

Lightweight Verifiable Computation with Constant-Size Proofs for Sequential State Transitions

Heewon Chung

Jeonbuk National University

Hyeonbum Lee

hyeonbumlee@snu.ac.kr

Seoul National University

Research Article

Keywords: incrementally verifiable computation, verifiable computation, state-transition system, data integrity

Posted Date: October 22nd, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-10995429/v1>

License:  This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Lightweight Verifiable Computation with Constant-Size Proofs for Sequential State Transitions

Heewon Chung · Hyeonbum Lee

Received: date / Accepted: date

Abstract Many applications, including IoT monitoring, verifiable stream analytics, and stateful audit services, maintain a state that evolves through a sequence of updates executed in an untrusted environment. Because the executing party may differ from the party that accepts the resulting state, an untrusted prover may apply an invalid transition, reuse an old state, or report an unreachable result, so checking the reported value alone does not suffice. We present an updatable verifiable computation scheme for state-transition systems: starting from an authentic running commitment, it certifies that the reported state is reachable within a fixed transition bound through some witness input sequence. The key idea is a linearly updatable quadratic arithmetic program family: each update reuses the previous proof and adds only the polynomial terms introduced by newly visible wires, so the prover avoids recursive proof composition and never re-proves the accumulated computation from scratch. On BN254, our scheme produces 160-byte proofs, and verification costs only 3 pairings plus a state-commitment increment check that scales with the state width and is independent of the number of steps. We evaluate sensor fusion, Hadamard products, MiMC-style cube chains, and repeated-squaring chains on a Raspberry Pi 5 edge gateway. For sensor fusion with $B = 1024$, our per-step proving at the final step is $2.6\text{--}2.7\times$ faster than Groth16 re-proving from a

single common reference string generated once for the maximum bound, and Groth16’s cumulative cost is $1.9\text{--}2.0\times$ ours. Against Nova’s compressed proof, ours is about $72\text{--}78\times$ smaller and verifies in $11.6\text{--}28.9\text{ ms}$ rather than $238\text{--}251\text{ ms}$.

Keywords incrementally verifiable computation · verifiable computation · state-transition system · data integrity

1 Introduction

Many applications maintain a state that evolves through a sequence of updates, where each update incorporates a fresh input into the previous state. For example, in an Internet of Things (IoT) monitoring [1, 2] or sensor-based manufacturing monitoring [3, 4], multiple measurements such as vibration, temperature, current, and acoustic signals can be fused into a running estimate of equipment condition, which is updated as new measurement batches arrive. In verifiable stream analytics, a service can update a vector state through element-wise operations such as the Hadamard product. In audit or ledger services, a compact state can be advanced by a MiMC-style cube permutation, while the arithmetic underlying verifiable delay functions (VDFs) repeatedly applies squaring. These examples differ in the application layer, but they share the same form: a new state is computed from the previous state and a transition input, and we call it a *state-transition system*.

A state-transition system can have security issues when such computations are outsourced or executed by an untrusted server because the server may use invalid transition inputs, reuse an old state, or replace the prescribed update computation with a cheaper one. One

H. Chung
Department of Software Engineering, Jeonbuk National University, Jeonju 54896, Republic of Korea
E-mail: heewonchung@jbnu.ac.kr

H. Lee (corresponding author)
BK21 FOUR Intelligence Computing, Seoul National University, Seoul 08826, Republic of Korea
E-mail: hyeonbumlee@snu.ac.kr

might attach signatures or commitments to the outputs, but this is still insufficient. Such attestations authenticate who produced or recorded a value, yet they do not prove that the value was obtained by applying the prescribed transition to the previous accepted state. They also do not rule out an invalid transition history unless the transition computation is checked. To address this issue, a verifier must be able to check that the reported state is reachable from the genesis state through some sequence that satisfies the fixed circuit. A naïve method is replaying all transitions from scratch, but it becomes increasingly expensive as the transition history grows. The setting also imposes compactness and latency requirements: the proof object should remain small enough for resource-constrained environments, and verification should be fast enough for online integrity checks.

Verifiable computation (VC) [5] is a strong candidate to support this functionality. It enables outsourcing computations to an untrusted party while retaining the ability to efficiently verify correctness of the result. Crucially, verifying the proof must be substantially cheaper than evaluating the function directly, and proof generation must be efficient. Among the resulting SNARK constructions [6, 7, 8, 9, 10, 11], Groth16 [6] is widely regarded as the most efficient in proof size and verification time, but it is designed to prove a single computation at once: it does not extend to state-transition systems, where the prover must re-prove the entire accumulated computation at every step, imposing a growing burden as transitions accumulate.

Incrementally verifiable computation (IVC) [12] addresses this by letting the prover incrementally extend a proof: at each step it shows only that the previous proof is valid and that the current state follows correctly from the previous one. As a result, the prover’s per-step cost is independent of the total running time, and the verifier needs only the latest proof.

Technical challenge Existing IVC approaches broadly fall into two categories, and both create difficulties in the setting above. The classical line relies on *recursive proof composition* [13]. Expressing a SNARK verifier as an arithmetic circuit requires pairing operations, which translate to approximately 15,000 constraints [14]. This overhead directly inflates the per-step proving cost, making recursive composition a persistent bottleneck for practical IVC deployment.

More recent folding-scheme-based approaches, such as Nova [15], avoid recursive proof composition and provide efficient per-step proving. However, their native fold-only proof has size $O(|C|)$ field elements and is not succinctly verifiable on its own; an additional SNARK

compression step is required to obtain a succinct proof. This compression step alone takes on the order of 7.6–8.8 seconds in our measurements (Section 6), and the resulting compressed proof still measures 11.5–12.5 KB across the benchmarked configurations.

Neither approach simultaneously provides the compact proofs, fast verification, and low update cost needed by bandwidth-constrained and latency-sensitive security applications. This brings us to the following motivating question:

Can we construct a verifiable computation scheme for state-transition computations that preserves Groth16-level proof compactness and verification speed while avoiding the prover overhead of recursive proof composition?

1.1 Our Contribution

This work presents a lightweight updatable verifiable computation scheme for state-transition systems. The construction is aimed at settings where a verifier must check the integrity of an evolving state without replaying the full transition history or handling large proof objects. The target workloads include streaming sensor-fusion updates, compact linear-algebra transitions, and cryptographic state transitions.

We introduce a *linearly updatable quadratic arithmetic program (QAP) family* for sequential circuit compositions. It gives a proof-update procedure that avoids recursive proof composition. The resulting construction has the following properties:

- **Integrity for states and transitions:** A proof certifies that the reported state follows from some sequence of witness inputs satisfying the prescribed transition circuit.
- **No recursive proof composition:** Proof updates are polynomial additions and encode no verifier circuit inside the next proving circuit.
- **Compact, constant-size proofs and verification:** A proof is $3\mathbb{G}_1 + 1\mathbb{G}_2$, the extra $\mathbb{G}_1$ element being a running commitment to the reported state, and verification is 3 pairings plus a state-commitment increment check of $O(|s_j|)$ scalar multiplications; both are independent of the step count and of B .

To state these guarantees precisely, we formalize *updatable verifiable computation* (UVC) as a proof system whose prover carries a proof and a local state across steps and whose verifier carries a constant-size digest of the accepted history. We state succinctness and local updatability as requirements, and define perfect completeness and *updatable knowledge soundness*. We

further show that the restriction to an authentic record is inherent rather than an artifact of the analysis: an adversary free to choose the verifier state can shift the running commitment by a public group operation and have an unmodified honest proof accepted for a false state. We prove perfect completeness and updatable knowledge soundness in the algebraic group model, relative to an authentic commitment. We evaluate the scheme on representative workloads matching this model: sensor fusion, Hadamard products, MiMC-style cube-permutation chains, and repeated-squaring chains. On BN254, the proof size is 160 bytes and verification takes approximately 11.6–28.9ms across the evaluated workloads. Compared with Nova’s compressed proof, our proof is 72–78× smaller and verification is substantially faster. On the sensor-fusion workload with $B = 1024$, Groth16’s cumulative proving time is $2.0\times$ ours at $K = 8$ and $1.9\text{--}2.0\times$ across $K \in \{4, 6, 8\}$, and at the end of the window our prover is $2.6\text{--}2.7\times$ faster per step. Both comparisons charge Groth16 a single common reference string (CRS) generated once for $\mathcal{C}_B$, the same one-time-setup structure our scheme uses.

1.2 Organization

The remainder of this paper is organized as follows. Section 2 surveys related work, and Section 3 formalizes the state-transition model, threat model, design goals, and integrity workflow. Section 4 recalls the QAP background and then defines the linearly updatable QAP family, and Section 5 builds the UVC syntax, security notions, and construction on that family. Section 6 reports the benchmark evaluation, and Section 7 concludes.

2 Related Work

2.1 SNARKs and Incrementally Verifiable Computation

QAP-based SNARKs Verifiable computation lets a client cryptographically check outsourced computation. The practical QAP-based SNARK pipeline runs from Pinocchio [16] through the highly optimized Groth16 scheme [6], still a common baseline for its 3-element proofs and constant-time verification, to prover-efficiency gains such as Polymath [17].

QAP composition and dynamic SNARKs Gennaro et al. [16] compose QAPs for functions sharing input and

output via the Chinese Remainder Theorem. We instead exploit the sequential structure of bounded state transitions, defining a family of QAPs that share one modulus and whose newly visible polynomial terms add linearly as the computation advances. Zhang et al. [18] propose Dynark, which updates a Groth16 proof in $O(d)$ group operations when d witness elements change within a *fixed* circuit, at the cost of $O(n \log n)$ per-witness preprocessing, while retaining zero-knowledge. This is complementary: Dynark targets witness updates for a static circuit (e.g., model re-training), whereas we target *circuit extension* from sequential composition, sacrificing zero-knowledge for additive proof updates.

Recursive proof composition and folding schemes IVC [12] lets a prover certify a long computation by repeatedly proving one more transition together with the validity of the previous proof. Classical realizations use recursive proof composition [13, 19], which must embed the previous proof’s verifier inside the next circuit, adding a large constraint overhead for pairing-based SNARKs. Folding schemes avoid encoding this verifier circuit: Kothapalli et al. [15] fold two R1CS instances into one via a random challenge, and later works extend folding to customizable constraint systems [20], non-uniform IVC [21], efficient on-chain verification [22], and lattice-based post-quantum settings [23] (surveyed in [24]). Folding supports long or unbounded computations, but its proofs are larger than a Groth16 proof unless compressed further, and Lee and Seo [25] show that the standard rewinding-based knowledge-soundness argument for such recursion holds only to logarithmic depth, though they also show Nova itself remains sound to polynomial depth under a strengthened model.

PCD and accumulation schemes Proof-carrying data [26] generalizes IVC to distributed computations, and accumulation schemes [27] aggregate verification obligations across complex proof systems. Our construction takes a narrower path, targeting bounded sequential state transitions and avoiding recursive verification via the algebraic structure of linearly updatable QAP extensions.

Table 1 compares this work with Groth16, folding-based IVC, and Dynark on trusted setup, proof size, verification and prover cost, and whether the step count is bounded.

Table 1 Comparison of our approach with Groth16, folding-scheme-based IVC, and Dynark. Here n denotes the number of constraints per step, B the maximum transition bound, d the number of changed witness elements, and $|C|$ the total circuit size; MSM denotes multi-scalar multiplication. On BN254, our $3\mathbb{G}_1+1\mathbb{G}_2$ proof is 160 bytes; a $2\mathbb{G}_1+1\mathbb{G}_2$ proof is 128 bytes.

	This work	Groth16	Folding IVC*	Dynark
Trusted setup	Yes	Yes	No	Yes
Proof size	$3\mathbb{G}_1+1\mathbb{G}_2$	$2\mathbb{G}_1+1\mathbb{G}_2$	$O(C)^{\P}$	$2\mathbb{G}_1+1\mathbb{G}_2$
Verification	3 pairings	3 pairings	$O(C)^{\P}$	3 pairings
Prover/step	$O(n)$ MSMs $+O(Bn \log Bn)$ poly	$O(n \log n)$	$O(C)$	$O(d)^{\ddagger}$
CRS size	$O(Bn)$	$O(C)$	$O(n)$	$O(n)$
Step count	Bounded (B)	N/A (single step)	Unbounded	N/A (fixed circuit)
Incremental update	Yes	No	Yes	No [‡]
Recursion overhead	None	–	None [§]	None
Post-hoc compression	None	–	Required	None

* Represents Nova [15], HyperNova [20], and MicroNova [22].

‡ Dynark updates a proof for a *fixed* circuit and it does not support circuit extension.

§ Folding replaces the recursive SNARK verifier with a lightweight in-circuit folding step; the $\sim 15,000$ -constraint pairing-verifier overhead applies to classical recursive composition [13, 14].

[¶] Native fold-only cost; succinct verification requires the separately measured compression step.

2.2 Integrity Protection for Distributed and Edge Systems

Sensor-data integrity and provenance Edge and IoT gateways often aggregate or transform data before a remote verifier or audit service observes it, so the verifier sees only a derived state while the raw inputs and intermediate updates were handled in an untrusted environment. Ramezan and Meamari [28] propose zk-IoT, using zero-knowledge proofs on blockchain platforms for IoT device authentication (~ 694 ms proving, 19 ms verification); Swalens et al. [29] introduce zkStream for trustworthy stream processing over sensor data, without constant-size cumulative proofs; Fiore and Tucker [30] develop efficient zero-knowledge proofs on signed data streams for a different verification model. A recent survey by Faraj et al. [31] documents bandwidth and complexity bottlenecks in IoT data provenance and observes that existing approaches do not fully certify the computational correctness of derived outputs.

Blockchain-integrated security services Blockchain-integrated systems use public logs, commitments, and smart contracts for tamper resistance in untrusted environments, targeting wireless IoT networks [32], collaborative storage in resource-limited clusters [33], healthcare data sharing [34], anonymous data exchange [35], and scalable edge-blockchain architectures [36], relying on hash-based commitments, homomorphic aggregation, and VDFs [37]. Our work is not a blockchain protocol, but it targets the same integrity requirement: a compact proof should convince a verifier that the current state follows from a valid sequence of updates.

3 System Model and Integrity Framework

We model the evolving computation as a state-transition system and its induced relation, give the participants, threat model, and design goals, and close with the integrity workflow used by the construction.

3.1 State-Transition System and Induced Relation

We model the evolving computation as a state-transition system, where at each step a transition input is accepted under the deployment policy and applied to the previous state to produce the next state.

Definition 1 (State-Transition System). A state-transition system is a tuple $\mathcal{ST} = (\Sigma, T, \text{StUp})$ where Σ is the set of states, T is the set of transitions, and StUp is a polynomial-time algorithm mapping from $T \times \Sigma$ to Σ . Each element in Σ and T can be represented by a bit-string of length polynomial in the security parameter, and $s_0 = \perp \in \Sigma$ is the genesis state, which our instantiations encode as the all-zero element of Σ .

For a transition sequence $\mathbf{t} = (t_1, \dots, t_j) \in T^j$, the j -fold composition of StUp is

$$\text{StUp}^j(\mathbf{t}, s_0) := \underbrace{\text{StUp}(t_j, \text{StUp}(t_{j-1}, \dots, \text{StUp}(t_1, s_0) \dots))}_{j\text{-fold composition of StUp}}.$$

When the maximum number of transitions is fixed at B , StUp^j is defined only for $j \in [1, B]$; the bound represents an operational horizon, such as a monitoring window, epoch, or audit period, and it fixes the relation generated for the construction.

Let RGen be a relation generator that takes a security parameter 1^λ and returns a polynomial-time decidable binary relation $\mathcal{R}$, whose language is the set of statements with a witness,

$$L_{\mathcal{R}} = \{ \phi \mid \exists w \text{ such that } (\phi, w) \in \mathcal{R} \}.$$

For the state-transition system, $\text{RGen}_{\mathcal{ST}}$ additionally takes the bound B and the public description of $\mathcal{ST} = (\Sigma, T, \text{StUp})$, and outputs $\mathcal{R}_{\mathcal{ST}}$. The public statement at step j is $\phi_j = (t_1, j, s_j)$, where t_1 is the first transition input and s_j is the claimed state after j transitions; only t_1 is public, so membership certifies reachability via *some* input sequence beginning with t_1 . Since the verifier ultimately checks an arithmetized statement, we take StUp to be given by an arithmetic circuit and let $\mathcal{C}_j$ denote j copies of it composed in sequence, so that a witness is an assignment to the wires of $\mathcal{C}_j$ that ϕ_j leaves free: the transition inputs $t_2, \dots, t_j$, the intermediate states $s_1, \dots, s_{j-1}$, and the internal wires of the j evaluations of StUp . Although the last two groups are determined by the inputs and s_0 , they belong to the witness because they are neither transmitted in ϕ_j nor recomputable by a verifier that never evaluates StUp ; Section 5.1 writes out $\mathcal{R}_{\mathcal{ST}}$ explicitly and gives the intermediate states a class of their own. Membership of (t_1, j, s_j) in $L_{\mathcal{R}_{\mathcal{ST}}}$ means the claimed state is reachable from the genesis state by a valid transition history of length j , and this relation fixes the statement the later proof must establish.

3.2 Participants, Threat Model, and Design Goals

The relation fixes the correctness condition; the following roles determine who produces and checks evidence for that condition. *Data sources* provide transition inputs, such as sensor readings, events, transactions, matrix blocks, vector updates, or cryptographic update values. The *prover* or service provider maintains the current state and computes the next state whenever a transition input is accepted. The *verifier* checks that a reported state is consistent with the prescribed transition computation and the claimed transition horizon. A deployment may additionally use an *audit layer*, such as a log server, blockchain, or monitoring service, to store state commitments, accepted-input digests, or verification receipts for later inspection. Here, prover and verifier are system roles; the proof algorithms invoked by these roles are introduced in Section 3.3.

Given these participants, the computationally untrusted party is the prover or service that maintains the evolving state. It may use an invalid transition computation, reuse an old state, or report a final state that is

inconsistent with its witness history. The verifier, relation description, CRS generation procedure, and public parameters are assumed to follow the protocol and to be fixed before the verification epoch begins. Against this adversary, the scheme should satisfy the following goals.

