PPAD Hardness from SIS

Arka Rai Choudhuri*
zkBricks

Abstract

Understanding the hardness of the complexity class PPAD is an important question in complexity theory given that it contains problems such as computing a Nash equilibrium. Evidence of hardness, however, has come primarily from cryptography, through a long line of work. Most recently, Chen, Freitag, Jin and Wichs (STOC 2025) based the hardness of PPAD on the polynomial hardness of the Learning with Errors (LWE) problem, together with a fine-grained average-case hardness assumption.

In this work, we base the hardness of PPAD on a different lattice problem, the Short Integer Solution (SIS) problem, which, unlike the assumptions underlying prior work, is not known to imply public-key encryption. Specifically, we show that PPAD is hard on average assuming either:

(i) polynomially hard SIS with a quasi-polynomial norm bound and a fine-grained average-case hardness assumption; or

(ii) subexponentially hard SIS alone.

Our main technical contribution is a succinct, incrementally updatable proof system for a class of exponentially large circuits with short descriptions. Building on the work of Wee (CRYPTO 2025) for succinct proof systems for circuits from SIS, we show how to make such proof systems satisfy these properties while preserving the unambiguity guarantee of the underlying scheme: it is hard to find two distinct accepted states for the same computation and time. Together, these are crucial for our PPAD hardness reductions.

*Email: arkarai.choudhuri@gmail.com.

Contents

1 Introduction **3**
- 1.1 Our Results . . . 3
- 1.2 Related Work . . . 4

2 Technical Overview **7**
- 2.1 The Proof-Based Template . . . 7
- 2.2 Tree Computations . . . 9
- 2.3 Mergeable Functional Commitments . . . 10
- 2.4 From Mergeable Functional Commitments to RSVL . . . 12
- 2.5 Providing the Hard Computation for RSVL . . . 13
- 2.6 A Mergeable Functional Commitment from SIS . . . 14
- 2.7 Organization . . . 21

3 Preliminaries **23**
- 3.1 Search Problems, RSVL, and EOML . . . 23
- 3.2 Short Integer Solutions . . . 24
- 3.3 Regular tree circuits . . . 25

4 Updatable Unambiguous SNARGs for Circuits and EOML Hardness **26**
- 4.1 Unambiguous updatable circuit SNARGs . . . 26
- 4.2 Average-case tree puzzles . . . 28
- 4.3 Hard RSVL instances . . . 29

5 Updatable Unambiguous SNARGs for Circuits from Mergeable Functional Commitments **33**
- 5.1 Arithmetic encoding of regular tree circuits . . . 33
- 5.2 Mergeable functional commitments . . . 34
- 5.3 Node proofs . . . 37
- 5.4 The uuSNARG construction . . . 41

6 A Mergeable Functional Commitment from SIS **44**
- 6.1 The commitment . . . 44
- 6.2 Layer proofs . . . 48
- 6.3 Merging . . . 51
- 6.4 Input commitments . . . 52
- 6.5 Norms and the main theorem . . . 54

7 Instantiations **59**
- 7.1 Planted search . . . 59
- 7.2 Clique polynomials . . . 60
- 7.3 Hardness from subexponential SIS alone . . . 61

8 AI Usage Disclosure **63**

A Worst-Case Hardness from ETH **68**

B From Worst-Case to Average-Case Clique Hardness **70**

C Hardness of UEOPL **72**

1 Introduction

Computing a Nash equilibrium and finding a Brouwer fixed point are central examples of total search problems: a solution is guaranteed to exist, but finding one may be computationally hard. Nash's theorem, for instance, guarantees an equilibrium in every finite game [Nas51]. Computing an equilibrium even in a two-player game [DGP06, CD06], as well as finding a Brouwer fixed point [Pap94], is complete for the complexity class PPAD (Polynomial Parity Arguments on Directed graphs). PPAD is one of several subclasses of TFNP, the class of total search problems [MP91], each defined by the combinatorial principle that guarantees a solution [Pap94]. For PPAD, this principle is a parity argument on directed graphs.

No polynomial-time algorithms are known for these problems, but establishing their hardness presents a challenge of its own. No total search problem can be NP-hard unless NP = coNP [MP91], so the usual approach of proving NP-hardness is unlikely to establish the hardness of PPAD. Cryptography provides another approach: a reduction can show that an efficient algorithm for PPAD would break a cryptographic assumption.

This approach has led to two lines of work. The first establishes PPAD hardness from obfustopia primitives, namely indistinguishability obfuscation [BPR15, GPS16, HY17] or private-key functional encryption [KS17], following an approach suggested by Abbott, Kane and Valiant [AKV04]. These primitives can be instantiated from well-founded assumptions [JLS21, JLS22]. The second uses succinct arguments for long computations whose proofs are *unambiguous* and can be updated incrementally [CHK+19a, CHK+19b, EFKP20, KPY20, LV20, JKKZ21, BCH+22, KLV23]. Using this approach, Chen, Freitag, Jin and Wichs [CFJW25] constructed unambiguous SNARGs for P from polynomially hard Learning with Errors (LWE). Combined with a fine-grained average-case hardness assumption, their construction gives PPAD hardness from polynomial LWE.

The concrete cryptographic assumptions underlying all of these results imply public-key encryption.[1] These include LWE, DDH and bilinear maps, as well as the assumptions used to instantiate obfustopia primitives.[2] It is unclear why public-key encryption should be necessary for PPAD hardness. The known barriers to using weaker primitives are only partial [RSS17] (see Section 1.2), leaving the following question.

Can we establish the hardness of PPAD *based on cryptographic assumptions that are not known to imply public-key encryption?*

1.1 Our Results

We answer this question for the Short Integer Solution (SIS) problem [Ajt96]. Given a uniformly random $n \times m$ matrix $\mathbf{A}$ modulo q, SIS asks for a nonzero integer vector $\mathbf{u}$ such that $\mathbf{Au} = \mathbf{0} \bmod q$ and every entry of $\mathbf{u}$ has absolute value at most a bound β. With appropriate parameters, SIS is as hard as worst-case lattice problems [Ajt96] and gives one-way functions and collision-resistant hash functions. It is not known to imply public-key encryption.[3]

Our first result combines polynomially hard SIS with a fine-grained average-case hardness assumption. Informally, this assumption is an average-case analogue of the time hierarchy theorem: for every polynomial bound p, there is a polynomial-time computable function that is hard to compute on random inputs by circuits of size p. A similar assumption is used by Chen, Freitag, Jin and Wichs [CFJW25], following [BPSS23].

[1] We consider the standard model. Relative to a random oracle, PPAD is hard assuming that #SAT is hard [CHK+19a], an assumption not known to imply public-key encryption.

[2] For example, indistinguishability obfuscation and one-way functions imply public-key encryption, and hardness of iterated squaring in RSA groups implies hardness of factoring. Komargodski and Segev [KS17] show that the quasi-polynomially secure private-key functional encryption used in their result, together with subexponentially secure one-way functions, implies public-key encryption.

[3] This concerns SIS as an average-case assumption. The worst-case hardness of approximating lattice problems, which is the standard evidence for SIS, is known to give public-key encryption [Reg05, Pei09].

Theorem 1.1 (Informal). *Assume that* SIS *is hard for polynomial-time adversaries, with a quasi-polynomial norm bound* $\beta = 2^{O(\log^3 n)}$*, and that there is a fine-grained average-case hard computation as above, with the tree structure described below. Then there is a polynomial-time samplable distribution of* PPAD *instances that is hard on average.*

Candidates for such a computation include exhaustive search for preimages of an exponentially hard one-way function, at increasing security parameters, and evaluation of the weighted k-clique polynomials, motivated by a non-uniform version of the worst-case clique-parity conjecture of Boix-Adserà, Brennan and Bresler [BBB19]. We discuss both instantiations in detail later in the paper.

Our second result removes the additional complexity assumption by strengthening the hardness required of SIS from polynomial to subexponential. Here, SIS plays the roles of both the proof system and the hard computation: we use different dimensions for the two roles (Section 7.3).

Theorem 1.2 (Informal). *Assume that* SIS *with norm bound* $\beta = 2^{n^{1/3}}$ *is hard for adversaries of size* $2^{n^{1/3}}$*, with the remaining parameters specified in Section 7.3. Then there is a polynomial-time samplable distribution of* PPAD *instances that is hard on average.*

Both results give hardness of the subclass CLS $\subseteq$ PPAD, since we construct hard instances of End-of-Metered-Line (EOML) [HY17].

Extensions. As in [BCH+22], we further obtain hardness of the smaller class UEOPL $\subseteq$ CLS, introduced by Fearnley, Gordon, Mehta and Savani [FGMS19], under the same assumptions (Section C). For worst-case hardness, polynomially hard SIS with a quasi-polynomial norm bound together with the non-uniform exponential time hypothesis (ETH) suffices. We prove this in Section A, following the approach of Bitansky and Gerichter [BG20].

Updatable proofs for circuits. Our technical contribution is a construction of unambiguous, updatable circuit SNARGs (uuSNARG) for a class of exponentially large circuits with short descriptions. These circuits have regular tree structure: each leaf performs a small computation, and each internal node combines the outputs of its children. The proofs can be updated incrementally, so the prover evaluates the tree while keeping only a polynomial-size state. Preprocessing, verification and each update take time polynomial in the circuit's description length. The state is unambiguous in the following sense: it is hard to find two distinct accepted states for the same circuit and time. Our reductions obtain the soundness they need from this guarantee: given an accepted state that differs from the honest one, they recompute the honest state, which gives two distinct accepted states. This recomputation must cost less than the adversaries against which state uniqueness holds. Together with a hard computation of the required tree structure, these properties suffice for our PPAD hardness reductions.

We construct this proof system from SIS by adapting the functional commitments underlying Wee's circuit SNARGs [Wee25].[4] Incrementally updatable proofs are already central to earlier constructions for uniform computations [CHK+19a, CHK+19b, EFKP20, KPY20, LV20, JKKZ21, BCH+22, KLV23, CFJW25]; here we obtain them for the structured circuits needed by our application.

1.2 Related Work

PPAD **hardness in obfustopia.** Abbott, Kane and Valiant [AKV04] suggested basing PPAD hardness on program obfuscation. Bitansky, Paneth and Rosen [BPR15] realized this approach from indistinguishability obfuscation and one-way functions, and Garg, Pandey and Srinivasan [GPS16] relied on polynomial

[4] As in [KPY20], the construction does not rely on the Fiat–Shamir transform or on correlation-intractable hashing.

hardness, through functional encryption. Hubáček and Yogev [HY17] extended these results to CLS, and Komargodski and Segev [KS17] replaced indistinguishability obfuscation with quasi-polynomially secure private-key functional encryption, together with subexponentially secure injective one-way functions. Recently Ben-David and Yogev [BDY26] gave a more direct construction of hard End-of-Line instances, from puncturable trapdoor permutations with full key and message domains, which they build from indistinguishability obfuscation and permutable pseudorandom permutations. Their reduction does not go through the Sink-of-Verifiable-Line problem, which all of the other aforementioned works use.

PPAD **hardness from unambiguous proofs.** Choudhuri, Hubáček, Kamath, Pietrzak, Rosen and Rothblum [CHK+19a] showed that PPAD is hard if the Fiat–Shamir transform for the sumcheck protocol is unambiguously sound and #SAT is hard, and instantiated the hash function assuming fully homomorphic encryption with almost-optimal circular security against quasi-polynomial-time adversaries. The same authors [CHK+19b] and Ephraim, Freitag, Komargodski and Pass [EFKP20] applied this approach to Pietrzak's protocol for iterated squaring [Pie19], basing PPAD hardness on iterated squaring in place of #SAT. Subsequent works obtained PPAD hardness through unambiguous incrementally verifiable computation from the KPY assumption on bilinear groups [KPY20], from LWE together with iterated squaring [LV20, BCH+22], from subexponential LWE [JKKZ21], and from subexponential DDH [KLV23]. Chen, Freitag, Jin and Wichs [CFJW25] obtained PPAD hardness from polynomial LWE together with a fine-grained average-case hardness assumption, through unambiguous SNARGs for P. Freitag and Wichs [FW26] study the stronger notion of unique SNARGs, in which every statement has at most one accepting proof, and build these from obfustopia assumptions.

SIS **and total search problems.** The connection between SIS and total search problems predates our work. Since SIS gives collision-resistant hash functions [Ajt96, GGH96], its average-case hardness gives average-case hardness in PWPP, the class of collision-finding problems [Jeř16], and hence in PPP. Sotiraki, Zampetakis and Zirdelis [SZZ18] also established completeness results for related problems: constrained SIS and the Blichfeldt problem are PPP-complete, while weak constrained SIS is PWPP-complete.[5] Göös, Kamath, Sotiraki and Zampetakis [GKSZ20] place restricted versions of SIS with a small modulus in PPA_p, the subclass of TFNP whose totality follows from counting arguments modulo a prime p. Since $\mathsf{PPAD} \subseteq \mathsf{PPA}_p$ [GKSZ20], hardness of PPAD implies hardness of PPA_p; the converse is not known.

Our results give hardness in CLS, and hence PPAD, from SIS itself (Fig. 1). This does not follow from collision resistance, since PWPP is not known to be contained in PPAD.

Barriers. As noted above, most constructions of hard PPAD instances go through Sink-of-Verifiable-Line, or a relaxation of it [CHK+19a]. In Sink-of-Verifiable-Line, the vertices of a line can be efficiently verified to lie on it, given their position, and the task is to find the last vertex. Rosen, Segev and Shahaf [RSS17] show that black-box constructions from one-way functions, or from any primitive that exists relative to a random injective trapdoor function oracle, cannot go through Sink-of-Verifiable-Line, and must produce instances with nearly exponentially many solutions. Hubáček, Kamath, Král and Slívová [HKKS20] rule out fully black-box constructions of hard total search problems from injective one-way functions, for a restricted class of reductions. These barriers concern constructions that use the underlying primitive as a black box. Our construction does not use SIS only as a one-way or collision-resistant function: the functional commitment relies on the linear structure of SIS, which is not available in these oracle models. Thus our construction is not black-box in the sense of these results, and they do not apply to it.

[5] Informally, constrained SIS restricts the binary vectors under consideration by additional linear equations, and asks for a vector mapped to zero or two vectors with the same image. Weak constrained SIS uses parameters that guarantee compression, and asks only for a collision.

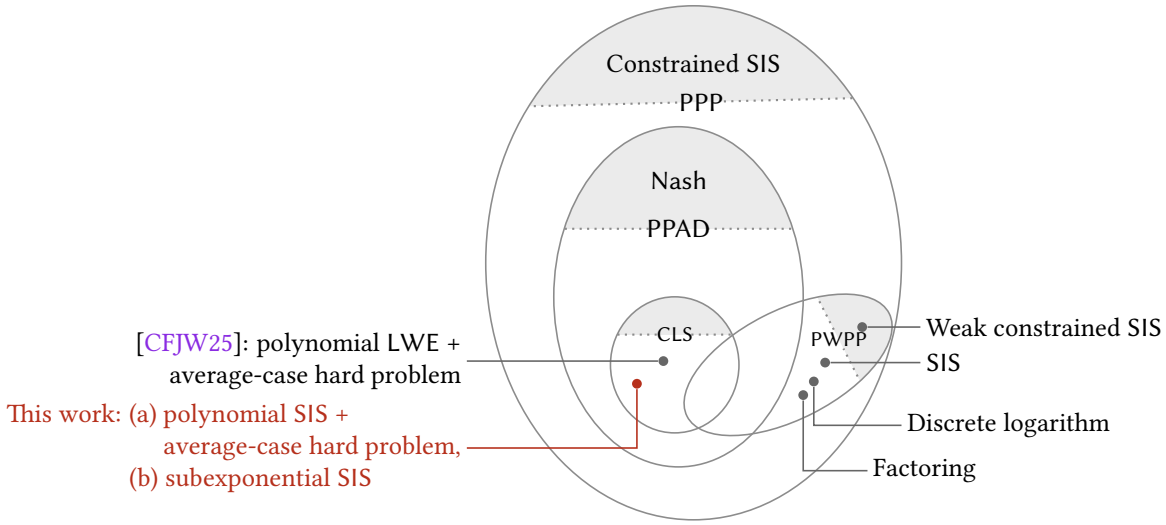


Figure 1: SIS and related problems in TFNP. In each class, the shaded region above the dotted line contains its complete problems. In red, our hard EOML instances, from (a) polynomial SIS and an average-case hard problem (Theorem 1.1) or (b) subexponential SIS (Theorem 1.2). The prior result [CFJW25] uses polynomial LWE and an average-case hard problem. Relative positions other than the containments CLS ⊆ PPAD ⊆ PPP and PWPP ⊆ PPP are schematic.

Succinct arguments from SIS. Wee [Wee25] constructed functional commitments and SNARGs for P/poly with unambiguous proofs from standard SIS, using only the algebraic structure of the commitment, without Fiat–Shamir or correlation intractability. Earlier lattice-based functional commitments rely on non-standard variants of SIS, or, from standard SIS [dCP23], do not support fast verification for general circuits, without which a SNARG for deterministic computations is trivial (see [Wee25]). We build on the commitment of [Wee25], and explain in the technical overview why it does not directly give the proof system we need.

2 Technical Overview

As stated in the introduction, we demonstrate the hardness of PPAD, building on the recent construction [Wee25] of succinct proofs for circuits (circuit SNARGs) based on SIS. In this overview, we describe the main ideas underlying our work. We start by describing the template used in prior works, connecting succinct proofs to PPAD. This will help pinpoint the exact security and efficiency properties needed from the underlying circuit SNARGs. We then describe our construction of the circuit SNARGs.

2.1 The Proof-Based Template

We start by describing here the succinct-proof-based template to show hardness of PPAD. This will be the template followed in this work as well. Let us fix the security parameter λ—this will let us talk about both the hardness and efficiency.

Relaxed Sink-of-Verifiable-Line. It has already been established in prior work [CHK+19a] that to show hardness of PPAD, it suffices to construct a hard instance of the Relaxed Sink-of-Verifiable-Line (RSVL) problem. An RSVL instance specifies a line of vertices, via two $\mathsf{poly}(\lambda)$-sized circuits, a successor circuit S and a verifier circuit V, together with a length L and a starting vertex v_0. For any vertex v,

$$\mathsf{S}(v) \to v' \qquad \mathsf{V}(v, i) \to 0/1$$

Let $v_i := \mathsf{S}^i(v_0)$ be the i-th vertex when the successor circuit S is applied i times to the starting vertex v_0. The vertices $\{v_i\}_i$ form a line, and V is promised to accept (v_i, i) for every $i \le L$.

A solution is either (i) the *sink*, a vertex v with $\mathsf{V}(v, L) = 1$; or (ii) a *false positive*, a pair (v, i) with $\mathsf{V}(v, i) = 1$ but $v \neq v_i$. So V certifies that a vertex lies at a given position on the line, and the *relaxation* is that V may also accept vertices not on the prescribed line, i.e., V is not an if and only if condition.

For any constructed RSVL instance to be hard against $\mathsf{poly}(\lambda)$-size adversaries, we need L to be super-polynomial (i.e., $L = \lambda^{\omega(1)}$) since otherwise the adversary can find the sink by starting with v_0 and applying the successor circuit a polynomial number of times to compute $v_L = \mathsf{S}^L(v_0)$.

From delegation to hard instances of RSVL**.** The structure of the RSVL computation draws a natural connection to delegating computation. We start with a computation which proceeds in steps, i.e., $z_0 \to z_1 \to \cdots \to z_L$ with $z_{t+1} = \mathsf{Step}(z_t)$. To each step z_t we attach a proof π_t that certifies that $z_t = \mathsf{Step}^t(z_0)$. The vertex is thus $v_t = (z_t, \pi_t)$, with V checking the proof, and $\mathsf{S}(z_t, \pi_t) = (z_{t+1}, \pi_{t+1})$.

We now describe the properties required of the computation and the proof system to make sure the above idea results in a hard RSVL instance:

- *Hard Computation.* Even with a super-polynomial L, an adversary might circumvent the use of S and directly compute z_L and the corresponding π_L. Thus, we require that computing z_L is hard for a polynomial size adversary.
- *(Computational) Unique Proofs.* To make it hard to find false positive solutions, it must be hard to find an accepting pair (z, π) at position t that differs from (z_t, π_t). When $z \neq z_t$, this is the standard soundness condition of the proof system since z is no longer a true statement. However, when $z = z_t$, we require a stronger property, *unambiguity* or computational uniqueness, where it is hard for a polynomial size adversary to find a *different* proof for a true statement.
- *Updatability.* The successor must be $\mathsf{poly}(\lambda)$ size, so π_{t+1} must be computable from (z_t, π_t) by a polynomial size circuit, without recomputing the proof from scratch.

The first two properties concern security, while the last property has to do with efficiency. There are many candidates for the hard computation, and the updatability notion is closely related to that of incrementally verifiable computation (IVC) [Val08]. However, despite the progress in building delegation schemes for (hard) computation [KPY19, CJJ22, WW22, KLVW23, CGJ+23], unambiguity remains a non-trivial property to achieve. The works of [KPY20, CFJW25] make proof systems for generic computations unambiguous, while other works make proof systems for specific computations unambiguous, such as the sumcheck protocol [CHK+19a, JKKZ21] and Pietrzak's protocol [Pie19] for iterated squaring [CHK+19b, EFKP20, LV20, BCH+22].

Since unambiguity is hard to achieve, we start from the circuit SNARGs of [Wee25], which are unambiguous SNARGs for P/poly based on SIS. However, circuit SNARGs do not directly fit the template above: they prove the output of a single circuit in one shot, whereas the template needs proofs for a long computation that are updated step by step. We first recall the properties of these SNARGs, and then the obstacles they raise for the template.

[Wee25] Circuit SNARGs. The core component of the delegation scheme in [Wee25] is a *chainable* functional commitment scheme [BCFL23]. An ordinary functional commitment opens a commitment C to x directly to the value $y = f(x)$, for a function f represented by a circuit. A chainable one instead opens C to a short commitment C' to y: the opening is a proof π that C' is a commitment to $f(x)$. Since an opening ends at a commitment, openings can be chained. A circuit of depth d is opened one layer at a time, along a chain $(C = C_0) \to C_1 \to \cdots \to (C_d = C')$ of commitments to the intermediate layer values, and the proof consists of the intermediate commitments together with one opening per layer. As a result, the proof grows with the depth of the circuit. This gives a SNARG for the statement $y = f(x)$: given x, y and the proof, the verifier recomputes C and C', which is possible since the commitment is deterministic, and checks the opening of each layer. The intermediate commitments are part of the proof, so the verifier never computes the intermediate layer values.

For Boolean circuits of size s and depth d, on ℓ-bit inputs, the SNARGs in [Wee25] have the following properties (ignoring log and λ terms):

- *Preprocessing.* The verifier preprocesses the circuit in time $O(s)$.
- *Proving.* The prover runs in time $O(s)$, and proofs have size $O(d)$.
- *Online verification.* After preprocessing, verification takes time $O(\ell + d)$.
- *Unambiguity.* The underlying functional commitment satisfies *strong binding*: it is hard to produce two different accepted pairs (C', π) for the same commitment C and function f, even with the same output commitment C'.[6]

Note that when $s = \mathrm{poly}(\lambda)$, strong binding also implies soundness: given an accepting proof for a wrong output, a reduction computes the honest proof in time $O(s)$. Comparing the claimed output and proof with the honest ones either gives two different accepted pairs (C', π), contradicting strong binding, or two different outputs with the same commitment, contradicting commitment binding.

Obstacles for RSVL from circuit SNARGs. Let us now try to instantiate the template with circuit SNARGs. Recall from the template above that the line must have super-polynomial length. Applying the circuit SNARG directly to the computation accumulated along the line therefore requires proofs for circuits of super-polynomial size, $s = \lambda^{\omega(1)}$. This then requires the following questions to be addressed:

[6] In [Wee25] strong binding is stated with output values instead: it is hard to produce two different accepted pairs (y, π) for the same C and f. We present it with output commitments to match the described chainable commitment view.

– *Preprocessing.* Preprocessing the circuit now takes time $\lambda^{\omega(1)}$, but it must be done when sampling the RSVL instance, which must take polynomial time.

– *Input commitment.* A circuit of size $\lambda^{\omega(1)}$ may have $\lambda^{\omega(1)}$ input wires, and thus V cannot hope to recompute this commitment.

– *Updatability.* The circuit SNARGs of [Wee25] compute the proofs in one shot in $\lambda^{\omega(1)}$ time. We need a notion of an intermediate state, and a method to update proofs without recomputing proofs from scratch.

– *Soundness.* The reduction above no longer runs in polynomial time: a false positive is a single accepted proof, and turning it into two openings requires computing the honest proof.

These obstacles are non-trivial for circuit SNARGs in general, but we do not need SNARGs for all circuits: we only need proofs for one long computation, which we are free to choose. We therefore choose a structured computation that lets us address the efficiency obstacles, which we describe next. We return to the soundness obstacle when establishing hardness of the resulting RSVL instances.

2.2 Tree Computations

We consider computations that can be represented as a complete binary tree of small circuits, such that all internal nodes are identical, as are the leaf nodes. Such a circuit has a compact description: the leaf circuit, the internal-node circuit and the depth. The simplest example, and the one that gives our hard computations, is that of an exhaustive search for some predicate.

In particular, let $\varphi : \{0,1\}^p \to \{0,1\}$ be a predicate, for instance a 3CNF formula. We search for a witness for φ along a complete binary tree of depth p. The nodes of the tree are indexed by strings $\eta \in \{0,1\}^{\leq p}$, the root is the empty string, the children of η are $\eta 0$ and $\eta 1$, and the leaves are the addresses $a \in \{0,1\}^p$. Each leaf checks whether its address is a witness, and each internal node passes up the witness found by its left child if there is one, and the result of its right child otherwise. Formally, each node outputs a pair of a flag and a candidate:

Leaf a: Output $u_a := (1, a)$ if $\varphi(a) = 1$, and $u_a := (0, 0^p)$ otherwise.

Internal node η: Output $u_\eta := u_{\eta 0}$ if the flag of $u_{\eta 0}$ is 1, and $u_\eta := u_{\eta 1}$ otherwise.

The root outputs the first witness in lexicographic order, if one exists. The computation has 2^p leaves but a short description: every leaf runs the same small circuit on its own address, and every internal node runs the same small circuit on its children's outputs. In particular, all subtrees of the same height compute the same circuit, and differ only in the addresses that their leaves read. We call such computations *tree computations*. We formalize this in the main body as regular tree circuits (Definition 3.8), where we also let the circuit at an internal node depend on its height.

Unlike a general circuit, a tree computation has no external input: the predicate φ is hardcoded into the leaf circuit, and each leaf reads only its own address. The structure by itself does not yet resolve any of the obstacles above. For instance, the addresses play the role of the input, so there are still 2^p of them to commit to. However, the tree does suggest a natural intermediate state, which we describe next.

Intermediate States of Tree Computations. We evaluate the leaves in lexicographic order, as in [CHK+19a, CHK+19b, EFKP20], following the incrementally verifiable computation of [Val08]. After the first t leaves have been evaluated, we keep track of the largest subtrees whose leaves have all been evaluated. There is one such subtree for each 1-bit of t, with 2^j leaves for bit j. For instance, with $p = 3$ and $t = 6 = (110)_2$, these are the subtrees rooted at 0, with four leaves, and at 10, with two leaves. Their roots

form the depth-first *frontier* [EFKP20, BCH+22] (Fig. 2).[7] The state at time t stores the output u_η of each frontier node η, so it has at most p entries. To advance the state, we evaluate the next leaf, and then, as long as the last two stored nodes are siblings, replace them by their parent. Each update therefore evaluates one leaf circuit and at most p internal-node circuits. At time 2^p, the frontier consists of the root alone, whose output is the result of the search.

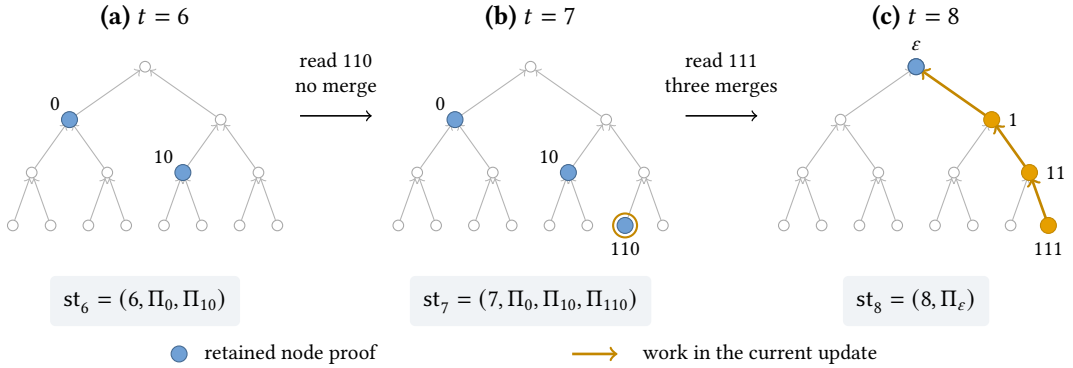


Figure 2: Three successive frontier states for $p = 3$. Filled blue nodes are the frontier nodes, and Π_η denotes the output u_η of node η together with its proof. The circled leaf in (b) and the orange path in (c) mark the work done by each update. Reading 110 adds a leaf proof. Reading 111 creates its leaf proof and merges $110, 111$ into 11, then $10, 11$ into 1, then $0, 1$ into the root ε. Only the root proof remains at $t = 8$. These merges are the carries of the binary counter $(110)_2 \to (111)_2 \to (1000)_2$.

For the template, each stored output must come with a proof, and replacing two siblings by their parent must merge their two proofs into a proof for the parent, without recomputing the subtree. We capture what this requires of the commitment in the notion of a *mergeable functional commitment*, which we describe next.

2.3 Mergeable Functional Commitments

To motivate mergeable functional commitments, consider the description above for the intermediate states, where there are multiple nodes in a state, each with its corresponding proof that may need to be merged when the state is updated. For simplicity, we assume from now on that each node runs a single layer, a function to which the commitment opens directly: F at the leaves and G at the internal nodes. Several layers per node are handled by chaining one proof per layer, as in [Wee25].

Now consider an internal node $\eta \in \{0,1\}^{p-1}$, i.e., a node that is the parent of two leaves. It applies the function G to the outputs of its child nodes $u_{\eta 0}$ and $u_{\eta 1}$. We write $C \xrightarrow{\mathsf{F}} C'$ for the statement that C' is a commitment to F applied to the vector committed in C. To determine what constitutes a *merge*, consider the setting where the leaf $\eta 1$ completed, with corresponding output $u_{\eta 1}$ and proof $\pi_{\eta 1}$ for the statement $C_{\eta 1} \xrightarrow{\mathsf{F}} C'_{\eta 1}$, where $C_{\eta 1}$ and $C'_{\eta 1}$ are the commitments to the address $\eta 1$ and to $u_{\eta 1}$. The state also holds the output $u_{\eta 0}$ and proof $\pi_{\eta 0}$ of its sibling, for the analogous statement $C_{\eta 0} \xrightarrow{\mathsf{F}} C'_{\eta 0}$. The merge step then constitutes the following (Fig. 3):

- From two proofs for the circuit F to a single proof for the circuit $\mathsf{F}^{\otimes 2}$. This in turn requires a way to merge the input commitments of two F circuits into an input commitment to their concatenation: from $C_{\eta 0}$ and $C_{\eta 1}$, we want to obtain the commitment C_η to $(\eta 0, \eta 1)$. Likewise, $C'_{\eta 0}$ and $C'_{\eta 1}$ should

[7] Equivalently, for $t < 2^p$, let a be the next leaf to be evaluated. The frontier consists of the left siblings of a and of its ancestors, ordered from left to right [EFKP20]. For $t = 6$, the next leaf is 110, and the left siblings of 11 and 1 are 10 and 0.

give the commitment C^*_η to $(u_{\eta 0}, u_{\eta 1})$, and the merged proof is a proof for $\mathsf{F}^{\otimes 2}$ with input commitment C_η and output commitment C^*_η.

– A new proof for G applied to $(u_{\eta 0}, u_{\eta 1})$, for the statement $C^*_\eta \xrightarrow{\mathsf{G}} C'_\eta$, where C'_η is the commitment to u_η. Since the input $(u_{\eta 0}, u_{\eta 1})$ is explicit and short, this proof is computed directly.

At this height, the prover could also prove $\mathsf{F}^{\otimes 2}$ from scratch. At a node of height h, however, the same step concerns $\mathsf{F}^{\otimes 2^h}$ on the 2^h leaves of its subtree. The first item must therefore use only the two proofs and the commitments, and not the committed vectors.

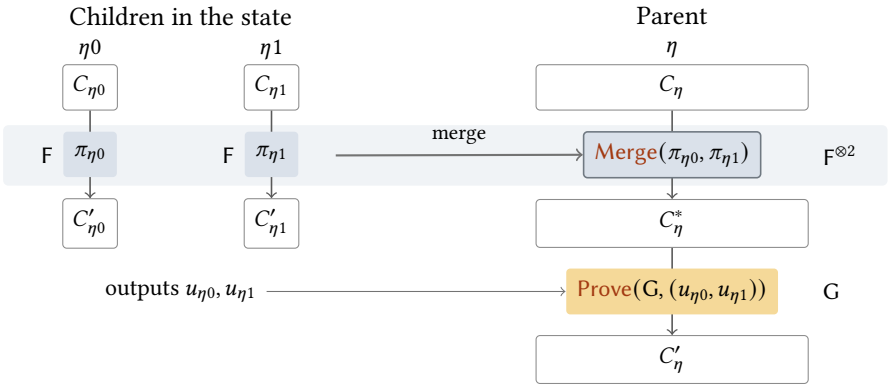


Figure 3: The merge step at a parent η of two leaves, with one layer per node. The proofs $\pi_{\eta 0}$ and $\pi_{\eta 1}$ for F are merged into a proof for $\mathsf{F}^{\otimes 2}$, using only the proofs and the commitments, and G is proved directly on the outputs $(u_{\eta 0}, u_{\eta 1})$.

The same merge applies at every height. The proof for a node of height h consists of one proof for each of the $h+1$ layers below it: $\mathsf{F}^{\otimes 2^h}$, followed by $\mathsf{G}^{\otimes 2^{h-1}}, \ldots, \mathsf{G}^{\otimes 2}, \mathsf{G}$. When two siblings are replaced by their parent, the proofs of the children are merged layer by layer, and only the parent's own copy of G is proved directly. Each update therefore performs at most p merges for each of the at most p new parents, which addresses updatability.

Efficient verification. The verifier checks the proof for a node η of height h as a chain. Starting from the commitment C_η to the 2^h addresses below η, it checks the proof for each layer in turn and recovers its output commitment, which is the input commitment of the next layer. It accepts if the last commitment is the commitment to the claimed output u_η. Two parts of this check depend on the size of the subtree, and correspond to the remaining efficiency obstacles:

1. *Input commitment.* The verifier must compute the input commitment C_η itself, in polynomial time, without listing the 2^h addresses.

2. *Preprocessing.* In [Wee25], the verifier preprocesses each layer into a short *layer digest*, in time proportional to the size of the layer. Here the layers $\mathsf{F}^{\otimes 2^h}$ and $\mathsf{G}^{\otimes 2^{h-1}}$ have size proportional to 2^h. We instead require that the digest of $\mathsf{F}^{\otimes 2^{h+1}}$ can be computed from the digest of $\mathsf{F}^{\otimes 2^h}$, and likewise for G, so that the digests for all heights up to p are computed with p such steps when the instance is sampled.

Mergeable functional commitments. We collect these requirements in the following algorithms, which we formalize in Definition 5.5. We state them for the layer F; the same algorithms apply to G.

