N.$$

For an honest response, $\alpha = \Delta_{e_0 \to e}$ satisfies both the share-update relation and the public digest-transition relation, so verification accepts.

Suppose instead that the returned tuple is malformed, in the sense that it is inconsistent with any common exponent for both $\widetilde{W}_{j,i} \mapsto W'_{j,i}$ and $\mathsf{acc}_{e_0} \mapsto \mathsf{acc}_e$. Expanding the accepted folded equation gives

$$W'_{j,i}\mathsf{acc}_e^{r_{j,i}} \equiv \widetilde{W}_{j,i}^{\alpha}\mathsf{acc}_{e_0}^{\alpha r_{j,i}} \pmod N.$$

Equivalently,

$$A_\alpha \equiv B_\alpha^{r_{j,i}} \pmod N, \qquad A_\alpha := W'_{j,i} \cdot (\widetilde{W}_{j,i}^{\alpha})^{-1}, \quad B_\alpha := \mathsf{acc}_{e_0}^{\alpha} \cdot \mathsf{acc}_e^{-1}.$$

If the item is consistent with a common exponent, then $A_\alpha = 1$ and $B_\alpha = 1$. If it is malformed, then for every candidate exponent extracted from an accepting PoKE, $(A_\alpha, B_\alpha) \neq (1, 1)$, and acceptance requires the Fiat–Shamir challenge to hit a challenge value satisfying the displayed algebraic relation. Because $r_{j,i}$ is modeled as a random-oracle output over the full row-indexed statement

$$(i, e_0, e, \mathsf{acc}_{e_0}, \mathsf{acc}_e, \widetilde{W}_{j,i}, W'_{j,i}),$$

the server cannot adapt the tuple after learning the challenge. The probability that a fixed malformed tuple obtains a challenge satisfying the required nontrivial relation is negligible over the challenge space, except by breaking the knowledge soundness of PoKE. Taking a union bound over the adversary's polynomially many random-oracle queries preserves negligibility.

Attribution follows from the authenticated response package and the public row-assignment map ψ. The item is received as part of resp_j, and row i belongs to the server-specific row set $T_j = \psi^{-1}(j)$. Thus a rejected item is attributable to server S_j.