- **State-transition integrity:** An accepting proof should certify that the reported state is reachable from the genesis state by some witness sequence of valid transition steps under StUp .
- **Succinct current-state verification:** Proof size and verification time should remain independent of the number of executed transitions within the epoch. Verification is stateful yet succinct: the verifier keeps only a constant-size commitment to the previously accepted state, not full history, so verification does not grow with the step count.
- **Update without recursive proof composition:** The prover should not encode a verifier circuit for the previous proof inside the next transition circuit.
- **Practical prover update cost:** Proof updates should add only the algebraic terms associated with the newly visible part of the computation.

Verifier state and trust model Verification in our construction is *stateful*: to check step j , the verifier holds a constant-size commitment D_{j-1} to the previously accepted state and accepts step j only by extending it. This statefulness constrains the trust model in two ways.

First, soundness is relative to a verifier state left by previously accepted steps. An accepting proof π_j certifies a valid history back to the genesis state only if D_{j-1} is authentic and non-malleable. We take this as a standing assumption, on par with honest CRS generation: the adversary controls the prover and the transition inputs, but not the accepted-state record. If a verifier accepts a forged commitment, we guarantee nothing for that step, since a shifted commitment makes a false state pass. This is not a limitation of our analysis but of any scheme whose verifier state is a linearly homomorphic commitment to the reported state; Appendix B.4 gives the explicit attack. In return for this assumption, verifying a single π_j suffices, that is, given an authenticated verifier state D_{j-1} , the verifier can validate the current proof without replaying the previous proofs. The authenticity of D_{j-1} serves as a checkpoint attesting that the preceding verification chain was accepted.

Second, a verifier that lacks D_{j-1} must first obtain it. A late-joining or restarted verifier cannot run the increment check without this commitment and retrieves it from the audit layer before verifying step j , thereafter its per-step cost is again constant. The audit layer is any

authenticated store of the running commitment, such as a persistent verifier that checked the earlier steps, an append-only log, or a smart contract that validated each increment. This is the same trust anchor a stateful single-step baseline would require, carried here as one constant-size commitment rather than a per-step-verified proof log.

Remark 1. Our scheme does not guarantee input authenticity; it does not detect false physical readings or forged external events. To rule out such inputs, an application must bind authenticated inputs or commitments to them into the relation, the public statement, or the audit record. Moreover, our scheme does not aim to hide the witness or the transition history; it is not a zero-knowledge proof system.

3.3 Integrity Workflow

We employ verifiable computation, formalized by Gennaro et al. [5]: a scheme for RGen consists of three probabilistic polynomial-time algorithms (Setup, Prove, Verify) such that for any relation $\mathcal{R} \leftarrow \text{RGen}(1^\lambda)$,

- $\sigma \leftarrow \text{Setup}(\mathcal{R})$ returns a CRS σ for $\mathcal{R}$.
- $\pi \leftarrow \text{Prove}(\mathcal{R}, \sigma, \phi, w)$ generates a proof π for $(\phi, w) \in \mathcal{R}$.
- $1/0 \leftarrow \text{Verify}(\mathcal{R}, \sigma, \phi, \pi)$ checks π against σ and ϕ .

This interface treats each step as an independent proving task and suffices for the base scheme of Appendix A.3, but it cannot express how one step reuses the previous one; Section 5.1 extends it to the updatable interface our construction realizes, in which the prover carries a proof and a local state and the verifier carries a constant-size digest of the accepted history.

Integrity of the reported state is then established in four steps: the deployment fixes $\mathcal{ST}$, the bound B , and $\mathcal{R}_{\mathcal{ST}} \leftarrow \text{RGen}_{\mathcal{ST}}(1^\lambda, B, \mathcal{ST})$, and generates $\sigma \leftarrow \text{Setup}(\mathcal{R}_{\mathcal{ST}})$; at step j the service accepts t_j and computes $s_j = \text{StUp}(t_j, s_{j-1})$; the prover forms $\phi_j = (t_1, j, s_j)$ and generates π_j from the witness for the transition history (in the updatable construction of Section 5.2, by updating π_{j-1} with only the algebraic terms introduced at the new transition horizon); and the verifier checks $\text{Verify}(\mathcal{R}_{\mathcal{ST}}, \sigma, \phi_j, \pi_j)$, rather than replaying the transition transcript or storing all intermediate witnesses. An accepting verification is evidence that the reported state belongs to $\mathcal{L}_{\mathcal{R}_{\mathcal{ST}}}$ for the claimed epoch.

4 Linearly Updatable QAP Families

This section gives the paper’s main technical construction: a family of QAPs, indexed by the composition step j , that all share one modulus and one CRS, and whose members grow only by adding the polynomials for the wires introduced at step j . The QAP reduction from arithmetic circuits and the base Groth16-style verifiable computation scheme it builds on are standard and are recalled in Appendix A.

4.1 Circuit Compositions and Notation

Notation To simplify the notation, we include the constant index 0 (the wire with $a_0 = 1$) in each index set I_j . Thus every sum $\sum_{i \in I_j}$ and set $\{\cdot\}_{i \in I_j}$ below also covers the constant wire.

Circuit compositions We assume that elements of Σ and T have fixed-length bit representations (with padding if needed) and that StUp has an arithmetic circuit representation $\mathcal{C}$. For each j , the j -fold composition $\mathcal{C}_j$ is the circuit obtained by sequentially connecting j copies of $\mathcal{C}$, where the output wires of the k -th copy become input wires of the $(k+1)$ -th copy.

4.2 The Linearly Updatable QAP Family

Given a state-transition system $\mathcal{ST} = (\Sigma, T, \text{StUp})$, we construct QAPs for compositions of StUp^j for $j \in [1, B]$, where B is the maximum transition bound. We then extend these to a *linearly updatable QAP family* where all QAPs share a common modulus and are linearly related.

QAPs for each $\mathcal{C}_j$ For $j = 1, \dots, B$, let I_j be the set of indices for all wires in $\mathcal{C}_j$ with unique indices. For $\mathcal{C}_j$, we have jn equations:

$$\sum_{i \in I_j} a_i u_{i,q} \cdot \sum_{i \in I_j} a_i v_{i,q} = \sum_{i \in I_j} a_i w_{i,q},$$

where $a_0 = 1$ and we assign the q -th equation for $q \in [1 + (k-1)n, kn]$ to the k -th copy of $\mathcal{C}$. Applying the same polynomial encoding as before, we obtain:

$$\sum_{i \in I_j} a_i u_i(X) \cdot \sum_{i \in I_j} a_i v_i(X) \equiv \sum_{i \in I_j} a_i w_i(X) \pmod{P_j(X)}$$

where $u_i(X)$, $v_i(X)$, $w_i(X)$ are degree- $(jn-1)$ polynomials interpolated at arbitrarily chosen distinct values r_q ’s, and $P_j(X) = \prod_{q=1}^{jn} (X - r_q)$. This gives a QAP $(\mathbb{F}, \text{aux}, I_j^0, \mathcal{P}_{I_j})$ where $\mathcal{P}_{I_j} := (\{u_i(X), v_i(X), w_i(X)\}_{i \in I_j}, P_j(X))$.

Index sets are nested, $I_1 \subset I_2 \subset \dots \subset I_B$, and the public class of a prefix is obtained by intersection, $I_j^{\text{io}} = I_B^{\text{io}} \cap I_j$, so a wire never changes class as j grows.

Crucially, we choose the *same* r_q for the q -th equation across all QAPs. This yields the following key properties.

Two facts about extensions are used later and verified in Appendix C.1: an assignment satisfying an extended QAP satisfies the original one, and the members of the family below do form an extension chain, since P_ℓ divides P_j for $\ell < j$ and $\mathcal{P}_{I_B}$ extends every $\mathcal{P}_{I_j}$.

Linearly updatable QAP family The QAP $\mathcal{P}_{I_j}$ above is the *naïve* reduction of $\mathcal{C}_j$ and serves only as a starting point: each step would carry its own modulus $P_j(X)$ and its own degree $jn - 1$, hence its own CRS. We instead use the family $\{\widehat{\mathcal{P}}_{I_j}\}_{j \in [1, B]}$ whose members all have degree $Bn - 1$ and share the modulus $P_B(X) = \prod_{q=1}^{Bn} (X - r_q)$,

$$\widehat{\mathcal{P}}_{I_j} := (\{\hat{u}_i(X), \hat{v}_i(X), \hat{w}_i(X)\}_{i \in I_j}, P_B(X)),$$

which is $\mathcal{P}_{I_B}$ restricted to the smaller index set, and is therefore an extension of $\mathcal{P}_{I_j}$. We call $\{(\mathbb{F}, \text{aux}, I_j^{\text{io}}, \widehat{\mathcal{P}}_{I_j})\}_{j \in [1, B]}$ a *linearly updatable QAP family* for $\mathcal{C}$. Its three defining features are that all members share $P_B(X)$, that passing from $\widehat{\mathcal{P}}_{I_j}$ to $\widehat{\mathcal{P}}_{I_{j+1}}$ only adds the polynomials indexed by $I_{j+1} \setminus I_j$, and that those polynomials are fixed once for the whole family, members differing only in the index set summed over. A single CRS therefore suffices for every step.

These features do not yet give completeness, because the honest assignment of $\mathcal{C}_j$ satisfies the gate equations only modulo $P_j(X)$ while the verifier checks modulo $P_B(X)$. At the extra roots, which correspond to future-step gates, prefix-visible wires may still appear; if they appear on both multiplicative sides, the restricted product can be nonzero even though the future output wire is absent, the numerator need not be divisible by $P_B(X)$, and the quotient $h_j(X)$ need not exist. A gate discipline rules this out.

4.3 Gate Discipline and Intermediate Completeness

A required gate discipline An honest assignment of $\mathcal{C}_j$ satisfies the naïve equation only modulo its own $P_j(X)$, but the family evaluates every step modulo $P_B(X)$, whose extra roots r_q with $q > jn$ belong to later gates; an honest assignment must satisfy the equation there too. The *gate discipline* secures this. Writing, for a prefix j and a future gate $q > jn$, the prefix-visible active

sets

$$\begin{aligned} U_q^{(j)} &= \{i \in I_j : u_{i,q} \neq 0\}, \\ V_q^{(j)} &= \{i \in I_j : v_{i,q} \neq 0\}, \\ W_q^{(j)} &= \{i \in I_j : w_{i,q} \neq 0\}. \end{aligned}$$

the discipline requires $W_q^{(j)} = \emptyset$ and either $U_q^{(j)} = \emptyset$ or $V_q^{(j)} = \emptyset$, for every $j \in [1, B]$ and $q > jn$: the verified function is unchanged, but constants and carried state are placed so that no future gate uses a prefix-visible wire on both multiplied sides; e.g., a future gate multiplying by 1 uses a fresh per-step unit wire rather than the shared R1CS constant wire. For knowledge extraction, each committed state coordinate is additionally assigned a fresh output wire with one designated output row: coefficient 1 on the W side there, 0 for every other committed state wire, and no W -side occurrence elsewhere.

Lemma 1 (Intermediate completeness). *Assume that the arithmetization follows the gate discipline above. If $\{a_i\}_{i \in I_j}$ is any satisfying assignment for $\mathcal{C}_j$, then the assignment over I_B that extends it by $a_i = 0$ for $i \in I_B \setminus I_j$ satisfies the extended QAP equation with the common modulus $P_B(X)$:*

$$\begin{aligned} &\left(\sum_{i \in I_j} a_i \hat{u}_i(X) \right) \left(\sum_{i \in I_j} a_i \hat{v}_i(X) \right) \\ &\quad \equiv \sum_{i \in I_j} a_i \hat{w}_i(X) \pmod{P_B(X)}. \end{aligned}$$

The proof, in Appendix C.1, checks the congruence root by root: at a root of a gate of $\mathcal{C}_j$ it is the ordinary R1CS constraint, and at a future root the gate discipline makes the restricted output side and one restricted multiplicative side vanish, so the restricted product vanishes too.

When a carried state value feeds a later step's non-linear computation, the arithmetization preserves the discipline by first binding that value to fresh step-local auxiliary wires used inside that step's gates, leaving prefix-restricted future gates with a structural zero on one multiplicative side; the benchmark circuits use this form.

5 Constructing Updatable Verifiable Computation

We now build an updatable verifiable computation scheme on the linearly updatable QAP family of Section 4.2, taking the single-step scheme of Appendix A.3 as the starting point. Section 5.1 defines the syntax

of an updatable verifiable computation (UVC) scheme together with its completeness and knowledge-soundness requirements. Section 5.2 instantiates that syntax, showing how the linear update property allows incremental proof updates without re-proving from scratch. Section 5.3 analyzes the resulting efficiency. Section 5.4 states the security guarantees; the proofs are deferred to Appendix C.

5.1 Syntax and Security Notions

The interface of Section 3.3 carries no proof or digest across steps, so neither *update without recursive composition* nor *succinct current-state verification* is expressible in it; we fix below the interface our construction realizes. Throughout, $\mathcal{ST} = (\Sigma, T, \text{StUp})$ has transition bound B and $\mathcal{R}_{\mathcal{ST}} \leftarrow \text{RGen}_{\mathcal{ST}}(1^\lambda, B, \mathcal{ST})$ is the relation of Section 3.1.

Wire classes The two-way split of Appendix A.2 cannot be used here, because a state has no fixed side: the prover reports s_j in ϕ_j , but at step $j+1$ the verifier reaches it only through the state it carries, whereas the per-step proof increments split along the classes and need them stable. We give the states a class of their own,

$$I = I^{\text{io}} \dot{\cup} I^{\text{st}} \dot{\cup} I^{\text{wt}} \quad (1)$$

with $\{0\}$ carrying the constant $a_0 = 1$ as before. The verifier recomputes I^{io} from ϕ_j at every step, never reads I^{st} individually but reaches it through vst_{j-1} , and never sees I^{wt} at all. Fig. 1 shows the situation the third class resolves.

For $\text{RGen}_{\mathcal{ST}}$, the I_j^{io} coordinates are determined by the anchor t_1 and the step index j , and the arithmetization identifies the coordinates of s_j with the indices in $I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}$, so $|I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}| = |s_j|$ for every j . Earlier state coordinates are not transmitted and are pinned by vst_{j-1} , which keeps ϕ_j constant-size while the public part of the assignment grows with j . The witness splits along the classes: the transition inputs t_k and the internal wires z_k occupy I_j^{wt} , while the intermediate states s_k occupy I_j^{st} . The prover therefore supplies only the I_j^{wt} coordinates, the rest being pinned by vst_{j-1} . We write

$$\omega_j := (a_i)_{i \in I_j^{\text{wt}} \setminus I_{j-1}^{\text{wt}}}$$

for the *witness increment* of step j : the transition input t_j together with the internal wires z_j of the j -th copy.

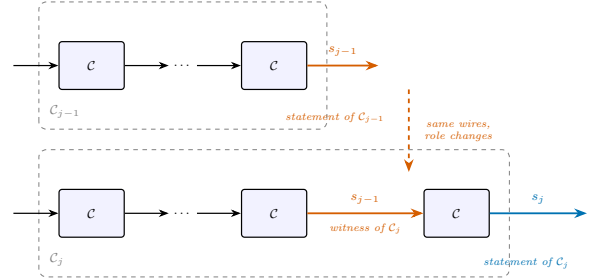


Fig. 1 Why the state needs a class of its own. The bundle s_{j-1} , which may be several wires, is the output of C_{j-1} and so belongs to its statement; inside C_j the same wires are internal, hence part of the witness. A class fixed once for the whole family can therefore call them neither public nor free, and I^{st} is the class that lets the verifier keep reaching them through vst_{j-1} once they become witness coordinates.

The relation The relation is the statement-witness relation of C_j , split by the classes of Eq. (1). Its statement is the triple $\phi = (t_1, j, s_j)$, which pins the coordinates of I_j^{io} together with the newest state slice: among the I_j^{io} coordinates only the anchor carries data, the others being the step-local units of the j visible copies, so over the full-bound class I_B^{io} they spell out $1^j 0^{B-j}$, the trailing zeros coming from the vanishing condition below. The witness is everything else that C_j leaves free, namely the wires of I_j^{wt} and the earlier state coordinates I_{j-1}^{st} , which were the output of C_{j-1} but are internal here. Finally, one QAP of degree $Bn-1$ serves every step, so an assignment is always indexed by $\{0\} \cup I_B$ and the index j carried in ϕ says where the prefix stops: coordinates beyond I_j are required to vanish. Altogether,

$$\mathcal{R}_{\mathcal{ST}} = \left\{ (\phi, w) \left| \begin{array}{l} \phi = (t_1, j, s_j), \\ w = (a_i)_{i \in I_j^{\text{wt}} \cup I_{j-1}^{\text{st}}}, \\ (a_i)_{i \in I_j^{\text{io}}} = (t_1, 1^j), \\ (a_i)_{i \in I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}} = s_j, \\ a_0 = 1, \quad a_i = 0 \text{ for } i \in I_B \setminus I_j, \\ \sum_{i \in \{0\} \cup I_B} a_i \hat{u}_i(X) \cdot \sum_{i \in \{0\} \cup I_B} a_i \hat{v}_i(X) \\ \equiv \sum_{i \in \{0\} \cup I_B} a_i \hat{w}_i(X) \pmod{P_B(X)} \end{array} \right. \right\}. \quad (2)$$

That condition is what makes the congruence modulo $P_B(X)$ the right statement for a prefix of length j , since the extra roots belong to gates of later copies and Lemma 1 shows a zero-padded satisfying assignment for C_j meets it there as well; since ϕ carries only t_1 and s_j , the statement stays constant-size even though the coordinates it pins grow with j , and we write ϕ_j for the statement of step j throughout. Appendix B.1 fixes the

prefix terminology: prefix circuit, prefix-visible and future wires, prefix assignment, and prefix-causality, that are used below and by the proofs. We assume throughout, and list among the hypotheses of Theorem 2, that the arithmetization is *faithful*: an assignment satisfies Eq. (2) exactly when the values it places on the transition inputs, intermediate states, and internal wires form a valid run of j copies of StUp from s_0 ending in s_j , so membership carries the reachability meaning of Section 3.1 directly.