- BaseCom($\mathbf{x}$) commits to a short vector $\mathbf{x}$.
- Node(C_0, C_1) merges two commitments into a commitment to the concatenation of the committed vectors.
- InputCom(p, η) computes the commitment C_η to the addresses below η, in polynomial time.
- Digest(F) computes the digest of one copy of F.
- DoubleDigest(F, σ) computes the digest of 2^{h+1} copies of F from the digest σ of 2^h copies.
- Prove(F, $\mathbf{x}$) computes the proof for one copy of F on an explicit input $\mathbf{x}$.
- Merge(F, σ, (C_0, π_0), (C_1, π_1)) merges two proofs for 2^h copies into a proof for 2^{h+1} copies, from the input commitments and the proofs alone.
- VerifyLayer(F, σ, C, π) decides whether to accept the proof π for the input commitment C, given the digest σ.
- OutputCom(F, σ, C, π) outputs the output commitment that π claims.

Merge is only required to combine honestly generated proofs: merging arbitrary accepted proofs need not give an accepted proof.

Binding. For security, we require that it is hard to find two different accepted proofs for the same layer, number of copies and input commitment. This is the strong binding of [Wee25], now also for merged proofs. We also require that it is hard to find two different inputs to BaseCom or Node with the same commitment.

We describe the ideas for constructing such a commitment from SIS in Section 2.6, and we assume the existence of such a commitment to build hard instances of RSVL.

2.4 From Mergeable Functional Commitments to RSVL

We now assume a mergeable functional commitment, and use it to construct the RSVL instance. The construction mostly follows prior work [CHK+19a, CHK+19b, EFKP20, BCH+22]: the vertices are the intermediate states of the tree computation, and the proofs on the frontier are now node proofs built from the mergeable functional commitment. When the instance is sampled, the digests of $\mathsf{F}^{\otimes 2^h}$ and $\mathsf{G}^{\otimes 2^h}$ for every $h \le p$ are computed with Digest and DoubleDigest, and hardwired into the circuits.

Node proofs. The proof for a node η of height h is $\Pi_\eta = (u_\eta, \pi_{\eta,0}, \dots, \pi_{\eta,h})$, consisting of the output of η and one layer proof for each layer below η: $\pi_{\eta,0}$ for F, and $\pi_{\eta,1}, \dots, \pi_{\eta,h}$ for G. The verifier starts from the input commitment $C_\eta = \mathsf{InputCom}(p, \eta)$. It checks each layer proof against the digest of its layer with VerifyLayer, and obtains the next commitment from that proof with OutputCom. The intermediate commitments are therefore not part of the node proof: each is determined by the preceding layer proof. That is, the verifier checks the chain

$$C_\eta = C_{\eta,0} \xrightarrow{\pi_{\eta,0}} C_{\eta,1} \xrightarrow{\pi_{\eta,1}} \cdots \xrightarrow{\pi_{\eta,h}} C_{\eta,h+1},$$

and accepts if $C_{\eta,h+1}$ is the commitment C'_η to u_η, which it computes with BaseCom and Node. For $h = 1$, this is the chain of Fig. 3, with $C_{\eta,1} = C^*_\eta$. This takes polynomial time, even though the subtree has 2^h leaves. To merge the proofs for two siblings, the prover merges their corresponding layer proofs layer by layer with Merge, as in Fig. 3, and proves only the final application of G directly with Prove, on the explicit outputs $(u_{\eta 0}, u_{\eta 1})$.

The instance. A vertex is a state of the tree computation,

$$v_t = \big(t, (\Pi_\eta)_{\eta \in \mathrm{frontier}(t)}\big),$$

where frontier(t) is the frontier at time t (Fig. 2), and the line has length $L = 2^p$. The successor evaluates the next leaf a and proves it, $\Pi_a = (\mathsf{F}(a), \mathsf{Prove}(\mathsf{F}, a))$, and then, as long as the last two node proofs are for siblings, replaces them by the proof for their parent, obtained with Merge and Prove as described above. It runs in polynomial time, since each update performs at most p merges of node proofs. The verifier $\mathsf{V}(v, t)$ checks that v has the form above for the time t, and verifies each node proof. At time 2^p, the frontier consists of the root, so the sink v_{2^p} contains the result of the search.

Uniqueness of the vertex. Suppose that $v \neq v'$ are two vertices accepted at the same time t. They differ in the proof for some frontier node η, and both verifications of this proof start from the same commitment $C_\eta = \mathsf{InputCom}(p, \eta)$. At the first layer proof on which they differ, the input commitments and the digests agree, so the two layer proofs break binding. If all layer proofs agree, so do all derived commitments, and the two outputs are different vectors with the same commitment $C_{\eta,h+1}$, which gives a collision for BaseCom or Node. This covers every value stored in the vertex, and neither vertex needs to be honest.

From uniqueness to a hard line. A false positive is a single accepted vertex $v \neq v_t$, while binding is broken only by two. The honest vertex v_t is also accepted, so v and v_t together break binding. This is why strong binding suffices, without soundness: a false positive may encode a wrong computation, and we never need to rule this out directly. However, a reduction that compares v with v_t must first compute v_t, in t steps, which is not polynomial when t is close to 2^p. This issue does not arise in [CHK+19a, CHK+19b, EFKP20, BCH+22], where the Fiat–Shamir proofs on the frontier are unambiguously sound: a single accepted proof other than the prescribed one already breaks the assumption, so the reduction never computes the honest vertex. For generic computations, [CFJW25] handle it with puzzles of increasing difficulty, following [BPSS23].

Finally, as in the template (Section 2.1), we make the sink hard to find by choosing a computation whose output is hard to compute: an accepted sink then reveals this output, except with the negligible probability controlled by the binding argument. We describe the hard computations that we use, and our two ways around the issue above, in the next subsection.

2.5 Providing the Hard Computation for RSVL

Recall that an adversary for the RSVL instance may output either the sink or a false positive. To make the sink hard to find, we choose a computation whose output is hard to compute, since an accepted sink reveals this output. For false positives to be hard to find, a reduction must turn a single false positive into a break of binding. As we saw in Section 2.4, the latter reduction computes the honest vertex at the time of the false positive, which may take up to 2^p steps.

Complexity leveraging. The most direct approach is to assume that binding holds even against adversaries that can afford this computation. Given a false positive v at time t, the reduction computes the honest vertex v_t in $t \leq 2^p$ steps, and v and v_t break binding. If the mergeable functional commitment is binding against adversaries of size $2^p \cdot \mathrm{poly}(\lambda)$, a single hard computation of length 2^p then suffices, and the sink is hard to find as long as this computation is hard for polynomial-size adversaries (Theorem 4.7). Since binding reduces to SIS, we obtain this from subexponential hardness of SIS by setting the dimension accordingly. A second SIS instance, of much smaller dimension and solved by exhaustive search, then provides the hard computation, which gives Theorem 1.2 from subexponential SIS alone (Section 7.3).

Puzzles of increasing difficulty. To rely only on polynomial hardness of SIS, the reduction must compute only polynomially many steps. Following Bitansky, Paneth, Shamir and Solomon [BPSS23] and [CFJW25],

the instance runs computations on puzzles of difficulty $2^0, 2^1, \ldots, 2^H$ in parallel, for $H = \log^2 \lambda$, one step of each per vertex, and freezes each once it completes. The range up to 2^H eventually contains every polynomial difficulty, although it is much smaller than 2^λ. Suppose that for every polynomial-size adversary, some polynomial difficulty t is hard for this adversary. A false positive before time t breaks binding, since the reduction computes the honest vertex in polynomial time. An accepted vertex after time t contains the frozen output of the puzzle of difficulty t. This output is correct by the same argument, since the reduction only replays this frozen computation for t steps, without computing the full vertex. So the adversary solves this puzzle (Theorem 4.5).

Fine-grained hard computations. It remains to find such puzzles. [BPSS23] obtain puzzles of increasing difficulty from worst-case hardness, but their construction uses fully homomorphic encryption (FHE), which implies public-key encryption, contrary to our goal. Like [CFJW25], we instead assume such hardness directly, as a fine-grained average-case hardness assumption: for every polynomial bound, some polynomial-time computation in a uniform family is hard on random inputs for circuits of that size. Since our proofs only apply to tree computations, we additionally require each puzzle to be computed by a tree computation. We give two candidates (Section 7):

- *Planted search.* Exhaustive search for a planted witness of length $\Theta(\log \lambda)$, such as a preimage of an exponentially hard one-way function, finishes in polynomial time, but is hard for any fixed polynomial budget once the length is a large enough multiple of $\log \lambda$.
- *Clique polynomials.* This candidate replaces search by summation: the leaves evaluate the contributions of vertex tuples to a weighted k-clique polynomial, and the internal nodes add them. Average-case hardness of these values is motivated by the clique-parity conjecture of Boix-Adserà, Brennan and Bresler [BBB19].

Together with polynomially hard SIS, either candidate gives our first result, Theorem 1.1. Finally, following Bitansky and Gerichter [BG20], exhaustive search for a satisfying assignment gives instances that are hard in the worst case, assuming polynomially hard SIS and the non-uniform exponential time hypothesis (Section A).

2.6 A Mergeable Functional Commitment from SIS

We now describe our construction of a mergeable functional commitment from SIS, starting from the functional commitment of Wee [Wee25]. Instead of presenting the final construction directly, we proceed in a few steps. These address merging and preprocessing for efficient verification together, since a single identity gives both the merged digest and the merged proof, while preserving binding. Finally, we describe the input commitments. Along the way, we point out where each algorithm of Section 2.3 appears. The formal construction and its analysis are in Section 6.

In this section, terms that depend only on the setup and the circuit, and can therefore be computed in preprocessing, are shown in blue. Terms that depend on the vector being committed, such as the commitment and its openings, are shown in red, and are not part of preprocessing.

Step 1: a one-level commitment. We start with the base level commitment, BaseCom. Fix a dimension d, polynomial in λ. To commit to a vector $\mathbf{x}$ of length ℓ, the setup contains a matrix $\mathbf{W}_i \in \{0,1\}^{d \times d}$ and a *verifier block* $\mathbf{V}_i \in \{0,1\}^{d \times d}$ for each position i. The commitment is the integer matrix

$$\mathbf{C} = \sum_i x_i \mathbf{W}_i.$$

An opening of position i is a short matrix $\mathbf{Z}_i(\mathbf{x})$ with

$$\mathbf{B}\,\mathbf{Z}_i(\mathbf{x}) = \begin{bmatrix} \mathbf{C}\,\mathbf{V}_i \\ -x_i\mathbf{I}_d \end{bmatrix} \begin{array}{l} \longleftarrow \text{ ties the opening to } \mathbf{C} \\ \longleftarrow \text{ claimed coordinate} \end{array} \tag{1}$$

for the public matrix

$$\mathbf{B} = \begin{bmatrix} \cdots & \mathbf{W}_j\mathbf{V}_k & \cdots \\ \cdots & -\delta_{jk}\mathbf{I}_d & \cdots \end{bmatrix},$$

whose block columns are indexed by pairs (j,k) of positions. The opening $\mathbf{Z}_i(\mathbf{x})$ has the block $x_j\mathbf{I}_d$ in block row (j,i) for every j, and zero blocks elsewhere. The top block row of $\mathbf{B}\mathbf{Z}_i(\mathbf{x})$ is then $\sum_j x_j\mathbf{W}_j\mathbf{V}_i = \mathbf{C}\mathbf{V}_i$, while the bottom block row keeps only the block row (i,i), which gives $-x_i\mathbf{I}_d$.

This is the commitment scheme of [Wee25], with two differences. First, [Wee25] computes modulo q, with uniform matrices $\mathbf{W}_i$, whereas we work over the integers, with binary matrices $\mathbf{W}_i$, which helps control the norms of commitments and openings. Second, $\mathbf{B}$ uses the identity matrix $\mathbf{I}_d$ where [Wee25] uses the gadget matrix $\mathbf{G}$. Looking ahead, [Wee25] cancels the gadget matrix with the bit decomposition $\mathbf{G}^{-1}$, which is not linear, and working over the integers lets us avoid it.

For binding, suppose that two short openings $\mathbf{Z}$ and $\mathbf{Z}'$ of the same commitment $\mathbf{C}$ at position i claim different values. Their difference $\mathbf{D} = \mathbf{Z} - \mathbf{Z}'$ is short and nonzero, and since both satisfy (1) with the same top block row $\mathbf{C}\mathbf{V}_i$, the top block row of $\mathbf{B}\mathbf{D}$ vanishes. Thus $\mathbf{D}$ gives an SIS solution for the top half $[\mathbf{W}_1\mathbf{V}_1|\mathbf{W}_1\mathbf{V}_2|\cdots]$ of $\mathbf{B}$, which in turn gives an SIS solution for $\mathbf{W} = [\mathbf{W}_1|\mathbf{W}_2|\cdots]$ or for $\mathbf{V} = [\mathbf{V}_1|\mathbf{V}_2|\cdots]$. By setting $\mathbf{W}$ and $\mathbf{V}$ to be the bit decompositions of uniform matrices modulo q, this gives a solution to standard SIS.

The setup and the openings grow with ℓ, which we ignore until Step 3. Finally, the commitment is linear, $\mathrm{Com}(\mathbf{x}) = \sum_o x_o\mathrm{Com}(\mathbf{e}_o)$, where the *basis commitment* $\mathrm{Com}(\mathbf{e}_o) = \mathbf{W}_o$ depends only on the setup. This view of the commitment will be useful shortly.

Step 2: layer proofs and their digests. As a next step towards a functional commitment, we extend openings of positions to openings of functions. Following [Wee25], we represent Boolean circuits as layers of quadratic polynomials over the integers, where a constant wire carrying the value 1 lets us express constant and linear terms in this form (Section 5.1). A layer f, such as F or G, has coordinates $(\mathrm{f})_o(\mathbf{x}) = \sum_{i,j}\alpha_{ijo}x_ix_j$, and a *gate* is a triple (i,j,o) with $\alpha_{ijo} \neq 0$. A gate is proved by combining two openings so that an intermediate term cancels. We show such intermediate terms in lavender: each appears in two equations, and cancels when the two are combined. We write $\overline{\mathbf{B}}$ and $\underline{\mathbf{B}}$ for the top and bottom block rows of $\mathbf{B}$, so that (1) reads $\overline{\mathbf{B}}\mathbf{Z}_i(\mathbf{x}) = \mathbf{C}\mathbf{V}_i$ and $\underline{\mathbf{B}}\mathbf{Z}_i(\mathbf{x}) = -x_i\mathbf{I}_d$. Multiplying the opening of i on the right by $\mathbf{M}\mathbf{V}_j$, for a matrix $\mathbf{M}$, and the opening of j by x_i gives

$$\mathbf{B}\,\mathbf{Z}_i(\mathbf{x})\mathbf{M}\mathbf{V}_j = \begin{bmatrix} \mathbf{C}\mathbf{V}_i\mathbf{M}\mathbf{V}_j \\ -x_i\mathbf{M}\mathbf{V}_j \end{bmatrix}, \qquad \mathbf{B}\,x_i\mathbf{Z}_j(\mathbf{x}) = \begin{bmatrix} x_i\mathbf{C}\mathbf{V}_j \\ -x_ix_j\mathbf{I}_d \end{bmatrix}.$$

Let $\mathbf{B}^*$ be $\mathbf{B}$ stacked with a copy of itself one block row lower and one block column to the right,

$$\mathbf{B}^* := \begin{bmatrix} \overline{\mathbf{B}} & \mathbf{0} \\ \underline{\mathbf{B}} & \overline{\mathbf{B}} \\ \mathbf{0} & \underline{\mathbf{B}} \end{bmatrix},$$

so that stacking the two openings gives

$$\mathbf{B}^*\begin{bmatrix} \mathbf{Z}_i(\mathbf{x})\mathbf{M}\mathbf{V}_j \\ x_i\mathbf{Z}_j(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \mathbf{C}\mathbf{V}_i\mathbf{M}\mathbf{V}_j \\ x_i\mathbf{C}\mathbf{V}_j - x_i\mathbf{M}\mathbf{V}_j \\ -x_ix_j\mathbf{I}_d \end{bmatrix}.$$

With $\mathbf{M} = \mathbf{C}$, the two intermediate terms cancel, and the stacked matrix is an opening of the product $x_i x_j$. Weighting each gate by its basis commitment and summing over the gates gives

$$\mathbf{B}^* \sum_{i,j,o} \alpha_{ijo} \begin{bmatrix} \mathbf{Z}_i(\mathbf{x})\mathbf{C}\mathbf{V}_j \\ x_i \mathbf{Z}_j(\mathbf{x}) \end{bmatrix} \mathsf{Com}(\mathbf{e}_o) = \begin{bmatrix} \mathbf{C}\,\mathbf{V}_\mathsf{f}[\mathbf{C}] \\ \mathbf{0} \\ -\mathsf{Com}(\mathsf{f}(\mathbf{x})) \end{bmatrix}$$

for the matrix $\mathbf{V}_\mathsf{f}[\mathbf{M}] := \sum_{i,j,o} \alpha_{ijo}\, \mathbf{V}_i\, \mathbf{M}\, \mathbf{V}_j\, \mathsf{Com}(\mathbf{e}_o)$. The last block row is the output commitment.

The layer digest. The matrix $\mathbf{V}_\mathsf{f}[\mathbf{C}]$ plays the role of a digest of the layer, but it depends on $\mathbf{C}$, so it cannot be computed ahead of time. However, $\mathbf{V}_\mathsf{f}[\mathbf{M}]$ is linear in $\mathbf{M}$. Writing $\mathbf{C} = \sum_{\mu,\nu} c_{\mu\nu}\mathbf{E}_{\mu\nu}$ in terms of the elementary matrices $\mathbf{E}_{\mu\nu}$, the verifier computes $\mathbf{V}_\mathsf{f}[\mathbf{C}] = \sum_{\mu,\nu} c_{\mu\nu}\mathbf{V}_\mathsf{f}[\mathbf{E}_{\mu\nu}]$ from the table of values $\mathbf{V}_\mathsf{f}[\mathbf{E}_{\mu\nu}]$, which depends only on the layer and the setup. This table is the *layer digest*, which the verifier computes in preprocessing, in time proportional to the number of gates of the layer, up to a factor polynomial in d. Afterwards, the verifier evaluates $\mathbf{V}_\mathsf{f}[\mathbf{C}]$ in time polynomial in d, independently of the size of the layer.

In [Wee25], the digest has the same form, with $\mathbf{C}$ and the weight $\alpha_{ijo}\mathsf{Com}(\mathbf{e}_o)$ replaced by their bit decompositions $\mathbf{G}^{-1}(\mathbf{C})$ and $\mathbf{G}^{-1}(\alpha_{ijo}\mathsf{Com}(\mathbf{e}_o))$. We show such bit decompositions in pink to indicate that they are not linear, and as we see below, this prevents the digest of a doubled layer from being computed from the old digest. Since we use $\mathbf{I}_d$ in place of $\mathbf{G}$, we do not need $\mathbf{G}^{-1}$, and do not run into this issue. In either case, the digest is a sum over all gates, and $\mathsf{f}^{\otimes 2^h}$ has 2^h times as many gates as f, for h up to p.

Why doubling changes the digest. Before turning to the issue with bit decompositions, we first describe how the digest changes when a layer is doubled. As noted above, computing the digest of $\mathsf{f}^{\otimes 2^h}$ takes time proportional to its number of gates, which is 2^h times that of f, so the verifier cannot compute it directly for large h (Section 2.3). Instead, we would like DoubleDigest to compute the digest of $\mathsf{f}^{\otimes 2^{h+1}}$ from the digest of $\mathsf{f}^{\otimes 2^h}$, without going over the gates (Fig. 4). The two copies in $\mathsf{f}^{\otimes 2^{h+1}}$ have identical gates, but they act on different positions: copy b reads and writes positions in half b of vectors that are twice as long. With the one-level commitment, this is a problem, since every position has its own setup matrices. The terms of copy 1 involve the verifier blocks $\mathbf{V}_i$ and basis commitments $\mathbf{W}_o$ of positions in the second half, and none of these matrices appear in the old digest. To compute the new digest from the old one, we would like the verifier block and basis commitment of each position in the long vector to be obtained from those of the corresponding position in the half by a fixed map, the same for every position. We enable this by committing to long vectors recursively, as in a Merkle tree, which we describe next.

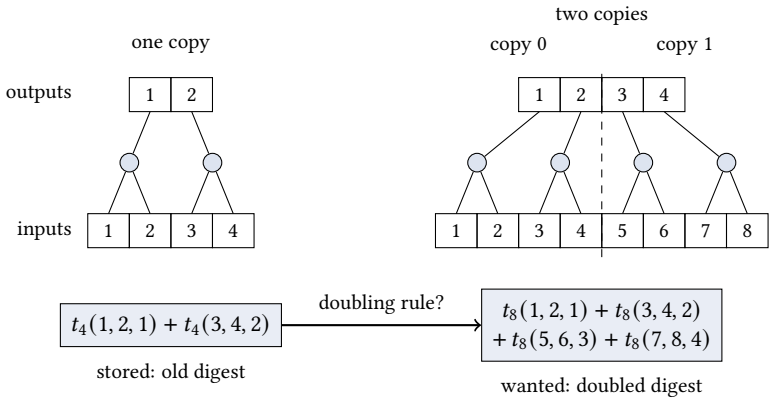


Figure 4: Doubling the digest of a layer with two gates. A gate on inputs i, j and output o contributes a term $t_\ell(i,j,o)$ for vectors of length ℓ. A doubling rule must compute the new sum from the old one alone.

Step 3: a linear Merkle-style recursion. So far, the setup and the openings grow with the length ℓ of the committed vector, and every position has setup matrices of its own. To shrink both, and to relate the positions of a long vector to those of its halves, we commit to vectors recursively. A vector $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1)$ of length $\ell > L_0$ is committed by first committing to its halves, $\mathbf{C}_b = \mathsf{Com}(\mathbf{x}_b)$, and then committing to the pair with BaseCom, now only on vectors of a fixed length L_0. This is the algorithm Node of Section 2.3, and the setup is now that of a single BaseCom.

The remaining question is how to pass the child commitments $\mathbf{C}_0, \mathbf{C}_1$ to BaseCom. In [Wee25], BaseCom receives their bits, $\mathbf{C} = \mathsf{BaseCom}(\mathsf{bits}([\mathbf{C}_0|\mathbf{C}_1]))$, which keeps its input short, as SIS-binding requires. However, this breaks the doubling rule we want. Consider a position o in half b of a vector twice as long. Its basis commitment is now BaseCom applied to the bits of its basis commitment in the half, so it is a fixed matrix times $\mathbf{G}^{-1}(\mathsf{Com}(\mathbf{e}_o))$. Similarly, its verifier block becomes a fixed matrix times $\mathbf{G}^{-1}(\mathbf{V}_o)$. Each term $\mathbf{V}_i\mathbf{M}\mathbf{V}_j\mathsf{Com}(\mathbf{e}_o)$ of the digest is therefore replaced by a product involving the bit decompositions of its factors. The old digest only stores the sum of the original terms, and since bit decomposition is not linear, these bit decompositions cannot be obtained from the sum. We therefore do not know how to compute the doubled digest from the old one when using the binary decomposition.

We instead commit directly to the entries of the children, and set

$$\mathsf{Node}(\mathbf{C}_0, \mathbf{C}_1) := \mathsf{BaseCom}(\mathsf{vec}([\mathbf{C}_0|\mathbf{C}_1])).$$

This makes the whole tree linear, but BaseCom only binds short inputs, and the entries of the children are integers that grow with the height. Each level of the tree multiplies their magnitude by a factor polynomial in d, so we pay for linearity with norms that accumulate over the levels, and hence with a larger SIS norm bound (see the end of this section). By linearity, the basis commitment of a position in half b propagates by a fixed linear map $\mathbf{N}_b$, given by $\mathsf{vec}(\mathsf{Node}(\mathbf{D}_0, \mathbf{D}_1)) = \mathbf{N}_0\,\mathsf{vec}(\mathbf{D}_0) + \mathbf{N}_1\,\mathsf{vec}(\mathbf{D}_1)$.

The openings are those of a Merkle tree. An opening of position i in half b consists of an opening of the parent commitment $\mathbf{C}$ to the child commitment $\mathbf{C}_b$, followed by the opening $\mathbf{Z}_i(\mathbf{x}_b)$ of position i in the half. The first opening is computed from the explicit child commitments alone. By the compactification trick of [Wee25], it is a matrix $\mathbf{Z}'_b$ with $\mathbf{B}\mathbf{Z}'_b = \left[\begin{smallmatrix}\mathbf{C}\mathbf{H}_b\\ -\mathbf{C}_b\end{smallmatrix}\right]$ for a fixed matrix $\mathbf{H}_b$, so that its last block row claims the child commitment $\mathbf{C}_b$. Consider first the case where each half has the base length L_0. Multiplying $\mathbf{Z}'_b$ by $\mathbf{V}_i$ and stacking it above the opening in the half, as in Step 2, gives

$$\mathbf{B}^* \begin{bmatrix} \mathbf{Z}'_b\mathbf{V}_i \\ \mathbf{Z}_i(\mathbf{x}_b) \end{bmatrix} = \begin{bmatrix} \mathbf{C}\,\mathbf{H}_b\mathbf{V}_i \\ \mathbf{C}_b\mathbf{V}_i - \mathbf{C}_b\mathbf{V}_i \\ -x_i\mathbf{I}_d \end{bmatrix} = \begin{bmatrix} \mathbf{C}\,\mathbf{H}_b\mathbf{V}_i \\ \mathbf{0} \\ -x_i\mathbf{I}_d \end{bmatrix}.$$

This is an opening of position i in the long vector, of the same form as (1) apart from the zero block row. The only change is the verifier block, which becomes $\mathbf{H}_b\mathbf{V}_i$.

Three objects are therefore associated with each position o: its basis commitment, which we write as the *weight vector* $\mathbf{w}_o := \mathsf{vec}(\mathsf{Com}(\mathbf{e}_o)) \in \mathbb{Z}^{d^2}$ with entries $w_{o,r}$, its verifier block $\mathbf{V}_o$, and its opening $\mathbf{Z}_o(\mathbf{x})$. In summary, when a vector becomes half b of a vector twice as long,

$$\mathbf{w}_o \;\mapsto\; \mathbf{N}_b\mathbf{w}_o, \qquad \mathbf{V}_i \;\mapsto\; \mathbf{H}_b\mathbf{V}_i, \qquad \mathbf{Z}_i(\mathbf{x}_b) \;\mapsto\; \begin{bmatrix} \mathbf{Z}'_b\mathbf{V}_i \\ \mathbf{Z}_i(\mathbf{x}_b) \end{bmatrix},$$

where each map takes the objects of a position in the half to those of the same position in the long vector, in the same way for every position. In particular, the new opening uses only the position's own verifier block $\mathbf{V}_i$, together with $\mathbf{Z}'_b$, which is shared by all positions in half b. For a vector of height g, each level adds one block row, and (1) becomes

$$\mathbf{B}_g\,\mathbf{Z}_i(\mathbf{x}) = \begin{bmatrix} \mathbf{C}\,\mathbf{V}_i \\ \mathbf{0} \\ -x_i\mathbf{I}_d \end{bmatrix}, \tag{2}$$

where $\mathbf{B}_g$ is block lower-bidiagonal with one copy of $\mathbf{B}$ per level, so that $\mathbf{B}_1 = \mathbf{B}^*$, and $\mathbf{0}$ has g block rows, one for each cancellation. Binding follows as in Step 1: two different short openings at the same position give images under $\mathbf{B}_g$ that agree in every block row except the last, and their difference gives a short SIS solution.

Step 4: mergeable summaries. With the linear recursion of Step 3, the verifier block and the basis commitment of a position change by fixed linear maps when a layer is doubled. We would like to apply these maps directly to the digest of Step 2,

$$\mathbf{V}_\mathrm{f}[\mathbf{M}] = \sum_{i,j,o} \alpha_{ijo}\, \mathbf{V}_i\, \mathbf{M}\, \mathbf{V}_j\, \mathrm{Com}(\mathbf{e}_o),$$

without going over the gates. For the verifier blocks, this works. Every term of the digest has the form $\mathbf{V}_i\mathbf{M}\mathbf{V}_j$ times a weight, and

$$(\mathbf{H}_b\mathbf{V}_i)\,\mathbf{M}\,(\mathbf{H}_b\mathbf{V}_j) = \mathbf{H}_b\big(\mathbf{V}_i\,(\mathbf{M}\mathbf{H}_b)\,\mathbf{V}_j\big),$$

so the outer $\mathbf{H}_b$ can be taken out of the sum over gates, and the inner one only changes the argument of the table. If the weights *did not change*, the gates of half b would therefore contribute

$$\sum_{i,j,o} \alpha_{ijo}\,(\mathbf{H}_b\mathbf{V}_i)\,\mathbf{M}\,(\mathbf{H}_b\mathbf{V}_j)\,\mathrm{Com}(\mathbf{e}_o) = \mathbf{H}_b\,\mathbf{V}_\mathrm{f}[\mathbf{M}\mathbf{H}_b],$$

which the verifier computes from the old digest at the matrix $\mathbf{M}\mathbf{H}_b$. The weight, however, is a problem. The digest multiplies each basis commitment $\mathrm{Com}(\mathbf{e}_o)$ into the term of its gate as a matrix, whereas $\mathbf{N}_b$ acts on its vectorization $\mathbf{w}_o$. Once the terms are summed, the coordinates of the individual weights are no longer available for $\mathbf{N}_b$ to act on.

We therefore keep a separate table for each coordinate of the weight. Writing $\mathrm{Com}(\mathbf{e}_o) = \sum_r w_{o,r}\mathbf{E}_r$, where $\mathbf{E}_r$ is the elementary matrix at coordinate $r \in [d^2]$, the digest of Step 2 splits as $\mathbf{V}_\mathrm{f}[\mathbf{M}] = \sum_r \mathbf{Q}_r[\mathbf{M}]\,\mathbf{E}_r$, with

$$\mathbf{Q}_r[\mathbf{M}] := \sum_{i,j,o} \alpha_{ijo} w_{o,r}\, \mathbf{V}_i\mathbf{M}\mathbf{V}_j.$$

The new layer digest is the table

$$\sigma = (\mathbf{Q}_r[\mathbf{E}_{\mu\nu}])_{r,\mu,\nu},$$

for $r \in [d^2]$ and $\mu, \nu \in [d]$. It consists of d^4 matrices of size $d \times d$, so its size is polynomial in d and independent of the number of copies. Digest(f) computes it directly for one copy of the layer. For $\mathrm{f}^{\otimes 2^{h+1}}$, the two halves each apply $\mathrm{f}^{\otimes 2^h}$, so every gate lies within one half. In half b, the verifier blocks are multiplied by $\mathbf{H}_b$ and the weight vectors by $\mathbf{N}_b$, which gives

$$\begin{aligned}
\mathbf{Q}_r[\mathbf{M}] &= \sum_{b\in\{0,1\}} \sum_{i,j,o} \alpha_{ijo}\,(\mathbf{N}_b\mathbf{w}_o)_r\,(\mathbf{H}_b\mathbf{V}_i)\,\mathbf{M}\,(\mathbf{H}_b\mathbf{V}_j) \\
&= \sum_{b\in\{0,1\}} \sum_{r'} (\mathbf{N}_b)_{r,r'}\,\mathbf{H}_b\Big(\sum_{i,j,o} \alpha_{ijo} w_{o,r'}\,\mathbf{V}_i\,(\mathbf{M}\mathbf{H}_b)\,\mathbf{V}_j\Big) \\
&= \sum_{b\in\{0,1\}} \sum_{r'} (\mathbf{N}_b)_{r,r'}\,\mathbf{H}_b\,\mathbf{Q}^{(b)}_{r'}[\mathbf{M}\mathbf{H}_b],
\end{aligned} \tag{3}$$

where in the first two lines, the inner sums are over the gates of $\mathrm{f}^{\otimes 2^h}$ in their positions in the half, and $\mathbf{Q}^{(b)}$ is the digest of half b. The last line has no sum over gates, and both halves have the same digest, so DoubleDigest applies (3) with $\mathbf{Q}^{(0)} = \mathbf{Q}^{(1)}$ (Fig. 5). This relies on every gate lying within one half. The

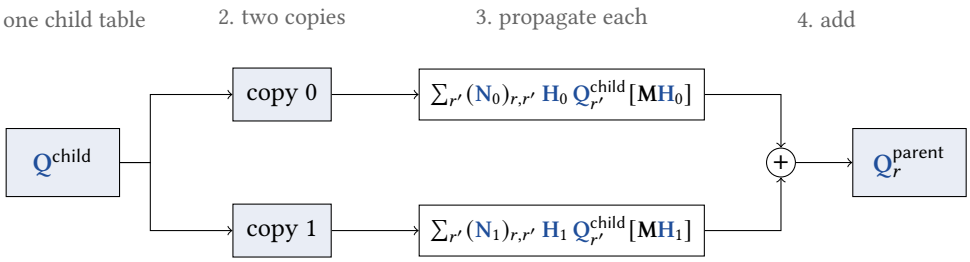


Figure 5: Doubling the layer digest by a merge (3). Both copies are the same layer, so one child table serves both, but copy b sits in half b of the parent and is transformed accordingly. The parent then becomes the child for the next doubling. Layer proofs merge in the same way, with the two children's own tables and one extra block row built from $\mathbf{Z}'$.

parent's own copy of G is different: it is a single copy whose gates combine the outputs of both children, so it is not two copies of a smaller layer and its digest does not split this way. Since its input is explicit and short, it is proved directly, as in Section 2.3.

Layer proofs. The layer proofs split in the same way. Recall from Step 2 that a gate is proved from two openings: the opening of i, multiplied by $\mathbf{M}\mathbf{V}_j$, and the opening of j, multiplied by x_i. Keeping the coordinate r of the weight separate, we sum each kind of opening over the gates on its own, which gives two summaries,

$$\mathbf{T}_r[\mathbf{M}] := \sum_{i,j,o} \alpha_{ijo} w_{o,r}\, \mathbf{Z}_i(\mathbf{x}) \mathbf{M}\mathbf{V}_j, \qquad \mathbf{L}_r := \sum_{i,j,o} \alpha_{ijo} w_{o,r}\, x_i \mathbf{Z}_j(\mathbf{x}),$$

Each satisfies its own equation, as each opening did in Step 2. For every r and every $\mathbf{M}$,

$$\mathbf{B}_g \mathbf{T}_r[\mathbf{M}] = \begin{bmatrix} \mathbf{C}\,\mathbf{Q}_r[\mathbf{M}] \\ \mathbf{0} \\ -\mathbf{M}\mathbf{R}_r \end{bmatrix}, \qquad \mathbf{B}_g \mathbf{L}_r = \begin{bmatrix} \mathbf{C}\,\mathbf{R}_r \\ \mathbf{0} \\ -y_r \mathbf{I}_d \end{bmatrix}, \tag{4}$$

where the intermediate value $\mathbf{R}_r := \sum_{i,j,o} \alpha_{ijo} w_{o,r}\, x_i \mathbf{V}_j$ collects the intermediate terms of Step 2 and links the two equations. Here y_r is coordinate r of $\mathrm{vec}(\mathrm{Com}(\mathrm{f}(\mathbf{x})))$, so the last block rows for $\mathbf{L}_r$ give the output commitment. As in Step 2, the two identities combine at $\mathbf{M} = \mathbf{C}$. Given $\mathbf{C}$, one computes $\mathbf{T}_r[\mathbf{C}]$ from the stored entries $\mathbf{T}_r[\mathbf{E}_{\mu\nu}]$ and $\mathbf{Q}_r[\mathbf{C}]$ from the digest, both by linearity, and stacking the two identities gives

$$\mathbf{B}_{2g+1} \begin{bmatrix} \mathbf{T}_r[\mathbf{C}] \\ \mathbf{L}_r \end{bmatrix} = \begin{bmatrix} \mathbf{C}\,\mathbf{Q}_r[\mathbf{C}] \\ \mathbf{0} \\ -\mathbf{C}\mathbf{R}_r + \mathbf{C}\mathbf{R}_r \\ \mathbf{0} \\ -y_r \mathbf{I}_d \end{bmatrix} = \begin{bmatrix} \mathbf{C}\,\mathbf{Q}_r[\mathbf{C}] \\ \mathbf{0} \\ -y_r \mathbf{I}_d \end{bmatrix},$$

where $\mathbf{B}_{2g+1}$ is again block lower-bidiagonal, and the two identities overlap in one block row, in which the intermediate terms cancel. For $g = 0$, this is the check of Step 2 with $\mathbf{B}_1 = \mathbf{B}^*$. This combined identity alone, however, does not bind the entire stored table. The verifier therefore does not check it directly. Instead, it checks the two identities separately at every stored entry, as described in Step 5, and these checks imply the combined identity. Since $\mathbf{R}_r$ only appears in these cancelling terms, it is not part of the layer proof, and the verifier derives it from $\mathbf{T}_r$, as we see in Step 5. The layer proof is

$$\pi = (\mathbf{T}_r[\mathbf{E}_{\mu\nu}], \mathbf{L}_r)_{r,\mu,\nu},$$

and Prove$(\mathsf{f}, \mathbf{x})$ computes it directly for one copy of the layer. It contains $\mathbf{T}_r$ at every elementary matrix, rather than only at $\mathbf{C}$, since a merge evaluates the tables of the children at the matrices $\mathbf{M}\mathbf{H}_b$. Merge then combines the two child proofs as in (3), using their input commitments and the common layer digest, with one extra block row computed from $\mathbf{Z}'_b$.