Definition 2 (Updatable verifiable computation). An *updatable verifiable computation* (UVC) scheme for $\text{RGen}_{\mathcal{ST}}$ is a triple of polynomial-time algorithms (Setup , Prove , Verify) together with fixed initial values (π_0, ζ_0) and vst_0 :

- $\sigma \leftarrow \text{Setup}(\mathcal{R}_{\mathcal{ST}})$ takes the relation and returns a CRS σ that is used by *every* step $j \in [1, B]$.
- $(\pi_j, \zeta_j) \leftarrow \text{Prove}(\mathcal{R}_{\mathcal{ST}}, \sigma, j, \phi_j; \omega_j, \pi_{j-1}, \zeta_{j-1})$ takes the step index j , the current statement ϕ_j , and the witness increment ω_j , together with the previous proof π_{j-1} and the previous prover state ζ_{j-1} , and returns the updated proof π_j and prover state ζ_j .
- $(b, \text{vst}_j) \leftarrow \text{Verify}(\mathcal{R}_{\mathcal{ST}}, \sigma, j, \phi_j, \pi_j, \text{vst}_{j-1})$ takes the verifier state vst_{j-1} carried from the previously accepted step and returns a bit $b \in \{0, 1\}$ together with an updated verifier state vst_j , which is defined only when $b = 1$.

The prover state ζ_j is local and neither transmitted nor read by Verify ; the verifier state vst_j is the only information the verifier retains between steps, produced by Verify itself when it accepts step j , so a verifier never holds a state that some accepted step did not hand it; in the construction of Section 5.2 this state is one group element of the accepted proof.

The scheme is *succinct* when $|\pi_j|$ and $|\text{vst}_j|$ are $\text{poly}(\lambda)$ and Verify runs in time $\text{poly}(\lambda, |\phi_j|)$, so that proof size, verifier storage, and verification time are independent of j , of B , and of the number of constraints n per step. It is *locally updatable* when the input of Prove at step j contains no wire value of $\mathcal{C}_{j-1}$ beyond what is carried inside $(\phi_j, \pi_{j-1}, \zeta_{j-1})$, with $|\zeta_{j-1}|$ independent of j ; equivalently, the prefix assignment $(a_i)_{i \in I_{j-1}}$ is not an input to Prove . Section 5.3 checks both for the construction. Local updatability constrains the prover’s *input* rather than its internal mechanism, so recursive and folding-based IVC schemes satisfy it too; what distinguishes our construction is that Prove never invokes Verify and never proves a statement about π_{j-1} , the update being a group-element addition determined by the QAP family.

Completeness Completeness is stated over an entire honest execution, since both π_{j-1} and vst_{j-1} are

outputs of earlier steps: for $\mathbf{t} = (t_1, \dots, t_J) \in T^J$ with $J \leq B$, the *honest execution* on $\mathbf{t}$ sets $s_j = \text{StUp}(t_j, s_{j-1})$ and $\phi_j = (t_1, j, s_j)$ for $j \in [1, J]$, takes ω_j to be the witness increment induced by the j -th circuit copy on (t_j, s_{j-1}) , and runs Prove then Verify for $j = 1, \dots, J$ in order.

Definition 3 (Perfect completeness). A UVC scheme has *perfect completeness* if, for every $\mathcal{ST}$, every bound B , every $J \in [1, B]$, and every $\mathbf{t} \in T^J$, the honest execution on $\mathbf{t}$ (generate $\mathcal{R}_{\mathcal{ST}} \leftarrow \text{RGen}_{\mathcal{ST}}(1^\lambda, B, \mathcal{ST})$ and $\sigma \leftarrow \text{Setup}(\mathcal{R}_{\mathcal{ST}})$, then for $j = 1, \dots, J$ in order run Prove on the honestly derived (ϕ_j, ω_j) with the previous proof and prover state and Verify on the resulting π_j with the verifier state carried from the previous step) outputs $b_j = 1$ for every $j \in [1, J]$ with probability 1, the probability taken over Setup ’s randomness. The formal experiment is given in Appendix B.2.

Updatable knowledge soundness What an accepting proof certifies depends on the verifier state it was checked against, and in our construction that state is $[D_{j-1}]_1$, a component of the previously accepted proof. We therefore let the adversary play the prover’s role for the whole run, not only the last step, and demand extraction once, from the last proof, for a witness of the whole prefix; letting the adversary produce the earlier proofs too, rather than having them supplied honestly, is what gives the notion its content, as Appendix B.3 explains alongside the formal experiment. At $j = 1$ the run has a single step and the display reduces to Definition 5.

Definition 4 (Updatable knowledge soundness). A UVC scheme has *updatable knowledge soundness* if, for every step index $j \in [1, B]$ and every non-uniform polynomial-time algebraic adversary $\mathcal{A}$, there is a non-uniform polynomial-time extractor $\chi_{\mathcal{A}}$ with the following guarantee. The adversary receives the public parameters $\text{pp} := (\mathcal{R}_{\mathcal{ST}}, \sigma)$ with $\text{pp} \leftarrow \text{Setup}(\text{RGen}_{\mathcal{ST}}(1^\lambda, B, \mathcal{ST}))$ and, before producing any proof, commits to the target step j and to every statement $\phi_1, \dots, \phi_j$ together with an initial auxiliary state aux_0 ; it then plays the prover’s role for the whole run, producing $\pi_1, \dots, \pi_j$ one at a time while carrying aux forward, and the experiment feeds each proof through Verify in sequence, starting from the scheme’s initial verifier state vst_0 and keeping the verifier state Verify returns. The extractor $\chi_{\mathcal{A}}$ is then run once, on the transcript $(\phi_i, \pi_i)_{i \leq j}$ together with the adversary’s random coins $r_{\mathcal{A}}$, and must output a witness w . The probability that every step is accepted, $b_1 = \dots = b_j = 1$, while extraction nonetheless fails is at most $\text{negl}(\lambda)$; extraction fails either because

$(\phi_j, w) \notin \mathcal{R}_{\mathcal{ST}}$, or because w disagrees with some earlier reported state, i.e., $w|_{\Delta I_i^{\text{st}}} \neq s_i$ for some $i < j$, where $\Delta I_i^{\text{st}} := I_i^{\text{st}} \setminus I_{i-1}^{\text{st}}$ is the state-wire increment at step i and s_i denotes the state component of $\phi_i = (t_1, i, s_i)$. The formal experiment is given in Appendix B.3.

The second disjunct is what makes the definition match the guarantee already used in Section 1 and established by the proof of Theorem 2: without it, membership in $\mathcal{R}_{\mathcal{ST}}$ alone says only that s_j is reachable in j steps through *some* history, not ruling out an adversary that reports one history through $\phi_1, \dots, \phi_{j-1}$ and satisfies the relation at ϕ_j with a different one.

At $j = 1$ the witness is that of a single circuit copy; for $j > 1$ the witness of Eq. (2) contains the whole intermediate history, so an accepting update certifies not merely the reported s_j but a valid path to it, and $[D_{j-1}]_1$ is what forces that path to be the one already certified.

For the construction of Section 5.2, $\text{vst}_{j-1} = [D_{j-1}]_1$ is part of π_{j-1} and, by chaining the accepted increment checks along the run, takes a closed form confined to I_{j-1}^{st} regardless of which proofs the adversary used to get there, which is the only property of the earlier steps the extraction argument uses. That the experiment computes vst_{j-1} itself rather than letting the adversary supply it is accordingly not an artifact of the proof: Appendix B.4 derives the closed form explicitly and gives an attack against an adversarially chosen verifier state.

5.2 Construction

Given a state-transition system $\mathcal{ST}$ with maximum transition bound B and its linearly updatable QAP family $\{(\mathbb{F}, \text{aux}, I_j^{\text{io}}, \widehat{P}_{I_j})\}_{j \in [1, B]}$ from Section 4.2, where the transition circuit follows the gate discipline stated there, we construct a UVC scheme in the sense of Definition 2. The *linear update* property of the QAP family, where all members share the common modulus $P_B(X)$ and updating from step j to $j + 1$ requires only adding polynomials for new wire indices $I_{j+1} \setminus I_j$, translates directly into an additive update mechanism for the proof components (A, B, C) . Since the CRS is generated once for the entire family, the verifier uses the *same* verification equation regardless of the step count j .

Notation Every polynomial appearing in the rest of the paper is the *extension* polynomial $\hat{u}_i(X)$, $\hat{v}_i(X)$, or $\hat{w}_i(X)$ of the linearly updatable QAP family of degree $Bn - 1$ and sharing the common modulus $P_B(X)$.

The naïve step- j polynomials $u_i(X)$, $v_i(X)$, $w_i(X)$ and the step- j modulus $P_j(X)$ of Section 4.2 are *never* used by the scheme. Accordingly, every formula below writes $\hat{u}_i(x)$, $\hat{v}_i(x)$, $\hat{w}_i(x)$ explicitly, evaluated at the trapdoor point x .

The key observation is that all equations in $\mathcal{R}_{j-1}$ are contained in $\mathcal{R}_j$, enabling proof updates by appending information for new equations. Since $I_{j-1} \subset I_j$ and the polynomials for indices in I_{j-1} are unchanged within the family, the proof component $A_j = \alpha + \sum_{i \in I_j} a_i \hat{u}_i(x)$ decomposes as

$$A_j = \alpha + \underbrace{\sum_{i \in I_{j-1}} a_i \hat{u}_i(x)}_{=A_{j-1}} + \sum_{i \in I_j \setminus I_{j-1}} a_i \hat{u}_i(x).$$

This holds symmetrically for B_j . For $j > 1$, the components C_j and D_j split along the step-invariant wire classes fixed by UVC.Setup: the free witness wires I_j^{wt} contribute to C_j , and the committed state wires I_j^{st} contribute to the commitment D_j . Because class membership does not change with the step index, the increments are simply $I_j^{\text{wt}} \setminus I_{j-1}^{\text{wt}}$ and $I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}$. Define $L_i(x) = \beta \hat{u}_i(x) + \alpha \hat{v}_i(x) + \hat{w}_i(x)$.

$$C_j = C_{j-1} + \sum_{i \in I_j^{\text{wt}} \setminus I_{j-1}^{\text{wt}}} a_i \frac{L_i(x)}{\delta} + \frac{(h_j(x) - h_{j-1}(x))P_B(x)}{\delta}$$

$$D_j = D_{j-1} + \sum_{i \in I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}} a_i \frac{L_i(x)}{\gamma}$$

$\sigma \leftarrow \text{UVC.Setup}(\mathcal{R}_{\mathcal{ST}})$

Choose random $\alpha, \beta, \gamma, \delta, x \in \mathbb{F}^*$. The wire classes are those of Eq. (1), fixed once for the full-bound circuit and obtained for each prefix by intersection. We instantiate them as follows: I_B^{io} contains the anchor t_1 and, for each circuit copy $k \in [1, B]$, a *step-local unit wire* unit_k prescribed to be 1 once copy k is visible and acting as a switch value that turns a not-yet-visible copy's constraints homogeneous; I_B^{st} contains the state outputs $s_1, \dots, s_B$, reached only through the running commitment; and I_B^{wt} contains the remaining wires, namely the transition inputs $t_2, \dots, t_B$ and all internal wires of the copies. Only the anchor t_1 is public among the inputs; the later inputs $t_2, \dots, t_B$ are free witnesses, which is why Theorem 2 certifies reachability through *some* witness input sequence rather than through a specific one. Appendix B.5 gives the full instantiation, its conventions (genesis handling, one wire per state), the switch-value argument in detail, and a worked step-2 example.

The CRS places $L_i(x)/\gamma$ on the two public classes and $L_i(x)/\delta$ on the free witnesses; I_B^{io} and I_B^{st} share the γ track and are separated only by the verifier's access

pattern. The full listing $\sigma = ([\sigma_1]_1, [\sigma_2]_2)$ is given in Appendix B.5.

The prover keeps local polynomial state $\zeta_j = (U_j, V_j, W_j, h_j)$. Its entries are defined by

$$\begin{aligned} U_j &= \sum_{i \in I_j} a_i \hat{u}_i, & V_j &= \sum_{i \in I_j} a_i \hat{v}_i, \\ W_j &= \sum_{i \in I_j} a_i \hat{w}_i, & h_j &= (U_j V_j - W_j) / P_B. \end{aligned}$$

This state is not part of the proof or verifier state, and a prover that does not retain it must reconstruct it from the full prefix assignment.

The initial values required by Definition 2 are $I_0 = \emptyset$ and

$$\begin{aligned} A_0 &= \alpha, & B_0 &= \beta, & C_0 &= D_0 = 0, \\ \zeta_0 &= (0, 0, 0, 0), & \text{vst}_0 &= [0]_1, \end{aligned}$$

so that $\pi_0 = ([\alpha]_1, [\beta]_2, [0]_1, [0]_1)$. This π_0 proves nothing; it seeds the update recursion. The constants α and β are the terms that A_j and B_j carry at every step, so starting from them makes the first call of **UVC.Prove** reproduce the base proof of Appendix A.3, while $C_0 = D_0 = 0$ and $\zeta_0 = 0$ are the empty sums over I_0 .

$(\pi_j, \zeta_j) \leftarrow \text{UVC.Prove}(\mathcal{R}_{\mathcal{ST}}, \sigma, j, \phi_j; \omega_j, \pi_{j-1}, \zeta_{j-1})$

Parse the previous proof and prover state as

$$\begin{aligned} \pi_{j-1} &= ([A_{j-1}]_1, [B_{j-1}]_2, [C_{j-1}]_1, [D_{j-1}]_1), \\ \zeta_{j-1} &= (U_{j-1}, V_{j-1}, W_{j-1}, h_{j-1}). \end{aligned}$$

Assemble the newly visible assignment $(a_i)_{i \in I_j \setminus I_{j-1}}$ from three sources: the free wires $I_j^{\text{wt}} \setminus I_{j-1}^{\text{wt}}$ take their values from the witness increment ω_j ; the state wires $I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}$ take the coordinates of s_j , read off the statement $\phi_j = (t_1, j, s_j)$; and the verifier-fixed wires $I_j^{\text{io}} \setminus I_{j-1}^{\text{io}}$ take their prescribed values, namely 1 for the step-local unit of the j -th copy and, when $j = 1$, the anchor coordinates read from t_1 . Compute $U_{\text{new}} = \sum_{i \in I_j \setminus I_{j-1}} a_i \hat{u}_i$, $V_{\text{new}} = \sum_{i \in I_j \setminus I_{j-1}} a_i \hat{v}_i$, and $W_{\text{new}} = \sum_{i \in I_j \setminus I_{j-1}} a_i \hat{w}_i$, then set $U_j = U_{j-1} + U_{\text{new}}$, $V_j = V_{j-1} + V_{\text{new}}$, $W_j = W_{j-1} + W_{\text{new}}$, and $h_j = (U_j V_j - W_j) / P_B$. The quotient exists by Lemma 1 and has degree at most $Bn - 2$. Compute the proof components as follows:

$$\begin{aligned} A_j &= A_{j-1} + \sum_{i \in I_j \setminus I_{j-1}} a_i \hat{u}_i(x) \\ B_j &= B_{j-1} + \sum_{i \in I_j \setminus I_{j-1}} a_i \hat{v}_i(x) \\ D_j &= D_{j-1} + \sum_{i \in I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}} a_i \frac{L_i(x)}{\gamma} \\ C_j &= C_{j-1} + \sum_{i \in I_j^{\text{wt}} \setminus I_{j-1}^{\text{wt}}} a_i \frac{L_i(x)}{\delta} + \frac{(h_j(x) - h_{j-1}(x)) P_B(x)}{\delta} \end{aligned}$$

Output $\pi_j = ([A_j]_1, [B_j]_2, [C_j]_1, [D_j]_1)$ and retain ζ_j locally.

Unwinding the four update rules gives the closed forms

$$\begin{aligned} A_j &= \alpha + \sum_{i \in I_j} a_i \hat{u}_i(x), & C_j &= \sum_{i \in I_j^{\text{wt}}} a_i \frac{L_i(x)}{\delta} + \frac{h_j(x) P_B(x)}{\delta}, \\ B_j &= \beta + \sum_{i \in I_j} a_i \hat{v}_i(x), & D_j &= \sum_{i \in I_j^{\text{st}}} a_i \frac{L_i(x)}{\gamma}, \end{aligned}$$

in which every component is determined by σ and the prefix assignment $(a_i)_{i \in I_j}$ alone, since $h_j = (U_j V_j - W_j) / P_B$ is itself so determined. The completeness proof verifies that the incremental rule reaches exactly these values. Note that the fourth component has no analogue in the base scheme of Appendix A.3; these are closed forms for the present scheme and not a comparison with **VC.Prove**.

$(b, \text{vst}_j) \leftarrow \text{UVC.Verify}(\mathcal{R}_{\mathcal{ST}}, \sigma, j, \phi_j, \pi_j, \text{vst}_{j-1})$

Parse $\text{vst}_{j-1} = [D_{j-1}]_1$, the running state commitment of the previously accepted step, and

$$\pi_j = ([A_j]_1, [B_j]_2, [C_j]_1, [D_j]_1).$$

Read the reported state s_j from ϕ_j and derive

$$[\tilde{D}_j]_1 := [D_{j-1}]_1 + \sum_{i \in I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}} a_i [L_i(x) / \gamma]_1.$$

Output $b = 1$ and $\text{vst}_j := [\tilde{D}_j]_1$ if and only if both

$$\begin{aligned} [D_j]_1 &= [\tilde{D}_j]_1, \\ [A_j]_1 \cdot [B_j]_2 &= [\alpha]_1 \cdot [\beta]_2 + ([Q_j]_1 + [D_j]_1) \cdot [\gamma]_2 + [C_j]_1 \cdot [\delta]_2, \end{aligned}$$

where $[Q_j]_1 = \sum_{i \in \{0\} \cup I_j^{\text{io}}} a_i [L_i(x) / \gamma]_1$ is the public term determined by the statement and the fixed relation: $a_0 = 1$, the anchor coordinates are read from t_1 , and every visible step-local unit has value 1 as determined by j . For a fixed relation, the contribution of a_0 and all visible step-local units is precomputed for each j in the verification key (or equivalently updated by one fixed group addition per accepted step), so evaluating Q_j online requires only the fixed-size anchor contribution and is independent of the prefix length. The reported s_j enters through the commitment-increment equation, not through Q_j .

As defined above, the prover outputs four group elements: $[A_j]_1$, $[B_j]_2$, $[C_j]_1$, and $[D_j]_1$. The element $[D_j]_1$ binds the reported current state to the commitment from the previous accepted step through the increment equation $[D_j]_1 = [D_{j-1}]_1 + \sum_{i \in I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}} a_i [L_i(x) / \gamma]_1$. Since the verifier derives $[\tilde{D}_j]_1$ on its own and accepts only when $[D_j]_1 = [\tilde{D}_j]_1$, the fourth element is redundant whenever the running commitment is available to

the verifier. If an authenticated bulletin board or audit layer supplies $[D_{j-1}]_1$, the verifier can derive $[D_j]_1$ from that commitment and the reported current-state coordinates instead of receiving $[D_j]_1$ from the prover. In this deployment, a proof becomes $[A_j]_1$, $[B_j]_2$, and $[C_j]_1$, comparable to Groth16.

Inspecting the three algorithms confirms the two requirements of Section 5.1: $|\pi_j| = 3\mathbb{G}_1 + 1\mathbb{G}_2$ and $|\text{vst}_j| = 1\mathbb{G}_1$ for every j , with **Verify** performing 3 pairings and $|s_j| + O(1)$ scalar multiplications in $\mathbb{G}_1$, while ζ_j is four polynomials of degree at most Bn and no wire value of $\mathcal{C}_{j-1}$ is an input to **Prove**. Moreover **UVC.Prove** makes no call to **UVC.Verify** and depends on π_{j-1} only through group addition, so no verifier circuit is arithmetized at any step.

5.3 Efficiency Analysis

Setup cost is dominated by the FFT/MSM work needed to evaluate the CRS, a one-time cost for a fixed circuit and bound. Base proving ($j = 1$) matches Groth16 for A_1, B_1 , but the quotient $h_1(X)$ is computed over the extended modulus $P_B(X)$, a factor- B overhead. Each update reuses the prover-local state $\zeta_{j-1} = (U_{j-1}, V_{j-1}, W_{j-1}, h_{j-1})$ rather than recomputing the quotient from scratch: writing $h_j = h_{j-1} + \Delta h_j$ and $U_{\text{new}}, V_{\text{new}}, W_{\text{new}} = \sum_{i \in I_j \setminus I_{j-1}} a_i \hat{u}_i, \hat{v}_i, \hat{w}_i$, the divisibility equations for steps $j-1$ and j give $W_{\text{new}} + \Delta h_j(X)P_B(X) = U_{\text{new}}V_{\text{new}} + U_{\text{new}} \sum_{i \in I_{j-1}} a_i \hat{v}_i + \sum_{i \in I_{j-1}} a_i \hat{u}_i \cdot V_{\text{new}}$, whose right-hand side has degree at most $2Bn-2$ and, by Lemma 1, is divisible by $P_B(X)$, so Δh_j has degree at most $Bn-2$ and is obtained by one FFT-based division.

None of the prover, verifier, or communication costs grow with the number of steps taken so far, and for moderate B (e.g., $B \leq 256$) the update cost remains competitive with folding's $O(|C|)$ per-step MSM cost without any recursion overhead, extending the comparison of Table 1 with a concrete threshold. Encoding all B steps into one CRS is accordingly the family's main deployment trade-off: a factor B in setup and storage against a one-step Groth16 CRS, in exchange for constant-size proofs and verification.

5.4 Security Analysis

We now show that the construction of Section 5.2 meets the requirements of Section 5.1, stating the target theorem first in the algebraic group model (Definition 6) under q -DLog (Definition 7).

Theorem 2 (Updatable VC security). *Let $N = Bn \geq 2$. Assume that the composed RICS is prefix-causal, faithful, and follows the gate discipline of Section 4.2, and that setup uses the stable disjoint partition above. Under q -DLog for $q = 2N - 1$, the updatable VC scheme of Section 5.2 has perfect completeness (Definition 3) and updatable knowledge soundness (Definition 4) against algebraic adversaries. Concretely, for every algebraic adversary whose run of j proofs is accepted throughout, the extractor outputs a satisfying assignment for $\mathcal{C}_j$ that agrees with the verifier-fixed and public coordinates and with every state reported along the run. By faithfulness, s_j is then reachable from genesis through some witness input sequence of length j .*

The two boundaries of Section 3.2 apply unchanged, the second of them, an adversarially chosen verifier state, being inherent rather than an artifact of the analysis, as Section 5.1 shows.

The proof is given in Appendix C.2: completeness is a direct induction resting on Lemma 1, and soundness treats an accepted update as an ordinary QAP proof in disguise, with the whole γ track $\{0\} \cup I_B^g \cup I_B^s$ playing the part of the public input, extracted once from the last proof with error uniform in j rather than compounding with the length of the run.

6 Performance Evaluation

We evaluate our updatable VC scheme on bounded state-transition circuits, comparing performance against Groth16 re-proving and Nova folding-scheme IVC. The evaluation asks whether proof size and verification latency remain constant across transition horizons, when proof updating becomes more efficient than re-proving the accumulated circuit, and how the fixed transition bound B affects setup and per-step cost across different workload classes.

6.1 Experimental Setup

Implementation and libraries Our implementation is written in C++ on top of the libsnark library [38], which provides finite-field and elliptic-curve arithmetic over the `alt_bn128` (BN254) pairing-friendly curve. We compare against two baselines: Groth16 and Nova. For Groth16, we use the implementation within the same libsnark codebase to ensure a fair comparison on identical field and curve operations; for Nova, we use the `nova-snark` library [39], a Rust implementation with BN254 support.

For Nova, we distinguish two costs: the per-step *folding* cost, which accumulates the R1CS instance, and the *SNARK compression* cost required to turn the non-succinct fold-only object into the reported succinct, compressed-verification proof. We use `nova-snark`'s Spartan-based compressing SNARK, instantiated as a `RelaxedR1CSSNARK` with a HyperKZG evaluation engine on the primary curve and an IPA evaluation engine on the secondary curve, not Groth16 compression. Per-step tables mark fold-only costs with † to give Nova the most favorable proving-time comparison, with the compression overhead reported separately. The resulting compressed proof serializes to 11,520–12,512 bytes across the benchmarked configurations. The source code is publicly available at <https://github.com/heewon-chung/libuvc>.

The authors wrote the initial implementation and used Claude (Anthropic) to assist with refactoring and debugging it. All code was reviewed and tested by the authors, who take full responsibility for the correctness of the reported results.

Testbed All experiments were executed on a Raspberry Pi 5 single-board computer (4-core Arm Cortex-A76, 7.9 GB RAM) running Debian, which is representative of the edge-gateway class of device that motivates this work. All reported timings are wall-clock measurements taken as the median over 10 runs. Because a Nova folding chain is inherently sequential, its timings are instead medians over five independent end-to-end repetitions of the whole chain.

We evaluate the protocol across four representative workload classes under the state-transition interface of Section 3: sensor fusion and Hadamard products for streaming and linear-algebra state updates, and a MiMC-style cube chain and repeated squaring for nonlinear, long sequential arithmetic (Sections 6.2–6.3). Across these workloads, we report CRS size, per-step and cumulative proving time, verification time, and proof size: the comparison with Groth16 measures the cost of avoiding full re-proving at each step, and the comparison with Nova separates fold-only performance from the compression step used to generate a verifiable succinct proof.

Choice of the Groth16 baseline Groth16 can be deployed against a growing transition history in three ways, and the choice matters for what the comparison means. Generating a fresh CRS for every prefix circuit $\mathcal{C}_j$ is the most direct translation of “re-prove from scratch”, but it charges the prover a full setup at every step, which a deployment would normally avoid by preparing keys offline. We therefore take as our

main baseline the variant that a careful deployment would actually run: a single CRS generated once for $\mathcal{C}_B$, reused at every step, with the prefix assignment zero-padded to $\mathcal{C}_B$. This gives Groth16 the same one-time-setup structure as our scheme. The third option is single-step chaining: prove only the fixed n -constraint transition circuit $\mathcal{C}(s_{j-1} \rightarrow s_j)$ with s_{j-1} and s_j as public inputs, and let the verifier link consecutive proofs itself. Because its circuit never grows, the proving time is genuinely fast, and our measurements put it 3.8–24.7× ahead of our prover in cumulative proving cost. The saving, however, moves the cost to the verifier. A proof certifies one transition only, so a party that wants assurance about the state after j steps must obtain and check all j proofs; a verifier that restarts or joins late has no way to recover the prefix it did not witness, and its work grows with the history. Our construction instead re-certifies the whole prefix from one constant-size proof and the stored commitment $[D_{j-1}]_1$, which is the property the comparison in this section is about. The remaining measurements therefore use fixed-CRS prefix re-proving as the Groth16 baseline, since it is the variant that offers the same guarantee to the verifier at the same setup cost structure as ours.

6.2 Sensor-Fusion Workload

The first workload is a sensor-fusion circuit that gives a concrete streaming example for edge or IoT monitoring. In this workload, a service repeatedly incorporates fresh measurements into a running state while a verifier needs compact evidence for the resulting state.

The sensor-fusion circuit models a weighted exponential moving average (EMA) aggregation of K sensor readings. Each step computes

$$s_j = \sum_{k=1}^K w_k \cdot x_k + \rho \cdot s_{j-1},$$

producing $K + 1$ R1CS constraints per step. Although the weights w_k and ρ are public constants, each circuit copy uses a fresh step-local verifier-fixed/public coordinate u_j whose prescribed value is 1 and whose wire is absent before step j . The k -th constraint is $x_k \cdot (w_k u_j) = p_k$, and the $(K + 1)$ -th constraint is $u_j \cdot (\sum_k p_k + \rho s_{j-1}) = s_j$. The verifier pins $u_j = 1$ through the public term Q_j , so u_j is not a free witness and the constraints compute exactly the displayed EMA update. At prefix $j - 1$, u_j , the new partial products, and s_j are absent; hence each future row has a structurally empty multiplicative side and output side as required by the gate discipline of Section 4.2. No

additional constraint is needed, so the circuit still has $K + 1$ constraints per step.

We set the decay factor $\rho = 1/10$ and assign sensor weights via inverse-variance weighting: sensor k (1-indexed) receives variance $\sigma_k^2 = k + 1$ and raw weight $1/\sigma_k^2$. The final weights are normalized so that $\sum_{k=1}^K w_k = 1 - \rho$. The genesis state is $s_0 = 0$. We instantiate at $K = 4, 6, 8$ sensors with step bounds $B \in \{64, 256, 1024\}$, representing monitoring windows of approximately 1 minute, 4 minutes, and 17 minutes at a 1 Hz sampling rate. This circuit-level configuration also determines the update arithmetic: each new step contributes $K + 1$ transition constraints, while the quotient update is computed over the extended degree- $B(K + 1)$ QAP domain. Thus, the algebraic update cost scales as $O(K + 1)$ scalar multiplications plus $O((K + 1)B \log((K + 1)B))$ polynomial arithmetic.

Table 2 reports per-step proving time at $B = 256$ and $B = 1024$, Fig. 2 plots cumulative proving cost at $B = 1024$, and Fig. 3 summarizes per-step verification time across all workloads and schemes. Crucially, our scheme achieves constant proving time regardless of step count, whereas Groth16 re-proving grows with the accumulated witness, as the next paragraphs detail.

CRS size With K sensors, each step of the fusion circuit produces $n = K + 1$ constraints. The CRS consists of $O((K + 1)B)$ group elements; for $K = 8$ and $B = 64$, this amounts to about 0.32 MB with compressed BN254 encodings (32 bytes per $\mathbb{G}_1$ element and 64 bytes per $\mathbb{G}_2$ element). For fixed $K = 8$, the same measurement gives about 1.27 MB for $B = 256$ and 5.08 MB for $B = 1024$. This footprint is fixed once the monitoring epoch is configured: increasing B enlarges the one-time setup/prover storage and CRS distribution cost, but it does not affect later proof size, per-proof communication, or verifier pairing work.

Per-step proving time Table 2 shows that our proving time is constant across all steps for a given bound B : for $K = 8$, it remains at ≈ 109 –113 ms with $B = 256$ and ≈ 323 –326 ms with $B = 1024$. The increase from $B = 256$ to $B = 1024$ reflects the larger extended QAP domain used in the quotient update, not dependence on the current step index j . Groth16 proving time, by contrast, grows with j : the padded assignment still drives an FFT over $\mathcal{C}_B$'s full evaluation domain, and only the multiexponentiations shrink with the sparsity of the padding. For $K = 8$, it rises from 359 ms at $j = 1$ to 864 ms at $j = 1024$, and already exceeds our $B = 1024$ cost at every measured step. Nova achieves a per-fold cost of ≈ 340 –359 ms beyond the first fold, depending on the configuration, which on this board is above our

per-step cost at both bounds; moreover, it only covers folding and excludes the SNARK compression.

Cumulative proving time Fig. 2 plots the cumulative proving cost of all steps up to step j , including the one-time setup of each scheme. Since our per-step cost is constant, the cumulative cost grows linearly in j , whereas Groth16 must re-prove the whole prefix at every step against the fixed $\mathcal{C}_B$ key, so its per-step cost rises with j and its cumulative cost grows superlinearly. Our initial cumulative cost exceeds Groth16's, because our one-time setup is the more expensive of the two, but the crossover arrives within a modest number of steps: for $B = 1024$, between $j = 32$ and $j = 64$ for all three sensor counts. At $B = 1024$ and $K = 8$, Groth16's cumulative cost reaches $2.0\times$ ours; the ratio is 1.9 – $2.0\times$ across the three sensor counts. Per step the same gap appears directly: at the final step with $B = 1024$, our prover is 2.6 – $2.7\times$ faster than Groth16 across all configurations, and it is faster at every measured step rather than only after a crossover. Nova's cumulative cost on the succinct-proof path, that is folding plus the one-time SNARK compression, starts above ours and stays above it at every measured step up to $j = 1024$ for all three sensor counts, since on this board its per-fold cost alone already exceeds our per-step update. Against Nova, our proof is $72\times$ smaller and verifies roughly $20\times$ faster. When a monitoring service demands that succinct proof at every step, the measured $K = 8$ fold and compression costs total ≈ 8.0 s at every step, roughly $25\times$ the proving time of our $B = 1024$ configuration at step 1024.

Verification time Our scheme verifies with only 3 pairings together with the state-commitment increment check, giving 11.6 ms verification at the final step across the sensor-fusion configurations (Fig. 3). Groth16 verifies with the same 3 pairings, but the fixed-CRS baseline must publish every reported state $s_1, \dots, s_B$ as a public input of $\mathcal{C}_B$, so its input-commitment multiexponentiation scales with B rather than with the state width: it costs 11.5–12.2 ms up to $j = 16$ and rises to 33.6–33.7 ms at $j = 1024$. Our verification performs the identical three pairings plus the $O(|s_j|)$ increment check, whose size depends only on the state width and not on B . Nova's verification takes 238–239 ms, roughly $20\times$ slower than our scheme on the sensor-fusion workload. As a result, both our scheme and Groth16 comfortably meet the real-time latency budget for edge verification.

Proof size Our scheme produces a proof of 4 group elements (3 in $\mathbb{G}_1$, 1 in $\mathbb{G}_2$), a constant 160 bytes independent of circuit size or step count; Groth16 proofs are

$2 \mathbb{G}_1 + 1 \mathbb{G}_2$, or 128 bytes. Nova’s compressed proof serializes to 11,520–12,512 bytes across the evaluated workloads, approximately 72–78× larger than ours.

6.3 Generality Across Workloads

To confirm that the sensor-fusion results of Section 6.2 are not an artifact of one circuit shape, we repeat the same comparison on two further workload classes: compact linear-algebra transitions (Hadamard products) and cryptographic or long sequential state transitions (a MiMC-style cube-permutation chain and a repeated-squaring chain).

Hadamard product The Hadamard product is a coordinate-wise vector update: for $\mathbf{a}, \mathbf{b} \in \mathbb{F}^d$, the product $\mathbf{a} \odot \mathbf{b}$ has i -th entry $a_i \cdot b_i$. In this benchmark, the state is a d -dimensional vector $\mathbf{s}_{\text{in}} \in \mathbb{F}^d$ and the transition input is a d -dimensional vector $\mathbf{t} \in \mathbb{F}^d$. Each step computes the Hadamard product $\mathbf{s}_{\text{out}} = \mathbf{s}_{\text{in}} \odot \mathbf{t}$, i.e., $s_{\text{out}}[i] = s_{\text{in}}[i] \cdot t[i]$ for $1 \leq i \leq d$. It yields exactly $n = d$ multiplicative constraints per step with no internal witness wires. Element-wise vector multiplication arises in verifiable stream analytics and integrity-protected aggregation, where repeated vector updates must be checked without replaying the whole transition history [34].

MiMC-style cube chain MiMC [40] is a SNARK-friendly block cipher whose round function consists of a cube map $x \mapsto x^3$. Our microbenchmark uses this cube-map pattern only. The state is a single field element $s_{\text{in}} \in \mathbb{F}$ and the transition input is a single field element $t \in \mathbb{F}$. Each step applies 90 rounds of a cube permutation: round r takes the previous round’s output w_{prev} (with $w_{\text{prev}} = s_{\text{in}}$ for $r = 1$) and computes

$$\begin{aligned} w_{\text{mix}} &= w_{\text{prev}} \cdot t, \\ w_{\text{sq}} &= w_{\text{mix}}^2, \\ w_{\text{cube}} &= w_{\text{sq}} \cdot w_{\text{mix}}. \end{aligned}$$

The last round’s w_{cube} is assigned as the state output s_{out} . Each round contributes 3 multiplicative constraints, producing $n = 270$ constraints per step.