Step 5: binding every stored entry. Finally, we return to binding. For the uniqueness argument of Section 2.4, it must be hard to find two different accepted layer proofs for the same input commitment. Since a merge uses every stored entry $\mathbf{T}_r[\mathbf{E}_{\mu\nu}]$, the verifier must bind all of them, and not only their combination at $\mathbf{C}$. As in Steps 1 and 3, an equation of the form

$$\mathbf{B}_g\mathbf{X} = \begin{bmatrix}\mathbf{A}\\ \mathbf{0}\\ *\end{bmatrix},$$

where the verifier can compute the top block row $\mathbf{A}$ itself, binds a short $\mathbf{X}$: two different short solutions have images that agree in every block row except the last, so their difference gives a short SIS solution. Both equations in (4) have this form, once the verifier knows their top block rows. For $\mathbf{T}_r[\mathbf{E}_{\mu\nu}]$, the top block row is $\mathbf{C}\mathbf{Q}_r[\mathbf{E}_{\mu\nu}]$, which VerifyLayer computes from the input commitment and the digest. This binds every stored entry of $\mathbf{T}_r$. For $\mathbf{L}_r$, the top block row is $\mathbf{C}\mathbf{R}_r$, and VerifyLayer obtains $\mathbf{R}_r$ from the entries of $\mathbf{T}_r$ it has just checked: it is the last block row of $\mathbf{B}_g\mathbf{T}_r[\mathbf{I}_d]$, up to sign, where $\mathbf{T}_r[\mathbf{I}_d] = \sum_\mu \mathbf{T}_r[\mathbf{E}_{\mu\mu}]$. The verifier also checks that the last block row of $\mathbf{B}_g\mathbf{T}_r[\mathbf{E}_{\mu\nu}]$ is $-\mathbf{E}_{\mu\nu}\mathbf{R}_r$, so that every stored entry satisfies (4), although the uniqueness argument below does not use this check. This binds $\mathbf{L}_r$, and its last block row gives the output coordinate y_r. In short, the verifier proceeds as

$$(\mathbf{C}, \mathbf{Q}) \xrightarrow{\text{bind}} \mathbf{T}_r[\mathbf{E}_{\mu\nu}] \text{ for all } \mu, \nu \xrightarrow{\text{derive}} \mathbf{R}_r \xrightarrow{\text{with C, bind}} \mathbf{L}_r \xrightarrow{\text{read off}} y_r,$$

and OutputCom outputs the output commitment, the matrix whose vectorization is $(y_r)_r$. Two accepted proofs for the same $\mathbf{C}$ that differ in $\mathbf{T}$ thus give an SIS solution. If they agree in $\mathbf{T}$, they derive the same $\mathbf{R}$, and the same argument applies to $\mathbf{L}$.

Input commitments. It remains to compute the input commitment C_η of a node η, as required in Section 2.3: the verifier must compute the commitment to the inputs of the 2^h leaves below η in polynomial time, without listing them. The key observation is that these inputs are highly structured, and that our commitment is linear. Recall that the input of a leaf is its address together with a coordinate equal to 1, which provides the constant wire of Step 2, padded to the base length L_0. Let $\mathbf{e}_0$ be the unit vector selecting the constant coordinate, and $\mathbf{e}_{j+1}$ the unit vector selecting the address bit a_j, numbered from the least significant bit. For example, take $p = 3$ and the node $\eta = 1$, whose leaves are 100, 101, 110, 111. Every leaf has the constant coordinate and the bit $a_2 = 1$ of η, so every leaf input contains $\mathbf{v} = \mathbf{e}_0 + \mathbf{e}_3$, while the bits a_1 and a_0 vary across the leaves. The concatenated input of η is therefore

$$\begin{aligned}\mathbf{x}_\eta = {} & (\mathbf{v}, \mathbf{v}, \mathbf{v}, \mathbf{v})\\ & + (\mathbf{0}, \mathbf{e}_1, \mathbf{0}, \mathbf{e}_1)\\ & + (\mathbf{0}, \mathbf{0}, \mathbf{e}_2, \mathbf{e}_2),\end{aligned}$$

where each tuple denotes the concatenation of four blocks, and the last two summands encode the bits a_0 and a_1 of each leaf. There is one such summand for each bit that varies below η, and since η has height 2, there are two. In general, the input of a node of height h is the sum of one repeated pattern and one pattern for each of the h varying bits, and by linearity, the input commitment is the sum of the commitments to these patterns.

Each pattern can be committed to without expanding it. Repeated blocks are easy to commit to: starting from $\mathsf{BaseCom}(\mathbf{v})$, each application of $\mathbf{C} \mapsto \mathsf{Node}(\mathbf{C}, \mathbf{C})$ doubles the number of copies. For a varying bit a_j, the pattern consists of 2^j zero blocks followed by 2^j copies of $\mathbf{e}_{j+1}$, repeated across the subtree. We first compute the commitment $\mathbf{C}$ to the 2^j copies of $\mathbf{e}_{j+1}$, combine it with the zero commitment as $\mathsf{Node}(\mathbf{0}, \mathbf{C})$, and then double the result until it covers all 2^h leaves. Each pattern thus takes $O(h)$ calls to Node, and the $h+1$ patterns take $O(p^2)$ calls overall. This gives the algorithm InputCom, without enumerating the leaves. Here we rely on the linearity of our commitment scheme once more

The cost: larger norms. We now return to the cost of removing the bit decompositions. Commitment entries grow by a factor polynomial in d at each level of the tree. For local circuits of polynomial description size and a tree of height at most H, honest layer proofs therefore have norm $2^{O((H+\log\lambda)\log d)}$. This growth does not compound across circuit layers, since the output commitment derived from each honest layer proof equals the commitment to the Boolean output of that layer, as if it had been committed afresh. The verifier checks an entry bound B_{thr}, and two distinct accepted layer proofs give an SIS solution of norm at most $2mB_{\mathsf{thr}}$, where m is polynomial in λ. For the polynomial-hardness route, $H = \log^2\lambda$ and $d = \mathsf{poly}(\lambda)$, so a quasi-polynomial bound $\beta = 2^{O(\log^3\lambda)}$ suffices. This is where limiting the largest puzzle difficulty improves the assumption. For the SIS-only route, we retain $H = \lambda$ and the larger norm bound $\beta = 2^{n^e}$, with n a sufficiently large polynomial in λ. Both parameter choices are verified in Section 6.5.

2.7 Organization

The rest of the paper is organized as follows. In Section 4, we define updatable unambiguous SNARGs (uuSNARG, Definition 4.1) for regular tree circuits (Definition 3.8). They capture the intermediate states and node proofs of Section 2.4, and we formalize them as a separate primitive since they may be of independent interest. In the same section, we combine a uuSNARG with hard computations to obtain hard instances of RSVL and EOML. In Section 5, we construct a uuSNARG from mergeable functional commitments, and in Section 6, we construct a mergeable functional commitment from SIS. We instantiate the hard computations in Section 7. The appendices give worst-case hardness from ETH, a worst-case to average-case reduction for clique polynomials, and hardness of UEOPL.

The primitives of this paper and the connections between them are summarized in Fig. 6. Each box and arrow refers to the definition or section where it appears in this paper, or cites the prior work it comes from.

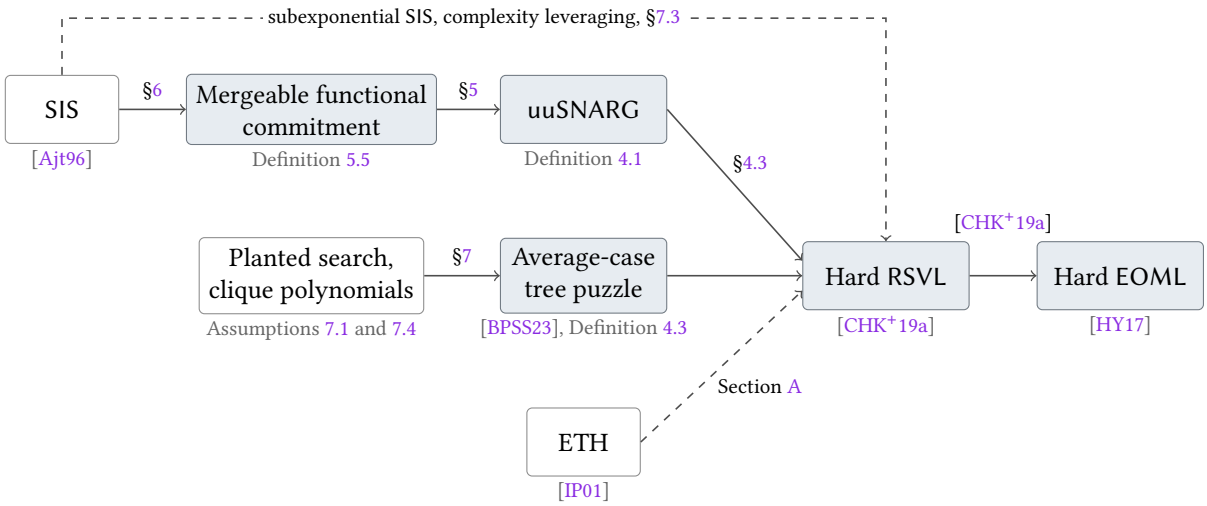


Figure 6: Organization of the paper. Shaded boxes are primitives and white boxes are assumptions. The dashed routes replace the average-case puzzle assumptions. With subexponential SIS, a smaller SIS instance gives a single hard computation, and complexity leveraging replaces the hierarchy of puzzles. With ETH, the uuSNARG gives RSVL instances that are hard in the worst case rather than on average.

3 Preliminaries

Notation. We write $\lambda \geq 2$ for the security parameter and $\mathsf{negl}(\lambda)$ for a negligible function of λ. For $n \in \mathbb{N}$ let $[n] = \{1, \ldots, n\}$, and for integers $a \leq b$ let $[a, b] = \{a, \ldots, b\}$. Vectors are bold lowercase letters, such as $\mathbf{v}$, and matrices are bold uppercase letters, such as $\mathbf{M}$. We write $\mathbf{e}_j$ for the j-th unit vector and $\mathbf{I}_d$ for the $d \times d$ identity matrix. Integer vectors and matrices are measured in the max-entry norm $\|\cdot\|$. We write $\otimes$ for the Kronecker product. Height bounds are rounded up to integers.

Gadget matrix. For a power of two q, the gadget matrix is $\mathbf{G}_n = \mathbf{I}_n \otimes (1, 2, \ldots, q/2) \in \mathbb{Z}^{n \times n \log q}$. For $\mathbf{W}' \in \mathbb{Z}_q^{n \times m}$ with entries in $[0, q)$, we write $\mathbf{G}^{-1}(\mathbf{W}') \in \{0,1\}^{n \log q \times m}$ for its entrywise binary decomposition, so that $\mathbf{G}_n \mathbf{G}^{-1}(\mathbf{W}') = \mathbf{W}' \bmod q$. Since the entries of $\mathbf{W}'$ lie in $[0, q)$, this equality also holds over $\mathbb{Z}$.

Adversaries. Unless stated otherwise, adversaries are non-uniform families $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ of probabilistic polynomial-size circuits, and a hardness statement asserts that every such adversary succeeds with probability at most $\mathsf{negl}(\lambda)$. All probabilities are taken over the adversary's coins as well as over the sampling shown in the experiment.

3.1 Search Problems, RSVL, and EOML

All of our hardness results pass through the following problem. Unlike problems in TFNP, its solutions need not be efficiently checkable: whether a pair (v, i) is a false positive depends on $\mathsf{S}^i(v_0)$, which may take super-polynomial time to compute.

Definition 3.1 (Relaxed Sink-of-Verifiable-Line [CHK+19a])**.** *An instance of* RSVL *consists of*

1. *a Boolean circuit* $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$ *(the successor);*
2. *a Boolean circuit* $\mathsf{V} : \{0,1\}^m \times [0, 2^m - 1] \to \{0,1\}$ *(the verifier);*
3. *an integer* $L \in [0, 2^m - 1]$*; and*
4. *a string* $v_0 \in \{0,1\}^m$*.*

Promise. *For every* $i \in [0, L]$, $\mathsf{V}(\mathsf{S}^i(v_0), i) = 1$.
Solutions. *One of the following:*

1. sink: *a vertex* v *with* $\mathsf{V}(v, L) = 1$*;*
2. false positive: *a pair* $(v, i) \in \{0,1\}^m \times [0, 2^m - 1]$ *with* $\mathsf{V}(v, i) = 1$ *and* $v \neq \mathsf{S}^i(v_0)$.

Definition 3.2 (End-of-Metered-Line [HY17])**.** *An instance of* EOML *consists of a successor circuit* $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$*, a predecessor circuit* $\mathsf{P} : \{0,1\}^m \to \{0,1\}^m$*, a meter circuit* $\mathsf{M} : \{0,1\}^m \to [0, 2^m]$*, and a string* $v_0 \in \{0,1\}^m$*, with the guarantee that* $\mathsf{P}(v_0) = v_0 \neq \mathsf{S}(v_0)$ *and* $\mathsf{M}(v_0) = 1$*. A solution is a vertex* v *satisfying one of the following:*

1. end of line: $\mathsf{P}(\mathsf{S}(v)) \neq v$*, or* $\mathsf{S}(\mathsf{P}(v)) \neq v \neq v_0$*;*
2. false source: $v \neq v_0$ *and* $\mathsf{M}(v) = 1$*;*
3. miscount: $\mathsf{M}(v) > 0$ *and* $\mathsf{M}(\mathsf{S}(v)) - \mathsf{M}(v) \neq 1$*, or* $\mathsf{M}(v) > 1$ *and* $\mathsf{M}(v) - \mathsf{M}(\mathsf{P}(v)) \neq 1$.

EOML lies in CLS $\subseteq$ PPAD [HY17, DP11]. Solutions of the first type are exactly the End-of-Line solutions of the graph given by the successor and predecessor circuits, so hardness of EOML implies hardness of End-of-Line.

A distribution $\mathcal{D}(1^\lambda)$ over RSVL instances, sampled in polynomial time, is *hard* if, for all sufficiently large λ, it is supported on instances satisfying the promise, and every adversary $\mathcal{A}$ outputs a solution to $\mathcal{D}(1^\lambda)$ with probability at most $\mathsf{negl}(\lambda)$. Hard distributions for EOML are defined analogously.

Theorem 3.3 ([CHK+19a]; see also [BCH+22]). *There is a randomized Karp reduction* $(\mathsf{R}_1, \mathsf{R}_2)$ *from* RSVL *to* EOML *with success probability* $|I|^{-O(1)}$.

In this reduction, the instance map R_1 is deterministic. Every vertex of $\mathsf{R}_1(I)$ that is not a self-loop, and in particular every solution, stores at most $\lceil \log L \rceil + 1$ pairs (v, i) accepted by V. Given a solution of $\mathsf{R}_1(I)$, the decoder R_2 picks one of these pairs uniformly at random, and outputs v if the pair is (v, L), and the pair itself otherwise. This is an RSVL solution with probability at least $1/(\lceil \log L \rceil + 1)$ [CHK+19a]. In particular, if $\mathcal{D}$ is a hard distribution over RSVL instances, then $\mathsf{R}_1(\mathcal{D})$ is a hard distribution over EOML instances, since composing a solver for $\mathsf{R}_1(\mathcal{D})$ with R_2 loses only a polynomial factor in its success probability. All of our results are statements about RSVL, followed by an application of Theorem 3.3.

3.2 Short Integer Solutions

We define below the short integer solutions (SIS) problem along with the variations and parameterization that will be relevant for this work.

Definition 3.4 (SIS [Ajt96]). *For lattice parameters* n, m, q, β *that depend on* λ, $\mathsf{SIS}_{n,m,q,\beta}$ *holds against size-*s *adversaries with advantage* δ *if for every adversary* $\mathcal{A}$ *of size* $s(\lambda)$,

$$\Pr\left[\mathbf{Au} = \mathbf{0} \bmod q \ \wedge \ 0 < \|\mathbf{u}\| \le \beta \ : \ \begin{array}{c} \mathbf{A} \leftarrow\!\$\ \mathbb{Z}_q^{n\times m} \\ \mathbf{u} \leftarrow \mathcal{A}(\mathbf{A}) \end{array}\right] \le \delta(\lambda).$$

We say that $\mathsf{SIS}_{n,m,q,\beta}$ is *polynomially hard* if it holds against size-$\mathrm{poly}(\lambda)$ adversaries with advantage $\mathrm{negl}(\lambda)$, and, for a constant $\nu > 0$, 2^{n^ν}*-hard* if it holds against size-2^{n^ν} adversaries with advantage $\mathrm{negl}(\lambda)$. Norms are max-entry norms. We use two norm regimes. The first, with $\beta = 2^{n^e}$, will be used for the result from subexponential SIS alone. For the result from polynomially hard SIS and an average-case tree puzzle, a quasi-polynomial norm bound suffices.

Assumption 3.5 (SIS_e). *Let* $0 < e < 1$ *be a constant. For every polynomially bounded* $n = n(\lambda) \ge \lambda^{\Omega(1)}$, *let* $\beta := 2^{n^e}$. *Then for every polynomially bounded* $m = m(\lambda)$ *and every power of two* q *with*

$$n\sqrt{m} \cdot \beta \ \le \ q \ \le \ 2^{\mathrm{poly}(n)},$$

$\mathsf{SIS}_{n,m,q,\beta}$ *is polynomially hard.*

Assumption 3.6 ($\mathsf{SIS}_{\mathrm{qp}}$). *Fix a sufficiently large absolute constant* $C \in \mathbb{N}$. *For every polynomially bounded* $n = n(\lambda) \ge \lambda^{\Omega(1)}$ *with* $n \ge 2$, *let* $\beta := 2^{C\lceil \log n \rceil^3}$. *Then for every polynomially bounded* $m = m(\lambda)$ *and every power of two* q *such that*

$$n\sqrt{m} \cdot \beta \le q$$

and q/β *is polynomially bounded in* n, $\mathsf{SIS}_{n,m,q,\beta}$ *is polynomially hard.*

Thus the norm bound is quasi-polynomial, and the modulus exceeds it by a polynomial factor. The concrete parameters and the requirement on C are given in Lemma 6.12.

Binary decompositions. The setup matrices of our construction are binary decompositions $\mathbf{G}^{-1}(\mathbf{W}')$ of uniformly random matrices $\mathbf{W}' \in \mathbb{Z}_q^{n\times m}$, with q a power of two, which we use over the integers. The following lemma shows that short integer kernel vectors of $\mathbf{G}^{-1}(\mathbf{W}')$ are SIS solutions for $\mathbf{W}'$, and hence hard to find.

Lemma 3.7 (Integer kernels give SIS solutions). *Let* $\mathbf{W}$ *be* $\mathbf{G}^{-1}(\mathbf{W}')$, *possibly padded with zero rows. If* $\mathbf{w} \in \mathbb{Z}^m$ *satisfies* $\mathbf{Ww} = \mathbf{0}$ *over* $\mathbb{Z}$ *and* $0 < \|\mathbf{w}\| \le \beta$, *then* $\mathbf{w}$ *is an* $\mathsf{SIS}_{n,m,q,\beta}$ *solution for* $\mathbf{W}'$.

Proof. Zero rows do not change the kernel, so $\mathbf{G}^{-1}(\mathbf{W}')\mathbf{w} = \mathbf{0}$, and $\mathbf{W}'\mathbf{w} = \mathbf{G}_n\mathbf{G}^{-1}(\mathbf{W}')\mathbf{w} = \mathbf{0}$ over $\mathbb{Z}$, hence also modulo q. □

3.3 Regular tree circuits

In this work, we focus on Boolean circuits that have repeated structure. The circuit is divided into blocks, which form a complete binary tree. A small circuit is evaluated at each leaf, and a small circuit at each internal node combines its two child outputs. The same leaf circuit is used at every leaf, and the same combining circuit is used at every node of a given height. This repetition gives a short description even when the tree has exponentially many leaves. Unlike a circuit that is evaluated on an input, a regular tree circuit has *no external input*: the leaf circuit is applied to the address of each leaf, and any input to the computation, such as a problem instance, is hardcoded into the local circuits. A regular tree circuit is therefore determined by its description, and so is its output.

Definition 3.8 (Regular tree circuit)**.** *A* regular tree circuit $C = (p, \mathsf{C}_0, \ldots, \mathsf{C}_p)$ *consists of an address length* p*, a Boolean leaf circuit* $\mathsf{C}_0 : \{0,1\}^p \to \{0,1\}^{v_0}$*, and Boolean combining circuits* $\mathsf{C}_h : \{0,1\}^{2v_{h-1}} \to \{0,1\}^{v_h}$ *for* $h \in [p]$*. The nodes of its tree are the strings* $\eta \in \{0,1\}^{\leq p}$*, where* η *has children* $\eta 0$ *and* $\eta 1$ *and height* $p - |\eta|$*. The root is* ε*, and the* $P = 2^p$ *leaves are the* addresses $a = a_{p-1} \cdots a_0 \in \{0,1\}^p$*, ordered lexicographically. The values at the nodes are defined bottom-up: for each leaf* a *and each node* η *of height* $h \geq 1$*,*

$$u_a := \mathsf{C}_0(a), \qquad u_\eta := \mathsf{C}_h(u_{\eta 0}, u_{\eta 1}).$$

The output of C *is* u_ε*, and its size* $|C|$ *is the total size of* $\mathsf{C}_0, \ldots, \mathsf{C}_p$*, where the size of a Boolean circuit counts its input wires as well as its gates.*

To illustrate the power of regular circuits, we show in the lemma below that a regular tree circuit can return the lexicographically first satisfying assignment of a circuit, with one leaf per candidate. Here the tree itself is exponential, but the description of the circuit is polynomial.

Lemma 3.9 (Exhaustive search)**.** *There is a polynomial-time algorithm that, given a Boolean circuit* $\varphi : \{0,1\}^n \to \{0,1\}$*, outputs a regular tree circuit* C_φ *with* $p = n$ *and* $|C_\varphi| \leq \mathrm{poly}(|\varphi|)$*, whose output is* $(1, a^*)$ *for the lexicographically first* $a^* \in \{0,1\}^n$ *with* $\varphi(a^*) = 1$*, and* $(0, 0^n)$ *if* φ *is unsatisfiable.*

Proof. The value at each node is a pair $(s, w) \in \{0,1\} \times \{0,1\}^n$ of a flag and a candidate. The leaf circuit is $\mathsf{C}_0(a) = (\varphi(a), \varphi(a) \cdot a)$, and every combining circuit is the same circuit $\mathsf{C}_h = \mathsf{C}$, which keeps the left pair if its flag is 1 and the right pair otherwise:

$$\mathsf{C}\big((s_L, w_L), (s_R, w_R)\big) = \begin{cases} (s_L, w_L) & \text{if } s_L = 1, \\ (s_R, w_R) & \text{otherwise.} \end{cases}$$

The leaf circuit has size $|\varphi| + O(n)$, and C is a multiplexer on $n + 1$ bits, of size $O(n)$. With $p = n$ combining circuits, $|C_\varphi| = |\varphi| + O(n^2) \leq \mathrm{poly}(|\varphi|)$. By induction on the height, u_η is $(1, a)$ for the lexicographically first satisfying a with prefix η, and $(0, 0^n)$ if there is none: the leaves below $\eta 0$ precede those below $\eta 1$, so the first satisfying leaf below η is below $\eta 0$ if $s_L = 1$, and is the first one below $\eta 1$ otherwise. Setting $\eta = \varepsilon$ completes the proof of the lemma. □

4 Updatable Unambiguous SNARGs for Circuits and EOML Hardness

Prior constructions of hard RSVL instances from succinct arguments [CHK+19a, CHK+19b, EFKP20, KPY20, LV20, JKKZ21, BCH+22, KLV23, CFJW25] rely, implicitly or explicitly, on unambiguous and incrementally updatable succinct arguments for long computations with short descriptions. These range from specific computations, such as counting satisfying assignments through the sumcheck protocol and iterated squaring, to general Turing machine computations. We formalize the analogous notion for regular tree circuits. We then show that such a SNARG, combined with an average-case hard problem computed by regular tree circuits, yields hard RSVL instances, and hence hard EOML instances by Theorem 3.3.

4.1 Unambiguous updatable circuit SNARGs

Fix a regular tree circuit C with $P = 2^p$ leaves (Definition 3.8). Since C has no external input, the computation is determined by its description, which is given to the preprocessing algorithm.

The prover starts from an initial state and applies a deterministic update P times to obtain the final state, and thus the output. We require that any intermediate state for a given time $t \in [0, P]$ can be verified, although the definition does not ascribe any semantic meaning to such an intermediate state, i.e. intermediate states don't provide a "partial output".

Our security property, *state uniqueness*, requires that it be hard to find two distinct accepting states for the same circuit and time. It is a collision-style analogue of unambiguous soundness: the adversary must output both states, rather than a single accepting state that differs from the honest one.

Height bound. Fix an efficiently computable, polynomially bounded function $H = H(\lambda) \geq 1$. The proof system supports regular tree circuits with address length at most H. Since H is part of the scheme, it is not an input to Setup, and security is still measured in λ. We use $H = \log^2 \lambda$, rounded up to an integer, for polynomially hard SIS and $H = \lambda$ for subexponentially hard SIS alone.

Syntax. An unambiguous updatable circuit SNARG (uuSNARG) for regular tree circuits is a tuple of algorithms (Setup, Preprocess, Initialize, Update, Verify, Output):

- $\mathsf{Setup}(1^\lambda, 1^N) \to \mathsf{crs}$. This is a PPT algorithm that takes as input the security parameter and a circuit size bound N, and outputs a common reference string crs.
- $\mathsf{Preprocess}(\mathsf{crs}, C) \to \mathsf{pp}$. This is a deterministic algorithm that takes as input the common reference string crs and a regular tree circuit $C = (p, \mathsf{C}_0, \ldots, \mathsf{C}_p)$, and outputs preprocessed parameters pp. We assume that pp includes the description of C.
- $\mathsf{Initialize}(\mathsf{crs}, \mathsf{pp}) \to \mathsf{st}_0$. This is a deterministic algorithm that takes as input the common reference string crs and the preprocessed parameters pp, and outputs an initial state st_0.
- $\mathsf{Update}(\mathsf{crs}, \mathsf{pp}, \mathsf{st}) \to \mathsf{st}'$. This is a deterministic algorithm that takes as input the common reference string crs, the preprocessed parameters pp and a state st, and outputs the next state st′.
- $\mathsf{Verify}(\mathsf{crs}, \mathsf{pp}, t, \mathsf{st}) \to b$. This is a deterministic algorithm that takes as input the common reference string crs, the preprocessed parameters pp, a time $t \in [0, P]$, where $P = 2^p$ for the circuit C given to Preprocess, and a state st, and outputs a bit $b \in \{0, 1\}$ indicating whether st is accepted for time t.
- $\mathsf{Output}(\mathsf{crs}, \mathsf{pp}, \mathsf{st}) \to y$. This is a deterministic algorithm that takes as input the common reference string crs, the preprocessed parameters pp and a state st, and outputs a claimed output y of C, or $\perp$.

We write Update^t for the t-fold application of $\mathsf{st} \mapsto \mathsf{Update}(\mathsf{crs}, \mathsf{pp}, \mathsf{st})$, and abbreviate $\mathsf{Output}(\mathsf{crs}, \mathsf{pp}, \mathsf{st})$ as $\mathsf{Output}(\mathsf{st})$ when crs and pp are clear.

Definition 4.1 (Unambiguous updatable circuit SNARG)**.** *A* uuSNARG *for regular tree circuits is a tuple* $(\mathsf{Setup}, \mathsf{Preprocess}, \mathsf{Initialize}, \mathsf{Update}, \mathsf{Verify}, \mathsf{Output})$ *that satisfies the following properties.*

Completeness. *For all $\lambda, N \in \mathbb{N}$, every regular tree circuit $C = (p, \mathsf{C}_0, \ldots, \mathsf{C}_p)$ with $p \le H$, $|C| \le N$ and output u_ε, and every $t \in [0, P]$,*

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{crs}, \mathsf{pp}, t, \mathsf{st}_t) = 1 & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N) \\ \wedge\ \mathsf{Output}(\mathsf{st}_P) = u_\varepsilon & \mathsf{pp} := \mathsf{Preprocess}(\mathsf{crs}, C) \\ & \mathsf{st}_0 := \mathsf{Initialize}(\mathsf{crs}, \mathsf{pp}) \\ & \mathsf{st}_t := \mathsf{Update}^t(\mathsf{st}_0),\ \mathsf{st}_P := \mathsf{Update}^P(\mathsf{st}_0) \end{array}\right] = 1.$$

(s, δ)-State uniqueness for a size bound N. *For a polynomial N, every adversary $\mathcal{A}$ of size $s(\lambda)$ satisfies*

$$\Pr\left[\begin{array}{l|l} p \le H \ \wedge\ |C| \le N(\lambda) \ \wedge\ t \in [0, P] & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{N(\lambda)}) \\ \wedge\ \mathsf{Verify}(\mathsf{crs}, \mathsf{pp}, t, \mathsf{st}) = \mathsf{Verify}(\mathsf{crs}, \mathsf{pp}, t, \mathsf{st}') = 1 & (C, t, \mathsf{st}, \mathsf{st}') \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge\ \mathsf{st} \neq \mathsf{st}' & \mathsf{pp} := \mathsf{Preprocess}(\mathsf{crs}, C) \end{array}\right] \le \delta(\lambda).$$

The circuit and the time are chosen adaptively after the crs*.*

Succinctness. *For all $\lambda, N \in \mathbb{N}$, there is a state length $\ell_{\mathsf{st}} = \ell_{\mathsf{st}}(\lambda, N) \le \mathsf{poly}(\lambda, N)$ such that, for every regular tree circuit $C = (p, \mathsf{C}_0, \ldots, \mathsf{C}_p)$ with $p \le H$ and $|C| \le N$:*

- Setup *runs in time* $\mathsf{poly}(\lambda, N)$*, so* $|\mathsf{crs}| \le \mathsf{poly}(\lambda, N)$*;*
- Initialize *outputs a state of length ℓ_{st}, and* Update *maps every state of length ℓ_{st} to a state of length ℓ_{st};*
- Preprocess *and* Initialize *run in time* $\mathsf{poly}(\lambda, N)$*;*
- Update*,* Verify *and* Output *run in time* $\mathsf{poly}(\lambda, N)$ *on* all *inputs whose state component has length ℓ_{st}.*

In particular, these bounds are polynomial in $p = \log P$, not in P.

We say that $\mathcal{U}$ has *state uniqueness* if, for every polynomial N and every polynomial s, it has (s, δ)-state uniqueness for the size bound N for some negligible function δ, which may depend on N and s.

We note that unlike prior notions of unambiguous soundness [CHK+19a, BCH+22], where an adversary is required to output a *single* accepting state st'_t that differs from $\mathsf{st}_t := \mathsf{Update}^t(\mathsf{st}_0)$, our notion more closely resembles the notion of strong binding of functional commitments in [Wee25], and the unambiguity of SNARGs in [CFJW25], which requires that it is hard to find two distinct accepting proofs for the same statement. As in [Wee25], we note below that when the honest state st_t can be computed efficiently, state uniqueness also implies soundness: the reduction computes st_t itself and outputs it together with the adversary's state. This holds for every time t up to the number of updates that the reduction can afford to compute.

Lemma 4.2 (From uniqueness to τ-bounded soundness)**.** *Let N be a polynomial, let $\tau = \tau(\lambda)$ be a time bound, and let $\mathcal{A}$ be an adversary of size $s(\lambda)$ that outputs either $\perp$ or a triple whose time satisfies $t \le \tau$. Suppose that*

$$\Pr\left[\begin{array}{l|l} p \le H \ \wedge\ |C| \le N(\lambda) \ \wedge\ t \in [0, P] & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{N(\lambda)}) \\ \wedge\ \mathsf{Verify}(\mathsf{crs}, \mathsf{pp}, t, \mathsf{st}) = 1 \ \wedge\ \mathsf{st} \neq \mathsf{st}_t & (C, t, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & \mathsf{pp} := \mathsf{Preprocess}(\mathsf{crs}, C) \\ & \mathsf{st}_t := \mathsf{Update}^t(\mathsf{Initialize}(\mathsf{crs}, \mathsf{pp})) \end{array}\right] \ge \epsilon(\lambda).$$

Then there is an adversary of size $s(\lambda) + (\tau+2)\cdot \mathsf{poly}(\lambda, N(\lambda))$ that breaks state uniqueness (Definition 4.1) for the size bound N with probability at least $\epsilon(\lambda)$.

Proof. The adversary $\mathcal{A}'$ runs $\mathcal{A}(\mathsf{crs})$. If the output is $\perp$, it aborts. Otherwise, it obtains (C, t, st). If C is not a well-formed regular tree circuit with $p \le H$ and $|C| \le N(\lambda)$, or if $t \notin [0, P]$, it aborts. These checks take time $\mathsf{poly}(\lambda, N(\lambda))$. Otherwise it computes $\mathsf{pp} := \mathsf{Preprocess}(\mathsf{crs}, C)$ and $\mathsf{st}_t := \mathsf{Update}^t(\mathsf{Initialize}(\mathsf{crs}, \mathsf{pp}))$, and outputs $(C, t, \mathsf{st}, \mathsf{st}_t)$. Since Initialize and Update are deterministic, st_t is well defined. Computing it takes one call to Preprocess, one to Initialize and $t \le \tau$ calls to Update, each of size $\mathsf{poly}(\lambda, N(\lambda))$ by succinctness, so $\mathcal{A}'$ has the stated size. By completeness, $\mathsf{Verify}(\mathsf{crs}, \mathsf{pp}, t, \mathsf{st}_t) = 1$. Hence whenever the event above occurs, $\mathcal{A}'$ outputs two distinct states accepted for the same circuit and time, which wins the state-uniqueness experiment. □

4.2 Average-case tree puzzles

To obtain hard RSVL instances, we pair a uuSNARG with a computation that takes a prescribed amount of time, but cannot be performed much faster on random inputs. We follow the syntax of the average-case puzzles of [BPSS23], as recalled in [CFJW25]: a function $\mathcal{F}$ with a difficulty parameter t, and a sampler $\mathcal{D}$ of random inputs. Our hardness requirement is weaker than theirs: one polynomially bounded difficulty per adversarial budget suffices, and we consider only solvers with negligible wrong-answer probability. We additionally require the computation to have regular tree structure, since our SNARGs apply to regular tree circuits: $\mathcal{F}(t, \cdot)$ must be the output of a regular tree circuit with t leaves and a short description.

Syntax. An *average-case tree puzzle* $(\mathcal{F}, \mathcal{D})$ consists of the following algorithms:

- $\mathcal{F}(t, x) \to y$. This is a deterministic algorithm that takes as input a difficulty parameter $t \in \mathbb{N}$ and a string $x \in \{0,1\}^*$, and outputs a string y of length at most $\mathsf{poly}(\log t, |x|)$.
- $\mathcal{D}(t, 1^\lambda) \to x$. This is a probabilistic algorithm that takes as input a difficulty parameter t and the security parameter, and outputs a string x of length at most $\mathsf{poly}(\log t, \lambda)$.

For a function $t(\cdot)$, we write $\mathcal{F}_{t,\lambda}(x)$ for $\mathcal{F}(t(\lambda), x)$ and $\mathcal{D}_{t,\lambda}$ for the distribution $\mathcal{D}(t(\lambda), 1^\lambda)$.

Definition 4.3 (Average-case tree puzzles)**.** *An average-case tree puzzle $(\mathcal{F}, \mathcal{D})$ satisfies the following properties.*

Efficiency.

- *$\mathcal{F}(t, x)$ runs in time $t \cdot \mathsf{poly}(\log t, |x|)$;*
- *$\mathcal{D}(t, 1^\lambda)$ runs in time $\mathsf{poly}(\log t, \lambda)$.*

Tree structure. *For every $t = 2^p$ and x, the value $\mathcal{F}(t, x)$ is the output of a regular tree circuit $C_{t,x} = (p, \mathsf{C}_0, \ldots, \mathsf{C}_p)$, and a deterministic algorithm computes the description of $C_{t,x}$ from (t, x) in time $\mathsf{poly}(p, |x|)$.*

Average-case hardness against non-faulty solvers. *For every polynomial p and every $r \in \mathbb{N}$, there is a function κ such that $t := 2^\kappa$ is polynomially bounded and the following holds. If an adversary $\mathcal{A}$ of size at most $\mathsf{p}(\lambda)$ satisfies*

$$\Pr\left[\mathcal{A}(x) \notin \{\perp, \mathcal{F}_{t,\lambda}(x)\} \;:\; x \leftarrow \mathcal{D}_{t,\lambda}\right] \le \mathsf{negl}(\lambda),$$

then, for all sufficiently large λ,

$$\Pr\left[\mathcal{A}(x) = \mathcal{F}_{t,\lambda}(x) \;:\; x \leftarrow \mathcal{D}_{t,\lambda}\right] < \lambda^{-r}.$$

4.3 Hard RSVL instances

We now combine a uuSNARG with an average-case tree puzzle to obtain a hard distribution of RSVL instances, and hence, by Theorem 3.3, of EOML instances. The instance runs the uuSNARG prover in parallel on puzzles of difficulty $2^0, 2^1, \ldots, 2^H$, one step per vertex. The computation for difficulty 2^k halts after 2^k steps, and from then on the state of that puzzle's computation is left unchanged, while the computations for larger difficulties continue. The reduction does not need to know which difficulty is hard for a given adversary: the instance contains every power-of-two difficulty up to 2^H. When $H = \omega(\log \lambda)$, this range eventually contains every polynomially bounded difficulty, and the analysis picks one that is hard for the adversary's budget. This follows the transform of [BPSS23], as described in [CFJW25], which runs computations on puzzles of all difficulties in parallel with the main computation.

Fix a uuSNARG $\mathcal{U} = (\mathsf{Setup}, \mathsf{Preprocess}, \mathsf{Initialize}, \mathsf{Update}, \mathsf{Verify}, \mathsf{Output})$ and an average-case tree puzzle $(\mathcal{F}, \mathcal{D})$. For a puzzle x of difficulty 2^k, let $C_{2^k,x}$ be the regular tree circuit with address length k that computes $\mathcal{F}(2^k, x)$ (Definition 4.3). Since H is polynomially bounded and these circuits are computed in polynomial time, there is a polynomial N such that $|C_{2^k,x}| \le N(\lambda)$ for all $k \le H$ and all x sampled by $\mathcal{D}(2^k, 1^\lambda)$.

Construction 4.4 (RSVL from a uuSNARG and an average-case tree puzzle)**.** The instance $\mathsf{RSVL}[\mathcal{U}, \mathcal{F}, \mathcal{D}] := (\mathsf{S}, \mathsf{V}, L, v_0)$ is output by the sampler in Fig. 7, with $L := 2^H$. A vertex is a canonical fixed-length encoding of $(j, \mathsf{st}^{(0)}, \ldots, \mathsf{st}^{(H)})$, a clock $j \in [0, L]$ and a state $\mathsf{st}^{(k)}$ of length $\ell_{\mathsf{st}}(\lambda, N(\lambda))$ for each C_k (Definition 4.1). Parsing a vertex fails unless it has this form; in particular, it fails if the clock lies outside $[0, L]$. Component k has *local clock* $t_k(i) := \min(i, 2^k)$: its computation halts after 2^k steps, and its state is not updated afterwards. The circuits S and V hardwire crs and $(\mathsf{pp}_k)_k$.

Claim 4.1 (Correctness)**.** *For every $i \in [0, L]$, $\mathsf{S}^i(v_0)$ is the vertex with clock i in which each component k holds its canonical state at its local clock $t_k(i)$. We write $\mathsf{st}_t^{(k)}$ for the canonical state of component k at time t. Moreover, $\mathsf{V}(\mathsf{S}^i(v_0), i) = 1$, so the* RSVL *promise holds. Finally,* S *and* V *are circuits of size* $\mathsf{poly}(\lambda)$.

Proof. The first statement follows by induction on i: component k is updated exactly at the steps $i < 2^k$, so after i steps it has been updated $\min(i, 2^k) = t_k(i)$ times. Acceptance follows from completeness of $\mathcal{U}$, since each canonical component state is accepted at its local clock. The size bound follows from succinctness of $\mathcal{U}$, since there are $H + 1$ components, each with $p \le H$ and $|C_k| \le N(\lambda)$. □

Theorem 4.5. *Let $\mathcal{U}$ be a* uuSNARG *with state uniqueness and height bound $H = \omega(\log \lambda)$, and let $(\mathcal{F}, \mathcal{D})$ be an average-case tree puzzle. Then* $\mathsf{RSVL}[\mathcal{U}, \mathcal{F}, \mathcal{D}]$ *is a hard distribution of* RSVL.

The instance sampler is fixed. Only the analysis chooses a difficulty, as a function of the adversary's size and success probability.

Proof. Suppose, towards contradiction, that an adversary $\mathcal{A}$ of size λ^a outputs an RSVL solution with probability at least λ^{-r} for infinitely many λ.

Let $\kappa : \mathbb{N} \to \mathbb{N}$ be a function such that $t := 2^\kappa$ is polynomially bounded. We will choose it in Case II. Since 2^κ is polynomially bounded, $\kappa = O(\log \lambda)$. Thus $\kappa \le H$ for all sufficiently large λ, and component κ is present in the instance. Run $\mathcal{A}$ on $\mathsf{RSVL}[\mathcal{U}, \mathcal{F}, \mathcal{D}]$ and write its output as (v, i), assigning $i = L$ when it outputs a sink v. Consider the events

$$\begin{aligned} \mathsf{Early}: &\ (v, i) \text{ is a solution with } i < t, \\ \mathsf{Late}: &\ (v, i) \text{ is a solution with } i \ge t. \end{aligned}$$

Sampler on input 1^λ:

1. $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{N(\lambda)})$. // one crs for all difficulties
2. For each $k \in [0, H]$:

   $z_k \leftarrow \mathcal{D}(2^k, 1^\lambda)$ // a puzzle of difficulty 2^k
   $C_k := C_{2^k, z_k}$ // the regular tree circuit computing $\mathcal{F}(2^k, z_k)$
   $\mathsf{pp}_k := \mathsf{Preprocess}(\mathsf{crs}, C_k)$
3. $v_0 := (0, \mathsf{Initialize}(\mathsf{crs}, \mathsf{pp}_0), \ldots, \mathsf{Initialize}(\mathsf{crs}, \mathsf{pp}_H))$. // every component at its initial state
4. Output $(\mathsf{S}, \mathsf{V}, 2^H, v_0)$.

Successor $\mathsf{S}(v)$:

1. Parse $v = (j, \mathsf{st}^{(0)}, \ldots, \mathsf{st}^{(H)})$. If this fails or $j = L$, output v.
   // malformed and final vertices are returned unchanged
2. For each k with $j < 2^k$: $\mathsf{st}^{(k)} \leftarrow \mathsf{Update}(\mathsf{crs}, \mathsf{pp}_k, \mathsf{st}^{(k)})$. // components with $j \geq 2^k$ have halted
3. Output $(j + 1, \mathsf{st}^{(0)}, \ldots, \mathsf{st}^{(H)})$.

Verifier $\mathsf{V}(v, i)$:

1. Parse $v = (j, \mathsf{st}^{(0)}, \ldots, \mathsf{st}^{(H)})$. If this fails or $j \neq i$, reject. // the clock must equal the index i
2. Accept if and only if $\mathsf{Verify}(\mathsf{crs}, \mathsf{pp}_k, t_k(i), \mathsf{st}^{(k)}) = 1$ for every $k \in [0, H]$.
   // each component is checked at its local clock

Figure 7: The sampler of $\mathsf{RSVL}[\mathcal{U}, \mathcal{F}, \mathcal{D}]$, and its successor and verifier circuits.

An early solution is necessarily a false positive, since a sink has index $L \geq t$. Every solution is early or late, so for infinitely many λ,

$$\Pr[\mathsf{Early}] + \Pr[\mathsf{Late}] \;\geq\; \lambda^{-r}. \tag{5}$$

We first show that early solutions have negligible probability for every such κ. We then choose κ so that late solutions would solve a hard puzzle.

Case I: Early solutions. We construct the following adversary against state uniqueness for the size bound N.

$\mathcal{A}_1(\mathsf{crs})$:

1. Sample $z_k \leftarrow \mathcal{D}(2^k, 1^\lambda)$ for $k \in [0, H]$, and form the instance $(\mathsf{S}, \mathsf{V}, L, v_0)$ as in Fig. 7, using crs.
2. Run $\mathcal{A}$ on the instance and obtain (v, i). If the output is malformed, $\mathsf{V}(v, i) = 0$, or $i \geq t$, abort.
3. Compute $\mathsf{S}^i(v_0)$. If it equals v, abort. Otherwise choose a component k in which the two vertices differ, and output $(C_k, t_k(i), \mathsf{st}^{(k)}, \mathsf{st}^{(k)}_{t_k(i)})$.

Replaying $i < t$ steps takes polynomial time. Whenever Early occurs, v and $\mathsf{S}^i(v_0)$ are distinct accepted vertices. Both parse canonically and have the same clock i. They therefore differ in some component k. Both states in that component are accepted at time $t_k(i)$, the canonical one by Claim 4.1. Since C_k has address length at most H and size at most $N(\lambda)$, the two states break state uniqueness. Thus $\Pr[\mathsf{Early}] \leq \mathsf{negl}(\lambda)$.

Case II: Late solutions. Once $i \geq t$, component κ has halted, and the verifier checks its state at its final local clock t. We use the output of this component to solve the puzzle of difficulty t.

Let $s_{\mathcal{B}}$ be a polynomial bounding the size of the solver $\mathcal{B}$ below, uniformly over the choice of $\kappa \leq H$. Such a bound exists: $\mathcal{B}$ samples an instance, runs $\mathcal{A}$ once, and evaluates V and Output. It does not replay any computation. Apply Definition 4.3 with size bound $s_{\mathcal{B}}$ and exponent $r+2$ to choose κ. Then $t = 2^{\kappa}$ is polynomially bounded, and every non-faulty solver of size at most $s_{\mathcal{B}}$ solves this puzzle with probability less than $\lambda^{-(r+2)}$ for all sufficiently large λ.

$\underline{\mathcal{B}(x)}$:

1. Sample $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{N(\lambda)})$ and $z_k \leftarrow \mathcal{D}(2^k, 1^{\lambda})$ for $k \neq \kappa$, and set $z_{\kappa} := x$. Form the instance as in Fig. 7.
2. Run $\mathcal{A}$ on the instance and obtain (v, i). If the output is malformed, $\mathsf{V}(v, i) = 0$, or $i < t$, output $\perp$.
3. Output $\mathsf{Output}(\mathsf{crs}, \mathsf{pp}_{\kappa}, \mathsf{st}^{(\kappa)})$, where $\mathsf{st}^{(\kappa)}$ is component κ of v.

When $x \leftarrow \mathcal{D}_{t,\lambda}$, the instance seen by $\mathcal{A}$ has exactly the distribution of Fig. 7. Every late solution passes the test in step 2. The test is efficiently checkable. It does not require deciding whether (v, i) is a solution.

We show that whenever this test passes, the extracted component is canonical except with negligible probability. Given crs, sample all the puzzles and run $\mathcal{A}$ as above. If the test passes, output $(C_{\kappa}, t, \mathsf{st}^{(\kappa)})$. Otherwise, abort. This is a polynomial-size algorithm that outputs an accepted state at time t. Since t is polynomially bounded, Lemma 4.2 and state uniqueness give

$$\Pr[\mathsf{V}(v, i) = 1 \ \wedge \ i \geq t \ \wedge \ \mathsf{st}^{(\kappa)} \neq \mathsf{st}_t^{(\kappa)}] \leq \mathsf{negl}(\lambda). \tag{6}$$

By completeness, the canonical state $\mathsf{st}_t^{(\kappa)}$ has output $\mathcal{F}(t, x)$. Thus, except with negligible probability, whenever step 2 passes, $\mathcal{B}$ outputs $\mathcal{F}(t, x)$, rather than an incorrect value or $\perp$. In particular, $\mathcal{B}$ is non-faulty, and it succeeds on every late solution apart from this negligible event. The choice of κ therefore gives, for all sufficiently large λ,

$$\Pr[\mathsf{Late}] \leq \Pr[\mathcal{B}(x) = \mathcal{F}_{t,\lambda}(x)] + \mathsf{negl}(\lambda) < \lambda^{-(r+2)} + \mathsf{negl}(\lambda) < \lambda^{-r}/2.$$

Together with Case I, this contradicts (5). □

Corollary 4.6. *Let $\mathcal{U}$ be a* uuSNARG *with state uniqueness and height bound $H = \omega(\log \lambda)$, and let $(\mathcal{F}, \mathcal{D})$ be an average-case tree puzzle. Then there is a hard distribution of* EOML.

Proof. Apply Theorem 3.3 to the hard distribution $\mathsf{RSVL}[\mathcal{U}, \mathcal{F}, \mathcal{D}]$ of Theorem 4.5. □

Hardness from complexity leveraging. In the preceding proof, we choose a difficulty t at which the puzzle is hard and the reduction can still replay the computation in polynomial time. If state uniqueness holds against larger adversaries, we can instead use the largest difficulty, $t = 2^H$. The uniqueness reduction can then replay the entire computation, while the puzzle solver remains polynomial-size: it only runs the original adversary and reads the selected component's output. Thus hardness at this one difficulty suffices. We will use this to base hardness on subexponentially hard SIS alone.

Theorem 4.7. *Let $\mathcal{U}$ be a* uuSNARG*, and let $(\mathcal{F}, \mathcal{D})$ satisfy the efficiency and tree structure properties of Definition 4.3. Put $L := 2^H$, and let c be a polynomial bounding the cost of sampling* $\mathsf{RSVL}[\mathcal{U}, \mathcal{F}, \mathcal{D}]$*, evaluating* S *or* V*, and executing one replayed step in Lemma 4.2. Suppose that:*

1. *For $x \leftarrow \mathcal{D}(L, 1^{\lambda})$, every polynomial-size adversary $\mathcal{A}$ satisfying*

$$\Pr[\mathcal{A}(x) \notin \{\perp, \mathcal{F}(L, x)\}] \leq \mathsf{negl}(\lambda)$$

also satisfies $\Pr[\mathcal{A}(x) = \mathcal{F}(L, x)] \leq \mathsf{negl}(\lambda)$.

2. *For every polynomial s, $\mathcal{U}$ has $(s + 2L \cdot c, \mathsf{negl}(\lambda))$-state uniqueness for the size bound N fixed above.*

Then $\mathsf{RSVL}[\mathcal{U}, \mathcal{F}, \mathcal{D}]$ *is a hard distribution of* RSVL.

Proof. Fix a polynomial-size adversary $\mathcal{A}$. Define Early and Late as in the proof of Theorem 4.5, with $\kappa := H$, so that $t = L$. Early solutions are false positives at indices $i < L$, and late solutions are accepted vertices at index L.

Case I: Early solutions. The adversary $\mathcal{A}_1$ from Case I of that proof samples an instance, runs $\mathcal{A}$, and replays $i < L$ steps. Its size is at most $s' + 2L \cdot c$ for a polynomial s' accounting for the original adversary and the polynomial overhead. Whenever Early occurs, it produces two distinct accepted states for one component at the same local clock. The assumed state uniqueness therefore gives $\Pr[\mathsf{Early}] \le \mathsf{negl}(\lambda)$.

Case II: Late solutions. Use the solver $\mathcal{B}$ from Case II with $\kappa = H$. Given $x \leftarrow \mathcal{D}(L, 1^\lambda)$, it places x in the component of difficulty L, runs $\mathcal{A}$, and returns that component's output if $\mathcal{A}$ supplies an accepted vertex at index L. Otherwise it returns $\perp$. This solver has polynomial size.

To establish that $\mathcal{B}$ is non-faulty, apply the same replay argument as in (6). The separate reduction to state uniqueness now replays L updates, so its size is at most $s'' + 2L \cdot c$ for a polynomial s''. The assumed uniqueness at this size implies that the accepted state supplied by $\mathcal{A}$ for the selected component differs from its canonical state with negligible probability. By completeness, the canonical state has output $\mathcal{F}(L, x)$. Hence $\mathcal{B}$ is non-faulty and outputs the correct answer on every late solution, apart from a negligible event.

The hardness of the puzzle at difficulty L now gives

$$\Pr[\mathsf{Late}] \le \Pr[\mathcal{B}(x) = \mathcal{F}(L, x)] + \mathsf{negl}(\lambda) \le \mathsf{negl}(\lambda).$$

Every solution is early or late, so $\mathcal{A}$ succeeds with negligible probability. □

Corollary 4.8. *Let $\mathcal{U}$ and $(\mathcal{F}, \mathcal{D})$ be as in Theorem 4.7. Then there is a hard distribution of* EOML.

Proof. Apply Theorem 3.3 to the hard distribution $\mathsf{RSVL}[\mathcal{U}, \mathcal{F}, \mathcal{D}]$ of Theorem 4.7. □

5 Updatable Unambiguous SNARGs for Circuits from Mergeable Functional Commitments

We now construct the uuSNARG for regular tree circuits required by Section 4. Its state after t updates must be accepted by Verify for time t, and it must be hard to find two different accepted states, while its size is polynomial in λ, even though t may be as large as 2^p, which is superpolynomial in λ when $p = \omega(\log \lambda)$.

As in [CHK+19a, CHK+19b, EFKP20, BCH+22], the state consists of proofs for subtrees. Call a subtree *completed* once all of its leaves have been visited. After t updates, the first t leaves have been visited, and the state holds one proof for each of the largest completed subtrees (Fig. 2). These subtrees have different heights, so there are at most $p + 1$ of them. The update visits the next leaf and proves it. Then, starting from this leaf and moving up the tree, as long as the current subtree and its sibling are both completed, it merges their proofs into a proof for their parent and discards them. Visiting the eighth of 8 leaves, for example, completes subtrees of heights 1, 2 and 3 in turn. The verifier checks each proof in the state against its subtree, so finding two different accepted states requires finding two different accepted proofs for the same subtree.

We obtain these proofs from a *mergeable functional commitment*, which we construct from SIS in Section 6 by adapting the functional commitment scheme in [Wee25]. We call the leaf and combining circuits of a regular tree circuit its *local circuits*. The commitment proves one layer of a local circuit at a time, for all copies of that layer in a subtree: given a commitment to their inputs, a layer proof determines a commitment to their outputs, which is the input commitment for the next layer. Proofs for the same layer on two sibling subtrees merge into a proof for their parent. Since both children apply the same layers, the proofs for two siblings are merged layer by layer, followed by proofs for the layers of the parent's combining circuit. By binding, it is hard to find two different accepted proofs for the same input commitment and layer. The input commitment of a subtree is computed from its addresses, so it is also hard to find two different accepted proofs for the same subtree.

In Section 5.1, we write the local circuits as sequences of quadratic layers, which are what our commitment supports. We define mergeable functional commitments in Section 5.2, construct proofs for subtrees in Section 5.3, and assemble the state, update and verifier of the uuSNARG in Section 5.4.

5.1 Arithmetic encoding of regular tree circuits

We start by encoding regular tree circuits in a form that our functional commitment supports. In particular, it supports layers in which each output is a sum of products of two inputs, with coefficients in $\{-1, 0, 1\}$, applied to Boolean inputs. Looking ahead, the restrictions on the coefficients keep the norms small in its SIS-based construction. A local circuit usually needs more than one such layer, so we write it as a sequence of layers, each reading the output of the previous one. Since every layer must receive Boolean inputs, the encoding must keep every value Boolean when the circuit is evaluated on the leaf addresses, which we show in Lemma 5.3.

Definition 5.1 (Quadratic layer)**.** *A* quadratic layer *is a function* $\mathsf{f} : \mathbb{Z}^{w^{\mathsf{in}}} \to \mathbb{Z}^{w^{\mathsf{out}}}$ *with coordinates* $\mathsf{f}_o(\mathbf{x}) = \sum_{j,l} \alpha_{jlo} x_j x_l$ *and coefficients* $\alpha_{jlo} \in \{-1, 0, 1\}$.

A quadratic layer is homogeneous, so it cannot add constants directly. Adding a *constant wire* c that carries the value 1 lets us express Boolean gates in this form: for wires u, v,

$$\mathsf{NAND}(u, v) = c^2 - u \cdot v, \qquad \mathsf{Copy}(u) = c \cdot u.$$

A constant bit $z \in \{0, 1\}$ is the term $z \cdot c^2$, in which z is a coefficient rather than a wire. Every Boolean circuit can be written with NAND gates and constants. We also need copy gates because a layer only reads

the outputs of the previous layer: a value that is used several layers later must be carried forward through the layers in between. These gates have coefficients in $\{-1, 0, 1\}$ and give Boolean outputs whenever their inputs are Boolean and $c = 1$.

A *layered circuit* is a sequence $\Gamma = (\gamma_1, \ldots, \gamma_d)$ of quadratic layers in which the output of each layer is the input of the next. It computes $\Gamma(\mathbf{x}) := (\gamma_d \circ \cdots \circ \gamma_1)(\mathbf{x})$, and we call $\gamma_1, \ldots, \gamma_d$ its layers. Each layer has degree 2, but we never compose them into a single function, whose degree would grow with d. An arithmetic tree description is a regular tree circuit whose local circuits are layered circuits instead of Boolean circuits.

Definition 5.2 (Arithmetic tree description)**.** *An* arithmetic tree description $\widehat{C} = (p, \Gamma_0, \ldots, \Gamma_p)$ *is defined as a regular tree circuit (Definition 3.8), except that the leaf circuit* $\Gamma_0 : \mathbb{Z}^{p+1} \to \mathbb{Z}^{v_0}$ *and the combining circuits* $\Gamma_h : \mathbb{Z}^{2v_{h-1}} \to \mathbb{Z}^{v_h}$ *are layered circuits, each with at least one layer, and that the leaf a receives its address with a constant wire prepended,* $\iota(a) := (1, a_0, \ldots, a_{p-1})$*. Thus the values at the nodes are*

$$u_a := \Gamma_0(\iota(a)), \qquad u_\eta := \Gamma_h(u_{\eta 0}, u_{\eta 1}),$$

and the output is u_ε*. We call* $\widehat{C}$ Boolean *if every wire in this evaluation is Boolean. Its size* $|\widehat{C}|$ *is the total number of layers, wires and nonzero coefficients of* $\Gamma_0, \ldots, \Gamma_p$*.*

As in a regular tree circuit, the only inputs are the leaf addresses, i.e. the circuit encoding determines the output. Requiring at least one layer in each local circuit makes every node's value the output of a layer at that node, hence a single vector, which the parent's first layer reads as one half of its input.

Lemma 5.3 (Compiling regular tree circuits)**.** *There is a deterministic polynomial-time algorithm* Encode *that maps a regular tree circuit* $C = (p, \mathsf{C}_0, \ldots, \mathsf{C}_p)$ *to a Boolean arithmetic tree description* $\widehat{C} = \mathsf{Encode}(C)$ *with the same* p*, of size* $|\widehat{C}| \le \mathrm{poly}(|C|)$*, whose output is* $(1, u_\varepsilon)$*, where* u_ε *is the output of* C*. More generally, the value of each node of* $\widehat{C}$ *is the value of the same node of* C *with a constant wire prepended, so the widths of* $\widehat{C}$ *exceed those of* C *by one.*

Proof. Rewrite each C_h using NAND, copy and constant gates, and put it into layered form by carrying wires forward with copy gates. If it has no gates, use a single copy layer. Replace each gate by its homogeneous quadratic form above and carry the constant wire as the first coordinate of each layer. Each level of gates thus becomes one quadratic layer, and Γ_h is the layered circuit with one layer for each level of C_h. At a combining node, obtain the new constant wire as the square of the left child's constant wire. The leaf circuit reads the address bits in their original order from $\iota(a)$.

Induction over the layers gives

$$\Gamma_0(\iota(a)) = (1, \mathsf{C}_0(a)), \qquad \Gamma_h((1, u), (1, v)) = (1, \mathsf{C}_h(u, v)).$$

Every wire in the evaluation of $\widehat{C}$ is Boolean, and every constant wire is 1. Gate replacement has constant overhead, while layering and copying add at most polynomial overhead. The encoding therefore has the claimed size and can be computed in polynomial time. □

5.2 Mergeable functional commitments

Consider a subtree of height h and a layer f of the leaf circuit. The leaf circuit runs at each of the 2^h leaves of the subtree, so f is applied in parallel to a wide vector made of 2^h blocks, one for each leaf. More generally, in a subtree of height $k + h$, a layer of the combining circuit for height k is applied in parallel to 2^h blocks, one for each node of height k in the subtree.

Given a commitment C to an input $\mathbf{x}$ of f, a *chainable* functional commitment [BCFL23, Wee25] provides a short proof that certifies a commitment C' to $\mathsf{f}(\mathbf{x})$, rather than $\mathsf{f}(\mathbf{x})$ itself. The verifier recovers C' from

the proof, so proofs for successive layers chain. We use a functional commitment for the function F_h that applies f in parallel to 2^h blocks. The commitment is *mergeable* if the proof for F_{h+1} on $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1)$ can be computed from the proofs for F_h on $\mathbf{x}_0$ and on $\mathbf{x}_1$ and the commitments to $\mathbf{x}_0$ and $\mathbf{x}_1$, without reading $\mathbf{x}$. Verification also uses a short *layer digest* of F_h, which depends only on f and h, and the layer digest of F_{h+1} is likewise computed from that of F_h. Thus the prover starts with $h = 0$ at a node of height k, where f is applied once, and each merge of two sibling subtrees increases h by one.

We use the height bound $H = H(\lambda)$ fixed in Section 4.1. The mergeable commitment supports up to 2^H copies of a layer, and the resulting uuSNARG supports trees of height at most H.

Blocks and layers. The commitment is parameterized by a base length L_0, a power of two, which is determined by λ and N. A vector has *height* h if its length is $L_0 2^h$. If $h \geq 1$, it splits into equal halves $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1)$.

The commitment applies to quadratic layers $\mathsf{f} : \mathbb{Z}^{L_0 2^a} \to \mathbb{Z}^{L_0 2^{a'}}$, whose input and output lengths are L_0 times a power of two. From now on, a *layer* means such a quadratic layer. Its size $|\mathsf{f}|$ is the number of nonzero coefficients plus $2^a + 2^{a'}$, and N bounds this size.

Definition 5.4 (Blockwise extension)**.** *Let* $\mathsf{f} : \mathbb{Z}^{L_0 2^a} \to \mathbb{Z}^{L_0 2^{a'}}$ *be a layer and* $h \geq 0$*. The* blockwise extension F_h *of* f *splits its input into* 2^h *consecutive* blocks *of length* $L_0 2^a$ *and applies* f *to each of them in parallel. Its* input height *is* $a + h$ *and its* output height *is* $a' + h$*, the heights of its input and output vectors. As for parallel repetition, we write* $\mathsf{F}_h = \mathsf{f}^{\otimes 2^h}$*. In particular,* $\mathsf{F}_0 = \mathsf{f}$ *and* $\mathsf{F}_{h+1} = \mathsf{F}_h^{\otimes 2}$.

Syntax. A mergeable functional commitment consists of the following algorithms, in four groups. The algorithms other than Setup take crs as an implicit input. The algorithms for layer digests depend only on the layer and are shown in blue, while the layer-proof algorithms depend on the committed input and are shown in red.

Setup and commitments.

- $\mathsf{Setup}(1^\lambda, 1^N) \to \mathsf{crs}$. This is a PPT algorithm that takes as input the security parameter and a size bound N for layers, and outputs a common reference string crs. It fixes a base length $L_0 = L_0(\lambda, N)$ with $H < L_0 \leq \mathrm{poly}(\lambda, N)$, which depends only on λ and N, and not on the randomness of Setup.
- $\mathsf{BaseCom}(\mathbf{x}) \to C$. This is a deterministic algorithm that takes as input a vector $\mathbf{x}$ of length L_0, and outputs a commitment C.
- $\mathsf{Node}(C_0, C_1) \to C$. This is a deterministic algorithm that takes as input two commitments C_0, C_1, and outputs a commitment C to the concatenation of the committed vectors.
- $\mathsf{Valid}(z) \to b$. This is a deterministic algorithm that takes as input a vector, a commitment or a layer proof z, and outputs a bit b indicating whether z has the format and size prescribed by crs. Binding only concerns inputs accepted by Valid. In our construction, Valid checks the dimensions and bounds the entries (Section 6.1).

Layer digests.

- $\mathsf{Digest}(\mathsf{f}) \to \sigma$. This is a deterministic algorithm that takes as input a layer f, and outputs a digest σ of f.
- $\mathsf{DoubleDigest}(\mathsf{f}, \sigma) \to \sigma'$. This is a deterministic algorithm that takes as input a layer f and a layer digest σ for 2^h copies of f, and outputs a layer digest σ' for 2^{h+1} copies.

Layer proofs.

- $\mathsf{Prove}(\mathsf{f}, \mathbf{x}) \to \pi$. This is a deterministic algorithm that takes as input a layer f and an input block $\mathbf{x}$, and outputs a layer proof π for one copy of f.
- $\mathsf{Merge}(\mathsf{f}, \sigma, (C_0, \pi_0), (C_1, \pi_1)) \to \pi$. This is a deterministic algorithm that takes as input a layer f, a layer digest σ for 2^h copies of f, and the input commitments and layer proofs of two siblings, and outputs a layer proof π for 2^{h+1} copies.
- $\mathsf{VerifyLayer}(\mathsf{f}, \sigma, C, \pi) \to b$. This is a deterministic algorithm that takes as input a layer f, a layer digest σ, an input commitment C and a layer proof π, and outputs a bit b indicating whether π is accepted.
- $\mathsf{OutputCom}(\mathsf{f}, \sigma, C, \pi) \to C'$. This is a deterministic algorithm that takes as input a layer f, a layer digest σ, an input commitment C and a layer proof π, and outputs the output commitment C' claimed by π.

Input commitments for the tree.

- $\mathsf{InputCom}(p, \eta) \to C$. This is a deterministic algorithm that takes as input an address length p and a node $\eta \in \{0,1\}^{\le p}$, and outputs a commitment C to the inputs of the leaves below η, as specified in Definition 5.5. It does so without enumerating these leaves.

Like Digest and DoubleDigest, the algorithm InputCom depends only on public data, and it is run by the verifier, which checks the proof for a node without the proofs for the node's children. A merge needs no separate input commitment: merging proofs of $C_0 \xrightarrow{\mathsf{F}_h} C_0'$ and $C_1 \xrightarrow{\mathsf{F}_h} C_1'$ gives a proof whose input commitment is $\mathsf{Node}(C_0, C_1)$. We write $\mathsf{Com}(\mathbf{x}) := \mathsf{BaseCom}(\mathbf{x})$ if $\mathbf{x}$ has height 0, and $\mathsf{Com}(\mathbf{x}) := \mathsf{Node}(\mathsf{Com}(\mathbf{x}_0), \mathsf{Com}(\mathbf{x}_1))$ otherwise. We write $\sigma_{\mathsf{f},h}$ for the layer digest of F_h, obtained by applying DoubleDigest to Digest(f) h times.

For a fixed layer f, define the *honest layer proof* of an input $\mathbf{x}$ of F_h by $\mathsf{Prove}_0(\mathbf{x}) := \mathsf{Prove}(\mathsf{f}, \mathbf{x})$ and

$$\mathsf{Prove}_{h+1}(\mathbf{x}) := \mathsf{Merge}\big(\mathsf{f}, \sigma_{\mathsf{f},h}, (\mathsf{Com}(\mathbf{x}_0), \mathsf{Prove}_h(\mathbf{x}_0)), (\mathsf{Com}(\mathbf{x}_1), \mathsf{Prove}_h(\mathbf{x}_1))\big).$$

A layer proof π *proves* $C \xrightarrow{\mathsf{F}_h} C'$ if $\mathsf{VerifyLayer}(\mathsf{f}, \sigma_{\mathsf{f},h}, C, \pi) = 1$ and $\mathsf{OutputCom}(\mathsf{f}, \sigma_{\mathsf{f},h}, C, \pi) = C'$. Since an output commitment is the next layer's input commitment, layer proofs chain: $C_0 \xrightarrow{\mathsf{f}_1} C_1 \xrightarrow{\mathsf{f}_2} \cdots$.

Definition 5.5 (Mergeable functional commitment)**.** *A* mergeable functional commitment *is a tuple of the algorithms above that satisfies the following properties.*

Completeness. *For all $\lambda, N \in \mathbb{N}$, every layer f with $|\mathsf{f}| \le N$, every $h \le H$, and every input $\mathbf{x}$ of F_h such that $\mathbf{x}$ and $\mathsf{F}_h(\mathbf{x})$ are Boolean,*

$$\Pr\left[\begin{array}{l} \mathsf{VerifyLayer}(\mathsf{f}, \sigma_{\mathsf{f},h}, C, \pi) = 1 \ \wedge\ \mathsf{Valid}(\pi) = 1 \\ \wedge\ \mathsf{OutputCom}(\mathsf{f}, \sigma_{\mathsf{f},h}, C, \pi) = \mathsf{Com}(\mathsf{F}_h(\mathbf{x})) \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N) \\ C := \mathsf{Com}(\mathbf{x}),\ \pi := \mathsf{Prove}_h(\mathbf{x}) \end{array}\right] = 1.$$

Moreover, $\mathsf{Valid}(z) = 1$ for every leaf vector and every commitment z in the commitment trees of $\mathbf{x}$ and $\mathsf{F}_h(\mathbf{x})$.