Repeated squaring The state is a single field element $s_{\text{in}} \in \mathbb{F}$ and the transition input is a single field element $t \in \mathbb{F}$. Each step first mixes the state with the transition

input via

$$\begin{aligned} w_1 &= s_{\text{in}} \cdot t, \\ w_2 &= w_1^2, \\ &\vdots \\ w_n &= w_{n-1}^2. \end{aligned}$$

The last round’s w_n is assigned as the state output s_{out} . This produces exactly n multiplicative constraints per step. Repeated modular squaring underlies practical VDF constructions [41] and delay-based protocols [37].

For these workloads the CRS size and quotient-update domain scale as $O(Bd)$ for the Hadamard product ($d \in \{16, 32\}$) and as $O(Bn)$ for the cryptographic circuits ($n = 270$ for the MiMC-style chain, $n \in \{2^{10}, 2^{14}\}$ for repeated squaring); the largest repeated-squaring instance is the intended stress case for the $O(Bn \log(Bn))$ polynomial arithmetic in the proof update.

Across the Hadamard, MiMC, and repeated-squaring circuits, our proof stays fixed at 160 bytes and Groth16’s at 128 bytes regardless of n or B (Table 3), and verification reproduces the same flat, step-independent profile already established for sensor fusion in Section 6.2 (Fig. 3), at 11.6 ms for the MiMC-style and repeated-squaring chains and at 20.0–28.9 ms for the Hadamard products, whose increment check scales with the state width: this proof-size and verification behavior is a property of the scheme, not of one circuit.

Our per-step proving time again remains flat while Groth16’s grows with the accumulated witness and eventually exceeds ours at every measured step (Table 3), reproducing the crossover behavior seen for sensor fusion. Nova’s fold-only cost is much lower on these large circuits, while the reported succinct-proof path additionally requires SNARK compression. Nova remains lower on the largest per-step circuits because folding avoids the $O(Bn \log(Bn))$ update arithmetic, but it pays with larger proofs and slower verification.

7 Conclusion

Our approach introduces a linearly updatable QAP family whose algebraic structure allows proof updates through polynomial additions over sequentially composed circuits, avoiding recursive proof composition while matching Groth16’s pairing count, staying within one group element of its proof size, and supporting efficient incremental updates for bounded-step sequential computations. We evaluated the construction on sensor fusion, Hadamard products,

Table 2 Sensor-fusion workload: per-step proving time (ms) measured on the Raspberry Pi 5 testbed (4 cores). Each cell reads $K=4$ / $K=6$ / $K=8$ for the number of sensors, and B is the step bound for our scheme.

Step	Proving time (ms), $K = 4/6/8$			
	Ours ($B=256$)	Ours ($B=1024$)	Groth16	Nova [†]
1	68.3 / 94.6 / 112	214 / 251 / 325	235 / 273 / 359	136 / 137 / 136
4	69.0 / 94.5 / 113	215 / 251 / 326	240 / 277 / 367	340 / 342 / 345
16	68.8 / 94.3 / 113	215 / 251 / 326	250 / 287 / 380	354 / 356 / 354
64	69.5 / 94.9 / 113	216 / 251 / 326	278 / 325 / 420	354 / 356 / 354
256	66.2 / 92.5 / 109	215 / 251 / 326	341 / 408 / 540	355 / 356 / 354
1024	—	213 / 251 / 323	559 / 669 / 864	359 / 355 / 359

[†] Nova entries are fold-only

SNARK compression (≈ 7.6 – 7.7 s) for compressed verification.

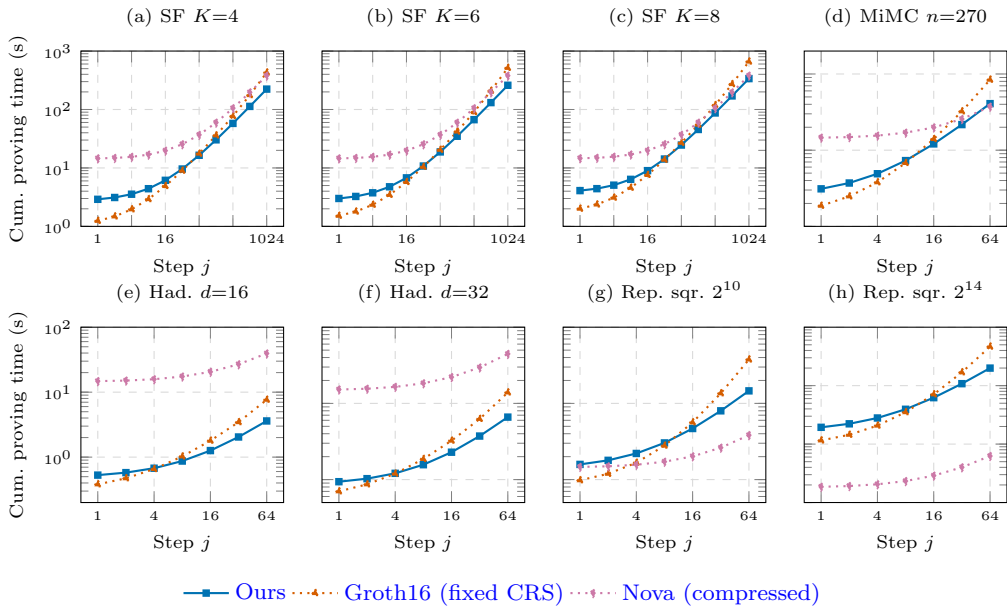


Fig. 2 Cumulative proving cost up to step j on the Raspberry Pi 5 testbed (4 cores), with every comparison bound-matched. It includes a one-time setup for all three schemes; Nova additionally includes its one-time SNARK compression.

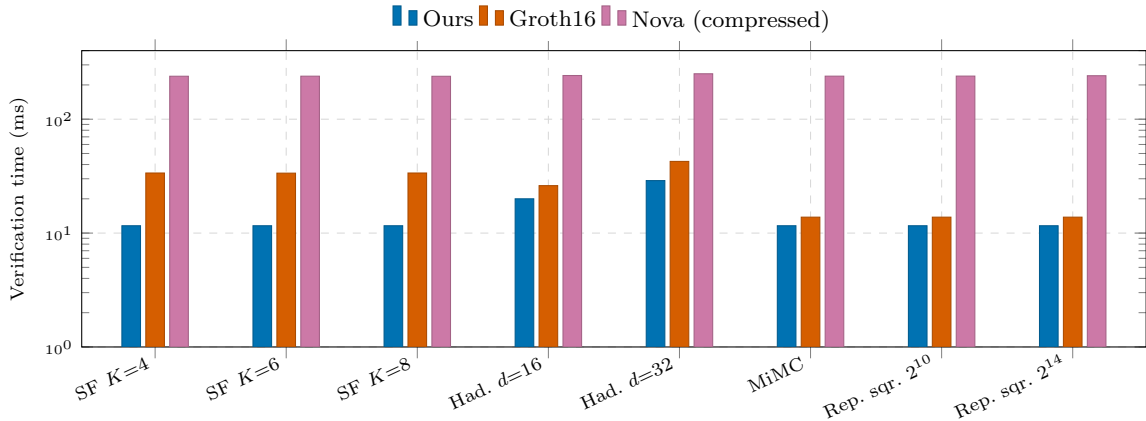


Fig. 3 Per-step verification time by workload and scheme (log scale) on the Raspberry Pi 5 testbed (4 cores): sensor fusion (SF) sensors, Hadamard products (Had.), the MiMC-style cube chain, and repeated squaring (Rep. sqr.).

Table 3 Per-step proving time (ms) across circuits, measured on the Raspberry Pi 5 testbed (4 cores). n is the number of constraints per step and B is the maximum steps for our scheme. Every proof is 160 bytes for our scheme and 128 bytes for Groth16 [6]; the compressed proof of Nova [15] is 11,520 bytes (MiMC-style cube chain, repeated squaring 2^{10}), 11,896 bytes (repeated squaring 2^{14}), 12,000 bytes (Hadamard $d=16$), and 12,512 bytes (Hadamard $d=32$).

Circuit	n	Step	Proving time (ms)			
			Ours ($B=16$)	Ours ($B=64$)	Groth16	Nova [†]
MiMC-style chain	270	1	240	598	592	140
		4	241	599	685	347
		16	241	598	989	358
		64	—	600	1,937	364
Hadamard product	16	1	20.2	48.1	91.1	155
		4	20.6	48.5	91.5	379
		16	20.8	49.5	98.3	389
		64	—	50.7	153	393
Hadamard product	32	1	35.8	88.9	165	194
		4	36.3	89.5	166	458
		16	36.5	91.2	175	468
		64	—	92.0	278	470
Repeated squaring	2^{10}	1	774	2,063	2,041	147
		4	776	2,066	2,535	365
		16	776	2,068	4,021	376
		64	—	2,071	9,202	374
Repeated squaring	2^{14}	1	9,090	28,536	27,871	304
		4	9,128	28,603	32,999	681
		16	9,143	28,628	50,558	697
		64	—	28,686	113,106	698

[†] Nova entries are fold-only; the reported succinct path adds one-time SNARK compression (≈ 7.6 – 8.8 s) for compressed verification.

a MiMC-style cube chain, and repeated squaring, with the main comparison of Section 6 collected on a four-core gateway-class Raspberry Pi 5. The resulting proof size, proving time, and verification numbers are summarized in the abstract and detailed in Section 6. Our scheme is accordingly not a universal replacement for existing verifiable-computation tools: Groth16 remains preferable for one-shot proofs or very short sequences, where our per-step and update machinery is unnecessary overhead, and folding schemes such as Nova remain more appropriate for unbounded computations in which proof size and verification latency are not the binding constraints, at the cost of a larger fold-only proof requiring an additional compression step.

Limitations and Further Work The bound B is fixed at setup and per-step arithmetic costs $O(Bn \log Bn)$ rather than the $O(n)$ of folding, so B is a deployment unit chosen conservatively: at 1 Hz it buys a one-minute window at $B = 64$ and seventeen minutes at $B = 1024$, and an unpredictable horizon must over-provision B or

stop at step B . Restarting with a fresh genesis state does not close that gap, since nothing in the relation binds consecutive executions and an adversary controlling the second prover may pick its genesis state freely; closing it would require binding each epoch’s genesis state as a public input of the next epoch’s relation across the CRS boundary, left to future work, so the results of Section 5 are proved for a single bounded execution only, with no claim about multi-epoch security. The verifier’s stored commitment D_{j-1} must also be authentic: Appendix B.4 shows an adversary free to choose the verifier state shifting it by a public group operation so an honest proof is accepted for a false state, so **vst** needs durable tamper-evident storage or a trusted audit log, and a compromised log voids the guarantee.

Beyond authenticated epoch boundaries, future work includes adding zero knowledge or another privacy layer, hybrid constructions combining our update mechanism with folding for unbounded computations, evaluation on microcontroller-class hardware, and extending the evaluation to further application classes. Sequential model inference is

the one we find most promising: a recurrent neural network (RNN) propagates a hidden state from one time step to the next [42], which instantiates $\mathcal{ST}$ with the hidden state as Σ and one recurrent step as StUp , so a client could check an outsourced inference from the final proof instead of re-executing the model. Two obstacles decide whether this is practical, and both are open. The activation must be arithmetized over $\mathbb{F}$, so what a proof certifies is the finite-field circuit rather than the real-valued model, and the constraint cost of that approximation is what would determine the achievable sequence length.

Acknowledgements This work was supported in part the AI Seoul Tech Research Support Program of the Seoul Future Foundation. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Data availability. The data supporting the findings of this study are the performance measurements reported in Section 6; no external datasets were analyzed. These measurements can be regenerated on the platforms described there using the benchmark scripts available at <https://github.com/heewon-chung/libuvc>.

Author contributions. CRediT (Contributor Roles Taxonomy, <https://credit.niso.org>):

- **Conceptualization:** Heewon Chung, Hyeonbum Lee
- **Data curation:** Heewon Chung, Hyeonbum Lee
- **Formal analysis:** Hyeonbum Lee
- **Funding acquisition:** Hyeonbum Lee
- **Investigation:** Heewon Chung
- **Methodology:** Heewon Chung
- **Project administration:** Heewon Chung
- **Resources:** Hyeonbum Lee
- **Software:** Heewon Chung
- **Supervision:** Heewon Chung
- **Validation:** Hyeonbum Lee
- **Visualization:** Heewon Chung
- **Writing – original draft:** Heewon Chung
- **Writing – review & editing:** Hyeonbum Lee

References

1. J. Hong, Y.G. Hong, X. de Foy, M. Kovatsch, E. Schooler, D. Kutscher. Internet of Things (IoT) edge challenges and functions. RFC 9556, IETF (2024). <https://datatracker.ietf.org/doc/html/rfc9556>
2. M. Frustaci, P. Pace, G. Aloï, G. Fortino, IEEE Internet of Things Journal **5**(4), 2483 (2018)
3. R. Teti, K. Jemielniak, G.E. O'Donnell, D. Dornfeld, CIRP Annals **59**(2), 717 (2010). DOI 10.1016/j.cirp.2010.05.010
4. N. Ghosh, Y.B. Ravi, A. Patra, S. Mukhopadhyay, S. Paul, A.R. Mohanty, A.B. Chattopadhyay, Mechanical Systems and Signal Processing **21**(1), 466 (2007). DOI 10.1016/j.ymssp.2005.10.010
5. R. Gennaro, C. Gentry, B. Parno, in *Advances in Cryptology – CRYPTO 2010* (Springer, 2010), pp. 465–482
6. J. Groth, in *Advances in Cryptology – EUROCRYPT 2016* (Springer, 2016), pp. 305–326
7. B. Bünz, B. Fisch, A. Szepieniec, in *Advances in Cryptology – EUROCRYPT 2020* (Springer, 2020), pp. 677–706
8. A. Gabizon, Z.J. Williamson, O. Ciobotaru. PLONK: permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019)
9. E. Ben-Sasson, I. Bentov, Y. Horesh, M. Riabzev, in *Advances in Cryptology – CRYPTO 2019* (Springer, 2019), pp. 701–732
10. S.T.V. Setty, in *Advances in Cryptology – CRYPTO 2020* (Springer, 2020), pp. 704–737
11. T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, D. Song, in *Advances in Cryptology – CRYPTO 2019* (Springer, 2019), pp. 733–764
12. P. Valiant, in *Theory of Cryptography – TCC 2008* (Springer, 2008), pp. 1–18
13. E. Ben-Sasson, A. Chiesa, E. Tromer, M. Virza, in *Advances in Cryptology – CRYPTO 2014* (Springer, 2014), pp. 276–294
14. S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, H. Wu, in *IEEE Symposium on Security and Privacy 2020* (IEEE, 2020), pp. 947–964
15. A. Kothapalli, S. Setty, I. Tzialla, in *Advances in Cryptology – CRYPTO 2022, LNCS*, vol. 13510 (Springer, 2022), LNCS, vol. 13510, pp. 693–723
16. R. Gennaro, C. Gentry, B. Parno, M. Raykova, in *Advances in Cryptology – EUROCRYPT 2013* (Springer, 2013), pp. 626–645
17. H. Lipmaa, in *Advances in Cryptology – CRYPTO 2024* (Springer, 2024)
18. T. Zhang, Y. Ouyang, Y. Zhang. Dynark: Making Groth16 dynamic. Cryptology ePrint Archive, Paper 2025/1897 (2025). <https://eprint.iacr.org/2025/1897>
19. J. Bonneau, I. Meckler, V. Rao, E. Shapiro. Mina: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Report 2020/352 (2020)
20. A. Kothapalli, S. Setty, in *Advances in Cryptology – CRYPTO 2024* (Springer, 2024), pp. 46–78

21. A. Kothapalli, S. Setty. SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758 (2022)
22. J. Zhao, S. Setty, W. Cui, G. Zaverucha, in *IEEE Symposium on Security and Privacy 2025* (IEEE, 2025)
23. D. Boneh, B. Chen, in *Advances in Cryptology – ASIACRYPT 2025* (Springer, 2025)
24. C.O. Sakwa, A.O. Anyembe, F. Li, Information Sciences (2025). DOI 10.1016/j.ins.2025.122144
25. H. Lee, J.H. Seo. On the security of Nova recursive proof system. Cryptology ePrint Archive, Report 2024/232 (2024)
26. A. Chiesa, E. Tromer, in *Innovations in Computer Science – ICS 2010* (2010), pp. 310–331
27. B. Bünz, A. Chiesa, P. Mishra, N. Spooner, in *Theory of Cryptography – TCC 2020* (Springer, 2020), pp. 1–38
28. G. Ramezan, E. Meamari. zk-IoT: Securing the Internet of Things with zero-knowledge proofs on blockchain platforms. arXiv:2402.08322 (2024). <https://arxiv.org/abs/2402.08322>
29. J. Swalens, L. Hoste, E. Heydari Beni, L. Trappeniers, in *Proceedings of the 25th International Middleware Conference* (ACM, 2024), pp. 252–265. DOI 10.1145/3652892.3700763
30. D. Fiore, I. Tucker, in *ACM Conference on Computer and Communications Security (CCS) 2022* (ACM, 2022)
31. O. Faraj, D. Megias, J. Garcia-Alfaro, ACM Computing Surveys (2025). DOI 10.1145/3718735
32. T.M. Duong, S. Kwon, IEEE Internet of Things Journal **12**(23), 50143 (2025). DOI 10.1109/JIOT.2025.3612031
33. K. Peng, J. Xie, L. Wei, J. Hu, X. Hu, T. Deng, M. Hu, IEEE Internet of Things Journal **11**(20), 33847 (2024). DOI 10.1109/JIOT.2024.3434583
34. S. Guan, Y. Cao, Y. Zhang, IEEE Internet of Things Journal (2024). DOI 10.1109/JIOT.2024.3487154
35. T. Wu, W. Wang, C. Zhang, W. Zhang, L. Zhu, K. Gai, H. Wang, IEEE Internet of Things Journal **10**(6), 5461 (2023). DOI 10.1109/JIOT.2022.3222453
36. H. Xia, C.N.A. Cobblah, Q. Xia, J. Gao, IEEE Internet of Things Journal **12**(15), 29811 (2025). DOI 10.1109/JIOT.2025.3568868
37. R. Xu, Y. Chen, in *2022 IEEE International Conference on Blockchain* (IEEE, 2022), pp. 348–355. DOI 10.1109/Blockchain55522.2022.00054
38. SCIPR Lab. libsnark: A C++ library for zkSNARK proofs. <https://github.com/scipr-lab/libsnark> (2024)
39. Microsoft Research. Nova: High-speed recursive arguments from folding schemes (Rust crate nova-snark, version 0.58.0). <https://github.com/microsoft/Nova> (2025)
40. M. Albrecht, L. Grassi, C. Rechberger, A. Roy, T. Tiessen, in *Advances in Cryptology – ASIACRYPT 2016* (Springer, 2016), pp. 191–219
41. D. Boneh, J. Bonneau, B. Bünz, B. Fisch, in *Advances in Cryptology – CRYPTO 2018, LNCS*, vol. 10991 (Springer, 2018), *LNCS*, vol. 10991, pp. 757–788. DOI 10.1007/978-3-319-96884-1_25
42. I.J. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, 2016)
43. M. Abe, S. Fehr, in *Advances in Cryptology – CRYPTO 2007* (Springer, 2007), pp. 439–457
44. G. Fuchsbauer, E. Kiltz, J. Loss, in *Advances in Cryptology – CRYPTO 2018* (Springer, 2018), pp. 33–62. EPrint 2017/620
45. K. Baghery, M. Kohlweiss, J. Siim, M. Volkhov, in *Financial Cryptography and Data Security – FC 2021* (Springer, 2021), pp. 457–475. EPrint 2020/811

A Preliminaries and Background

A.1 Knowledge Soundness, the Algebraic Group Model, and the q -DLog Assumption

The workflow above relies on the following notions. An index set I is a set of consecutive positive integers starting from 1, and $\mathbb{F}$ is a finite field of prime order p . We write $f(X) = \text{Lagrange}_r(\mathbf{a})$ for the polynomial of degree at most $n-1$ that Lagrange interpolation recovers from the n distinct points $(r_i, a_i) \in \mathbb{F}^2$, so that $f(r_i) = a_i$ for $1 \leq i \leq n$, where $\mathbf{r} = (r_1, \dots, r_n)$ and $\mathbf{a} = (a_1, \dots, a_n)$.