Input commitments. *For every $p \le H$ and every node η of height $h = p - |\eta|$,*

$$\mathsf{InputCom}(p, \eta) = \mathsf{Com}\big((\iota(\eta b))_{b \in \{0,1\}^h}\big),$$

where each $\iota(\eta b) \in \mathbb{Z}^{p+1}$ is extended with zeros to length $L_0 > p$.

(s, δ)-**Binding for a size bound** N. *For a polynomial N, every adversary $\mathcal{A}$ of size $s(\lambda)$ satisfies*

$$\Pr\left[\mathsf{COLL}(z) \ \vee \ \mathsf{BIND}(z) \ : \ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{N(\lambda)}) \\ z \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \delta(\lambda),$$

where the events are defined as follows.

- $\mathsf{COLL}(z)$ *holds if z is a* tree collision*: either $z = (\mathbf{u}, \mathbf{u}')$ with $\mathsf{Valid}(\mathbf{u}) = \mathsf{Valid}(\mathbf{u}') = 1$, $\mathbf{u} \neq \mathbf{u}'$ and $\mathsf{BaseCom}(\mathbf{u}) = \mathsf{BaseCom}(\mathbf{u}')$, or $z = ((C_0, C_1), (C_0', C_1'))$ with $\mathsf{Valid}(C_b) = \mathsf{Valid}(C_b') = 1$ for $b \in \{0, 1\}$, $(C_0, C_1) \neq (C_0', C_1')$ and $\mathsf{Node}(C_0, C_1) = \mathsf{Node}(C_0', C_1')$.*
- $\mathsf{BIND}(z)$ *holds if z is a* binding pair*: $z = (\mathsf{f}, h, C, \pi, \pi')$ with $|\mathsf{f}| \leq N(\lambda)$, $h \leq H$, $\mathsf{Valid}(C) = \mathsf{Valid}(\pi) = \mathsf{Valid}(\pi') = 1$, $\pi \neq \pi'$ and $\mathsf{VerifyLayer}(\mathsf{f}, \sigma_{\mathsf{f},h}, C, \pi) = \mathsf{VerifyLayer}(\mathsf{f}, \sigma_{\mathsf{f},h}, C, \pi') = 1$.*

Efficiency. *For all $\lambda, N \in \mathbb{N}$:*

- $\mathsf{Setup}(1^\lambda, 1^N)$ *runs in time* $\mathsf{poly}(\lambda, N)$.
- $\mathsf{BaseCom}$, Node, Valid, Digest, $\mathsf{DoubleDigest}$, Prove, Merge, $\mathsf{VerifyLayer}$ *and* $\mathsf{OutputCom}$ *run in time polynomial in the length of their inputs, including* crs.
- *For every layer* f *with* $|\mathsf{f}| \leq N$ *and every $h \leq H$, the layer digest $\sigma_{\mathsf{f},h}$ has length* $\mathsf{poly}(\lambda, N)$.
- *Every vector, commitment or layer proof z with* $\mathsf{Valid}(z) = 1$ *has length* $\mathsf{poly}(\lambda, N)$.
- *For every $p \leq H$ and node $\eta \in \{0,1\}^{\leq p}$,* $\mathsf{InputCom}(p, \eta)$ *runs in time* $\mathsf{poly}(\lambda, N)$*, even though η may have $2^{p-|\eta|}$ leaves below it.*

A binding pair corresponds to a break of strong binding in [Wee25]: two distinct accepted openings of the same commitment. Since a layer proof determines its output commitment, the two proofs either give different output commitments, which breaks evaluation binding, or give the same output commitment but differ elsewhere. Excluding the second case is what strong binding adds to evaluation binding, and we need it because a state contains its layer proofs: two such proofs would give two distinct accepted states for the same computation. For the same reason, binding covers the entire layer proof, including any data retained for merging. On honest inputs, merging combines statements side by side: proofs of $C_0 \xrightarrow{\mathsf{F}_h} C_0'$ and $C_1 \xrightarrow{\mathsf{F}_h} C_1'$ give a proof of $\mathsf{Node}(C_0, C_1) \xrightarrow{\mathsf{F}_{h+1}} \mathsf{Node}(C_0', C_1')$. Completeness only concerns honest proofs. Merging arbitrary accepted proofs need not give an accepted proof.

5.3 Node proofs

In this subsection, we describe the proof held for a node of the tree. Its statement is that the computation on the subtree of the node, starting from the leaf addresses, yields a claimed output. The statement is efficiently computable, since the commitment to the leaf addresses is given by $\mathsf{InputCom}$. The proof consists of a layer proof of the mergeable functional commitment (Definition 5.5) for each layer of this computation. We then give algorithms that prove a leaf and merge the proofs of two siblings. Throughout, fix a regular tree circuit C with $p \leq H$ and $|C| \leq N$, and let $\widehat{C} = \mathsf{Encode}(C) = (p, \Gamma_0, \ldots, \Gamma_p)$.

Layers. List the layers of $\Gamma_0, \Gamma_1, \ldots, \Gamma_p$ in order as $\mathsf{f}_1, \ldots, \mathsf{f}_D$. Let j_h be the number of layers in $\Gamma_0, \ldots, \Gamma_h$, with $j_{-1} := 0$. For $k \in [0, p]$, the layers of Γ_k are f_i for $i \in I_k$, where

$$I_k := \{j_{k-1} + 1, \ldots, j_k\}.$$

Let k_i be the index of the local circuit containing layer i, that is, $i \in I_{k_i}$. Thus layer i belongs to Γ_{k_i} and is applied at the nodes of height k_i. In the full tree there are 2^{p-k_i} such nodes, so f_i is applied to 2^{p-k_i} blocks. The first layer of Γ_0 acts on the leaf input $\iota(a)$, and for $h \geq 1$, the first layer of Γ_h acts on the concatenation of the two child outputs. Every other layer, in the leaf circuit or in a combining circuit, acts on the output of the previous layer at the same node.

Remark 5.6 (Padding). *Commitments are only defined for vectors whose length is L_0 times a power of two, and the blocks of each layer must align with the commitment tree. We therefore pad each output block with zeros to the least length $L_0 2^a$ that accommodates it, and each leaf input $\iota(a)$, of length $p + 1 \le L_0$, to a single block of length L_0. The input of the first layer of Γ_h is then the concatenation of two padded child outputs. When the preprocessing computes $\mathsf{f}_1, \ldots, \mathsf{f}_D$ from $\widehat{C}$ (Section 5.4), it re-indexes their coefficients onto these padded positions, with zero coefficients on terms involving input padding and on output padding coordinates. Consecutive layers then share exactly the same intermediate vector and commitment tree.*

Padding adds no nonzero coefficients. For an output block of width w, its padded height satisfies $2^a \le 2\max(1, w)$, and concatenating two child blocks doubles this quantity. Thus each padded f_i is a layer of size at most $N_{\mathsf{a}}(N)$ for a fixed polynomial N_{a} independent of L_0.

For simplicity of exposition, we leave the padding implicit from now on: $\mathsf{f}_1, \ldots, \mathsf{f}_D$ denote the re-indexed layers, and every input and output of a layer, including the claimed output in a node proof, denotes its padded form.

The statement at a node. A node η of height h represents the computation on the 2^h addresses with prefix η, through layer j_h. Its input commitment is

$$C_\eta := \mathsf{InputCom}(p, \eta),$$

and its output is u_η. Within this subtree, each layer $i \le j_h$ is applied to 2^{h-k_i} blocks and has layer digest $\sigma_{\mathsf{f}_i, h-k_i}$. The statement always concerns the entire subtree of η. The input commitment is computed by InputCom in time $\mathrm{poly}(\lambda, N)$, without enumerating the 2^h addresses, so verifying a proof for η takes polynomial time. Constructing such a proof from scratch would evaluate the whole subtree. The incremental prover instead reuses the proofs of the two children, so each update proves one leaf and performs at most p merges. A state keeps proofs only for the nodes of the frontier (Section 5.4).

Node proofs and verification. A node proof for η contains a claimed output and, for each local circuit Γ_k with $k \le h$, the proofs for its layers:

$$\Pi = (y, \boldsymbol{\pi}_0, \ldots, \boldsymbol{\pi}_h), \qquad \boldsymbol{\pi}_k = (\pi_i)_{i \in I_k}.$$

We call $\boldsymbol{\pi}_k$ the *circuit proof* for Γ_k. It contains one layer proof for each layer of Γ_k. The claimed output y is a single block of polynomial length: the last layer j_h belongs to Γ_h and is applied only once in the subtree, at η. The node verifier VerifyNode recomputes the commitments along the chain of layers, starting from the input commitment C_η, which it computes with InputCom.

VerifyNode(η, Π), for a node η of height h:

1. Parse $\Pi = (y, \boldsymbol{\pi}_0, \ldots, \boldsymbol{\pi}_h)$, and each $\boldsymbol{\pi}_k$ as $(\pi_i)_{i \in I_k}$, in the fixed polynomial-length encoding. Reject if this fails, if the padding is not canonical, if some integer does not have its canonical encoding, or if y is not Boolean of the prescribed length.
2. Set $C_{\eta,0} := \mathsf{InputCom}(p, \eta)$. // the input commitment C_η, without enumerating the leaves
3. For $k = 0, \ldots, h$: // the local circuits, in order
   For $i \in I_k$: // the layers of Γ_k
   Reject unless $\mathsf{Valid}(\pi_i) = 1$ and $\mathsf{VerifyLayer}(\mathsf{f}_i, \sigma_{\mathsf{f}_i, h-k}, C_{\eta,i-1}, \pi_i) = 1$.
   Set $C_{\eta,i} := \mathsf{OutputCom}(\mathsf{f}_i, \sigma_{\mathsf{f}_i, h-k}, C_{\eta,i-1}, \pi_i)$.
   Reject unless $\mathsf{Valid}(C_{\eta,i}) = 1$.
4. Reject unless every leaf vector and every commitment in the commitment tree of y is accepted by Valid.
5. Accept if and only if $C_{\eta,j_h} = \mathsf{Com}(y)$. // the chain ends at the claimed output

Thus VerifyNode verifies the chain

$$C_\eta = C_{\eta,0} \xrightarrow{\mathsf{f}_1} C_{\eta,1} \xrightarrow{\mathsf{f}_2} \cdots \xrightarrow{\mathsf{f}_{j_h}} C_{\eta,j_h} = \mathsf{Com}(y),$$

where each arrow denotes the appropriate blockwise extension. The commitments $C_{\eta,i}$ are recomputed during verification and are not stored in Π. A node proof is thus the opening of the functional commitment for layered circuits obtained by chaining one layer proof per layer, as in [BCFL23, Wee25], and its length grows with the number of layers.

The *prescribed node proof* is $\Pi_\eta := (u_\eta, \boldsymbol{\pi}_{\eta,0}, \ldots, \boldsymbol{\pi}_{\eta,h})$ with $\boldsymbol{\pi}_{\eta,k} = (\pi_{\eta,i})_{i \in I_k}$, where $\pi_{\eta,i}$ is the honest proof Prove_{h-k_i} for the input of layer i within the subtree. Keeping a proof for each layer lets us merge the circuit proofs of two children layer by layer, before proving the local circuit of their parent.

The following algorithms compute node proofs. The first proves one local circuit Γ_k applied once, to an explicit input. The other two use it to prove a leaf, and to merge the node proofs of two siblings. They use the layer digests $\sigma_{\mathsf{f}_i,h}$, and return a fixed failure string if a check fails. In particular, ProveCircuit checks that every intermediate vector is Boolean, which keeps all lengths polynomial even on arbitrary inputs.

ProveCircuit$(k, \mathbf{x})$, for $k \in [0, p]$ and an input $\mathbf{x}$ of Γ_k:

1. Reject unless $\mathbf{x}$ is Boolean. Set $\mathbf{x}_{j_{k-1}} := \mathbf{x}$.
2. For $i \in I_k$: set $\pi_i := \mathsf{Prove}(\mathsf{f}_i, \mathbf{x}_{i-1})$ and $\mathbf{x}_i := \mathsf{f}_i(\mathbf{x}_{i-1})$, and reject unless $\mathbf{x}_i$ is Boolean.
   // each layer acts on one block
3. Output $(\mathbf{x}_{j_k}, \boldsymbol{\pi}_k)$ with $\boldsymbol{\pi}_k := (\pi_i)_{i \in I_k}$.

ProveLeaf(a), for a leaf $a \in \{0,1\}^p$:

1. Output ProveCircuit$(0, \iota(a))$. // the node proof (y, π_0)

MergeNode$(\eta, \Pi^{(0)}, \Pi^{(1)})$, for a node η of height $h \geq 1$:

1. For $b \in \{0,1\}$: run VerifyNode$(\eta b, \Pi^{(b)})$, and output a fixed failure string if it rejects. Otherwise, parse $\Pi^{(b)} = (y^{(b)}, \boldsymbol{\pi}_0^{(b)}, \ldots, \boldsymbol{\pi}_{h-1}^{(b)})$ with $\boldsymbol{\pi}_k^{(b)} = (\pi_i^{(b)})_{i \in I_k}$, and let $C_{\eta b,0} = C_{\eta b}, C_{\eta b,1}, \ldots$ be the commitments recomputed by VerifyNode.
2. For $k = 0, \ldots, h-1$: // the local circuits shared with the children
   For $i \in I_k$: set $\pi_i := \mathsf{Merge}\big(\mathsf{f}_i, \sigma_{\mathsf{f}_i,h-1-k}, (C_{\eta 0,i-1}, \pi_i^{(0)}), (C_{\eta 1,i-1}, \pi_i^{(1)})\big)$. // merge layer by layer
   Set $\boldsymbol{\pi}_k := (\pi_i)_{i \in I_k}$.
3. Set $(y, \boldsymbol{\pi}_h) :=$ ProveCircuit$(h, (y^{(0)}, y^{(1)}))$. // prove Γ_h once, at η
4. Output $(y, \boldsymbol{\pi}_0, \ldots, \boldsymbol{\pi}_h)$.

Lemma 5.7 (Node proofs). *For every* crs *in the support of* $\mathsf{Setup}(1^\lambda, 1^{N_\mathsf{a}(N)})$*, the following hold.*

Completeness. VerifyNode *accepts every prescribed node proof.*

Leaves and merges. *The algorithms* ProveCircuit, ProveLeaf *and* MergeNode *run in time* $\mathrm{poly}(\lambda, N)$ *on all inputs of the prescribed lengths, and* $\mathsf{ProveLeaf}(a) = \Pi_a$ *and* $\mathsf{MergeNode}(\eta, \Pi_{\eta 0}, \Pi_{\eta 1}) = \Pi_\eta$.

Uniqueness. *Given two different node proofs for the same node that are both accepted by* VerifyNode*, one can compute in polynomial time a tree collision or a binding pair for a layer of size at most* $N_\mathsf{a}(N)$.

Proof. Fix a node η of height h.

Completeness. We show that, on the prescribed proof Π_η, the commitments recomputed by VerifyNode follow the honest evaluation of the subtree. The initial commitment $C_{\eta,0} = C_\eta$ commits to its leaf inputs, by the input-commitment property of Definition 5.5. Every layer input and output in this evaluation is Boolean by Lemma 5.3. Thus, if $C_{\eta,i-1}$ is the commitment to the honest input of layer i, completeness of the functional commitment ensures that $\pi_{\eta,i}$ is accepted and that $C_{\eta,i}$ is the commitment to its honest output. By Remark 5.6, this is also the input commitment for the next layer. Inductively, the verifier reaches $C_{\eta,j_h} = \mathsf{Com}(u_\eta)$. The prescribed encoding is canonical, and completeness also supplies all the required Valid checks, so VerifyNode accepts.

Leaves and merges. On an honest input to Γ_k, the algorithm ProveCircuit evaluates this local circuit and proves each of its layers once. Its calls to Prove therefore give precisely the honest layer proofs Prove_0. All intermediate values are Boolean, so no check fails. In particular, on a leaf input $\iota(a)$, it returns Π_a, proving $\mathsf{ProveLeaf}(a) = \Pi_a$.

Now consider $\mathsf{MergeNode}(\eta, \Pi_{\eta 0}, \Pi_{\eta 1})$ for $h \geq 1$. Completeness ensures that the two child proofs pass verification and that the recovered commitments are their honest intermediate commitments. We check the two parts of the parent proof separately.

- For a circuit Γ_k with $k < h$, each child proof certifies 2^{h-1-k} copies of each layer f_i, $i \in I_k$. The parent must certify their concatenation, comprising 2^{h-k} copies. Step 2 applies Merge to the two child proofs and their input commitments, exactly as in the recursive definition of Prove_{h-k}. It therefore produces the prescribed layer proof $\pi_{\eta,i}$ for every $i \in I_k$, and hence the prescribed circuit proof $\boldsymbol{\pi}_{\eta,k}$.
- The remaining circuit Γ_h is evaluated only at η, on the two child outputs $(u_{\eta 0}, u_{\eta 1})$. By the observation about ProveCircuit above, step 3 returns its output u_η and its prescribed circuit proof $\boldsymbol{\pi}_{\eta,h}$.

Together, these are exactly the fields of Π_η.

For the running time, all three algorithms make polynomially many calls to algorithms that run in time polynomial in their input lengths. In ProveCircuit, every layer is evaluated on a Boolean vector of polynomial length; if its output is not Boolean, the algorithm rejects before using it as the next layer's input. Thus all intermediate lengths remain polynomial even on arbitrary inputs. In MergeNode, verification checks the lengths of the child proofs and recovered commitments before they are passed to Merge. Each merge therefore also takes polynomial time. This proves the claimed running-time bounds.

Uniqueness. Let Π and Π' be distinct accepted proofs for η. Compare their layer proofs in the order used by VerifyNode.

Suppose first that some layer proof differs, and let i be the first such layer. Both verifier chains start at C_η, and all preceding layer proofs agree. Since OutputCom is deterministic, the two chains reach the same input commitment $C_{\eta,i-1}$. Thus $\pi_i \neq \pi'_i$ are both accepted for the same layer f_i, layer digest $\sigma_{\mathsf{f}_i,h-k_i}$ and input commitment $C_{\eta,i-1}$. These give a binding pair. The layer has size at most $N_{\mathsf{a}}(N)$, its height $h - k_i$ is at most $p \leq H$, and the proofs and input commitment are valid: validity of $C_{\eta,0}$ follows from completeness, while all subsequent commitments and layer proofs are checked by VerifyNode.

Otherwise, all layer proofs agree, so all recovered commitments agree as well. Canonical encoding then implies that the two node proofs differ in their claimed outputs $y \neq y'$. The final verification checks give

$$\mathsf{Com}(y) = C_{\eta,j_h} = \mathsf{Com}(y').$$

Compute the two commitment trees and start at their equal roots. At an internal node, if the pairs of child commitments differ, they give a collision for Node. If the pairs agree, descend into a child whose underlying vectors differ; its two commitments are still equal. If this process reaches a leaf, it gives distinct leaf

vectors with the same commitment under BaseCom. All the vectors and commitments used in this collision pass Valid, as checked by VerifyNode. Both the comparison of layer proofs and the search through the commitment trees take polynomial time, since the proofs and explicit outputs have polynomial length. □

5.4 The uuSNARG construction

We now assemble the node proofs of Section 5.3 into the state of a uuSNARG. Each update visits one leaf and merges the proofs of the subtrees that this leaf completes. The verifier checks each retained node proof independently.

The frontier and the state. The leaves are visited in lexicographic order, so after t updates the completed subtrees are those whose leaves are among the first t. Let $\mathrm{frontier}(t)$ consist of the nodes whose subtree is completed but whose parent's subtree is not, together with the root once all leaves have been visited. The frontier is empty at $t = 0$ and consists only of the root at $t = 2^p$. In general, its subtrees have distinct heights, namely the positions of the nonzero bits of t. Thus $\mathrm{frontier}(t)$ contains at most $p + 1$ nodes, and their addresses and left-to-right order can be computed from t in time polynomial in p.

The prescribed state after t updates is

$$\mathsf{st}_t = \big(t, (\Pi_\eta)_{\eta \in \mathrm{frontier}(t)}\big),$$

with node proofs listed from left to right (Fig. 2). The clock determines which node each proof belongs to, so the state does not store node addresses. We use a fixed state length depending only on λ, N, with unused slots set to zero and all padding canonical. No other working data are retained.

Construction 5.8 (uuSNARG from a mergeable functional commitment)**.** The algorithms use the state encoding above. Parsing a state checks that its clock lies in $[0, 2^p]$, that it contains one node proof of the prescribed length for each node of the corresponding frontier, and that all padding and all integer encodings are canonical. A state that fails these checks is returned unchanged by Update, rejected by Verify, and mapped to $\perp$ by Output. The same holds for every state when $\mathsf{pp} = \perp$, and in that case Initialize outputs the all-zero string of the fixed state length. These are format checks. The node proofs themselves are verified by VerifyNode.

Setup$(1^\lambda, 1^N)$: Run the setup of the mergeable functional commitment with size bound $N_{\mathrm{a}}(N)$ and output its crs.

Preprocess(crs, C):

1. If C is malformed, has $p > H$, or has size greater than N, output pp $:= \perp$.
2. Compute $\widehat{C} := \mathsf{Encode}(C)$ and its padded layers $\mathsf{f}_1, \ldots, \mathsf{f}_D$ as in Remark 5.6.
3. For every layer i, compute $\sigma_{\mathsf{f}_i,0} := \mathsf{Digest}(\mathsf{f}_i)$ and then $\sigma_{\mathsf{f}_i,1}, \ldots, \sigma_{\mathsf{f}_i,p}$ by repeated DoubleDigest.
4. Output pp, consisting of C, $\widehat{C}$, the padded layers and these layer digests.

Initialize(crs, pp): Output the canonical encoding of the clock 0 and the empty frontier.

Update(crs, pp, st):

1. Parse st as $(t, (\Pi_\eta)_{\eta \in \mathrm{frontier}(t)})$. If $t = 2^p$, output st.
2. Let $a \in \{0,1\}^p$ be the binary representation of t, which is the next leaf in lexicographic order. Append $\mathsf{ProveLeaf}(a)$ to the list. // visit the next leaf
3. While the last two proofs in the list belong to siblings $\eta 0, \eta 1$: // merge upward, one height at a time

   Replace them by $\Pi_\eta := \mathsf{MergeNode}(\eta, \Pi_{\eta 0}, \Pi_{\eta 1})$, associated with their parent η. If the subtree of the sibling of η is completed, Π_η is now one of the last two proofs and is merged in the next iteration.
4. Output the canonical encoding of the clock $t + 1$ and the resulting list.

If any subroutine fails or any proof exceeds its allotted length, stop and output the original state st.

Verify(crs, pp, t, st):

1. Reject unless $t \in [0, 2^p]$ and st parses as $(t, (\Pi_\eta)_{\eta \in \mathrm{frontier}(t)})$.
2. Accept if and only if $\mathsf{VerifyNode}(\eta, \Pi_\eta) = 1$ for every $\eta \in \mathrm{frontier}(t)$.

Output(crs, pp, st):

1. Parse st. If its clock is not 2^p, output $\perp$.
2. Read the claimed output y from the root proof. If $y = (1, u, \mathbf{0})$, where $u \in \{0,1\}^{v_p}$ for the output length v_p of C_p and $\mathbf{0}$ is the prescribed padding, output u. Otherwise, output $\perp$. // remove the constant wire and padding

Theorem 5.9 (uuSNARG from a mergeable functional commitment)**.** *Let a mergeable functional commitment be given (Definition 5.5). Then Construction 5.8 satisfies completeness and succinctness of Definition 4.1. For every polynomial N, there is a polynomial c such that if an adversary of size s breaks state uniqueness for the size bound N with probability ϵ, then an adversary of size $s + c$, given the commitment's* crs *for the size bound $N_{\mathrm{a}}(N)$, outputs a tree collision or a binding pair with probability at least ϵ.*

Proof. For completeness and succinctness, fix a regular tree circuit C with $p \le H$ and $|C| \le N$.

Completeness. We show by induction on t that the state after t updates is the prescribed state st_t. This holds at $t = 0$, since Initialize outputs the empty frontier. Suppose it holds at time $t < 2^p$. The next update visits the leaf with index t. Appending this leaf and repeatedly replacing the last two siblings by their parent gives exactly $\mathrm{frontier}(t + 1)$. Each replacement combines two completed subtrees, and the loop stops at the first node whose parent's subtree is not completed. By Lemma 5.7, ProveLeaf produces the prescribed leaf proof and each call to MergeNode produces the prescribed parent proof. Thus the update returns st_{t+1}.

The same lemma ensures that every node proof in st_t passes VerifyNode, so Verify accepts at every time t. At $t = 2^p$, the only retained proof is the root proof. By Lemma 5.3 and the padding convention, its

claimed output is $(1, u_\varepsilon, \mathbf{0})$, where u_ε is the output of C. Hence $\mathsf{Output}(\mathrm{st}_{2^p}) = u_\varepsilon$.

Succinctness. The setup runs in time $\mathrm{poly}(\lambda, N_{\mathrm{a}}(N)) = \mathrm{poly}(\lambda, N)$. Encoding and padding produce polynomially many layers, each of size at most $N_{\mathrm{a}}(N)$. For each layer, preprocessing computes one layer digest and doubles it at most p times. Since all these digests have polynomial length, Preprocess also takes time $\mathrm{poly}(\lambda, N)$.

A state contains a clock and at most $p + 1 \le H + 1$ node proofs of polynomial length, so a fixed polynomial state length suffices. The frontier is computed from the clock in polynomial time. Initialization writes the empty state, and output reads the root's claimed output. An update computes one leaf proof and performs at most p node merges, and verification checks at most $p + 1$ node proofs. By Lemma 5.7 and the efficiency of the functional commitment, all these operations take time $\mathrm{poly}(\lambda, N)$. In particular, VerifyNode computes each input commitment using InputCom, without enumerating the leaves of its subtree. The format checks and the bounds on subroutine inputs give the same running-time bounds on arbitrary states of the prescribed length. Every update returns either a canonically encoded state of that length or its original input, as required.

State uniqueness. Given the commitment's crs for size bound $N_{\mathrm{a}}(N)$, the reduction passes it to the state-uniqueness adversary and receives $(C, t, \mathrm{st}, \mathrm{st}')$. This is exactly the crs distribution of the construction. The reduction checks the circuit and time bounds, computes $\mathsf{pp} := \mathsf{Preprocess}(\mathsf{crs}, C)$, and runs Verify on both states. If either state is rejected or the states agree, it aborts.

Otherwise, the two states have the same clock and hence the same frontier and ordering of node proofs. Their padding is canonical, and there are no other retained fields. Thus some node $\eta \in \mathrm{frontier}(t)$ has two different node proofs, both accepted by VerifyNode. The reduction finds such a node by comparing the two lists and applies Lemma 5.7 to obtain a tree collision or a binding pair for a layer of size at most $N_{\mathrm{a}}(N)$. All work apart from running the adversary has size bounded by a polynomial c, and the reduction succeeds whenever the adversary breaks state uniqueness. Its size is therefore at most $s + c$, with success probability at least ϵ. □

6 A Mergeable Functional Commitment from SIS

We construct the mergeable functional commitment of Definition 5.5 from SIS, adapting the functional commitment of Wee [Wee25]. Beyond proving a single layer, the construction must support two operations: merging the proofs for two adjacent groups of copies of a layer, and computing the commitment to a subtree's leaf addresses without listing them. Both rely on commitments and openings being linear over the integers. The price of this linearity, as we shall see, is larger norms.

We define the commitment and its openings in Section 6.1, layer proofs in Section 6.2, and merging in Section 6.3. Section 6.4 computes input commitments, and Section 6.5 bounds the norms, fixes the parameters and proves the main theorem. Throughout, quantities fixed by the crs and the layer are shown in blue, and quantities that depend on the committed input in red. Except when sampling the setup matrices, all arithmetic is over $\mathbb{Z}$.

Kronecker products and compactification. We write $\otimes$ for the Kronecker product and $\mathrm{vec}(\mathbf{M})$ for the vector obtained by stacking the columns of $\mathbf{M}$, and use the identities

$$(\mathbf{A}\otimes\mathbf{B})(\mathbf{C}\otimes\mathbf{D}) = (\mathbf{AC})\otimes(\mathbf{BD}), \qquad (\mathbf{B}^\top\otimes\mathbf{A})\,\mathrm{vec}(\mathbf{X}) = \mathrm{vec}(\mathbf{AXB}).$$

An opening of a vector $\mathbf{u}$ reveals it in the expanded form $\mathbf{u}^\top\otimes\mathbf{I}_d$ (Lemma 6.2). When $\mathbf{u} = \mathrm{vec}(\mathbf{M})$ vectorizes a matrix $\mathbf{M}\in\mathbb{Z}^{d\times c}$, such as a pair of child commitments, this expanded form has cd^2 columns. *Compactification* is the fixed linear map that shrinks it back to the c columns of $\mathbf{M}$: multiplying on the right by the matrix $\mathbf{J}_c$ below recovers $\mathbf{M}$ itself. In [Wee25], $\mathbf{u}$ is the bit decomposition of a matrix over $\mathbb{Z}_q$, and the same map also recomposes the bits. Over the integers no decomposition is needed. We state the corresponding lemma below and provide a proof.

Lemma 6.1 (Compactification)**.** *For $c, d\in\mathbb{N}$, let $\mathbf{J}_c := \mathbf{I}_c\otimes\mathrm{vec}(\mathbf{I}_d)\in\{0,1\}^{cd^2\times c}$. For every $\mathbf{M}\in\mathbb{Z}^{d\times c}$,*

$$(\mathrm{vec}(\mathbf{M})^\top\otimes\mathbf{I}_d)\,\mathbf{J}_c = \mathbf{M}. \tag{7}$$

Proof. Write $\mathbf{M} = [\mathbf{m}_1|\cdots|\mathbf{m}_c]$. Then $\mathrm{vec}(\mathbf{M})^\top = \sum_j \mathbf{e}_j^\top\otimes\mathbf{m}_j^\top$ and $(\mathbf{m}_j^\top\otimes\mathbf{I}_d)\,\mathrm{vec}(\mathbf{I}_d) = \mathbf{m}_j$, so the left-hand side is $\sum_j \mathbf{e}_j^\top\otimes\mathbf{m}_j = \mathbf{M}$. □

6.1 The commitment

Parameters. On input $(1^\lambda, 1^N)$, set $\Lambda := H + \lceil\log N\rceil + 2$. The construction uses an SIS dimension n, a modulus q, an SIS norm bound β and an acceptance threshold B_{thr}, all determined by λ, N and the fixed choice of height bound H. Their values are fixed in Section 6.5, and until then we only need the lengths derived from them. Let d be $n\log q$ rounded up to a power of two, and set $m := 2d^3$, $L_0 := 2d^2$ and $\ell_B := L_0 m$. The crs contains 1^λ and 1^N, from which every algorithm recomputes these parameters. For the matrices $\mathbf{W}', \mathbf{V}'\in\mathbb{Z}_q^{n\times m}$ in the crs, let $\mathbf{W}, \mathbf{V}\in\{0,1\}^{d\times m}$ be $\mathbf{G}^{-1}(\mathbf{W}')$ and $\mathbf{G}^{-1}(\mathbf{V}')$, padded with zero rows. These two matrices are binary.

Commitment algorithms. We define Setup, BaseCom, Node and Valid of Definition 5.5 as follows. A commitment is a $d\times d$ integer matrix. The base commitment BaseCom commits to a vector of length L_0 by applying $\mathbf{W}$. It is used at the leaves, and also at every internal node: Node concatenates the two child commitments, vectorizes them into a vector of length $L_0 = 2d^2$, and applies BaseCom to it. All algorithms in this section check input lengths, dimensions and height bounds before performing the indicated computations, and return a fixed failure symbol on malformed inputs.

Setup$(1^\lambda, 1^N)$:

1. Compute the parameters above and those of Section 6.5.
2. Sample $\mathbf{W}', \mathbf{V}' \leftarrow \mathbb{Z}_q^{n\times m}$ uniformly, and output $\mathrm{crs} := (1^\lambda, 1^N, \mathbf{W}', \mathbf{V}')$.

BaseCom$(\mathbf{u})$, for $\mathbf{u} \in \mathbb{Z}^{L_0}$:

1. Output $\mathbf{W}(\mathbf{u} \otimes \mathbf{I}_d)$.

Node$(\mathbf{C}_0, \mathbf{C}_1)$, for $\mathbf{C}_0, \mathbf{C}_1 \in \mathbb{Z}^{d\times d}$:

1. Output BaseCom$(\mathrm{vec}([\mathbf{C}_0|\mathbf{C}_1]))$.

Valid(z):

1. Accept if and only if z is a vector in $\mathbb{Z}^{L_0}$, a matrix in $\mathbb{Z}^{d\times d}$ or a layer proof of input height at most $\Lambda - 2$ (Section 6.2), with entries in $[-B_{\mathrm{thr}}, B_{\mathrm{thr}}]$. The dimensions of z determine which case applies, since $L_0 = 2d^2 \neq d^2$, and for a layer proof they also determine its input height g, since $\mathbf{T}$ has $(g+1)\ell_B$ rows.

Recall from Section 5.2 that a vector of length $\ell = L_0 2^g$ has *height* g, and that its commitment $\mathrm{Com}(\mathbf{x})$ applies BaseCom if $g = 0$ and otherwise applies Node to the commitments of its two halves. We restate this recursion in the boxes below, alongside the verification key and the opening.

Verification matrices. Openings are checked against fixed matrices built from $\mathbf{W}$ and $\mathbf{V}$. Let

$$\mathbf{B}_{\mathrm{top}} := \mathbf{W}(\mathbf{I}_{L_0} \otimes \mathbf{V}) \in \mathbb{Z}^{d\times \ell_B}, \qquad \mathbf{B}_{\mathrm{bot}} := -\big(\mathrm{vec}(\mathbf{I}_{L_0})^\top \otimes \mathbf{I}_d\big) \in \mathbb{Z}^{d\times \ell_B}.$$

Stacking them gives the matrices

$$\mathbf{B} := \begin{bmatrix} \mathbf{B}_{\mathrm{top}} \\ \mathbf{B}_{\mathrm{bot}} \end{bmatrix} \in \mathbb{Z}^{2d\times \ell_B}, \qquad \mathbf{B}_g := \begin{bmatrix} \mathbf{B}_{\mathrm{top}} & & & \\ \mathbf{B}_{\mathrm{bot}} & \mathbf{B}_{\mathrm{top}} & & \\ & \mathbf{B}_{\mathrm{bot}} & \ddots & \\ & & \ddots & \mathbf{B}_{\mathrm{top}} \\ & & & \mathbf{B}_{\mathrm{bot}} \end{bmatrix} \in \mathbb{Z}^{(g+2)d\times(g+1)\ell_B}$$

for $g \geq 0$. Each of the $g+1$ block columns of $\mathbf{B}_g$ contains a copy of $\mathbf{B}$, one block row below the previous one, and blank blocks are zero. In particular, $\mathbf{B}_0 = \mathbf{B}$, and for $g \geq 1$ the first block column of $\mathbf{B}_g$ is $\mathbf{B}$ followed by zeros, while deleting the first block row and block column leaves $\mathbf{B}_{g-1}$:

$$\mathbf{B}_g = \left[\begin{array}{c|c} \mathbf{B}_{\mathrm{top}} & \mathbf{0} \\ \hline \mathbf{B}_{\mathrm{bot}} & \multirow{2}{*}{$\mathbf{B}_{g-1}$} \\ \mathbf{0} & \end{array}\right],$$

where the top block row has d rows and the first block column has ℓ_B columns.

Vector commitment. As in the vector commitment of Wee [Wee25], we associate with each length ℓ a verification key $\overline{\mathbf{V}}_\ell$ and with each vector $\mathbf{x}$ an opening matrix $\mathbf{Z}_\mathbf{x}$. They are computed by the algorithms VerKey and Open below. Each algorithm is given by a base case and a recursive step, shown in the two boxes below. The key depends only on the crs and ℓ. The opening matrix is used to define the layer proofs but is not stored in full. Let $\mathbf{J} := \mathbf{J}_{2d}$ be the matrix of Lemma 6.1, and split $\mathbf{J} = [\mathbf{J}_0|\mathbf{J}_1]$ and $\mathbf{VJ} = [\mathbf{H}_0|\mathbf{H}_1]$ into halves with $\mathbf{J}_b \in \{0,1\}^{m\times d}$ and $\mathbf{H}_b \in \mathbb{Z}^{d\times d}$.

Base case $\ell = L_0$.

$\underline{\mathsf{Com}(\mathbf{x}), \text{ for } \mathbf{x} \in \mathbb{Z}^{L_0}:}$

1. Output $\mathsf{BaseCom}(\mathbf{x}) = \mathbf{W}(\mathbf{x} \otimes \mathbf{I}_d)$.

$\underline{\mathsf{VerKey}(1^{L_0}):}$

1. Output $\overline{\mathbf{V}}_{L_0} := \mathbf{V}$.

$\underline{\mathsf{Open}(\mathbf{x}), \text{ for } \mathbf{x} \in \mathbb{Z}^{L_0}:}$

1. Output $\mathbf{Z}_{\mathbf{x}} := \mathbf{x} \otimes \mathbf{I}_{L_0 d}$.

Recursive step from ℓ to $\ell/2$.

$\underline{\mathsf{Com}(\mathbf{x}), \text{ for } \mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1) \in \mathbb{Z}^{\ell}:}$

1. Compute $\mathbf{C}_b := \mathsf{Com}(\mathbf{x}_b)$ for $b \in \{0, 1\}$.
2. Output $\mathsf{Node}(\mathbf{C}_0, \mathbf{C}_1) = \mathsf{BaseCom}(\mathbf{u}_{\mathbf{x}})$, where $\mathbf{u}_{\mathbf{x}} := \mathrm{vec}([\mathbf{C}_0 | \mathbf{C}_1]) \in \mathbb{Z}^{L_0}$.

$\underline{\mathsf{VerKey}(1^{\ell}):}$

1. Compute $\overline{\mathbf{V}}_{\ell/2} := \mathsf{VerKey}(1^{\ell/2})$.
2. Output $\overline{\mathbf{V}}_{\ell} := \mathbf{VJ}(\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2}) \in \mathbb{Z}^{d \times \ell d}$.

$\underline{\mathsf{Open}(\mathbf{x}), \text{ for } \mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1) \in \mathbb{Z}^{\ell}:}$

1. Compute $\mathbf{u}_{\mathbf{x}}$ as in $\mathsf{Com}(\mathbf{x})$ and $\overline{\mathbf{V}}_{\ell/2} := \mathsf{VerKey}(1^{\ell/2})$.
2. Compute $\mathbf{Z}_{\mathbf{u}_{\mathbf{x}}} := \mathsf{Open}(\mathbf{u}_{\mathbf{x}})$, a base-case opening, and $\mathbf{Z}_{\mathbf{x}_b} := \mathsf{Open}(\mathbf{x}_b)$ for $b \in \{0, 1\}$.
3. Output
$$\mathbf{Z}_{\mathbf{x}} := \begin{bmatrix} \mathbf{Z}_{\mathbf{u}_{\mathbf{x}}} \mathbf{J}(\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2}) \\ [\mathbf{Z}_{\mathbf{x}_0} \,|\, \mathbf{Z}_{\mathbf{x}_1}] \end{bmatrix}.$$

For $\mathbf{x}$ of height g, this gives $\mathbf{Z}_{\mathbf{x}} \in \mathbb{Z}^{(g+1)\ell_B \times \ell d}$. For $j \in [\ell]$, let $\mathbf{V}_j$ and $\mathbf{Z}_j(\mathbf{x})$ be the j-th blocks of d columns of $\overline{\mathbf{V}}_{\ell}$ and $\mathbf{Z}_{\mathbf{x}}$. The matrix $\mathbf{V}_j$ is fixed by the crs, the height and the coordinate j, whereas $\mathbf{Z}_j(\mathbf{x})$ depends on the input. The following lemma is the verification relation of the vector commitment.

Lemma 6.2 (Opening identity). *For every $\mathbf{x} \in \mathbb{Z}^{\ell}$ of height g, $\mathsf{Com}(\mathbf{x})$ and $\mathbf{Z}_{\mathbf{x}}$ are linear in $\mathbf{x}$, and*

$$\mathbf{B}_g \mathbf{Z}_{\mathbf{x}} = \begin{bmatrix} \mathsf{Com}(\mathbf{x})\, \overline{\mathbf{V}}_{\ell} \\ \mathbf{0} \\ -\mathbf{x}^{\top} \otimes \mathbf{I}_d \end{bmatrix},$$

where $\mathbf{0}$ has g block rows. In particular, for every $j \in [\ell]$,

$$\mathbf{B}_g \mathbf{Z}_j(\mathbf{x}) = \begin{bmatrix} \mathsf{Com}(\mathbf{x}) \mathbf{V}_j \\ \mathbf{0} \\ -x_j \mathbf{I}_d \end{bmatrix}.$$

Proof. Linearity holds because each step of the recursion is linear. We prove the identity by induction on g. $\underline{\text{Base case } g = 0}$. Since $m = L_0 d$, the mixed-product rule gives $\mathbf{B}_{\mathrm{top}} \mathbf{Z}_{\mathbf{x}} = \mathbf{W}(\mathbf{x} \otimes \mathbf{V}) = \mathsf{Com}(\mathbf{x})\mathbf{V}$ and $\mathbf{B}_{\mathrm{bot}} \mathbf{Z}_{\mathbf{x}} = -\big(\mathrm{vec}(\mathbf{I}_{L_0})^{\top}(\mathbf{x} \otimes \mathbf{I}_{L_0})\big) \otimes \mathbf{I}_d = -\mathbf{x}^{\top} \otimes \mathbf{I}_d$.

Inductive step from $\ell/2$ to ℓ. Write $\mathbf{C}_b := \mathsf{Com}(\mathbf{x}_b)$, $\mathbf{u} := \mathbf{u}_{\mathbf{x}}$ and $\mathbf{C} := \mathsf{Com}(\mathbf{x}) = \mathsf{Com}(\mathbf{u})$. By the induction hypothesis for $\mathbf{x}_0$ and $\mathbf{x}_1$, placed side by side,

$$\mathbf{B}_{g-1}[\mathbf{Z}_{\mathbf{x}_0}|\mathbf{Z}_{\mathbf{x}_1}] = \begin{bmatrix} [\mathbf{C}_0|\mathbf{C}_1](\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2}) \\ \mathbf{0} \\ -\mathbf{x}^\top \otimes \mathbf{I}_d \end{bmatrix}, \tag{8}$$

where $\mathbf{0}$ has $g-1$ block rows. The top row still contains the child commitments $\mathbf{C}_0$ and $\mathbf{C}_1$. To cancel it, we use an opening of the parent commitment $\mathbf{C}$, which reveals $\mathbf{u} = \mathrm{vec}([\mathbf{C}_0|\mathbf{C}_1])$ and hence the child commitments. The base case for $\mathbf{u}$ gives

$$\mathbf{B}\mathbf{Z}_{\mathbf{u}} = \begin{bmatrix} \mathbf{C}\mathbf{V} \\ -\mathbf{u}^\top \otimes \mathbf{I}_d \end{bmatrix}.$$

Multiplying on the right by $\mathbf{J}$, Lemma 6.1 with $\mathbf{M} = [\mathbf{C}_0|\mathbf{C}_1]$ turns the revealed vector $\mathbf{u} = \mathrm{vec}([\mathbf{C}_0|\mathbf{C}_1])$ back into the matrix $[\mathbf{C}_0|\mathbf{C}_1]$. Multiplying further by $\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2}$ matches the top row of (8):

$$\mathbf{B}\,\mathbf{Z}_{\mathbf{u}}\mathbf{J}(\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2}) = \begin{bmatrix} \mathbf{C}\,\mathbf{V}\mathbf{J}(\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2}) \\ -[\mathbf{C}_0|\mathbf{C}_1](\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2}) \end{bmatrix} = \begin{bmatrix} \mathbf{C}\,\overline{\mathbf{V}}_{\ell} \\ -[\mathbf{C}_0|\mathbf{C}_1](\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2}) \end{bmatrix}. \tag{9}$$

The key $\overline{\mathbf{V}}_\ell$ is, by definition, the key that the top row of (9) carries. Since the first block column of $\mathbf{B}_g$ is $\mathbf{B}$ followed by zeros and the remaining block columns form $\mathbf{B}_{g-1}$ below a zero block row,

$$\mathbf{B}_g\mathbf{Z}_{\mathbf{x}} = \begin{bmatrix} \mathbf{B}\,\mathbf{Z}_{\mathbf{u}}\mathbf{J}(\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2}) \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{B}_{g-1}[\mathbf{Z}_{\mathbf{x}_0}|\mathbf{Z}_{\mathbf{x}_1}] \end{bmatrix}.$$

By (9) and (8), the first block row is $\mathbf{C}\,\overline{\mathbf{V}}_\ell$, the second block row is $-[\mathbf{C}_0|\mathbf{C}_1](\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2}) + [\mathbf{C}_0|\mathbf{C}_1](\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2}) = \mathbf{0}$, and the remaining block rows are $g-1$ zero block rows followed by $-\mathbf{x}^\top \otimes \mathbf{I}_d$. □

Unrolling the recursion, $\mathbf{Z}_j(\mathbf{x})$ consists of $g+1$ block rows of ℓ_B rows each, which we number $0, \ldots, g$ from the top. For $k < g$, block row k opens the node at depth k on the path to x_j to its two children, and block row g opens the leaf to x_j, as in a Merkle path. Each of the g middle block rows of $\mathbf{B}_g$ checks that the value revealed at one depth equals what the next depth computes from its own commitment.

Recall from Definition 5.5 that binding is broken by one of two events: a *tree collision* (COLL), two different inputs to BaseCom or Node with the same output, or a *binding pair* (BIND), two different layer proofs accepted for the same input commitment. We do not define binding for the vector commitment on its own. The next two lemmas turn each event into a short integer kernel vector of $\mathbf{W}$ or $\mathbf{V}$, which by Lemma 3.7 is an SIS solution for $\mathbf{W}'$ or $\mathbf{V}'$. The first lemma is used for binding pairs, in Lemma 6.7. There, the two proofs give matrices $\mathbf{X} \neq \mathbf{X}'$ that satisfy verification equations with the same first $g+1$ block rows: the top block row is determined by the shared input commitment and by data the two proofs share (the layer digest, or a common derived table), and the next g block rows are zero. Only the last block row, which carries the claimed values, may differ. Thus the first $g+1$ block rows of $\mathbf{B}_g\mathbf{D}$ vanish for the difference $\mathbf{D} := \mathbf{X} - \mathbf{X}'$. The second lemma handles tree collisions. Both follow the case analysis of [Wee25].

Lemma 6.3 (Bidiagonal kernels). *Let $\mathbf{D}$ be a nonzero integer matrix with $(g+1)\ell_B$ rows such that the first $g+1$ block rows of $\mathbf{B}_g\mathbf{D}$ are zero. Then one can compute in polynomial time a nonzero $\mathbf{w}$ with $\mathbf{W}\mathbf{w} = \mathbf{0}$ and $\|\mathbf{w}\| \le m\|\mathbf{D}\|$, or a nonzero $\mathbf{w}$ with $\mathbf{V}\mathbf{w} = \mathbf{0}$ and $\|\mathbf{w}\| \le \|\mathbf{D}\|$.*

Proof. Write $\mathbf{D}$ in block rows $\mathbf{D}_1, \ldots, \mathbf{D}_{g+1}$ of ℓ_B rows each, let $\mathbf{D}_i$ be the first nonzero one, and let $\mathbf{v}$ be a nonzero column of $\mathbf{D}_i$. Block row i of $\mathbf{B}_g\mathbf{D}$ is $\mathbf{B}_{\mathrm{top}}\mathbf{D}_i + \mathbf{B}_{\mathrm{bot}}\mathbf{D}_{i-1}$, where $\mathbf{D}_0 := \mathbf{0}$, and $\mathbf{D}_{i-1} = \mathbf{0}$ by the choice

of i. Since $i \le g+1$, this block row is zero, so $\mathbf{B}_{\text{top}}\mathbf{v} = \mathbf{W}(\mathbf{I}_{L_0} \otimes \mathbf{V})\mathbf{v} = \mathbf{0}$. Split $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_{L_0})$ into blocks of m entries, so that $(\mathbf{I}_{L_0} \otimes \mathbf{V})\mathbf{v} = (\mathbf{V}\mathbf{v}_1, \ldots, \mathbf{V}\mathbf{v}_{L_0})$.
Case 1: $(\mathbf{I}_{L_0} \otimes \mathbf{V})\mathbf{v} \neq \mathbf{0}$. Let $\mathbf{w} := (\mathbf{I}_{L_0} \otimes \mathbf{V})\mathbf{v}$. Then $\mathbf{w} \neq \mathbf{0}$ and $\mathbf{W}\mathbf{w} = \mathbf{B}_{\text{top}}\mathbf{v} = \mathbf{0}$. Since $\mathbf{V}$ is binary with m columns, $\|\mathbf{w}\| \le m\|\mathbf{v}\| \le m\|\mathbf{D}\|$.
Case 2: $(\mathbf{I}_{L_0} \otimes \mathbf{V})\mathbf{v} = \mathbf{0}$. Then $\mathbf{V}\mathbf{v}_k = \mathbf{0}$ for every k. Since $\mathbf{v} \neq \mathbf{0}$, some $\mathbf{v}_k$ is nonzero, and $\mathbf{w} := \mathbf{v}_k$ satisfies $\mathbf{V}\mathbf{w} = \mathbf{0}$ and $\|\mathbf{w}\| \le \|\mathbf{D}\|$. □

Lemma 6.4 (Tree collisions). *If* $\mathbf{u} \neq \mathbf{u}'$ *satisfy* $\mathsf{BaseCom}(\mathbf{u}) = \mathsf{BaseCom}(\mathbf{u}')$, *then* $\mathbf{w} := (\mathbf{u} - \mathbf{u}') \otimes \mathbf{e}_1$ *is nonzero,* $\mathbf{W}\mathbf{w} = \mathbf{0}$ *and* $\|\mathbf{w}\| \le \|\mathbf{u}\| + \|\mathbf{u}'\|$. *In particular, every tree collision gives a kernel vector of* $\mathbf{W}$ *of norm at most* $2B_{\text{thr}}$.

Proof. $\mathbf{W}((\mathbf{u} - \mathbf{u}') \otimes \mathbf{I}_d) = \mathbf{0}$, and its first column is $\mathbf{W}\mathbf{w}$. A collision for Node is a collision for BaseCom on the distinct vectors $\mathrm{vec}([\mathbf{C}_0|\mathbf{C}_1])$ and $\mathrm{vec}([\mathbf{C}'_0|\mathbf{C}'_1])$. □

6.2 Layer proofs

The opening identity lets us check a single coordinate of a committed vector against its commitment. A layer proof combines these openings to show that a second commitment contains the output of a layer. By linearity, the output commitment is a weighted sum of products of input coordinates, with weights determined by the layer. As discussed in the technical overview, we aggregate the corresponding openings with the same weights into *summary tables*. These tables have polynomial size even when the layer is repeated exponentially many times, and they are checked by equations that are linear in the input commitment.

A merge uses the whole of a child's table $\mathbf{T}$, not only the part that determines the output commitment. Verification therefore checks every entry of a layer proof. Then any two different accepted proofs for the same input commitment give a short kernel vector, even when they claim the same output commitment (Lemma 6.7).

Fix a layer $\mathsf{f} : \mathbb{Z}^{L_0 2^a} \to \mathbb{Z}^{L_0 2^{a'}}$ and a height h, and let F_h be its blockwise extension (Definition 5.4), with input height $g = a + h$ and output height $g' = a' + h$. Let ℓ and ℓ' be its input and output lengths. Write $(\mathsf{F}_h)_o(\mathbf{x}) = \sum_{i,j} \alpha_{ijo} x_i x_j$ for $o \in [\ell']$. For each output coordinate o, let $\mathbf{w}_o := \mathrm{vec}(\mathsf{Com}(\mathbf{e}_o)) \in \mathbb{Z}^{d^2}$ be the vectorized commitment to the o-th unit vector of length ℓ', with entries $w_{o,r}$ for $r \in [d^2]$. Expanding $\mathsf{F}_h(\mathbf{x})$ in the unit vectors and using linearity of Com and vec, the output commitment satisfies

$$\begin{aligned}\mathbf{y} := \mathrm{vec}\big(\mathsf{Com}(\mathsf{F}_h(\mathbf{x}))\big) &= \mathrm{vec}\Big(\mathsf{Com}\Big(\sum_o (\mathsf{F}_h)_o(\mathbf{x})\,\mathbf{e}_o\Big)\Big) = \sum_o (\mathsf{F}_h)_o(\mathbf{x})\ \mathrm{vec}\big(\mathsf{Com}(\mathbf{e}_o)\big) \\ &= \sum_o (\mathsf{F}_h)_o(\mathbf{x})\,\mathbf{w}_o = \sum_{i,j,o} \alpha_{ijo} x_i x_j\,\mathbf{w}_o,\end{aligned}$$

with entries $y_r = \sum_{i,j,o} \alpha_{ijo} w_{o,r} x_i x_j$. For $r \in [d^2]$ and $\mathbf{M} \in \mathbb{Z}^{d \times d}$, define the *summaries*

$$\mathbf{Q}_r[\mathbf{M}] := \sum_{i,j,o} \alpha_{ijo} w_{o,r}\,\mathbf{V}_i \mathbf{M} \mathbf{V}_j \in \mathbb{Z}^{d \times d}, \qquad \mathbf{T}_r[\mathbf{M}] := \sum_{i,j,o} \alpha_{ijo} w_{o,r}\,\mathbf{Z}_i(\mathbf{x}) \mathbf{M} \mathbf{V}_j \in \mathbb{Z}^{(g+1)\ell_B \times d},$$
$$\mathbf{R}_r := \sum_{i,j,o} \alpha_{ijo} w_{o,r}\,x_i \mathbf{V}_j \in \mathbb{Z}^{d \times d}, \qquad \mathbf{L}_r := \sum_{i,j,o} \alpha_{ijo} w_{o,r}\,x_i \mathbf{Z}_j(\mathbf{x}) \in \mathbb{Z}^{(g+1)\ell_B \times d}.$$

The table $\mathbf{Q}$ depends only on the layer f, the height h and the crs, and will form the layer digest. The proof will contain $\mathbf{T}$ and $\mathbf{L}$. The verifier will derive $\mathbf{R}$ from $\mathbf{T}$. Here $\mathbf{T}$ and $\mathbf{R}$ are linear in $\mathbf{x}$, while $\mathbf{L}$ is quadratic.

We view each $\mathbf{Q}_r$ and $\mathbf{T}_r$ as a function of its matrix argument $\mathbf{M}$, and both functions are linear in $\mathbf{M}$. Verification evaluates them only at elementary matrices and at $\mathbf{I}_d$, where the identities of Lemma 6.5 below

determine $\mathbf{R}_r$ and the output commitment. Merging needs other arguments: Lemma 6.8 expresses the parent's tables at $\mathbf{M}$ through the children's tables at $\mathbf{M}\mathbf{H}_b$. This is why a layer proof contains the whole function $\mathbf{T}_r$, not just its value at $\mathbf{I}_d$. By linearity, it suffices to store its values on the elementary matrices $\mathbf{E}_{\mu\nu}$, each of which has a single 1 in position (μ, ν). For example,

$$\mathbf{T}_r[\mathbf{M}] = \sum_{\mu,\nu} M_{\mu\nu}\, \mathbf{T}_r[\mathbf{E}_{\mu\nu}].$$

As r ranges over $[d^2]$ and μ, ν over $[d]$, the table $\mathbf{Q}$ has d^6 integer entries, $\mathbf{T}$ has $(g+1)\ell_B d^5$ and $\mathbf{L}$ has $(g+1)\ell_B d^3$. These counts are polynomial in d and g, independently of the number of copies of the layer.

Lemma 6.5 (Summary identities). *Let* $\mathbf{C} := \mathrm{Com}(\mathbf{x})$. *For every* r *and every* $\mathbf{M}$,

$$\mathbf{B}_g\mathbf{T}_r[\mathbf{M}] = \begin{bmatrix} \mathbf{C}\,\mathbf{Q}_r[\mathbf{M}] \\ \mathbf{0} \\ -\mathbf{M}\mathbf{R}_r \end{bmatrix}, \qquad \mathbf{B}_g\mathbf{L}_r = \begin{bmatrix} \mathbf{C}\,\mathbf{R}_r \\ \mathbf{0} \\ -y_r\mathbf{I}_d \end{bmatrix}.$$

Proof. Fix r and write $\gamma_{ijo} := \alpha_{ijo} w_{o,r}$, a scalar. By the definition of $\mathbf{T}_r$ and Lemma 6.2,

$$\mathbf{B}_g\mathbf{T}_r[\mathbf{M}] = \sum_{i,j,o} \gamma_{ijo} \left(\mathbf{B}_g\mathbf{Z}_i(\mathbf{x})\right)\mathbf{M}\mathbf{V}_j = \sum_{i,j,o} \gamma_{ijo} \begin{bmatrix} \mathbf{C}\mathbf{V}_i\mathbf{M}\mathbf{V}_j \\ \mathbf{0} \\ -x_i\mathbf{M}\mathbf{V}_j \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{C}\sum_{i,j,o}\gamma_{ijo}\mathbf{V}_i\mathbf{M}\mathbf{V}_j \\ \mathbf{0} \\ -\mathbf{M}\sum_{i,j,o}\gamma_{ijo}x_i\mathbf{V}_j \end{bmatrix} = \begin{bmatrix} \mathbf{C}\,\mathbf{Q}_r[\mathbf{M}] \\ \mathbf{0} \\ -\mathbf{M}\mathbf{R}_r \end{bmatrix}.$$

In the bottom block row, the scalar x_i moves past $\mathbf{M}$, which stays on the left. Similarly, by the definition of $\mathbf{L}_r$ and Lemma 6.2,

$$\mathbf{B}_g\mathbf{L}_r = \sum_{i,j,o} \gamma_{ijo}\, x_i\, \mathbf{B}_g\mathbf{Z}_j(\mathbf{x}) = \sum_{i,j,o} \gamma_{ijo}\, x_i \begin{bmatrix} \mathbf{C}\mathbf{V}_j \\ \mathbf{0} \\ -x_j\mathbf{I}_d \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{C}\sum_{i,j,o}\gamma_{ijo}x_i\mathbf{V}_j \\ \mathbf{0} \\ -\left(\sum_{i,j,o}\gamma_{ijo}x_ix_j\right)\mathbf{I}_d \end{bmatrix} = \begin{bmatrix} \mathbf{C}\,\mathbf{R}_r \\ \mathbf{0} \\ -y_r\mathbf{I}_d \end{bmatrix}.$$

The last step uses the definition of $\mathbf{R}_r$ and the formula for y_r above. □

The *layer digest* of F_h is $\sigma_{\mathsf{f},h} := (h, \mathbf{Q})$, where $\mathbf{Q}$ denotes the table $(\mathbf{Q}_r[\mathbf{E}_{\mu\nu}])_{r,\mu,\nu}$, and a *layer proof* is a pair $\pi = (\mathbf{T}, \mathbf{L})$ of tables $(\mathbf{T}_r[\mathbf{E}_{\mu\nu}])_{r,\mu,\nu}$ and $(\mathbf{L}_r)_r$. The *honest* layer proof of $\mathbf{x}$ consists of its summaries. For a single copy of the layer, the layer digest and the honest layer proof are computed directly from the definitions, in time polynomial in $|\mathsf{f}|$ and d.

Digest(f):

1. Compute the summaries $\mathbf{Q}_r[\mathbf{E}_{\mu\nu}]$ of $\mathsf{F}_0 = \mathsf{f}$ for all r, μ, ν, and let $\mathbf{Q}$ be their table.
2. Output $\sigma := (0, \mathbf{Q})$.

Prove(f, x):

1. Compute the summaries $\mathbf{T}_r[\mathbf{E}_{\mu\nu}]$ and $\mathbf{L}_r$ of $\mathbf{x}$ for $\mathsf{F}_0 = \mathsf{f}$ for all r, μ, ν, and let $\mathbf{T}$ and $\mathbf{L}$ be their tables.
2. Output $\pi := (\mathbf{T}, \mathbf{L})$.

Lemma 6.5 suggests the following verification. It checks both identities, the first in checks 2 and 3 and the second in check 4, and also recovers the output commitment. Here and below, r ranges over $[d^2]$, indexing the entries of a vectorized commitment, and μ, ν range over $[d]$.

VerifyLayer$(\mathsf{f}, \sigma, \mathbf{C}, \pi)$:

1. Parse $\sigma = (h, \mathbf{Q})$ and $\pi = (\mathbf{T}, \mathbf{L})$, and let $g := a + h$. Check the prescribed dimensions, including $\mathbf{C} \in \mathbb{Z}^{d \times d}$, and reject if $\mathsf{Valid}(\pi) = 0$.
2. For all r, μ, ν, reject unless the first $g + 1$ block rows of $\mathbf{B}_g \mathbf{T}_r[\mathbf{E}_{\mu\nu}]$ are
$$\begin{bmatrix} \mathbf{C}\mathbf{Q}_r[\mathbf{E}_{\mu\nu}] \\ \mathbf{0} \end{bmatrix},$$
where $\mathbf{0}$ has g block rows.
3. For each r, compute $\mathbf{T}_r[\mathbf{I}_d] = \sum_\mu \mathbf{T}_r[\mathbf{E}_{\mu\mu}]$, and let $\mathbf{R}_r$ be the matrix for which the last block row of $\mathbf{B}_g \mathbf{T}_r[\mathbf{I}_d]$ is $-\mathbf{R}_r$. For all r, μ, ν, reject unless the last block row of $\mathbf{B}_g \mathbf{T}_r[\mathbf{E}_{\mu\nu}]$ is $-\mathbf{E}_{\mu\nu}\mathbf{R}_r$.
4. For all r, reject unless
$$\mathbf{B}_g \mathbf{L}_r = \begin{bmatrix} \mathbf{C}\mathbf{R}_r \\ \mathbf{0} \\ -y_r \mathbf{I}_d \end{bmatrix}$$
for an integer y_r, where $\mathbf{0}$ has g block rows.
5. Accept.

OutputCom$(\mathsf{f}, \sigma, \mathbf{C}, \pi)$:

1. If VerifyLayer$(\mathsf{f}, \sigma, \mathbf{C}, \pi) = 0$, output $\perp$.
2. Parse σ, π and g as in VerifyLayer. For each r, let y_r be the integer for which the last block row of $\mathbf{B}_g \mathbf{L}_r$ is $-y_r \mathbf{I}_d$, which exists by check 4 of VerifyLayer.
3. Set $\mathbf{y} := (y_r)_r \in \mathbb{Z}^{d^2}$, and output $\mathbf{C}' := (\mathbf{y}^\top \otimes \mathbf{I}_d)\mathbf{J}_d \in \mathbb{Z}^{d \times d}$. // the matrix with $\mathrm{vec}(\mathbf{C}') = \mathbf{y}$, by Lemma 6.1

The verifier first checks $\mathbf{T}$ against the input commitment and the layer digest, then derives $\mathbf{R}$ and uses it to check $\mathbf{L}$. In particular, $\mathbf{R}$ is not an additional field supplied by the prover. Checking every elementary matrix binds the entire $\mathbf{T}$ table, including the entries used in a subsequent merge. In OutputCom, the last block rows of $\mathbf{B}_g\mathbf{L}_1, \ldots, \mathbf{B}_g\mathbf{L}_{d^2}$, placed side by side, form $-\mathbf{y}^\top \otimes \mathbf{I}_d$, and compactification turns this expanded form back into the matrix $\mathbf{C}'$.

Lemma 6.6 (Completeness of layer proofs)**.** *Let f be a layer and $h \geq 0$, such that F_h has input height $g \leq \Lambda - 2$. Let $\sigma := (h, \mathbf{Q})$, where $\mathbf{Q}$ is the table of summaries of F_h, let $\mathbf{x}$ be an input of F_h, and let π be the honest layer proof of $\mathbf{x}$. If every entry of π has absolute value at most B_{thr}, then*

$$\mathsf{VerifyLayer}(\mathsf{f}, \sigma, \mathsf{Com}(\mathbf{x}), \pi) = 1 \qquad \textit{and} \qquad \mathsf{OutputCom}(\mathsf{f}, \sigma, \mathsf{Com}(\mathbf{x}), \pi) = \mathsf{Com}(\mathsf{F}_h(\mathbf{x})).$$

Proof. Let $\mathbf{C} := \mathsf{Com}(\mathbf{x})$. The proof has the prescribed dimensions, and $\mathsf{Valid}(\pi) = 1$ since $g \leq \Lambda - 2$ and its entries are bounded by B_{thr}. Check 2 holds by Lemma 6.5 with $\mathbf{M} = \mathbf{E}_{\mu\nu}$. Since $\mathbf{T}_r$ is linear in its argument, $\sum_\mu \mathbf{T}_r[\mathbf{E}_{\mu\mu}] = \mathbf{T}_r[\mathbf{I}_d]$, and Lemma 6.5 with $\mathbf{M} = \mathbf{I}_d$ shows that check 3 derives the honest $\mathbf{R}_r$. Check 3 then holds by Lemma 6.5 with $\mathbf{M} = \mathbf{E}_{\mu\nu}$, and check 4 holds by the second identity of Lemma 6.5, with y_r the entries of $\mathbf{y} = \mathrm{vec}(\mathsf{Com}(\mathsf{F}_h(\mathbf{x})))$. Finally, OutputCom outputs $(\mathbf{y}^\top \otimes \mathbf{I}_d)\mathbf{J}_d = \mathsf{Com}(\mathsf{F}_h(\mathbf{x}))$ by Lemma 6.1. □

Lemma 6.7 (Uniqueness of layer proofs). *Let* f *be a layer,* σ *a layer digest and* $\mathbf{C} \in \mathbb{Z}^{d\times d}$. *Given two layer proofs* $\pi \neq \pi'$ *with* $\mathsf{VerifyLayer}(\mathsf{f}, \sigma, \mathbf{C}, \pi) = \mathsf{VerifyLayer}(\mathsf{f}, \sigma, \mathbf{C}, \pi') = 1$, *one can compute in polynomial time a nonzero* $\mathbf{w} \in \mathbb{Z}^m$ *with* $\mathbf{W}\mathbf{w} = \mathbf{0}$ *or* $\mathbf{V}\mathbf{w} = \mathbf{0}$, *and* $\|\mathbf{w}\| \le 2mB_{\mathsf{thr}}$.

Proof. Write the two accepted proofs as $(\mathbf{T}, \mathbf{L})$ and $(\mathbf{T}', \mathbf{L}')$, and let $\sigma = (h, \mathbf{Q})$ and $g := a + h$. Both proofs pass Valid in check 1, so their entries have absolute value at most B_{thr}.
Case 1: $\mathbf{T} \neq \mathbf{T}'$. Choose r, μ, ν for which

$$\mathbf{D} := \mathbf{T}_r[\mathbf{E}_{\mu\nu}] - \mathbf{T}'_r[\mathbf{E}_{\mu\nu}] \neq \mathbf{0}.$$

By check 2, the first $g+1$ block rows of $\mathbf{B}_g\mathbf{D}$ vanish. Also, $\|\mathbf{D}\| \le 2B_{\mathsf{thr}}$, so Lemma 6.3 gives the required kernel vector.
Case 2: $\mathbf{T} = \mathbf{T}'$. Then check 3 derives the same $\mathbf{R}$ from both proofs. Since the proofs differ, $\mathbf{L}_r \neq \mathbf{L}'_r$ for some r. Check 4 makes the first $g+1$ block rows of $\mathbf{B}_g(\mathbf{L}_r - \mathbf{L}'_r)$ vanish, and Lemma 6.3 again applies. These cases cover every field of the proof. □

6.3 Merging

For $h \ge 1$, the blockwise extension $\mathsf{F}_h = \mathsf{F}_{h-1}^{\otimes 2}$ applies F_{h-1} to each half of its input $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1)$, so every gate of F_h is a gate of F_{h-1} in one of the halves. We call the summaries of F_{h-1} on $\mathbf{x}_0$ and $\mathbf{x}_1$ the summaries of the children. Since commitments are linear, the parent's verification keys $\mathbf{V}_i$, openings $\mathbf{Z}_i(\mathbf{x})$ and output weights $w_{o,r}$ (the entries of the commitments $\mathbf{w}_o$ to unit vectors) are fixed linear functions of the children's. Substituting these into the definitions expresses each table of the parent as a linear combination of the children's tables, with coefficients determined by the setup and the two input commitments. A merge therefore needs only the two child proofs and their input commitments, and not the input vectors themselves.

Since Node is linear, there are integer matrices $\mathbf{N}_0, \mathbf{N}_1 \in \mathbb{Z}^{d^2\times d^2}$, depending only on the setup, with

$$\mathrm{vec}\big(\mathsf{Node}(\mathbf{D}_0, \mathbf{D}_1)\big) = \mathbf{N}_0\,\mathrm{vec}(\mathbf{D}_0) + \mathbf{N}_1\,\mathrm{vec}(\mathbf{D}_1) \qquad \text{for all } \mathbf{D}_0, \mathbf{D}_1 \in \mathbb{Z}^{d\times d}.$$

If output coordinate o lies in half b, the parent's unit vector $\mathbf{e}_o$ has the child's unit vector in half b and zeros in the other half, so $\mathbf{w}_o = \mathbf{N}_b\mathbf{w}_o^{(b)}$, that is,

$$w_{o,r} = \sum_{r'} (\mathbf{N}_b)_{r,r'}\, w^{(b)}_{o,r'},$$

where the child uses the local index of o.