We require the usual perfect completeness, namely that an honestly generated proof for a valid statement-witness pair always verifies; the updatable analogue is stated as Definition 3.

Definition 5 (Knowledge Soundness). We say that $(\text{Setup}, \text{Prove}, \text{Verify})$ satisfies knowledge soundness if for all non-uniform polynomial-time algebraic adversaries $\mathcal{A}$ there exists a non-uniform polynomial-time extractor $\chi_{\mathcal{A}}$ such that

$$\Pr \left[\begin{array}{l} (\phi, w) \notin \mathcal{R} \\ 1 \leftarrow \text{Verify}(\mathcal{R}, \sigma, \phi, \pi) \end{array} \middle| \begin{array}{l} \mathcal{R} \leftarrow \text{RGen}(1^\lambda); \\ \sigma \leftarrow \text{Setup}(\mathcal{R}); \\ ((\phi, \pi); w) \leftarrow (\mathcal{A} \parallel \chi_{\mathcal{A}})(\mathcal{R}, \sigma) \end{array} \right] \leq \text{negl}(\lambda),$$

where the notation $(y; z) \leftarrow (\mathcal{A} \parallel \chi_{\mathcal{A}})(x)$ means that $\mathcal{A}$ on input x outputs y and $\chi_{\mathcal{A}}$ on the same input (including random coins) outputs z .¹

¹ This notation is used by Abe and Fehr [43] and by Groth [6].

Definition 6 (Algebraic Group Model (AGM)). Following Fuchsbaauer et al. [44], an adversary $\mathcal{A}$ is *algebraic* (in the bilinear group model) if, whenever it outputs a group element $[Y]_1 \in \mathbb{G}_1$, it also outputs coefficients $c_1, \dots, c_k \in \mathbb{F}$ with $Y = \sum_{i=1}^k c_i z_i$, expressing Y in the group elements $[z_1]_1, \dots, [z_k]_1$ it has received from the CRS and oracle queries. The same requirement applies to $\mathbb{G}_2$.

Definition 7 (q -DLog Assumption). For a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ of prime order p with generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$, the q -DLog assumption states that no probabilistic polynomial-time algorithm, given

$$(\{[x^i]_1\}_{i=0}^q, \{[x^i]_2\}_{i=0}^q) \quad \text{for } x \leftarrow \mathbb{F}_p^*,$$

outputs x except with negligible probability in λ . Writing $N = Bn \geq 2$, we assume $q = 2N - 1$ in both source groups, which is what the extraction argument of Section 5.4 needs: the powers up to $2N - 1$ let a reduction simulate the whole CRS from a q -DLog challenge, and they bound the degree of the polynomial whose roots yield the trapdoor.

A.2 Reduction from Arithmetic Circuits to Quadratic Arithmetic Programs

We recall the standard reduction [16] only to the extent later sections use it. An arithmetic circuit $\mathcal{C}$ with fan-in two gates over $\mathbb{F}$ yields n quadratic constraints in the wire values $a_i \in \mathbb{F}$ ($a_0 = 1$, $i \in I$ an index set), which interpolation over n distinct points $r_1, \dots, r_n$ turns into a single congruence modulo $P(X) = \prod_{q=1}^n (X - r_q)$:

$$\begin{aligned} \sum_{i \in \{0\} \cup I} a_i u_i(X) \cdot \sum_{i \in \{0\} \cup I} a_i v_i(X) \\ \equiv \sum_{i \in \{0\} \cup I} a_i w_i(X) \pmod{P(X)}. \end{aligned} \quad (3)$$

The resulting QAP is the tuple

$$(\mathbb{F}, aux, I^{\text{io}}, \{u_i(X), v_i(X), w_i(X)\}_{i \in \{0\} \cup I}, P(X))$$

, $\deg u_i, v_i, w_i < n$, $\deg P = n$, defining a relation whose statement assigns $I^{\text{io}} \subset I$ and whose witness assigns $I^{\text{wt}} := I \setminus I^{\text{io}}$, a pair being in the relation when both, with $a_0 = 1$, satisfy Eq. (3).

A.3 Verifiable Computation for QAP

This subsection recalls the single-step scheme our construction starts from; it is Groth16 with the zero-knowledge randomization removed, and is included here because the later sections build on its exact form rather than because it is new. We present a verifiable computation scheme for a QAP generator, taking the index sets to be $I^{\text{io}} = [1, \ell]$ and $I^{\text{wt}} = [\ell + 1, m]$, so that the relation of Appendix A.2 reads: $\phi = (a_1, \dots, a_\ell)$, $w = (a_{\ell+1}, \dots, a_m)$, and together with $a_0 = 1$ they satisfy Eq. (3) for polynomials $u_i(X), v_i(X), w_i(X)$ of degree $n - 1$ and a degree- n modulus $P(X)$. For a valid statement-witness pair, there exists $h(X)$ of degree $n - 2$ satisfying

$$\sum_{i=0}^m a_i u_i(X) \cdot \sum_{i=0}^m a_i v_i(X) = \sum_{i=0}^m a_i w_i(X) + h(X)P(X).$$

Our VC scheme is a variant of Groth's zk-SNARK [6] with randomization removed from the proving algorithm. This removal is intentional: while randomization is essential for zero-knowledge, it is unnecessary for verifiable computation and, more importantly, it introduces non-linearity that would prevent the additive proof update mechanism we develop in Section 5.2. Specifically, Groth16 uses random blinding factors $r, s \in \mathbb{F}$ so that $A = \alpha + \sum a_i u_i(x) + r\delta$ and $B = \beta + \sum a_i v_i(x) + s\delta$; the cross-term $rs\delta^2$ in the product $A \cdot B$ makes C depend non-linearly on the randomness, preventing additive decomposition of proof components across steps.

Our verifiable computation scheme Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be an asymmetric (Type-3) bilinear group of prime order p , where $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a non-degenerate bilinear map and no efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ is assumed. The field $\mathbb{F}$ used in QAP constructions is identified with $\mathbb{F}_p$, the scalar field of the bilinear group. We write $[a]_1 = g^a$, $[b]_2 = h^b$, and $[a]_1 \cdot [b]_2 = [ab]_T$ for generators $g \in \mathbb{G}_1$, $h \in \mathbb{G}_2$.

Write $L_i(x) = \beta u_i(x) + \alpha v_i(x) + w_i(x)$. Both VC.Setup and VC.Verify are exactly Groth16's, so we state them as a difference rather than repeat them. VC.Setup($\mathcal{R}$) samples $\alpha, \beta, \gamma, \delta, x \leftarrow \mathbb{F}^*$ and outputs the Groth16 CRS $\sigma = ([\sigma_1]_1, [\sigma_2]_2)$: the powers $\{x^i\}_{i=0}^{n-1}$ in both groups, the quotient track $\{x^i P(x)/\delta\}_{i=0}^{n-2}$, the statement track $\{L_i(x)/\gamma\}_{i=0}^\ell$ and the witness track $\{L_i(x)/\delta\}_{i=\ell+1}^m$ in $\mathbb{G}_1$, together with α, β, δ in $\mathbb{G}_1$ and β, γ, δ in $\mathbb{G}_2$. VC.Verify($\mathcal{R}, \sigma, (a_1, \dots, a_\ell), \pi$) parses $\pi = ([A]_1, [B]_2, [C]_1)$ and accepts iff the Groth16 equation $[A]_1 \cdot [B]_2 = [\alpha]_1 \cdot [\beta]_2 + \sum_{i=0}^\ell a_i [L_i(x)/\gamma]_1 \cdot [\gamma]_2 + [C]_1 \cdot [\delta]_2$ holds. Appendix B.5 writes out the updatable CRS, which differs from σ only in the index partition and in the polynomial tracks the update needs. The single difference is in the prover, which sets the Groth16 blinding factors to $r = s = 0$ and outputs the deterministic $\pi = ([A]_1, [B]_2, [C]_1)$ given by

$$\begin{aligned} A &= \alpha + \sum_{i=0}^m a_i u_i(x) \\ B &= \beta + \sum_{i=0}^m a_i v_i(x) \\ C &= \sum_{i=\ell+1}^m a_i \frac{L_i(x)}{\delta} + \frac{h(x)P(x)}{\delta} \end{aligned}$$

The prover does not know the trapdoor x ; it computes $[A]_1$ using the CRS elements $\{[x^i]_1\}_{i=0}^{n-1}$ as a basis for polynomial evaluation, and similarly for $[B]_2$ and $[C]_1$ using the corresponding CRS elements in $\mathbb{G}_1$ and $\mathbb{G}_2$.

This base scheme is a verifiable computation scheme in the sense of Section 3.3. Its perfect completeness is a direct calculation, and its knowledge soundness under Definitions 6 and 7 is recalled in Section 5.4.

Remark 2 (Proof malleability). Removing the blinding factors r and s makes the honest prover deterministic for a fixed CRS and witness, but it does not change setup, verification, or the set of adversarial proofs accepted by the Groth16 verifier. Therefore the Groth16 knowledge extractor applies unchanged; the restriction removes honest-prover randomness and zero knowledge. Because the CRS and verifier are unchanged, the public Groth16 re-randomization procedure of Baghery et al. [45, Fig. 1] remains applicable to any valid proof. We consequently claim neither proof uniqueness nor non-malleability. This is consistent with Definition 4,

which asks for extraction from an accepting proof and not for uniqueness of the accepting proof. In integrity-oriented deployments, non-zero-knowledge is acceptable when the verified inputs, state values, or model parameters are public, or when the deployment only needs integrity relative to values committed by the surrounding system.

B Formal Definitions and CRS Specification

B.1 Prefix Terminology

The proofs below and in Appendix C use the following derived terms, introduced with the relation in Section 5.1.

We use *prefix* throughout for the length- j initial segment of a bounded execution and fix the derived terms here. The *prefix circuit* is C_j , the first j copies of the transition circuit, and its index set is $I_j \subseteq I_B$; a wire is *prefix-visible* at j when its index lies in I_j and *future* otherwise. A *prefix assignment* is a map $(a_i)_{i \in I_j}$, and we identify it with the assignment over I_B obtained by setting $a_i = 0$ on future wires, which is the form Eq. (2) uses; the *prefix statement* and *prefix witness* are its restrictions to the statement and witness coordinates, that is, the ϕ and w of a pair in $\mathcal{R}_{ST}$ at step j . A gate is a *prefix row* when it belongs to one of the first j copies, so its root is r_q with $q \leq jn$, and *prefix-restricted* describes a future gate with its future wires deleted. Finally, the composed RICS is *prefix-causal* when no prefix row carries a coefficient on a future wire,

$$u_{i,q} = v_{i,q} = w_{i,q} = 0 \quad \text{whenever } i \in I_B \setminus I_j \text{ and } q \leq jn,$$

which holds by construction here, since a gate of copy k touches only wires of copy k and the state carried from copy $k-1$. Prefix-causality is what allows a full-bound assignment to be cut back to C_j , and it is used only for that; the gate discipline of Section 4.2 constrains the complementary case, a prefix-visible wire appearing in a future gate, and is what makes an honest prefix assignment satisfy the congruence at the later roots.

B.2 Perfect Completeness Experiment

Definition 3 states perfect completeness in prose; the formal experiment is as follows. A UVC scheme has *perfect completeness* if for every $\mathcal{ST}$, every bound B , every $J \in [1, B]$, and every $\mathbf{t} \in T^J$,

$$\Pr \left[\bigwedge_{j=1}^J b_j = 1 \mid \begin{array}{l} \mathcal{R}_{ST} \leftarrow \text{RGen}_{ST}(1^\lambda, B, ST); \\ \sigma \leftarrow \text{Setup}(\mathcal{R}_{ST}); \\ \text{for } j = 1, \dots, J : \\ \quad (\pi_j, \zeta_j) \leftarrow \text{Prove}(\mathcal{R}_{ST}, \sigma, j, \phi_j); \\ \quad \omega_j, \pi_{j-1}, \zeta_{j-1}; \\ \quad (b_j, \text{vst}_j) \leftarrow \\ \quad \quad \text{Verify}(\mathcal{R}_{ST}, \sigma, j, \phi_j, \pi_j, \text{vst}_{j-1}) \end{array} \right] = 1,$$

where $(\phi_j, \omega_j)_{j \in [1, J]}$ is the honest execution on $\mathbf{t}$ (Section 5.1).

B.3 Updatable Knowledge Soundness Experiment

Definition 4 states updatable knowledge soundness in prose; we first record why the experiment is shaped as it is.

Letting the adversary produce the earlier proofs, and not only the last one, is what gives the notion its content. Had the

prefix been generated honestly, the assignment underlying it would already be known to the experiment, and the coordinates that extraction had to supply would be only those of the last copy; the guarantee would then say nothing about a prover that misbehaved earlier and covered its tracks. With every proof adversarial, the algebraic representations of all of them are available, and a single extraction at the end recovers the entire assignment, so the accepting run certifies one history rather than a last step glued to a trusted past. Fixing the statements in advance costs nothing, since *Prove* and *Verify* are deterministic and public and the adversary can compute intermediate values itself; it also matches the fact that statements are public data settled with the relation.

The formal experiment is as follows. A UVC scheme has *updatable knowledge soundness* if for every step index $j \in [1, B]$ and every non-uniform polynomial-time algebraic adversary $\mathcal{A}$ there is a non-uniform polynomial-time extractor $\chi_{\mathcal{A}}$ such that

$$\Pr \left[\begin{array}{l} b_1 = \dots = b_j = 1 \wedge \\ ((\phi_j, w) \notin \mathcal{R}_{ST} \vee \\ \exists i < j : w|_{\Delta I_i^{\text{st}}} \neq s_i) \end{array} \mid \begin{array}{l} \text{pp} \leftarrow \text{Setup}(\text{RGen}_{ST}(1^\lambda, B, ST)); \\ (j, (\phi_i)_{i \leq j}, \text{aux}_0) \leftarrow \mathcal{A}(\text{pp}); \\ \text{for } i = 1, \dots, j : \\ \quad (\pi_i, \text{aux}_i) \leftarrow \mathcal{A}(\text{aux}_{i-1}); \\ \quad (b_i, \text{vst}_i) \leftarrow \text{Verify}(i, \phi_i, \pi_i, \text{vst}_{i-1}); \\ w \leftarrow \chi_{\mathcal{A}}((\phi_i, \pi_i)_{i \leq j}, r_{\mathcal{A}}) \end{array} \right] \leq \text{negl}(\lambda),$$

where $r_{\mathcal{A}}$ are the random coins of $\mathcal{A}$, vst_0 is the scheme's initial verifier state, $\Delta I_i^{\text{st}} := I_i^{\text{st}} \setminus I_{i-1}^{\text{st}}$, and s_i denotes the state component of $\phi_i = (t_1, i, s_i)$ (Section 5.1).

B.4 Why the Verifier State Must Come from Verify

Definition 4 requires the verifier state at step j to be the state returned by *Verify* at the previously accepted step, rather than an arbitrary group element supplied by the adversary. This restriction is necessary for our construction.

Recall that the verifier state carried into step j is $\text{vst}_{j-1} = [D_{j-1}]_1$. Since $\text{vst}_0 = [0]_1$ and every accepted step satisfies the state-commitment increment check, chaining these checks gives

$$\text{vst}_{j-1} = [D_{j-1}]_1 = \left[\sum_{i \in I_{j-1}^{\text{st}}} \hat{s}_i \frac{L_i(x)}{\gamma} \right]_1, \quad (4)$$

where the coefficients $\hat{s}_i$ are the coordinates of the states reported in the previously accepted statements $\phi_1, \dots, \phi_{j-1}$. Thus, an honestly carried verifier state has support only on the state coordinates already accepted before step j .

We next show what goes wrong if the adversary is allowed to choose vst_{j-1} arbitrarily. Suppose that

$$\pi_j = ([A_j]_1, [B_j]_2, [C_j]_1, [D_j]_1)$$

is a valid proof for $\phi_j = (t_1, j, s_j)$ with an authentic previous state $[D_{j-1}]_1$. Let $s'_j \neq s_j$ be another reported state, and let a_i and a'_i denote the corresponding coordinates on $I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}$. An adversary that may choose the verifier state can instead provide

$$[D'_{j-1}]_1 := [D_{j-1}]_1 + \sum_{i \in I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}} (a_i - a'_i) \left[\frac{L_i(x)}{\gamma} \right]_1. \quad (5)$$

When verifying the false statement $\phi'_j = (t_1, j, s'_j)$, the verifier then computes

$$\begin{aligned} [\tilde{D}'_j]_1 &= [D'_{j-1}]_1 + \sum_{i \in I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}} a'_i \left[\frac{L_i(x)}{\gamma} \right]_1 \\ &= [D_{j-1}]_1 + \sum_{i \in I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}} a_i \left[\frac{L_i(x)}{\gamma} \right]_1 = [D_j]_1. \end{aligned}$$

Hence the state-commitment increment check still accepts.

Moreover, the main pairing equation is unchanged. The reported state s_j does not appear in the public term Q_j ; it is bound to the proof through $[D_j]_1$ and the increment check. Since the adversary leaves π_j , and in particular $[D_j]_1$, unchanged, the same pairing equation that accepted ϕ_j also accepts ϕ'_j under the shifted verifier state of Eq. (5).

The shifted state of Eq. (5) cannot arise from a sequence of accepted executions starting from vst_0 , because it introduces a compensating contribution on the newest state slice $I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}$ and therefore does not have the form of Eq. (4). Consequently, the requirement that vst_{j-1} be obtained from a previously accepted execution of **Verify** is necessary for the soundness of our construction.

B.5 Full CRS of UVC.Setup

Section 5.2 instantiates the wire classes of Eq. (1): I_B^{io} holds the anchor t_1 and, for each copy $k \in [1, B]$, a *step-local unit wire* unit_k prescribed to be 1 once copy k is visible; I_B^{st} holds the state outputs $s_1, \dots, s_B$; and I_B^{wt} holds the remaining wires, the transition inputs $t_2, \dots, t_B$ and all internal wires. This subsection records the three conventions that instantiation relies on and then gives the full CRS. First, the state output wire of copy k is the state input wire of copy $k+1$; no separate copy constraint is introduced, so each s_k occupies exactly one index. Second, the genesis state s_0 is not a wire: it is hardwired as a constant in the constraints of copy 1, consistently with $\text{vst}_0 = [0]_1$ and $I_0^{\text{st}} = \emptyset$. Third, only the anchor t_1 is public; the later inputs $t_2, \dots, t_B$ are free witnesses, which is why Theorem 2 certifies reachability through *some* witness input sequence rather than through a specific one.

Section 4.2 states the gate discipline this arrangement realizes; Appendix D works the switch-value argument out on a concrete circuit, together with the verifier's step-2 computation.

Each state-commitment wire follows the designated state-output-row condition in Section 4.2. Let $L_i(x) = \beta \hat{u}_i(x) + \alpha \hat{v}_i(x) + \hat{w}_i(x)$. The CRS places $L_i(x)/\gamma$ on the two public classes and $L_i(x)/\delta$ on the free witnesses; note that I_B^{io} and I_B^{st} share the γ track and are separated only by the verifier's access pattern. Output $\sigma = ([\sigma_1]_1, [\sigma_2]_2)$:

$$\begin{aligned} \sigma_1 &= \left(\alpha, \beta, \delta, \left\{ \frac{x^i P_B(x)}{\delta} \right\}_{i=0}^{Bn-2}, \{ \hat{u}_i(x), \hat{v}_i(x), \hat{w}_i(x) \}_{i \in I_B}, \right. \\ &\quad \left. \left\{ \frac{L_i(x)}{\gamma} \right\}_{i \in \{0\} \cup I_B^{\text{io}} \cup I_B^{\text{st}}}, \left\{ \frac{L_i(x)}{\delta} \right\}_{i \in I_B^{\text{wt}}} \right) \\ \sigma_2 &= \left(\beta, \gamma, \delta, \{ \hat{u}_i(x), \hat{v}_i(x), \hat{w}_i(x) \}_{i \in I_B} \right) \end{aligned}$$

C Proofs

C.1 Proofs for the Linearly Updatable QAP Family

This section proves the two statements of Section 4.2; we first fix the notion of extension they use.

Definition 8. Given a tuple of polynomials

$$\mathcal{P}_I := (\{u_i(X), v_i(X), w_i(X)\}_{i \in \{0\} \cup I}, P(X))$$

for some index set I , we say that

$$\hat{\mathcal{P}}_{\hat{I}} := (\{\hat{u}_i(X), \hat{v}_i(X), \hat{w}_i(X)\}_{i \in \{0\} \cup \hat{I}}, \hat{P}(X))$$

for some index set $\hat{I}$ is an *extension* of $\mathcal{P}_I$ if $I \subset \hat{I}$ and for $i \in I$:

$$\begin{aligned} \hat{u}_i(X) &\equiv u_i(X) \bmod P(X), \\ \hat{v}_i(X) &\equiv v_i(X) \bmod P(X), \\ \hat{w}_i(X) &\equiv w_i(X) \bmod P(X), \\ \hat{P}(X) &\equiv 0 \bmod P(X) \text{ and } \hat{P}(X) \neq 0. \end{aligned}$$

We extend the terminology to QAPs: a QAP is an *extension* of another when its polynomial tuple is.

The first statement below justifies working with the extended QAP in place of the naïve one and is not used later; the second, Lemma 1, makes the prover's quotient exist at every step and is the one ingredient that Appendix C.2.1 takes from this section.

Lemma 3 (Extension soundness for the family). *Let $\hat{\mathcal{P}}_I$ be an extension of $\mathcal{P}_I$. Then:*

1. *every assignment satisfying the QAP of $\hat{\mathcal{P}}_I$ satisfies the QAP of $\mathcal{P}_I$;*
2. *for $1 \leq \ell < j$, $P_\ell(X)$ divides $P_j(X)$, and $\mathcal{P}_{I_B}$ is an extension of $\mathcal{P}_{I_j}$ for every $j \in [1, B-1]$, so item 1 applies throughout the family.*

Proof. Item 1 follows by reducing the congruence of $\hat{\mathcal{P}}_I$ modulo $\hat{P}(X)$ further modulo $P(X)$: by Definition 8 it becomes Eq. (3), so every assignment satisfying the extended QAP satisfies the original one. For item 2, the divisibility $P_\ell \mid P_j$ is immediate from the construction of the r_q , and $I_j \subset I_B$; moreover $u_i(X)$ interpolates at $(r_q, u_{i,q})$ for $q \in [1, jn]$ while $\hat{u}_i(X)$ interpolates at those points and more, so $\hat{u}_i(X) - u_i(X)$ vanishes at every such r_q , giving $\hat{u}_i(X) \equiv u_i(X) \bmod P_j(X)$, and the same for v_i and w_i . $\square$

Proof of Lemma 1. Let $F_j(X)$ denote the difference between the left- and right-hand sides above. Since $P_B(X) = \prod_{q=1}^{Bn} (X - r_q)$ has distinct roots, it suffices to prove $F_j(r_q) = 0$ for every $q \in [1, Bn]$. If $q \leq jn$, then gate q belongs to $\mathcal{C}_j$ and all of its active wires are included in the satisfying assignment, so $F_j(r_q) = 0$ follows from the ordinary RICS constraint for that gate. If $q > jn$, then $F_j(r_q)$ is the same gate equation restricted to prefix-visible wires. By the gate discipline, the restricted output sum is zero and at least one of the two restricted multiplicative sums is structurally zero; hence the restricted product is also zero. Thus $F_j(r_q) = 0$ for every future gate as well. The critical case is the next step, where carried state may still be prefix-visible; later steps have no prefix-visible non-unit wires by construction. Hence F_j vanishes at every root of P_B , proving the congruence. $\square$

C.2 Security Proof of Theorem 2

We prove Theorem 2 as stated there, writing $N = Bn \geq 2$ throughout and using its hypotheses: the composed R1CS is prefix-causal, faithfully represents the transition relation at the prescribed verifier-fixed values, follows the gate discipline of Section 4.2, and uses the stable disjoint partition $I_B = \{0\} \dot{\cup} I_B^{\text{io}} \dot{\cup} I_B^{\text{st}} \dot{\cup} I_B^{\text{wt}}$, and the two-source q -DLog assumption holds with $q = 2N - 1$. Its two assertions are independent and only the second is conditional: perfect completeness (Appendix C.2.1) is a direct calculation resting only on Lemma 1, proved in Appendix C.1, while updatable knowledge soundness (Appendix C.2.2) is a self-contained argument in the algebraic group model under Definitions 6 and 7.

C.2.1 Perfect completeness

Proof of Theorem 2, perfect completeness. Fix $\mathcal{ST}$, a bound B , some $J \in [1, B]$, and a transition sequence $t \in T^J$, and consider the honest execution on t . At each step it produces the statement $\phi_j = (t_1, j, s_j)$ and the witness increment ω_j , and the pair (ϕ_j, w_j) they assemble lies in $\mathcal{R}_{\mathcal{ST}}$ by construction, the underlying assignment $(a_i)_{i \in I_j}$ being the wire values of $\mathcal{C}_j$ on t . We show by induction on j that UVC.Verify accepts every step, so that $b_1 = \dots = b_J = 1$. The argument is an identity check and nothing probabilistic enters.

Base case. The initial values are

$$\pi_0 = ([\alpha]_1, [\beta]_2, [0]_1, [0]_1), \quad \zeta_0 = (0, 0, 0, 0), \quad \text{vst}_0 = [0]_1,$$

with $I_0 = \emptyset$, so the update rule applied to π_0 gives

$$\begin{aligned} A_1 &= \alpha + \sum_{i \in \{0\} \cup I_1} a_i \hat{u}_i(x), & D_1 &= \sum_{i \in I_1^{\text{st}}} a_i \frac{L_i(x)}{\gamma}, \\ B_1 &= \beta + \sum_{i \in \{0\} \cup I_1} a_i \hat{v}_i(x), & C_1 &= \sum_{i \in I_1^{\text{wt}}} a_i \frac{L_i(x)}{\delta} + \frac{h_1(x)P_B(x)}{\delta}, \end{aligned}$$

where $L_i(x) = \beta \hat{u}_i(x) + \alpha \hat{v}_i(x) + \hat{w}_i(x)$ and $h_1 = (U_1 V_1 - W_1)/P_B$. The assignment $(a_i)_{i \in I_1}$ satisfies the R1CS of $\mathcal{C}_1$, so Lemma 1 makes $U_1 V_1 - W_1$ divisible by P_B ; hence h_1 is a polynomial of degree at most $Bn - 2$, the CRS contains the elements needed to evaluate it, and $\pi_1 = ([A_1]_1, [B_1]_2, [C_1]_1, [D_1]_1)$ is well defined.

For the increment equation, the verifier starts from $\text{vst}_0 = [0]_1$ and $I_0^{\text{st}} = \emptyset$, so it recomputes

$$[\tilde{D}_1]_1 = \text{vst}_0 + \sum_{i \in I_1^{\text{st}}} a_i [L_i(x)/\gamma]_1 = \sum_{i \in I_1^{\text{st}}} a_i [L_i(x)/\gamma]_1 = [D_1]_1.$$

For the main equation, the extended QAP identity $U_1 V_1 - W_1 = h_1 P_B$ evaluated at x gives

$$\begin{aligned} A_1 B_1 &= \left(\alpha + \sum_{i \in \{0\} \cup I_1} a_i \hat{u}_i(x) \right) \left(\beta + \sum_{i \in \{0\} \cup I_1} a_i \hat{v}_i(x) \right) \\ &= \alpha \beta + \sum_{i \in \{0\} \cup I_1} a_i L_i(x) + h_1(x) P_B(x). \end{aligned}$$

The disjoint partition $I_1 = I_1^{\text{io}} \dot{\cup} I_1^{\text{st}} \dot{\cup} I_1^{\text{wt}}$ splits the middle sum along the two CRS tracks,

$$\begin{aligned} \sum_{i \in \{0\} \cup I_1^{\text{io}}} a_i L_i(x) &= Q_1 \gamma, & \sum_{i \in I_1^{\text{st}}} a_i L_i(x) &= D_1 \gamma, \\ \sum_{i \in I_1^{\text{wt}}} a_i L_i(x) + h_1(x) P_B(x) &= C_1 \delta, \end{aligned}$$

so that $A_1 B_1 = \alpha \beta + (Q_1 + D_1) \gamma + C_1 \delta$. Applying the pairing to both sides yields

$$[A_1]_1 \cdot [B_1]_2 = [\alpha]_1 \cdot [\beta]_2 + ([Q_1]_1 + [D_1]_1) \cdot [\gamma]_2 + [C_1]_1 \cdot [\delta]_2,$$

which is the main verification equation. Both checks hold identically, so UVC.Verify accepts and retains $\text{vst}_1 = [\tilde{D}_1]_1 = [D_1]_1$.

Inductive step. Let $1 \leq k < J$ and suppose the claim holds at step k : the proof π_k was accepted, its components are

$$\begin{aligned} A_k &= \alpha + \sum_{i \in \{0\} \cup I_k} a_i \hat{u}_i(x), & D_k &= \sum_{i \in I_k^{\text{st}}} a_i \frac{L_i(x)}{\gamma}, \\ B_k &= \beta + \sum_{i \in \{0\} \cup I_k} a_i \hat{v}_i(x), & C_k &= \sum_{i \in I_k^{\text{wt}}} a_i \frac{L_i(x)}{\delta} + \frac{h_k(x)P_B(x)}{\delta}, \end{aligned}$$

and the prover state is $\zeta_k = (U_k, V_k, W_k, h_k)$ with $U_k = \sum_{i \in I_k} a_i \hat{u}_i$, $V_k = \sum_{i \in I_k} a_i \hat{v}_i$, $W_k = \sum_{i \in I_k} a_i \hat{w}_i$, and $h_k = (U_k V_k - W_k)/P_B$. Since UVC.Verify accepted step k , its increment check forced $[D_k]_1 = [\tilde{D}_k]_1$ and it retained $\text{vst}_k = [\tilde{D}_k]_1$, so the group element the verifier now holds is the fourth component of π_k itself.

The honest execution supplies $(\phi_{k+1}, w_{k+1}) \in \mathcal{R}_{\mathcal{ST}}$ at the next step, and its assignment restricts on I_k to the one above, because $\mathcal{C}_{k+1}$ contains $\mathcal{C}_k$ as its first k copies and both are evaluated on the same t . The update rule accumulates $U_{k+1} = U_k + \sum_{i \in I_{k+1} \setminus I_k} a_i \hat{u}_i$ and likewise for V_{k+1} and W_{k+1} , so these are again the sums over I_{k+1} , and Lemma 1 applied to $\mathcal{C}_{k+1}$ makes $h_{k+1} = (U_{k+1} V_{k+1} - W_{k+1})/P_B$ a polynomial. Substituting the displayed forms of A_k, B_k, C_k, D_k into the update rule gives

$$\begin{aligned} A_{k+1} &= A_k + \sum_{i \in I_{k+1} \setminus I_k} a_i \hat{u}_i(x) = \alpha + \sum_{i \in \{0\} \cup I_{k+1}} a_i \hat{u}_i(x), \\ B_{k+1} &= B_k + \sum_{i \in I_{k+1} \setminus I_k} a_i \hat{v}_i(x) = \beta + \sum_{i \in \{0\} \cup I_{k+1}} a_i \hat{v}_i(x), \\ D_{k+1} &= D_k + \sum_{i \in I_{k+1}^{\text{st}} \setminus I_k^{\text{st}}} a_i \frac{L_i(x)}{\gamma} = \sum_{i \in I_{k+1}^{\text{st}}} a_i \frac{L_i(x)}{\gamma}, \\ C_{k+1} &= C_k + \sum_{i \in I_{k+1}^{\text{wt}} \setminus I_k^{\text{wt}}} a_i \frac{L_i(x)}{\delta} + \frac{(h_{k+1}(x) - h_k(x))P_B(x)}{\delta} \\ &= \sum_{i \in I_{k+1}^{\text{wt}}} a_i \frac{L_i(x)}{\delta} + \frac{h_{k+1}(x)P_B(x)}{\delta}, \end{aligned}$$

each sum telescoping because $I_k^* \subset I_{k+1}^*$ for $\star \in \{\text{io}, \text{st}, \text{wt}\}$, the classes being fixed once for the whole family, and because the two assignments agree on the prefix. The components therefore have the same closed form at step $k+1$.

For the increment equation, UVC.Verify recomputes

$$\begin{aligned} [\tilde{D}_{k+1}]_1 &= \text{vst}_k + \sum_{i \in I_{k+1}^{\text{st}} \setminus I_k^{\text{st}}} a_i [L_i(x)/\gamma]_1 \\ &= [D_k]_1 + \sum_{i \in I_{k+1}^{\text{st}} \setminus I_k^{\text{st}}} a_i [L_i(x)/\gamma]_1 = [D_{k+1}]_1, \end{aligned}$$

the second equality being the induction hypothesis $\text{vst}_k = [D_k]_1$ and the third the update rule the prover applied: verifier and prover begin the increment from the same group element and add the same terms. The main equation holds by the computation of the base case with I_{k+1} in place of

I_1 , since that computation used only the closed forms of the four components, the QAP identity at step $k+1$, and the disjoint partition. Hence UVC.Verify accepts π_{k+1} and retains $\text{vst}_{k+1} = [D_{k+1}]_1$, which is the induction hypothesis at step $k+1$.

Conclusion. By induction, UVC.Verify accepts every step of the honest execution, so $b_j = 1$ for all $j \in [1, J]$. Both verification equations were established as identities in $\mathbb{F}$, so this holds with probability 1; since $\mathcal{ST}$, B , J , and t were arbitrary, the scheme has perfect completeness in the sense of Definition 3. $\square$

C.2.2 Updatable knowledge soundness

Proof of Theorem 2, updatable knowledge soundness. Fix a step index j and an algebraic adversary $\mathcal{A}$, and write $J_{\text{pub}} = \{0\} \cup I_B^{\text{io}} \cup I_B^{\text{st}}$ for the γ track of the CRS and $J_{\text{wit}} = I_B^{\text{wt}}$ for the δ track. Write $\varepsilon(\lambda)$ for the probability that the extraction below fails. Suppose the experiment accepts all j steps of the run and write $\pi_j = ([A]_1, [B]_2, [C]_1, [D]_1)$ for the last proof. No induction is needed: extraction is demanded once, from π_j , and the earlier verification constraints enter only through the state.