Lemma 6.8 (Merging summaries). *Let child* $b \in \{0,1\}$ *have summaries* $\mathbf{Q}^{(b)}, \mathbf{T}^{(b)}, \mathbf{R}^{(b)}, \mathbf{L}^{(b)}$, *and let* $\mathbf{Z}' := \mathbf{Z}_{\mathbf{u}}$ *for* $\mathbf{u} = \mathrm{vec}([\mathsf{Com}(\mathbf{x}_0)|\mathsf{Com}(\mathbf{x}_1)])$. *Then the parent's summaries are*

$$\mathbf{Q}_r[\mathbf{M}] = \sum_{b,r'} (\mathbf{N}_b)_{r,r'}\, \mathbf{H}_b\, \mathbf{Q}^{(b)}_{r'}[\mathbf{M}\mathbf{H}_b], \qquad \mathbf{T}_r[\mathbf{M}] = \sum_{b,r'} (\mathbf{N}_b)_{r,r'} \begin{bmatrix} \mathbf{Z}'\mathbf{J}_b\mathbf{Q}^{(b)}_{r'}[\mathbf{M}\mathbf{H}_b] \\ \mathbf{T}^{(b)}_{r'}[\mathbf{M}\mathbf{H}_b] \end{bmatrix},$$

$$\mathbf{R}_r = \sum_{b,r'} (\mathbf{N}_b)_{r,r'}\, \mathbf{H}_b\mathbf{R}^{(b)}_{r'}, \qquad \mathbf{L}_r = \sum_{b,r'} (\mathbf{N}_b)_{r,r'} \begin{bmatrix} \mathbf{Z}'\mathbf{J}_b\mathbf{R}^{(b)}_{r'} \\ \mathbf{L}^{(b)}_{r'} \end{bmatrix}.$$

Proof. Since $\mathsf{F}_h = \mathsf{F}_{h-1}^{\otimes 2}$, every gate of F_h lies in one half b, and its coefficient is the coefficient α_{ijo} of F_{h-1} for the indices i, j, o local to that half. For such a gate, the recursions for $\overline{\mathbf{V}}$ and $\mathbf{Z}$ in Section 6.1 give

$$\mathbf{V}_i = \mathbf{H}_b\mathbf{V}^{(b)}_i, \qquad \mathbf{Z}_i(\mathbf{x}) = \begin{bmatrix} \mathbf{Z}'\mathbf{J}_b\mathbf{V}^{(b)}_i \\ \mathbf{Z}^{(b)}_i(\mathbf{x}_b) \end{bmatrix},$$

and $w_{o,r} = \sum_{r'} (\mathbf{N}_b)_{r,r'} w^{(b)}_{o,r'}$, where on the left the indices refer to the parent's coordinates. Substituting these into the definition of $\mathbf{Q}_r$,

$$\begin{aligned}\mathbf{Q}_r[\mathbf{M}] &= \sum_b \sum_{i,j,o} \alpha_{ijo} w_{o,r}\, \mathbf{V}_i \mathbf{M} \mathbf{V}_j = \sum_b \sum_{i,j,o} \alpha_{ijo} \sum_{r'} (\mathbf{N}_b)_{r,r'} w^{(b)}_{o,r'} \left(\mathbf{H}_b \mathbf{V}^{(b)}_i\right) \mathbf{M} \left(\mathbf{H}_b \mathbf{V}^{(b)}_j\right) \\ &= \sum_{b,r'} (\mathbf{N}_b)_{r,r'}\, \mathbf{H}_b \Big(\sum_{i,j,o} \alpha_{ijo} w^{(b)}_{o,r'}\, \mathbf{V}^{(b)}_i (\mathbf{M}\mathbf{H}_b) \mathbf{V}^{(b)}_j \Big) = \sum_{b,r'} (\mathbf{N}_b)_{r,r'}\, \mathbf{H}_b\, \mathbf{Q}^{(b)}_{r'}[\mathbf{M}\mathbf{H}_b].\end{aligned}$$

The third step moves the scalars to the front and regroups $\mathbf{M}(\mathbf{H}_b \mathbf{V}^{(b)}_j) = (\mathbf{M}\mathbf{H}_b)\mathbf{V}^{(b)}_j$. This is why the children's tables are evaluated at $\mathbf{M}\mathbf{H}_b$, which in general differs from $\mathbf{H}_b\mathbf{M}$. The formulas for $\mathbf{R}$, $\mathbf{T}$ and $\mathbf{L}$ follow in the same way. In $\mathbf{R}_r$, the single factor $\mathbf{V}_j$ contributes one factor $\mathbf{H}_b$. In $\mathbf{T}_r$ and $\mathbf{L}_r$, the first block row comes from the first block row $\mathbf{Z}'\mathbf{J}_b\mathbf{V}^{(b)}_i$ of $\mathbf{Z}_i(\mathbf{x})$, which yields $\mathbf{Z}'\mathbf{J}_b\mathbf{Q}^{(b)}_{r'}[\mathbf{M}\mathbf{H}_b]$ and $\mathbf{Z}'\mathbf{J}_b\mathbf{R}^{(b)}_{r'}$, and the remaining block rows come from $\mathbf{Z}^{(b)}_i(\mathbf{x}_b)$, which yields the children's tables $\mathbf{T}^{(b)}_{r'}[\mathbf{M}\mathbf{H}_b]$ and $\mathbf{L}^{(b)}_{r'}$. □

These identities give both DoubleDigest and Merge. Every evaluation at $\mathbf{M}\mathbf{H}_b$ is computed as a linear combination of the stored elementary-matrix entries.

Both halves of F_h apply the same blockwise extension, so the children's $\mathbf{Q}$ tables agree.

DoubleDigest(f, σ):

1. Parse $\sigma = (h, \mathbf{Q})$.
2. Compute the parent table $\mathbf{Q}'$ by the first identity of Lemma 6.8, with $\mathbf{Q}^{(0)} = \mathbf{Q}^{(1)} = \mathbf{Q}$.
3. Output $(h+1, \mathbf{Q}')$.

Merge$(\mathsf{f}, \sigma, (\mathbf{C}_0, \pi_0), (\mathbf{C}_1, \pi_1))$:

1. Parse $\sigma = (h, \mathbf{Q})$ and $\pi_b = (\mathbf{T}^{(b)}, \mathbf{L}^{(b)})$ for $b \in \{0,1\}$, and derive $\mathbf{R}^{(b)}$ from $\mathbf{T}^{(b)}$ as in check 3 of VerifyLayer.
2. Set $\mathbf{u} := \mathrm{vec}([\mathbf{C}_0|\mathbf{C}_1])$ and $\mathbf{Z}' := \mathsf{Open}(\mathbf{u}) = \mathbf{u} \otimes \mathbf{I}_{L_0 d}$. // $\mathbf{u}$ has height 0
3. Compute the parent tables $(\mathbf{T}, \mathbf{L})$ by Lemma 6.8, with $\mathbf{Q}^{(0)} = \mathbf{Q}^{(1)} = \mathbf{Q}$, and output them.

Thus a merge uses only the layer digest, the two child proofs and their input commitments. It does not read the committed vectors. On honest child proofs, step 1 of Merge recovers the honest tables $\mathbf{R}^{(b)}$, since by Lemma 6.5 with $\mathbf{M} = \mathbf{I}_d$ the last block row of $\mathbf{B}_g \mathbf{T}^{(b)}_r[\mathbf{I}_d]$ is $-\mathbf{R}^{(b)}_r$. By induction on h, Lemma 6.8 therefore shows that the recursively defined $\mathsf{Prove}_h(\mathbf{x})$ of Section 5.2 consists exactly of the summaries of $\mathbf{x}$ for F_h. The same induction, using the first identity of Lemma 6.8, shows that $\sigma_{\mathsf{f},h} = (h, \mathbf{Q})$ for the table $\mathbf{Q}$ of summaries of F_h. This establishes merging for honest proofs.

Merging plays no role in uniqueness. Lemma 6.7 applies to any two proofs accepted by VerifyLayer at the same height, however they were produced, and VerifyLayer checks a merged proof in the same way as any other. Since Merge is run only by the honest prover, it suffices that it maps honest child proofs to the honest parent proof. We do not require arbitrary accepted child proofs to merge into an accepted parent.

6.4 Input commitments

Just as DoubleDigest computes a layer digest without visiting the gates, InputCom computes an input commitment without visiting the leaves, by applying Node to copies of commitments to repeated patterns. The verifier runs it in VerifyNode, since it checks the proof for a node without the proofs for the node's children. The input of a node η is $(\iota(\eta b))_{b \in \{0,1\}^h}$, where $\iota(a) = (1, a_0, \ldots, a_{p-1})$ is padded with zeros to length L_0. In this subsection, number the coordinates of a padded address starting at 0: $\mathbf{e}_0$ selects the constant coordinate, and $\mathbf{e}_{j+1}$ selects address bit a_j. Each unit vector has length L_0. Then $\iota(a) = \mathbf{e}_0 + \sum_j a_j \mathbf{e}_{j+1}$: the

constant coordinate, plus one unit vector for each address bit equal to 1. Summed over the leaves of a node, the input is therefore a sum of a few repeated patterns, and by linearity, so is its commitment. Recall that $\mathrm{Com}(\mathbf{v}) = \mathrm{Node}(\mathrm{Com}(\mathbf{v}_0), \mathrm{Com}(\mathbf{v}_1))$ for every $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1)$, and that the commitment to a zero vector is $\mathbf{0}$.

Lemma 6.9 (Input commitments)**.** *For $h \le p$ and $c \in [0, p]$, let $\mathbf{U}_h[c]$ be the commitment to 2^h copies of $\mathbf{e}_c$, and for $j < h$, let $\mathbf{U}'_h[j] := \mathrm{Com}\big((b_j \mathbf{e}_{j+1})_{b \in \{0,1\}^h}\big)$, with the strings $b = b_{h-1} \cdots b_0$ in lexicographic order. Starting from $\mathbf{U}_0[c] = \mathrm{Com}(\mathbf{e}_c)$, for $h \ge 1$ we have*

$$\begin{aligned} \mathbf{U}_h[c] &= \mathrm{Node}(\mathbf{U}_{h-1}[c], \mathbf{U}_{h-1}[c]), \\ \mathbf{U}'_h[j] &= \mathrm{Node}(\mathbf{U}'_{h-1}[j], \mathbf{U}'_{h-1}[j]) \quad \text{for } j < h-1, \qquad \mathbf{U}'_h[h-1] = \mathrm{Node}(\mathbf{0}, \mathbf{U}_{h-1}[h]). \end{aligned}$$

For a node $\eta = a_{p-1} \cdots a_h$ of height h,

$$\mathrm{Com}\big((\iota(\eta b))_{b \in \{0,1\}^h}\big) = \mathbf{U}_h[0] + \sum_{j<h} \mathbf{U}'_h[j] + \sum_{j \ge h:\, a_j = 1} \mathbf{U}_h[j+1].$$

Proof. For $h' \le p$, write $\mathbf{a}_{h'}[c] := (\mathbf{e}_c)_{b \in \{0,1\}^{h'}}$ for $2^{h'}$ copies of $\mathbf{e}_c$ and $\mathbf{p}_{h'}[j] := (b_j \mathbf{e}_{j+1})_{b \in \{0,1\}^{h'}}$, so that $\mathbf{U}_{h'}[c] = \mathrm{Com}(\mathbf{a}_{h'}[c])$ and $\mathbf{U}'_{h'}[j] = \mathrm{Com}(\mathbf{p}_{h'}[j])$. For the address ηb, the bits a_j with $j < h$ are those of b, and the bits with $j \ge h$ are those of η. Hence

$$\iota(\eta b) = \mathbf{e}_0 + \sum_{j<h} b_j \mathbf{e}_{j+1} + \sum_{j \ge h} a_j \mathbf{e}_{j+1}.$$

Concatenating over $b \in \{0,1\}^h$ in lexicographic order, and noting that the bits a_j of η do not depend on b,

$$(\iota(\eta b))_{b \in \{0,1\}^h} = \mathbf{a}_h[0] + \sum_{j<h} \mathbf{p}_h[j] + \sum_{j \ge h} a_j \, \mathbf{a}_h[j+1].$$

Applying the linear map Com, and using $a_j \in \{0,1\}$, gives the formula for the input commitment.

For the recurrences, let $h' \ge 1$ and split the strings $b \in \{0,1\}^{h'}$ by their most significant bit $b_{h'-1}$. The first half of each vector corresponds to $b_{h'-1} = 0$, the second to $b_{h'-1} = 1$, and in both halves the remaining bits range over $\{0,1\}^{h'-1}$ in lexicographic order. Therefore

$$\begin{aligned} \mathbf{a}_{h'}[c] &= \big(\mathbf{a}_{h'-1}[c], \mathbf{a}_{h'-1}[c]\big), \\ \mathbf{p}_{h'}[j] &= \big(\mathbf{p}_{h'-1}[j], \mathbf{p}_{h'-1}[j]\big) \quad \text{for } j < h'-1, \\ \mathbf{p}_{h'}[h'-1] &= \big(\mathbf{0}, \mathbf{a}_{h'-1}[h']\big). \end{aligned}$$

The first two hold because a bit other than $b_{h'-1}$ takes the same values in both halves, and the third because $b_{h'-1}$ is 0 in the first half and 1 in the second. Applying $\mathrm{Com}(\mathbf{v}) = \mathrm{Node}(\mathrm{Com}(\mathbf{v}_0), \mathrm{Com}(\mathbf{v}_1))$ and $\mathrm{Com}(\mathbf{0}) = \mathbf{0}$ gives the three recurrences. □

This gives the following algorithm, which makes $O(p^2)$ calls to Node.

InputCom(p, η), for a node $\eta = a_{p-1} \cdots a_h$ of height h:

1. For $c \in [0, p]$: set $\mathbf{U}_0[c] := \mathrm{BaseCom}(\mathbf{e}_c)$.
2. For $h' = 1, \ldots, h$: compute $\mathbf{U}_{h'}[c]$ for $c \in [0, p]$ and $\mathbf{U}'_{h'}[j]$ for $j < h'$ by the recurrences of Lemma 6.9.
3. Output $\mathbf{U}_h[0] + \sum_{j<h} \mathbf{U}'_h[j] + \sum_{j \ge h:\, a_j = 1} \mathbf{U}_h[j+1]$.

6.5 Norms and the main theorem

The verifier bounds the absolute values of proof entries by B_{thr}. We must choose this cutoff large enough for every honest proof and small enough that the kernel vector extracted from two accepted proofs has norm at most β. We first bound honest entries for Boolean inputs, then choose the parameters and check both inequalities. We give one parameter choice for $H = \log^2 \lambda$ and another for $H = \lambda$. In each choice, fixed constants make the completeness and extraction bounds hold for every security parameter.

Recap of dimensions. Recall from Section 6.1 that $\Lambda = H + \lceil \log N \rceil + 2$, that d is $n \log q$ rounded up to a power of two, and that $m = 2d^3$, $L_0 = 2d^2$ and $\ell_B = L_0 m = 4d^5$. The matrices $\mathbf{W}, \mathbf{V}$ are binary with d rows and m columns. For a layer f with $|\mathsf{f}| \le N$ and $h \le H$, the input and output heights of F_h are at most $\Lambda - 2$, since $2^a, 2^{a'} \le N$. For a vector $\mathbf{x}$ of height g and length $\ell = L_0 2^g$, and a layer proof of input height g, the matrices of Sections 6.1 and 6.2 have the following dimensions.

commitment	$\mathrm{Com}(\mathbf{x})$	$d \times d$
verification key	$\overline{\mathbf{V}}_\ell$	$d \times \ell d$
opening	$\mathbf{Z}_{\mathbf{x}}$	$(g+1)\ell_B \times \ell d$
verification matrix	$\mathbf{B}_g$	$(g+2)d \times (g+1)\ell_B$
digest, summary entries	$\mathbf{Q}_r[\mathbf{E}_{\mu\nu}]$, $\mathbf{R}_r$	$d \times d$
summary entries	$\mathbf{T}_r[\mathbf{E}_{\mu\nu}]$, $\mathbf{L}_r$	$(g+1)\ell_B \times d$

It remains to choose n, q, β and B_{thr}. The lengths d, m, L_0 and ℓ_B then follow.

Lemma 6.10 (Honest norms). *For every Boolean $\mathbf{x}$ of height g,*

$$\|\mathrm{Com}(\mathbf{x})\| \le L_0^{g+1}, \qquad \left\|\overline{\mathbf{V}}_\ell\right\| \le (2d^2)^g, \qquad \|\mathbf{Z}_{\mathbf{x}}\| \le (16d^9)^{g+2}.$$

Proof. An entry of $\mathbf{W}(\mathbf{x} \otimes \mathbf{I}_d)$ is a sum of at most L_0 entries of $\mathbf{x}$, so it is at most $L_0\|\mathbf{x}\|$. At a parent, $\mathbf{u}_{\mathbf{x}}$ has length L_0, so induction gives L_0^{g+1}. The entries of $\mathbf{VJ}$ are sums of d bits, so each level of $\overline{\mathbf{V}}_\ell$ multiplies the bound by at most $2d^2$. For $g = 0$, the opening matrix has norm at most 1. For $g \ge 1$, its top block is $\mathbf{Z}_{\mathbf{u}}\mathbf{J}(\mathbf{I}_2 \otimes \overline{\mathbf{V}}_{\ell/2})$. Since $\|\mathbf{Z}_{\mathbf{u}}\| = \|\mathbf{u}\| \le L_0^g$ and the two inner dimensions are m and $2d$, this block has norm at most

$$2dm\, L_0^g (2d^2)^{g-1} = 2^{2g+1} d^{4g+2} \le (16d^9)^{g+2}.$$

The remaining rows are the children's openings, which satisfy the same bound by induction. □

Lemma 6.11 (Summary norms). *Let f be a layer with $|\mathsf{f}| \le N$ and let $h \le H$. The digest table $\mathbf{Q}$ of F_h, the summary tables $\mathbf{T}, \mathbf{R}, \mathbf{L}$ of F_h on every Boolean input, and every commitment in the commitment trees of Boolean inputs and outputs of F_h, have norm at most $2^{C_{\mathrm{norm}} \Lambda \log d}$, where $C_{\mathrm{norm}} := 41$.*

Proof. Let s be the number of nonzero coefficients of F_h, so $s \le 2^h N$. Each entry of a summary table is a sum of at most s terms. A term consists of a coefficient $\alpha_{ijo} w_{o,r}$, of absolute value at most $L_0^{g'+1}$ since $\alpha_{ijo} \in \{-1, 0, 1\}$ and $\mathbf{w}_o$ is the commitment to a Boolean vector of height g', times a product of at most two of the matrices bounded in Lemma 6.10 and an elementary matrix. There are at most three inner dimensions, each of size at most ℓ_B. If $s = 0$, every summary is zero. Otherwise, its norm is at most 2^Δ, where

$$\Delta := \log s + 3 \log \ell_B + (g' + 1) \log L_0 + 2(g+2) \log(16d^9).$$

Since $2^a, 2^{a'} \le |\mathsf{f}| \le N$ and $h \le H$, we have $g, g' \le \Lambda - 2$ and $\log s \le \Lambda$. Substituting $\ell_B = 4d^5$ and $L_0 = 2d^2$, and using $g' + 1 \le \Lambda$ and $g + 2 \le \Lambda$,

$$\Delta \le \Lambda + (6 + 15 \log d) + \Lambda(1 + 2 \log d) + 2\Lambda(4 + 9 \log d) = 10\Lambda + 6 + 15 \log d + 20\Lambda \log d.$$

Since $\Lambda \geq 2$ and $\log d \geq 1$, we have $10\Lambda \leq 10\Lambda \log d$, $6 \leq 3\Lambda \log d$ and $15 \log d \leq 7.5\Lambda \log d$, so $\Delta \leq 40.5\Lambda \log d \leq 41\Lambda \log d$. The bounds of Lemma 6.10 on commitments are smaller. □

Requirements on the parameters. The parameters must satisfy four requirements, each coming from a different part of the analysis.

- *Completeness.* Valid must accept every honest proof and commitment, so by Lemma 6.11 we need $2^{C_{\text{norm}}\Lambda \log d} \leq B_{\text{thr}}$.
- *Extraction.* Two accepted layer proofs, or two colliding inputs, have entries in $[-B_{\text{thr}}, B_{\text{thr}}]$, so their difference $\mathbf{D}$ has norm at most $2B_{\text{thr}}$. A tree collision gives the kernel vector $(\mathbf{u} - \mathbf{u}') \otimes \mathbf{e}_1$ of norm at most $2B_{\text{thr}}$ (Lemma 6.4). For a binding pair, Case 1 of Lemma 6.3 may output $\mathbf{w} = (\mathbf{I}_{L_0} \otimes \mathbf{V})\mathbf{v}$ for a column $\mathbf{v}$ of $\mathbf{D}$. Each entry of $\mathbf{w}$ is a sum of at most m entries of $\mathbf{v}$, since $\mathbf{V}$ is binary with m columns, so $\|\mathbf{w}\| \leq 2mB_{\text{thr}}$ (Lemma 6.7). By Lemma 3.7, these kernel vectors are SIS solutions if their norm is at most β, so we need $2mB_{\text{thr}} \leq \beta$.
- *Modulus.* Assumptions 3.5 and 3.6 require $n\sqrt{m} \cdot \beta \leq q$.
- *Base length.* Definition 5.5 requires $H < L_0$, so that each leaf input, of length $p + 1 \leq H + 1$, fits in a single block.

The first two requirements determine the size of the norm bound. Up to constant factors in the exponent, they hold with $\log B_{\text{thr}}$ and $\log \beta$ both of order $\Lambda \log d$, since $\log m = O(\log d)$. As d is polynomial in n, this is $\Lambda \log n$ up to constants, and $\Lambda = H + \lceil \log N \rceil + 2$ is governed by the height bound. Neither assumption lets us choose β freely: Assumption 3.6 fixes $\log \beta = C\lceil \log n \rceil^3$, and Assumption 3.5 fixes $\log \beta = n^e$. The parameters we control are the dimension n, and in the first case the constant C, and we choose them so that this β is large enough. For $H = \log^2 \lambda$, we take $n \geq \lambda, N$, so that $\Lambda = O(\log^2 n)$ and $\log \beta = O(\log^3 n)$, a quasi-polynomial norm bound. Since this has the same form as $\log \beta = C\lceil \log n \rceil^3$, it suffices that C is a large enough constant, which is why Assumption 3.6 asks for one. For $H = \lambda$, we have $\Lambda > \lambda$, so our bound $2^{C_{\text{norm}}\Lambda \log d}$ on honest norms is exponential in $\lambda \log d$. We take $\beta = 2^{n^e}$ and choose n polynomial in Λ, large enough that n^e is at least a large constant multiple of $\Lambda \log d$. In both cases, q exceeds β by a polynomial factor, which gives the modulus requirement, and $L_0 = 2d^2 \geq 2n^2$ exceeds H.

Parameters for $H = \log^2 \lambda$. Fix the constant C in Assumption 3.6, with $C \geq 492$, and set $K := C/2 \geq 246$. Choose a constant n_0 large enough that

$$2K\lceil \log n \rceil^3 + 6\lceil \log n \rceil + 7 \leq n \qquad \text{for every } n \geq n_0.$$

Set $n := \max\{n_0, \lambda, N\}$ and $\rho := \lceil \log n \rceil$, and take

$$B_{\text{thr}} := 2^{K\rho^3}, \qquad \beta := 2^{2K\rho^3}, \qquad q := 2^{2K\rho^3 + 6\rho + 7}.$$

The lengths d, m, L_0, ℓ_B are derived from n, q as before. Choosing n first avoids any circular dependence through d or m. In terms of the requirements above, B_{thr} is chosen to dominate the completeness exponent, the choice $\beta = B_{\text{thr}}^2$ leaves room for the factor $2m$ from extraction, and q exceeds β by a polynomial factor, which covers the modulus requirement. We verify these in Lemma 6.12 below.

Lemma 6.12 (Quasi-polynomial norm parameters)**.** *For this choice of parameters, for every $\lambda \geq 2$ and $N \geq 1$,*

$$2^{C_{\text{norm}}\Lambda \log d} \leq B_{\text{thr}}, \qquad 2mB_{\text{thr}} \leq \beta, \qquad n\sqrt{m} \cdot \beta \leq q, \qquad H < L_0.$$

If N is polynomial in λ, then $n, d, m, \log q$ are polynomial in λ, $n \geq \lambda$, and these are the parameters of Assumption 3.6.

Proof. Increase n_0 if necessary so that $\rho \geq 2$. We first bound d and Λ in terms of ρ. By the choice of n_0, $\log q = 2K\rho^3 + 6\rho + 7 \leq n$. Rounding up to a power of two at most doubles, so $d \leq 2n \log q \leq 2n^2$, and since $n \leq 2^\rho$, $\log d \leq 1 + 2\rho \leq 3\rho$. Since $\lambda, N \leq n$, we have $\log \lambda \leq \rho$ and $\lceil \log N \rceil \leq \rho$, and $H = \lceil \log^2 \lambda \rceil \leq \rho^2$ because ρ^2 is an integer. Using $\rho \geq 2$,

$$\Lambda = H + \lceil \log N \rceil + 2 \leq \rho^2 + \rho + 2 \leq 2\rho^2.$$

For completeness, this gives

$$C_{\mathsf{norm}} \Lambda \log d \leq 41 \cdot 2\rho^2 \cdot 3\rho = 246\rho^3 \leq K\rho^3 = \log B_{\mathsf{thr}},$$

where the last inequality is $K \geq 246$. This is where the requirement $C = 2K \geq 492$ comes from. For extraction, $\log(2m) = \log(4d^3) = 2 + 3 \log d \leq 2 + 9\rho \leq K\rho^3$, so

$$\log(2mB_{\mathsf{thr}}) = \log(2m) + K\rho^3 \leq 2K\rho^3 = \log \beta.$$

For the modulus, $m = 2d^3 \leq 16n^6$ gives $\sqrt{m} \leq 4n^3$, and $n^6 \leq 2^{6\rho}$, so

$$n\sqrt{m} \leq 4n^4 < 128n^6 \leq 2^{6\rho+7} = q/\beta.$$

For the base length, $d \geq n \log q \geq n \geq 2^{\rho-1}$, so $L_0 = 2d^2 \geq 2n^2 \geq 4^\rho/2 > \rho^2 \geq H$. Finally, if N is polynomial in λ, then so is $n = \max\{n_0, \lambda, N\}$, and $\log q \leq n$, $d \leq 2n^2$ and $m = 2d^3$ are polynomial as well. Moreover, $\beta = 2^{C\rho^3}$ with $C = 2K$, and q is a power of two with $q/\beta = 2^{6\rho+7} \leq 2^{13}n^6$, since $\rho \leq \log n + 1$. These are the parameters of Assumption 3.6. □

Parameters for $H = \lambda$. Fix a constant $0 < e < 1$, and let $K = K(e)$ be the constant of Lemma 6.13 below. Here $\Lambda = \lambda + \lceil \log N \rceil + 2$. Set

$$B_{\mathsf{thr}} := 2^{\lceil K\Lambda^{5/4} \rceil}, \qquad n := \left\lceil (2K\Lambda^{5/4})^{1/e} \right\rceil, \qquad \beta := 2^{n^e}, \qquad q := 2^{\lceil n^e \rceil + \lceil 6 \log n \rceil + 7}.$$

The lengths d, m, L_0 and ℓ_B are then derived from n and q as in Section 6.1. In terms of the requirements above, $\log d = O_e(\log K + \log \Lambda)$ here, so the completeness exponent $C_{\mathsf{norm}} \Lambda \log d$ is at most $K\Lambda^{5/4}$ for a large enough constant K. Since d depends on n, defining n directly in terms of $\Lambda \log d$ would be circular, so we bound $\Lambda \log d$ by $K\Lambda^{5/4}$. Any exponent larger than 1 in place of $5/4$ would do, since $\log d$ grows only logarithmically in K and Λ. An exponent $a > 1$ would give the constraint $\nu > e/a$ in place of $\nu > 4e/5$ in Corollary 6.16, and we use $5/4$ for concreteness. The dimension n is chosen so that $n^e \geq 2K\Lambda^{5/4}$, which leaves room for the factor $2m$ from extraction, and q again exceeds β by a polynomial factor.

Lemma 6.13 (Parameters)**.** *There is a constant $K = K(e)$ such that, for all λ and N,*

$$2^{C_{\mathsf{norm}} \Lambda \log d} \leq B_{\mathsf{thr}}, \qquad 2mB_{\mathsf{thr}} \leq \beta, \qquad n\sqrt{m} \cdot \beta \leq q, \qquad \lambda < L_0.$$

Moreover, if N is polynomial, then $n, \log q, d, m$ are polynomial in λ, $n \geq \lambda^{1/e}$, and $q \leq 2^{2\lceil n^e \rceil}$ for all sufficiently large λ. In particular, n, m, q, β are in the range of Assumption 3.5.

Proof. Since $\log q \leq 8n$, we have $d \leq 16n^2$ and $\log d \leq 4 + 2 \log n = O_e(\log K + \log \Lambda)$. Choose $K = K(e) \geq 1$ large enough that, for every $\Lambda \geq 2$,

$$C_{\mathsf{norm}} \Lambda \log d \leq K\Lambda^{5/4}, \qquad 3 + 3 \log d \leq K\Lambda^{5/4}.$$

Such a constant exists because $\log d = O_e(\log K + \log \Lambda)$ and $\log \Lambda = O(\Lambda^{1/4})$. The first inequality gives $2^{C_{\mathsf{norm}} \Lambda \log d} \leq B_{\mathsf{thr}}$. Using the second and the ceiling in the definition of B_{thr}, together with $\log(2m) = 2 + 3 \log d$ and $\log B_{\mathsf{thr}} \leq K\Lambda^{5/4} + 1$, gives

$$\log(2mB_{\mathsf{thr}}) \leq K\Lambda^{5/4} + 3 + 3 \log d \leq 2K\Lambda^{5/4} \leq n^e.$$

Next, $\log\sqrt{m} = (1 + 3\log d)/2 \le 6.5 + 3\log n$, so $\log(n\sqrt{m}\cdot\beta) \le n^e + 4\log n + 6.5 \le \log q$. For the base length, $L_0 > d \ge \log q \ge n^e \ge 2K\Lambda^{5/4}$, and the first inequality on K gives $K\Lambda^{5/4} \ge C_{\mathsf{norm}}\Lambda\log d \ge \Lambda > \lambda$. In particular, $n \ge (2K\Lambda^{5/4})^{1/e} \ge \lambda^{5/(4e)} \ge \lambda^{1/e}$, since $K \ge 1$ and $\Lambda > \lambda$, and n is polynomial in λ when N is, since then $\Lambda = O(\lambda)$. The remaining claims follow from the definitions, since $\lceil 6\log n\rceil + 7 \le \lceil n^e\rceil$ for all sufficiently large n. □

Theorem 6.14 (Mergeable functional commitment from SIS). *With either parameter choice above and its corresponding height bound H, the construction of this section satisfies efficiency, completeness and input commitments of Definition 5.5. Moreover, every tree collision or binding pair can be mapped in polynomial time to a nonzero kernel vector of $\mathbf{W}$ or $\mathbf{V}$ of norm at most β. Consequently, for every polynomial N there is a polynomial c such that, for every s, if $\mathsf{SIS}_{n,m,q,\beta}$ with the parameters for the size bound N holds against size $s + c$ with advantage δ, then the construction is $(s, 2\delta)$-binding for the size bound N.*

Proof. We verify the properties of Definition 5.5 in turn, and then reduce binding to SIS.

Efficiency. The base length satisfies $H < L_0 \le \mathrm{poly}(\lambda, N)$ by Lemmas 6.12 and 6.13. Objects accepted by Valid and digests have $\mathrm{poly}(d, \Lambda)$ integer entries of $O(\log B_{\mathsf{thr}})$ bits, by Lemma 6.11 and the corresponding parameter lemma for layer digests. Both parameter choices give polynomial bit lengths. Each algorithm computes polynomially many integer matrix products of polynomial dimension, and rejects inputs of the wrong dimensions.

Completeness. Let $\mathbf{x}$ and $\mathsf{F}_h(\mathbf{x})$ be Boolean, with $|\mathsf{f}| \le N$ and $h \le H$. By Section 6.3, $\mathsf{Prove}_h(\mathbf{x})$ is the honest layer proof of $\mathbf{x}$ for F_h, and $\sigma_{\mathsf{f},h} = (h, \mathbf{Q})$ for the table $\mathbf{Q}$ of summaries of F_h. Since $2^a \le |\mathsf{f}| \le N$ and $h \le H$, the input height satisfies $g \le \Lambda - 2$. By Lemma 6.11 and the corresponding parameter lemma, the entries of $\mathsf{Prove}_h(\mathbf{x})$ and of all leaf vectors and commitments in the trees of $\mathbf{x}$ and $\mathsf{F}_h(\mathbf{x})$ have absolute value at most B_{thr}, so they are accepted by Valid. Lemma 6.6 then shows that $\mathsf{Prove}_h(\mathbf{x})$ proves $\mathsf{Com}(\mathbf{x}) \xrightarrow{\mathsf{F}_h} \mathsf{Com}(\mathsf{F}_h(\mathbf{x}))$.

Input commitments. This is Lemma 6.9.

Kernel vectors. A binding pair gives a kernel vector of norm at most $2mB_{\mathsf{thr}}$ by Lemma 6.7, and a tree collision gives one of norm at most $2B_{\mathsf{thr}}$ by Lemma 6.4. Both are at most β by Lemmas 6.12 and 6.13.

Binding. Given an SIS challenge $\mathbf{A}$, set $\mathbf{W}' := \mathbf{A}$ or $\mathbf{V}' := \mathbf{A}$, each with probability $1/2$, and sample the other matrix uniformly. The resulting crs has the distribution of Setup, independently of this choice. Run the binding adversary and map its output to a kernel vector as above. With probability $1/2$, the kernel vector belongs to the matrix holding the challenge, and Lemma 3.7 makes it an SIS solution. □

Combining Theorem 6.14 with Theorem 5.9 gives our uuSNARG.

Theorem 6.15 (uuSNARG from SIS). *Let $\mathcal{U}$ be Construction 5.8 instantiated with either parameter choice for the commitment of this section, with its corresponding height bound H. Then $\mathcal{U}$ is a* uuSNARG *for regular tree circuits that satisfies completeness and succinctness. Moreover, for every polynomial N there is a polynomial c such that, for every s, if $\mathsf{SIS}_{n,m,q,\beta}$ with the parameters for the size bound $N_{\mathsf{a}}(N)$ holds against size $s + c$ with advantage δ, then $\mathcal{U}$ has $(s, 2\delta)$-state uniqueness for the size bound N. In particular, Assumption 3.6 gives state uniqueness for $H = \log^2\lambda$, and Assumption 3.5 gives state uniqueness for $H = \lambda$.*

Proof. Completeness and succinctness follow from Theorems 5.9 and 6.14. For uniqueness, let $\mathcal{A}$ break state uniqueness with probability ϵ. Given an SIS challenge $\mathbf{A}$, the reduction sets $\mathbf{W}' := \mathbf{A}$ or $\mathbf{V}' := \mathbf{A}$, each with probability $1/2$, and samples the other matrix uniformly. The resulting crs has the distribution of Setup, independently of which matrix holds the challenge. By Theorem 5.9, the reduction obtains a tree collision or a binding pair with probability at least ϵ, which Theorem 6.14 maps to a kernel vector of $\mathbf{W}$ or $\mathbf{V}$ of norm at most β. With probability $1/2$, it is a kernel vector for the matrix that holds the challenge, and then Lemma 3.7 makes it an SIS solution. Taking c to include both the compiler reduction and the kernel extraction, the resulting SIS adversary has size at most $s + c$. Hence $\delta \geq \epsilon/2$. The last claims follow from Lemmas 6.12 and 6.13. □

Corollary 6.16 (Leveraged uniqueness from SIS)**.** *Use the parameter choice for $H = \lambda$, and let $\nu > 4e/5$ be a constant. If $\mathsf{SIS}_{n,m,q,\beta}$, with the parameters of Theorem 6.15, holds against size 2^{n^ν} with negligible advantage, then $\mathcal{U}$ has the state uniqueness required by Theorem 4.7.*

Proof. Theorem 4.7 requires $(s + 2^{\lambda+1}c', \mathsf{negl}(\lambda))$-state uniqueness for every polynomial s and a fixed polynomial c'. Since $n^\nu \geq (2K\Lambda^{5/4})^{\nu/e} \geq \lambda^{5\nu/(4e)}$ and $5\nu/(4e) > 1$, the size $s + 2^{\lambda+1}c' + c$ is below 2^{n^ν} for all sufficiently large λ, so Theorem 6.15 applies. □

7 Instantiations

We instantiate the hard computations used in Section 4.2. Planted witness search and clique polynomials give average-case tree puzzles, and hence hard EOML distributions under polynomially hard SIS. With subexponentially hard SIS, a smaller SIS instance supplies the hard computation itself. In Section A, we also obtain worst-case hardness from polynomially hard SIS and non-uniform ETH. For planted search and clique polynomials, $\mathcal{U}$ denotes the uuSNARG of Theorem 6.15 with $H = \log^2 \lambda$ and the quasi-polynomial norm parameters of Lemma 6.12. Each hard computation is the output of a regular tree circuit. The exhaustive searches use the circuits C_φ of Lemma 3.9, and for the clique polynomials we describe the circuit directly.

7.1 Planted search

Suppose finding a planted witness of length ℓ requires exponential-size circuits. Taking ℓ to be a sufficiently large multiple of $\log \lambda$ then gives a puzzle that is hard for a prescribed polynomial budget, while exhaustive search still takes polynomial time in λ.