The verifier state. Chaining the accepted increment checks from $\text{vst}_0 = [0]_1$ gives

$$\text{vst}_{j-1} = [D_{j-1}]_1, \quad D_{j-1} = \sum_{i \in I_{j-1}^{\text{st}}} \hat{s}_i \frac{L_i(x)}{\gamma}, \quad (6)$$

where the $\hat{s}_i$ are the states reported in $\phi_1, \dots, \phi_{j-1}$, which the experiment knows because $\mathcal{A}$ fixed every statement before producing any proof. Each check was verified by the experiment, so this holds however $\mathcal{A}$ produced the earlier proofs, and their effect on the verification equation enters only through this one group element; their representations are used separately, in the extraction below. For $j=1$ the sum is empty and $\text{vst}_0 = [0]_1$.

Extraction. Because $\mathcal{A}$ is algebraic, it accompanies each group element it outputs with a representation in terms of everything it has seen, that is, the CRS and the proofs it produced earlier; substituting the earlier representations, which $\mathcal{A}$ also supplied, expresses the components of π_j in the CRS basis alone, so we may write

$$\begin{aligned} A &= A_\alpha \alpha + A_\beta \beta + A_\delta \delta + \sum_{i \in I_B} (A_{u,i} \hat{u}_i(x) + A_{v,i} \hat{v}_i(x) + A_{w,i} \hat{w}_i(x)) \\ &\quad + \sum_{i \in J_{\text{pub}}} A_{\gamma,i} \frac{L_i(x)}{\gamma} + \sum_{i \in J_{\text{wit}}} A_{\delta,i} \frac{L_i(x)}{\delta} + \sum_{i=0}^{Bn-2} A_{h,i} \frac{x^i P_B(x)}{\delta}, \end{aligned}$$

with known coefficients, and similarly for B over the $\mathbb{G}_2$ part of the CRS and for C and D over the $\mathbb{G}_1$ part. The extractor $\chi_{\mathcal{A}}$ outputs

$$w := (C_{\delta,i})_{i \in I_j^{\text{wt}}} \cup (\hat{s}_i)_{i \in I_{j-1}^{\text{st}}},$$

the coefficients that $\mathcal{A}$ placed on the elements $[L_i(x)/\delta]_1$ of the δ track, restricted to the free wires of the first j copies, together with the earlier states, which are witness coordinates at step j and are read off the statements.

To see that this succeeds, treat $\alpha, \beta, \gamma, \delta, x$ as formal variables, so that each component of π_j is a known Laurent polynomial in them. Separating the two denominators, write

$$A = P_A + \frac{G_A}{\gamma} + \frac{H_A}{\delta}, \quad C = P_C + \frac{G_C}{\gamma} + \frac{H_C}{\delta}, \quad B = B_\gamma \gamma + R_B A \cdot B = P_A B_\gamma \gamma + P_A R_B + G_A B_\gamma + \frac{G_A R_B}{\gamma} + \frac{H_A B_\gamma \gamma}{\delta} + \frac{H_A R_B}{\delta},$$

where P_A, P_C, R_B carry no γ^{-1} or δ^{-1} and

$$G_A = \sum_{i \in J_{\text{pub}}} A_{\gamma,i} L_i(x), \quad H_A = \sum_{i \in J_{\text{wit}}} A_{\delta,i} L_i(x) + A_h(x) P_B(x),$$

with G_C and H_C defined the same way from C . That B has no γ^{-1} or δ^{-1} part is a property of the setup: σ_2 contains β, γ, δ and the extension polynomials, and no element of either track, which is where the asymmetry of the pairing is used.

The increment check removes the one unknown from the public side. Acceptance of step j forces

$$[D]_1 = \text{vst}_{j-1} + \sum_{i \in I_j^{\text{st}} \setminus I_{j-1}^{\text{st}}} \bar{a}_i [L_i(x)/\gamma]_1 = \sum_{i \in I_j^{\text{st}}} \bar{a}_i [L_i(x)/\gamma]_1,$$

the second equality by Eq. (6), with the coefficients on the newest slice read off the state reported in ϕ_j and those on I_{j-1}^{st} the states reported earlier in the run; note that this fixes the *value* of $[D]_1$ and uses no uniqueness property of the representation $\mathcal{A}$ supplied for it. Writing $\bar{\mathbf{a}}$ for the vector on J_{pub} that is 1 at 0, the verifier-determined values on I_j^{io} , the states reported along the run on I_{j-1}^{st} , the state of ϕ_j on the newest slice, and zero on every coordinate of J_{pub} belonging to a later copy, the accepted main equation becomes

$$A \cdot B = \alpha \beta + \sum_{i \in J_{\text{pub}}} \bar{a}_i L_i(x) + C \delta. \quad (7)$$

Let F denote the difference of the two sides of Eq. (7), multiplied by $\gamma \delta$ to clear denominators; it is a polynomial in the formal variables whose degree in x is at most $3N$, since $A \gamma \delta$ has x -degree at most $2N-2$ through the quotient track and B at most $N-1$, with coefficients computable from $\mathcal{A}$'s output. Suppose F is not the zero polynomial. Because an algebraic adversary chooses its coefficients after seeing concrete encodings, we cannot pass directly from $F \neq 0$ to the vanishing of a nonzero univariate polynomial at the trapdoor; we use the embedding of Fuchsbaier et al. [44] instead. Given a q -DLog challenge $([z^k]_1, [z^k]_2)_{k \leq q}$ with $q = 2N-1$, the reduction samples $y_1, \dots, y_5 \leftarrow \mathbb{F}^*$ and sets

$$\alpha = y_1 z, \quad \beta = y_2 z, \quad \gamma = y_3 z, \quad \delta = y_4 z, \quad x = y_5 z,$$

so that every CRS element becomes a known combination of the challenge powers: clearing denominators by the common factor $\gamma \delta$ in $\mathbb{G}_1$ leaves the basis $\delta \gamma \alpha, \delta \gamma \beta, \delta \gamma \hat{u}_i(x), \delta L_i(x), \gamma L_i(x)$, and $\gamma x^k P_B(x)$, whose largest total degree is $\deg(\gamma x^{N-2} P_B(x)) = 1 + (N-2) + N = 2N-1$, and the $\mathbb{G}_2$ elements have smaller degree. The reduction can therefore present the CRS to $\mathcal{A}$ without knowing z , and q is fixed by the maximum total degree of the polynomials defining the CRS rather than by the degree of the verification equation. Running $\mathcal{A}$ on this CRS yields coefficients and hence F , which is a nonzero polynomial in five variables of total degree at most $2N-1$ in each argument. Substituting the embedding turns it into the univariate $F(y_1 Z, \dots, y_5 Z)$, and by Schwartz–Zippel over the hidden y_i this is nonzero except with probability at most $\deg F / (|\mathbb{F}| - 1)$; when it is nonzero, z is among its at most $\deg F$ roots and the reduction outputs it. Collecting the two losses, the case $F \neq 0$ occurs with probability at most $\varepsilon(\lambda)$, the q -DLog advantage of the reduction plus a term negligible in λ .

Assume therefore that F vanishes identically and compare coefficients. Expanding the left side of Eq. (7),

while the right side is $\alpha\beta + \sum_i \bar{a}_i L_i(x) + \delta P_C + \frac{\delta G_C}{\gamma} + H_C$.

Note that σ_2 contains β, γ, δ and the extension polynomials but not α , so B has no α component; this is where the asymmetry of the pairing enters. Five comparisons pin the shape of the proof.

1. *The δ^{-1} terms.* Collecting them on the left gives $H_A R_B / \delta$ and $H_A B \gamma / \delta$, that is, $H_A \cdot B / \delta$, while the right side has none; hence $H_A \cdot B = 0$. If $B = 0$ the left side vanishes while the right retains the monomial $\alpha\beta$, so $B \neq 0$ and therefore $H_A = 0$: A places no weight on the free-witness track or on the quotient track.
2. *The $\alpha\beta$ term.* No other product contributes this monomial, since $G_A B \gamma$ is a combination of the $L_i(x)$ and carries α or β but never both, so $A_\alpha B_\beta = 1$. In particular $A_\alpha \neq 0$ and $B_\beta \neq 0$.
3. *The γ^{-1} terms.* They give $G_A R_B = \delta G_C$. Splitting $R_B = B_\delta \delta + R'_B$ with $R'_B = B_\beta \beta + B(x)$ free of δ , the δ -free part reads $G_A R'_B = 0$, and $R'_B \neq 0$ because $B_\beta \neq 0$; hence $G_A = 0$ and then $\delta G_C = 0$, so $G_C = 0$ as well. Neither A nor C places weight on the public track.
4. *The β^2 and γ terms.* The right side has no β^2 , so $A_\beta B_\beta = 0$ and thus $A_\beta = 0$; it has no γ , so $A_\alpha B_\gamma = 0$ and thus $B_\gamma = 0$. What remains is $A = A_\alpha \alpha + A_\delta \delta + A(x)$ and $B = B_\beta \beta + B_\delta \delta + B(x)$ with $A(x), B(x)$ of degree below N .
5. *The remaining terms.* Write $m_i := \bar{a}_i$ for $i \in J_{\text{pub}}$ and $m_i := C_{\delta,i}$ for $i \in J_{\text{wit}}$, so that the right side of Eq. (7) is $\alpha\beta + \sum_i m_i L_i(x) + C_h(x) P_B(x) + \delta P_C$. Comparing the monomials $\alpha f(x), \beta f(x)$, and those free of $\alpha, \beta, \gamma, \delta$ gives

$$A_\alpha B(x) = \sum_i m_i \hat{v}_i(x), \quad A(x) B_\beta = \sum_i m_i \hat{u}_i(x),$$

$$A(x) B(x) = \sum_i m_i \hat{w}_i(x) + C_h(x) P_B(x).$$

The scalars A_δ and B_δ are unconstrained: every monomial they produce carries a factor δ and is absorbed by δP_C , which is consistent with the re-randomization noted in Appendix A.3 and does not affect m .

Multiplying the first two identities and using $A_\alpha B_\beta = 1$ gives $A(x) B(x) = (\sum_i m_i \hat{u}_i(x)) (\sum_i m_i \hat{v}_i(x))$, so the third reads

$$\sum_i m_i \hat{u}_i(x) \cdot \sum_i m_i \hat{v}_i(x) = \sum_i m_i \hat{w}_i(x) + C_h(x) P_B(x),$$

an identity of polynomials in x , so $\{m_i\}_{i \in I_B}$ satisfies the full-bound QAP modulo $P_B(X)$, with $m_0 = 1$, agreeing with $\bar{\mathbf{a}}$ on J_{pub} and with the extracted w on J_{wit} . In particular it carries the verifier-determined values on I_j^{io} , the states reported along the run on I_{j-1}^{st} , the state of ϕ_j on the newest slice, and zero on every public and state coordinate of a later copy. Future witness coefficients need not vanish, since they are the $C_{\delta,i}$ that $\mathcal{A}$ chose; they are discarded when the assignment is restricted to I_j below, which by prefix causality leaves every prefix row unchanged.

The disjoint partition is what makes the last point stick: a state coordinate is queried only as $L_i(x)/\gamma$ and never as $L_i(x)/\delta$, so $\mathcal{A}$ has no element with which to move it into C and away from the value that $[D]_1$ fixes.

Because every proof of the run was produced by $\mathcal{A}$, this single extraction already covers the prefix: the coefficients on J_{wit} carry the transition inputs and internal wires of all j copies, not merely those of the last one. Restricting $\{m_i\}$ to I_j leaves a satisfying assignment for the R1CS of C_j : by

prefix causality the first jn rows carry no coefficient on a wire outside I_j , so discarding those coordinates changes none of them. Padding the restriction with $a_i = 0$ for $i \in I_B \setminus I_j$ meets the vanishing condition of Eq. (2), and Lemma 1 restores the congruence modulo $P_B(X)$ at the roots of the later copies. The disjoint partition is what makes the restriction meaningful on the state wires as well, since no $i \in I_B^{\text{st}}$ has an element $L_i(x)/\delta$ with which $\mathcal{A}$ could have offset the state that $[D]_1$ fixes. Hence $(\phi_j, w) \in \mathcal{R}_{\mathcal{ST}}$ except with probability $\varepsilon(\lambda)$.

It remains to rule out the second disjunct of Definition 4. The extractor sets the coordinates $w|_{I_i^{\text{st}} \setminus I_{i-1}^{\text{st}}}$, for $i < j$, to the states $\hat{s}_i$ reported in $\phi_1, \dots, \phi_{j-1}$, so they agree with the run by construction. This is not vacuous: the assignment $\{m_i\}$ produced by the comparison above already carries $\bar{a}_i = \hat{s}_i$ on I_{j-1}^{st} , because the accepted increment checks forced $[D]_1$ to that value, so the relation is satisfied with those very coordinates rather than with some other history the adversary might have preferred. Both disjuncts therefore fail except with probability $\varepsilon(\lambda)$.

Error analysis. Recovering the CRS representation of π_j does walk back through the run, since $\mathcal{A}$ may have written each proof in terms of the earlier ones; unwinding takes j substitutions. That unwinding is deterministic linear algebra on coefficient vectors $\mathcal{A}$ has already supplied, however, and not a re-execution of $\mathcal{A}$: the extractor is run once and never rewinds. The only probabilistic step is the dichotomy above, applied to the single verification equation of step j , so the failure probability is $\varepsilon(\lambda)$ and does not compound with the length of the run. Writing $\text{Adv}_{\mathcal{A},j}^{\text{uks}}(\lambda)$ for the winning probability of Definition 4 at step j ,

$$\text{Adv}_{\mathcal{A},j}^{\text{uks}}(\lambda) \leq \varepsilon(\lambda) \quad \text{for every } j \in [1, B],$$

a bound uniform in j rather than one growing with it. This is where the notion differs from rewinding-based extraction for recursive and folding-based IVC, in which an extractor is applied once per level and the loss compounds with depth, as discussed in Section 2.

Finally, faithfulness of the arithmetization identifies the extracted w with a valid sequence of j transitions from the genesis state whose intermediate states are the ones reported along the run and whose final state is the one reported in ϕ_j , which is the reachability conclusion of Theorem 2. $\square$

The extraction never assumes that the adversary ran the honest update algorithm, and the conclusion certifies reachability through *some* witness input sequence; authenticity, ordering, and freshness of external events need the input-binding mechanisms of Section 3.2. Nor may the verifier state be adversarially supplied: as shown in Section 5.1, an arbitrary group element defeats the increment check by a public group operation.

D A Worked Instance of the Wire Classes

Let $\text{StUp}(t, s) = (s+t)^2$ with $s_0 = 0$ and a single-field-element state, so copy k has two constraints,

$$(s_{k-1} + t_k) \cdot \text{unit}_k = z_k, \quad z_k \cdot z_k = s_k,$$

with z_k internal. For $B = 3$ the classes are

$$\begin{aligned} I_3^{\text{io}} &= \{\text{unit}_1, \text{unit}_2, \text{unit}_3, t_1\} \\ I_3^{\text{st}} &= \{s_1, s_2, s_3\} \\ I_3^{\text{wt}} &= \{t_2, t_3, z_1, z_2, z_3\} \end{aligned}$$

and the prefixes are $I_1 = \{0, \text{unit}_1, t_1, z_1, s_1\}$ and $I_2 = I_1 \cup \{\text{unit}_2, t_2, z_2, s_2\}$. At step 2 the verifier holds $\phi_2 = (t_1, 2, s_2)$ and $\text{vst}_1 = [D_1]_1$. It forms Q_2 from $a_0 = 1$, $\text{unit}_1 = \text{unit}_2 = 1$ and the anchor t_1 , and it forms $[\tilde{D}_2]_1 = \text{vst}_1 + s_2 [L_i(x)/\gamma]_1$ for the single index $i \in I_2^{\text{st}} \setminus I_1^{\text{st}}$, *without knowing* s_1 . This is the separation of Eq. (1) in concrete terms. *The units as switch values.* Take the prefix $j = 2$, so that the vanishing condition sets $\text{unit}_3 = t_3 = z_3 = s_3 = 0$ while the prefix wires, among them s_2 , keep their values. Copy 3 then reads

$$(s_2 + 0) \cdot 0 = 0, \quad 0 \cdot 0 = 0,$$

and both hold whatever s_2 is: the zero on unit_3 has switched the copy off. This is what the congruence modulo $P_B(X)$ needs at the roots of copy 3, and it is the content of Lemma 1 on this circuit.

Three variants show that the arrangement is not incidental. First, suppose copy 3 used the global wire 0 in place of unit_3 , so that its first constraint were $(s_2 + t_3) \cdot 1 = z_3$. The vanishing condition cannot switch that coefficient off, since $a_0 = 1$ at every prefix, so the constraint would read $(s_2 + 0) \cdot 1 = 0$ and force $s_2 = 0$: an honest prefix would fail the congruence because of its own current state. Second, suppose the units lay in I_B^{wt} instead. They would then be free wires, and a prover could set $\text{unit}_2 = 0$ while step 2 is *visible*, turning $(s_1 + t_2) \cdot 0 = z_2$ into $z_2 = 0$ and switching a live step off; placing them in I_B^{ro} is what fixes them to 1 up to j . Third, the units do not by themselves suffice. Had the transition been written with the carried state on both multiplicative sides, say $s_{k-1} \cdot s_{k-1} = z_k$, no unit would appear in that row, and the prefix $j = 2$ would give $s_2 \cdot s_2 = 0$ and again force $s_2 = 0$. This is the second half of the gate discipline: a future gate must have an empty prefix-visible output side and at least one empty prefix-visible multiplicative side, and the circuit above satisfies it because unit_k occupies one side of every first-row product and z_k , an internal wire, occupies both sides of every second-row product.