Assumption 7.1 (Exponentially hard planted search)**.** *There is a polynomial-time sampler* $\mathsf{Puz}(1^\ell) \to (z, w)$ *with* $w \in \{0,1\}^\ell$ *and* $\mathsf{R}(z, w) = 1$*, for a polynomial-time relation* R*, and a constant* $\gamma > 0$*, such that for all sufficiently large* ℓ*, every circuit of size at most* $2^{\gamma\ell}$ *given* z *outputs some* w' *with* $\mathsf{R}(z, w') = 1$ *with probability at most* $2^{-\gamma\ell}$*.*

Planting guarantees that every sampled instance has a witness, so that the trivial answer $(0, 0^\ell)$ is never correct. For example, take w uniform in $\{0,1\}^\ell$ and $z = g(w)$ for an exponentially hard one-way function g, with $\mathsf{R}(z, w') = 1$ if $g(w') = z$. Define the puzzle by setting $\ell := \lfloor \log t \rfloor$:

- $\mathcal{D}_{\mathsf{puz}}(t, 1^\lambda)$ samples $(z, w) \leftarrow \mathsf{Puz}(1^\ell)$ and outputs z.
- $\mathcal{F}_{\mathsf{puz}}(t, z)$ is the output of C_φ (Lemma 3.9) for $\varphi := \mathsf{R}(z, \cdot)$ on $\{0,1\}^\ell$. It is $(1, w)$ for the first witness w.

Lemma 7.2. *Under Assumption 7.1,* $(\mathcal{F}_{\mathsf{puz}}, \mathcal{D}_{\mathsf{puz}})$ *is an average-case tree puzzle.*

Proof. The sampler runs in time $\mathsf{poly}(\log t)$. Exhaustive search takes time $2^\ell \cdot \mathsf{poly}(\ell, |z|) \le t \cdot \mathsf{poly}(\log t, |z|)$, and Lemma 3.9 gives the required tree structure.

For hardness, fix a polynomial p and $r \in \mathbb{N}$, and choose a so that $\mathsf{p}(\lambda) \le \lambda^a$ for all sufficiently large λ. Set

$$\kappa(\lambda) := \left\lceil \frac{a + r + 1}{\gamma} \log \lambda \right\rceil, \qquad t := 2^\kappa \le 2\lambda^{(a+r+1)/\gamma}.$$

At this difficulty the planted witness has length κ, and $2^{\gamma\kappa} \ge \lambda^{a+r+1} > \mathsf{p}(\lambda)$. Any solver returning $\mathcal{F}_{\mathsf{puz}}(t, z)$ therefore returns a witness for z, so Assumption 7.1 bounds its success probability by

$$2^{-\gamma\kappa} \le \lambda^{-(r+1)} < \lambda^{-r}.$$

This bound holds even without the non-faulty requirement. □

Corollary 7.3. *Assumption 3.6 together with Assumption 7.1 implies that there is a hard distribution of* EOML*.*

Proof. By Theorem 6.15, Assumption 3.6 gives a uuSNARG with state uniqueness and height bound $H = \omega(\log \lambda)$. Apply Corollary 4.6 with the puzzle from Lemma 7.2. □

7.2 Clique polynomials

Our second puzzle evaluates the weighted clique polynomial. Let $v = \lambda$ and $\mathbb{F} = \mathbb{F}_{2^\lambda}$, represented in a fixed polynomial-time constructible polynomial basis. For edge weights $\mathbf{w} \in \mathbb{F}^{\binom{v}{2}}$, the weighted k-clique polynomial sums, over all sets of k vertices, the product of the weights of the edges among them:

$$P_k(\mathbf{w}) := \sum_{1 \le i_1 < \cdots < i_k \le v} \prod_{\alpha < \beta} w_{i_\alpha i_\beta}.$$

On Boolean edge weights, a set of k vertices contributes 1 exactly when it is a k-clique, so P_k counts the k-cliques. Since $\mathbb{F}$ has characteristic two, this count is taken modulo 2: P_k is the parity of the number of k-cliques [GR18, BBB19]. We assume that increasing k eventually makes evaluation hard for any fixed polynomial budget, even for solvers that may output $\perp$ instead of an answer, as long as they rarely output a wrong value.

Assumption 7.4 (Average-case clique-polynomial hardness)**.** *For every polynomial s and every $r \in \mathbb{N}$, there is a constant $k \ge 3$ such that the following holds for uniform $\mathbf{w} \leftarrow \mathbb{F}^{\binom{\lambda}{2}}$. If an adversary $\mathcal{A}$ of size at most $s(\lambda)$ satisfies*

$$\Pr[\mathcal{A}(\mathbf{w}) \notin \{\perp, P_k(\mathbf{w})\}] \le \mathsf{negl}(\lambda),$$

then, for all sufficiently large λ,

$$\Pr[\mathcal{A}(\mathbf{w}) = P_k(\mathbf{w})] < \lambda^{-r}.$$

In Section B, we justify this assumption from a non-uniform version of the worst-case clique-parity conjecture that requires hardness at every sufficiently large input length of Boix-Adserà, Brennan and Bresler [BBB19].

The difficulty t determines the clique size k. Let $b := \lceil \log(v+1) \rceil$ be the number of bits in a vertex label, and let $k := \lfloor \log t / b \rfloor$ be the largest clique size for which every list of k labels fits in $\log t$ bits, so that enumerating these lists takes at most t steps. The puzzle is then defined as follows:

- $\mathcal{D}_{\mathsf{clq}}(t, 1^\lambda)$ samples uniform edge weights $\mathbf{w} \leftarrow \mathbb{F}^{\binom{v}{2}}$ and outputs them.
- $\mathcal{F}_{\mathsf{clq}}(t, \mathbf{w}) := P_k(\mathbf{w})$.[8]

For $t = 2^p$, $\mathcal{F}_{\mathsf{clq}}(t, \mathbf{w})$ is the output of the following regular tree circuit with address length p, in which $\mathbf{w}$ is hardwired and field operations are implemented by Boolean circuits. A leaf reads the first kb bits of its address as k vertex labels $i_1, \ldots, i_k$ of b bits each, and outputs

$$\prod_{\alpha < \beta} w_{i_\alpha i_\beta}$$

if $1 \le i_1 < \cdots < i_k \le v$ and the remaining $p - kb$ address bits are zero, and 0 otherwise. Each combining circuit adds its two inputs in $\mathbb{F}$. Every set of k vertices is listed in increasing order by exactly one leaf, and all other leaves output 0, so the root outputs $P_k(\mathbf{w})$.

Lemma 7.5. *Under Assumption 7.4, $(\mathcal{F}_{\mathsf{clq}}, \mathcal{D}_{\mathsf{clq}})$ is an average-case tree puzzle.*

Proof. The sampler runs in time $\mathrm{poly}(\lambda)$, and the local circuits have size $\mathrm{poly}(k, \lambda)$. Since $k \le \log t$, their descriptions can be computed in time $\mathrm{poly}(\log t, |\mathbf{w}|)$, and evaluating the tree takes time $t \cdot \mathrm{poly}(\log t, |\mathbf{w}|)$.

[8] The definition of P_k applies to every $k \ge 0$, with an empty sum equal to 0 and an empty product equal to 1. Thus $P_k(\mathbf{w}) = 0$ for $k > v$, and for $k \le 1$ every set of k vertices contributes 1.

For hardness, fix a polynomial p and $r \in \mathbb{N}$, and take k from Assumption 7.4 for this budget and exponent. Set $\kappa := kb$. Then

$$t = 2^{kb} \leq (2\lambda + 2)^k$$

is polynomially bounded. At this difficulty, the puzzle evaluates exactly P_k on the uniform weights of Assumption 7.4, so that assumption gives the required hardness bound. □

Corollary 7.6. *Assumption 3.6 together with Assumption 7.4 implies that there is a hard distribution of* EOML.

Proof. Combine Theorem 6.15, Lemma 7.5 and Corollary 4.6, using $H = \log^2 \lambda = \omega(\log \lambda)$. □

7.3 Hardness from subexponential SIS alone

For this subsection, take $H = \lambda$ and let $\mathcal{U}$ be the uuSNARG of Theorem 6.15 with the parameters of Lemma 6.13.

We now use SIS for both parts of Theorem 4.7. At a large dimension, subexponential hardness gives state uniqueness against adversaries that replay the entire line. At a smaller dimension, it makes finding an SIS solution hard for polynomial-size adversaries. Exhaustive search for this smaller solution is the computation proved by the uuSNARG.

Assumption 7.7 (Subexponentially hard SIS_e)**.** *Let e and ν be constants with $0 < e < 5/9$ and $4e/5 < \nu < 1-e$. For all parameters n, m, q, β as in Assumption 3.5 that additionally satisfy $q \leq 2^{2\lceil n^e \rceil}$, $\mathsf{SIS}_{n,m,q,\beta}$ is 2^{n^ν}-hard.*

We use the parameter range $4e/5 < \nu < 1 - e$; the lower bound is required by Corollary 6.16. For a discussion of SIS parameter choices and lattice attacks, see Micciancio and Regev [MR09]. The condition $e < 5/9$ ensures that such a ν exists. For instance, $e = \nu = 1/3$ satisfies all of these conditions. The lower bound on ν comes from the dimension of the uuSNARG, and the upper bound on the modulus must accommodate both dimensions. For $H = \lambda$, Lemma 6.13 sets $\beta = 2^{n^e}$ above the threshold $B_{\mathsf{thr}} = 2^{\lceil K\Lambda^{5/4} \rceil}$, with $\Lambda \geq \lambda$, so the dimension of the uuSNARG is $n \approx (2K\Lambda^{5/4})^{1/e} \geq \lambda^{5/(4e)}$. Theorem 4.7 needs state uniqueness against adversaries of size about 2^λ, which replay the entire line, and Assumption 7.7 gives security against size 2^{n^ν} with $n^\nu \geq \lambda^{5\nu/(4e)}$. This exceeds λ by a polynomial factor when $\nu > 4e/5$, where the $5/4$ is the exponent in B_{thr}. Rearranging $4e/5 < 1 - e$ gives $e < 5/9$. The upper bound $q \leq 2^{2\lceil n^e \rceil}$ on the modulus covers the modulus of the uuSNARG, which exceeds β only by a polynomial factor, and the modulus of the puzzle below, which we set equal to this upper bound.

The puzzle is an SIS instance $\hat{\mathbf{A}} \in \mathbb{Z}_{\hat{q}}^{\hat{n} \times \hat{m}}$ that is just small enough to be solved by exhaustive search over at most 2^λ candidate addresses. We first describe the choice of parameters for the example $e = \nu = 1/3$.

- *A solution exists.* With $\hat{m} > \hat{n} \log \hat{q}$ columns, there are more vectors in $\{0,1\}^{\hat{m}}$ than possible images, so two of them collide, and their difference is a nonzero kernel vector with entries in $\{-1, 0, 1\}$.
- *Exhaustive search tries at most 2^λ candidate addresses.* Each entry of such a vector, one of $-1, 0, 1$, takes two bits, so the search ranges over $2^{2\hat{m}}$ addresses, and we need $2\hat{m} \leq \lambda$.
- *The modulus is one that Assumption 7.7 allows.* As in Assumption 3.5, at dimension $\hat{n}$ the assumption concerns solutions of norm at most $\beta = 2^{\hat{n}^{1/3}}$ and requires a modulus of at least $\hat{n}\sqrt{\hat{m}} \cdot \beta$. Assumption 7.7 additionally bounds the modulus from above by roughly β^2. We take $\hat{q} \approx \beta^2 = 2^{2\hat{n}^{1/3}}$, which is in this range. Our solutions have norm $1 \leq \beta$, so finding one breaks the assumption.
- *The dimension is polynomial in λ.* The first item now needs $\hat{m} \approx \hat{n} \log \hat{q} = 2\hat{n}^{4/3}$ columns, and the second needs $2\hat{m} \leq \lambda$. The largest dimension satisfying both is $\hat{n} \approx (\lambda/4)^{3/4}$.
- *The puzzle is hard for polynomial-size solvers.* The assumption rules out solvers of size $2^{\hat{n}^{1/3}} \approx 2^{(\lambda/4)^{1/4}}$ at dimension $\hat{n}$. This is superpolynomial in λ, so no polynomial-size solver finds a solution.

The dimension $\hat{n}$ is much smaller than the dimension n of the uuSNARG of Theorem 6.15, whose security must hold against adversaries running for 2^λ steps, so that $n^{1/3}$ must exceed λ; for $e = 1/3$, Lemma 6.13 gives $n = \Theta(\lambda^{15/4})$, compared to $\hat{n} \approx (\lambda/4)^{3/4}$. For general e and ν, the norm exponent $1/3$ becomes e, the exponent $4/3$ becomes $1 + e$, and the exponent $1/3$ in the adversary size becomes ν, and $\hat{n} \approx (\lambda/4)^{1/(1+e)}$. Formally, let $\hat{n}$ be the largest positive integer such that $2\hat{m} \le \lambda$, where

$$\hat{q} := 2^{2\lceil \hat{n}^e \rceil}, \qquad \hat{m} := \hat{n} \log \hat{q} + 1,$$

and set $\hat{n} := 1$ for the finitely many λ for which no such integer exists. An address $a \in \{0,1\}^p$ *encodes* a vector $\mathbf{u} \in \{-1,0,1\}^{\hat{m}}$ if its first $\min(p, 2\hat{m})$ bits, padded with zeros to length $2\hat{m}$, encode the entries of $\mathbf{u}$ by $01, 00, 10$ for $-1, 0, 1$, and its remaining bits are zero. Define the puzzle by setting $p := \lfloor \log t \rfloor$:

- $\mathcal{D}_{\mathsf{sis}}(t, 1^\lambda)$ samples a uniform matrix $\hat{\mathbf{A}} \leftarrow \mathbb{Z}_{\hat{q}}^{\hat{n} \times \hat{m}}$ and outputs it.
- $\mathcal{F}_{\mathsf{sis}}(t, \hat{\mathbf{A}})$ is the output of C_φ (Lemma 3.9) for the predicate $\varphi : \{0,1\}^p \to \{0,1\}$ that accepts a if it encodes a vector $\mathbf{u} \neq \mathbf{0}$ with $\hat{\mathbf{A}}\mathbf{u} = \mathbf{0} \bmod \hat{q}$. It is $(1, a)$ for the first such a, or $(0, 0^p)$ if no such address exists.

Lemma 7.8. *$(\mathcal{F}_{\mathsf{sis}}, \mathcal{D}_{\mathsf{sis}})$ satisfies the efficiency and tree structure properties of Definition 4.3. Under Assumption 7.7, every polynomial-size adversary outputs $\mathcal{F}_{\mathsf{sis}}(2^\lambda, \hat{\mathbf{A}})$, for $\hat{\mathbf{A}} \leftarrow \mathcal{D}_{\mathsf{sis}}(2^\lambda, 1^\lambda)$, with negligible probability.*

Proof. The sampler and witness predicate run in polynomial time, so efficiency and tree structure follow from Lemma 3.9. We prove hardness for sufficiently large λ, when $p = \lambda \ge 2\hat{m}$. At this difficulty, the search includes every vector in $\{-1,0,1\}^{\hat{m}}$.

A nonzero solution exists for every $\hat{\mathbf{A}}$: since $2^{\hat{m}} > \hat{q}^{\hat{n}}$, two distinct binary vectors have the same image under $\hat{\mathbf{A}}$, and their difference has entries in $\{-1,0,1\}$. Thus a correct puzzle output yields an SIS solution of norm 1.

It remains to check that the assumption applies at this dimension. We have $\hat{m} = \Theta(\hat{n}^{1+e})$, so the maximal choice of $\hat{n}$ gives $\hat{n} = \Theta(\lambda^{1/(1+e)})$. Moreover,

$$\frac{\hat{q}}{2^{\hat{n}^e}} \ge 2^{\hat{n}^e} \ge \hat{n}\sqrt{\hat{m}}$$

for all sufficiently large λ. These are therefore parameters covered by Assumption 7.7. Since $\nu > 0$, its adversarial budget $2^{\hat{n}^\nu}$ exceeds every polynomial in λ, including the cost of decoding a puzzle output. The assumption bounds the probability of finding the resulting norm-one solution by $\mathsf{negl}(\lambda)$. □

Corollary 7.9. *Assumption 7.7 alone implies that there is a hard distribution of* EOML.

Proof. We check the two conditions of Theorem 4.7. At the dimension used by Theorem 6.15, the modulus satisfies $q \le 2^{2\lceil n^e \rceil}$ for all sufficiently large λ, so Assumption 7.7 applies. Since $\nu > 4e/5$, Corollary 6.16 gives state uniqueness against adversaries of the size required by Theorem 4.7. At the smaller dimension $\hat{n}$, Lemma 7.8 gives hardness of the puzzle at difficulty 2^λ. Applying Corollary 4.8 proves the result. □

8 AI Usage Disclosure

Claude Opus 5.5 and ChatGPT 6.1 Astra were used in developing the solution. In late 2025, before using AI for this problem, the author had attempted to use the functional commitment scheme of Wee [Wee25] to obtain PPAD hardness via a mergeable functional commitment, as outlined in the overview, although the abstraction considered at the time was different. However, the author was unable to modify the scheme to instantiate this abstraction. Recently, while using ChatGPT to explore the minimal assumptions required to establish PPAD hardness, the author proposed this route to the model and asked whether modifications to the aforementioned scheme could yield the desired abstraction. The core idea of keeping the proof as summary tables emerged from that interaction. The solution was simplified over several iterations and independently checked by the author. AI also made the connection to the fine-grained complexity of cliques, which gives a second instantiation of the hard computation.

The author then wrote up the solution, using AI for editorial assistance and for identifying gaps and issues in the writing and proofs. The author takes full responsibility for the claims in this work, and any errors are the author's own.

References

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996. 3, 5, 22, 24

[AKV04] Tim Abbott, Daniel Kane, and Paul Valiant. On algorithms for Nash equilibria. Unpublished manuscript. http://web.mit.edu/tabbott/Public/final.pdf, 2004. 3, 4

[BBB19] Enric Boix-Adserà, Matthew Brennan, and Guy Bresler. The average-case complexity of counting cliques in Erdős-Rényi hypergraphs. In David Zuckerman, editor, *60th FOCS*, pages 1256–1280. IEEE Computer Society Press, November 2019. 4, 14, 60, 70

[BCFL23] David Balbás, Dario Catalano, Dario Fiore, and Russell W. F. Lai. Chainable functional commitments for unbounded-depth circuits. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part III*, volume 14371 of *LNCS*, pages 363–393. Springer, Cham, November / December 2023. 8, 34, 39

[BCH+22] Nir Bitansky, Arka Rai Choudhuri, Justin Holmgren, Chethan Kamath, Alex Lombardi, Omer Paneth, and Ron D. Rothblum. PPAD is as hard as LWE and iterated squaring. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 593–622. Springer, Cham, November 2022. 3, 4, 5, 8, 10, 12, 13, 24, 26, 27, 33, 72

[BDY26] Shany Ben-David and Eylon Yogev. The power of permutable PRPs: Towards trapdoor permutations with full key and message domains. Cryptology ePrint Archive, Paper 2026/2333, 2026. 5

[BG20] Nir Bitansky and Idan Gerichter. On the cryptographic hardness of local search. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 6:1–6:29. LIPIcs, January 2020. 4, 14, 68

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In Venkatesan Guruswami, editor, *56th FOCS*, pages 1480–1498. IEEE Computer Society Press, October 2015. 3, 4

[BPSS23] Nir Bitansky, Omer Paneth, Dana Shamir, and Tomer Solomon. Non-interactive universal arguments. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 132–158. Springer, Cham, August 2023. 3, 13, 14, 22, 28, 29

[CD06] Xi Chen and Xiaotie Deng. Settling the complexity of two-player Nash equilibrium. In *47th FOCS*, pages 261–272. IEEE Computer Society Press, October 2006. 3

[CFJW25] Liyan Chen, Cody Freitag, Zhengzhong Jin, and Daniel Wichs. Unambiguous SNARGs for P from LWE with applications to PPAD hardness. In Michal Koucký and Nikhil Bansal, editors, *57th ACM STOC*, pages 944–954. ACM Press, June 2025. 3, 4, 5, 6, 8, 13, 14, 26, 27, 28, 29, 72

[CGJ+23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 635–668. Springer, Cham, August 2023. 8

[CHK+19a] Arka Rai Choudhuri, Pavel Hubácek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a nash equilibrium is no easier than breaking Fiat-Shamir. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1103–1114. ACM Press, June 2019. 3, 4, 5, 7, 8, 9, 12, 13, 22, 23, 24, 26, 27, 33

[CHK+19b] Arka Rai Choudhuri, Pavel Hubacek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. PPAD-hardness via iterated squaring modulo a composite. Cryptology ePrint Archive, Report 2019/667, 2019. 3, 4, 5, 8, 9, 12, 13, 26, 33

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd FOCS*, pages 68–79. IEEE Computer Society Press, February 2022. 8

[dCP23] Leo de Castro and Chris Peikert. Functional commitments for all functions, with transparent setup and from SIS. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 287–320. Springer, Cham, April 2023. 6

[DGP06] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. In Jon M. Kleinberg, editor, *38th ACM STOC*, pages 71–78. ACM Press, May 2006. 3

[DP11] Constantinos Daskalakis and Christos H. Papadimitriou. Continuous local search. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 790–804, 2011. 23, 69

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 125–154. Springer, Cham, May 2020. 3, 4, 5, 8, 9, 10, 12, 13, 26, 33

[FGMS19] John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *ICALP 2019*, volume 132 of *LIPIcs*, pages 56:1–56:15. Schloss Dagstuhl, July 2019. 4, 72

[FW26] Cody Freitag and Daniel Wichs. Unique SNARGs with adaptive security: Constructions and black-box separations. In *CRYPTO (1)*, volume 16800 of *Lecture Notes in Computer Science*, pages 190–217. Springer, 2026. 5

[GGH96] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Collision-free hashing from lattice problems. Cryptology ePrint Archive, Report 1996/009, 1996. 5

[GKSZ20] Mika Göös, Pritish Kamath, Katerina Sotiraki, and Manolis Zampetakis. On the complexity of modulo-q arguments and the Chevalley–Warning theorem. In *35th Computational Complexity Conference (CCC 2020)*, volume 169 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:42. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. 5

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 579–604. Springer, Berlin, Heidelberg, August 2016. 3, 4

[GR18] Oded Goldreich and Guy N. Rothblum. Counting t-cliques: Worst-case to average-case reductions and direct interactive proof systems. In Mikkel Thorup, editor, *59th FOCS*, pages 77–88. IEEE Computer Society Press, October 2018. 60

[HKKS20] Pavel Hubácek, Chethan Kamath, Karel Král, and Veronika Slívová. On average-case hardness in TFNP from one-way functions. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 614–638. Springer, Cham, November 2020. 5

[HY17] Pavel Hubácek and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. In Philip N. Klein, editor, *28th SODA*, pages 1352–1371. ACM-SIAM, January 2017. 3, 4, 5, 22, 23, 69

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. 22, 68

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. 68

[Jeř16] Emil Jeřábek. Integer factoring and modular square roots. *Journal of Computer and System Sciences*, 82(2):380–394, 2016. 5

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 708–721. ACM Press, June 2021. 3, 4, 5, 8, 26

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021. 3

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Cham, May / June 2022. 3

[KLV23] Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. SNARGs and PPAD hardness from the decisional Diffie-Hellman assumption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 470–498. Springer, Cham, April 2023. 3, 4, 5, 26

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 1545–1552. ACM Press, June 2023. 8

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1115–1124. ACM Press, June 2019. 8

[KPY20] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. Delegation with updatable unambiguous proofs and PPAD-hardness. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 652–673. Springer, Cham, August 2020. 3, 4, 5, 8, 26

[KS17] Ilan Komargodski and Gil Segev. From minicrypt to obfustopia via private-key functional encryption. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 122–151. Springer, Cham, April / May 2017. 3, 5

[LV20] Alex Lombardi and Vinod Vaikuntanathan. Fiat-Shamir for repeated squaring with applications to PPAD-hardness and VDFs. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 632–651. Springer, Cham, August 2020. 3, 4, 5, 8, 26

[MP91] Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991. 3

[MR09] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer, 2009. 61

[Nas51] John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951. 3

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994. 3

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 333–342. ACM Press, May / June 2009. 3

[Pie19] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019. 5, 8

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. 3

[RSS17] Alon Rosen, Gil Segev, and Ido Shahaf. Can PPAD hardness be based on standard cryptographic assumptions? In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 747–776. Springer, Cham, November 2017. 3, 5

[SZZ18] Katerina Sotiraki, Manolis Zampetakis, and Giorgos Zirdelis. PPP-completeness with connections to cryptography. In Mikkel Thorup, editor, *59th FOCS*, pages 148–158. IEEE Computer Society Press, October 2018. 5

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, March 2008. 8, 9

[Wee25] Hoeteck Wee. Functional commitments and SNARGs for P/poly from SIS. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 617–640. Springer, Cham, August 2025. 4, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 27, 33, 34, 37, 39, 44, 45, 47, 63

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 433–463. Springer, Cham, August 2022. 8

A Worst-Case Hardness from ETH

The corollaries of Section 7 give hard distributions of EOML, and in particular imply EOML $\notin$ FP/poly, but each of them relies on an average-case assumption beyond polynomially hard SIS, or on subexponentially hard SIS. In this section, we show that for worst-case hardness, polynomially hard SIS and the exponential time hypothesis suffice. The latter is a worst-case assumption, and accordingly the result gives no hard distribution.

We follow Bitansky and Gerichter [BG20], who obtain PLS $\not\subseteq$ FP/poly from a bilinear-map assumption and the exponential time hypothesis, and replace the average-case puzzle by exhaustive search for a satisfying assignment. Compared with their result, our conclusion extends from PLS to CLS.

Assumption A.1 (Non-uniform ETH [IP01, IPZ01])**.** *There is a constant $\delta > 0$ such that no family of circuits of size $2^{\delta n}$ decides satisfiability of all* 3CNF *formulas with n variables for all sufficiently large n.*

We use the non-uniform variant of the hypothesis, as in [BG20], in its infinitely-often form: every such family fails for infinitely many n.

Overview. Given a formula x with n variables, we build a line of length 2^n that runs exhaustive search over all assignments, with a uuSNARG state certifying each step. The end of the line reveals a satisfying assignment if one exists, so an EOML solver that reaches it decides whether x is satisfiable. The main question is how to choose the security parameter. State uniqueness follows from polynomially hard SIS only if the line can be replayed in time polynomial in λ (Lemma 4.2), so we take $\lambda = 2^{n/\alpha}$ for a constant α, which makes the length $2^n = \lambda^\alpha$ polynomial. Thus the formula has only $n = \alpha \log \lambda$ variables, and all efficiency is measured in λ, which determines the size of the EOML instance, and not in n. In particular, the reduction from satisfiability takes time $2^{O(n/\alpha)}$, which is exponential in n, but at a rate that becomes small as α grows. This is why we rely on Assumption A.1, which only rules out algorithms below a fixed rate δ. Since the uuSNARG is succinct, the size of the instance is a polynomial in λ that does not depend on α. A polynomial-time solver then runs in time $\lambda^{O(1)} = 2^{O(n/\alpha)}$, where the hidden constant depends on the solver but not on α. Choosing α large enough, after fixing the solver, brings this below $2^{\delta n}$ and contradicts Assumption A.1. Finally, the formula is a worst-case input and not a sample from a distribution. We therefore bound the probability of an accepted non-canonical state for the worst formula of each size, which a non-uniform adversary can have hardwired, and we remove the randomness of the solver by repetition and a union bound over all formulas.

Encoding formulas. We encode a 3CNF formula with n variables canonically, by the characteristic vector of its set of clauses among the at most $(2n)^3$ clauses with three literals, repeating a literal in a clause with fewer than three and dropping clauses that contain a variable and its negation. Every formula can be brought into this form in polynomial time without changing its satisfiability, so Assumption A.1 applies to canonically encoded formulas. In particular, there are at most $2^{(2n)^3}$ formulas with n variables.

The instances. Use the uuSNARG of Theorem 6.15 with $H = \log^2 \lambda$ and the parameters of Lemma 6.12. Fix a constant $\alpha \geq 1$. For a 3CNF formula x with n variables, set

$$\lambda := 2^{\lceil n/\alpha \rceil}, \qquad L := 2^n \leq \lambda^\alpha.$$

Let $C_x := C_\varphi$ be the circuit of Lemma 3.9 for $\varphi := x$ on $\{0,1\}^n$. Its output is $(1, w)$ for the first satisfying assignment w of x, if one exists. Since $|C_x| \leq \mathrm{poly}(n)$ and $\lambda \geq 2^{n/\alpha}$, there is an $n_0(\alpha)$ such that, for all $n \geq n_0(\alpha)$, we have $n \leq \alpha \log \lambda \leq H$ and C_x has size at most $N(\lambda)$ for a fixed polynomial N independent

of α and x. We consider only such n. The height bound and setup are fixed independently of α; only this lower threshold on n may depend on α.

The RSVL instance I_x is the instance of Fig. 7 with the single circuit C_x in place of the puzzles: it samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{N(\lambda)})$, sets $\mathsf{pp} := \mathsf{Preprocess}(\mathsf{crs}, C_x)$, and has length L. A vertex encodes (j, st) with $j \in [0, L]$, the successor increments j and applies Update while $j < L$, and $\mathsf{V}((j, \mathsf{st}), i)$ accepts if $j = i$ and $\mathsf{Verify}(\mathsf{crs}, \mathsf{pp}, i, \mathsf{st}) = 1$. By completeness, the promise of RSVL holds. The instance I_x has size at most $\lambda^{c_0'}$, and we pad the EOML instance $\mathsf{R}_1(I_x)$ of Theorem 3.3 to a common size $m(\lambda) \le \lambda^{c_0}$, where c_0, c_0' are constants independent of α and x.

Theorem A.2 (Worst-case hardness)**.** *Assume Assumptions A.1 and 3.6. Then no family of polynomial-size circuits solves* EOML *on all instances. Consequently,* CLS*,* PPAD *and* PLS *are not contained in* FP/poly*, since* EOML $\in$ CLS $\subseteq$ PPAD $\cap$ PLS *[HY17, DP11].*

Proof. Suppose that, for a constant η and all sufficiently large m, a circuit of size m^η solves EOML on all instances of size m. Let $\mathcal{A}$ denote this circuit for the size $m(\lambda)$ of our padded instances. We use it to decide satisfiability, and choose α at the end of the proof. Let $(\mathsf{R}_1, \mathsf{R}_2)$ be the reduction of Theorem 3.3. We write $\mathsf{R}_1(I)$ for the EOML instance it produces deterministically from an RSVL instance I, and $\mathsf{R}_2(I, z; \rho)$ for the output of the decoder on a solution z to that instance, using fresh coins ρ. We write a decoded sink v as (v, L).

The SAT solver. The randomized circuit $\mathcal{B}$ runs the following experiment T times independently, and outputs "unsatisfiable" if no run outputs "satisfiable".

One run of $\mathcal{B}(x)$:

1. Sample I_x and independent decoder coins ρ.
2. Compute $z := \mathcal{A}(\mathsf{R}_1(I_x))$ and $(v, i) := \mathsf{R}_2(I_x, z; \rho)$.
3. If $i = L$, $v = (L, \mathsf{st})$, and $\mathsf{Output}(\mathsf{crs}, \mathsf{pp}, \mathsf{st}) = (1, w)$ for an assignment w satisfying x, output "satisfiable".

Since $\mathcal{B}$ checks every assignment before accepting, it errs only on satisfiable formulas.

Success of one run. Fix a satisfiable x. Since $\mathcal{A}$ solves every instance, Theorem 3.3 gives an RSVL solution with probability at least λ^{-c_1}, for a constant c_1 independent of α and x. If the solution is the canonical sink, then by completeness its output is $(1, w)$ for the first satisfying assignment w, and the run succeeds. Otherwise, it is an accepted vertex (i, st) with $i \le L$ and $\mathsf{st} \ne \mathsf{st}_i$. This includes sinks other than the canonical one.

We bound the probability of the second case uniformly over x. For each λ, hardwire a formula x that maximizes this probability among all formulas whose number of variables n satisfies $2^{\lceil n/\alpha \rceil} = \lambda$, and consider the algorithm that, given crs, runs one experiment of $\mathcal{B}(x)$ with this crs. If the decoded pair (v, i) is an accepted vertex, that is, v parses as (i, st) with $i \le L$ and $\mathsf{V}(v, i) = 1$, it outputs (C_x, i, st). Otherwise, it outputs $\perp$. It has size $\lambda^{c_0\eta} \cdot \mathrm{poly}(\lambda)$ and outputs either $\perp$ or a time at most $\tau := L \le \lambda^\alpha$, which is polynomial for fixed α. By Lemma 4.2 and the state uniqueness of the uuSNARG of Theorem 6.15, it outputs an accepted state other than the canonical one with negligible probability. Hence, for all sufficiently large λ, each run succeeds with probability at least $\lambda^{-c_1} - \mathrm{negl}(\lambda) \ge \lambda^{-c_1}/2$.

Fixing the coins. Take $T := \lceil 2\lambda^{c_1}(\lambda + 1) \rceil$. Then $\mathcal{B}$ errs on each formula with probability at most $e^{-(\lambda+1)}$. There are at most $2^{(2n)^3} \le 2^\lambda$ canonically encoded formulas with n variables for all sufficiently large n, since $\lambda \ge 2^{n/\alpha}$, so by a union bound, some fixing of the coins is correct on all of them. The resulting circuit never replays a computation, so it has size at most $\lambda^{c_0\eta + c_2} = 2^{(c_0\eta + c_2)\lceil n/\alpha \rceil}$, for a constant c_2 independent of α. For $\alpha > (c_0\eta + c_2)/\delta$, this is less than $2^{\delta n}$ for all sufficiently large n, contradicting Assumption A.1. □

B From Worst-Case to Average-Case Clique Hardness

We justify Assumption 7.4 from a worst-case clique-parity hypothesis. Boix-Adserà, Brennan and Bresler conjecture that, for constant k, any randomized algorithm computing the parity of the number of k-cliques with error less than $1/3$ takes time $v^{\Omega(k)}$ on v-vertex graphs in the worst case [BBB19]. They also note that their conjectures follow from randomized ETH. We use the graph case as a hierarchy: for every polynomial budget, some constant k makes clique parity hard. Our premise additionally requires non-uniform hardness at all sufficiently large lengths. These are additions to the cited algorithmic conjecture.

Intuition for the hypothesis. Counting the k-cliques of a v-vertex graph by brute force takes about v^k steps, one for each set of k vertices. The conjecture says that, even for the parity of the count and with randomness, no algorithm improves this exponent by more than a constant factor. For each fixed k, the problem is therefore solvable in polynomial time, but the exponent of the polynomial grows with k. This is the shape needed by the puzzles of increasing difficulty in Section 2.5: counting the cliques by enumerating the sets of k vertices is a polynomial-time computation with the tree structure of Section 7.2, and for every polynomial budget some constant k makes the problem too hard for that budget. The two additions match how the assumption is used. Non-uniform hardness means that we even allow a different circuit for each graph size, without requiring a single efficient algorithm that constructs these circuits. We need this because an adversary in Assumption 7.4 is a circuit, so the reduction below built from it is a circuit family. Hardness must also hold at every sufficiently large security parameter, not only at infinitely many, because an evaluator that breaks Assumption 7.4 need only succeed at infinitely many security parameters, and the reduction must reach a contradiction at those.

Notation. Let $v = \lambda$ be the number of vertices, and let $\mathbb{F} = \mathbb{F}_{2^\lambda}$, a field of characteristic two. A vector $\mathbf{w} \in \mathbb{F}^{\binom{v}{2}}$ assigns a field element to each possible edge. Recall the weighted clique polynomial

$$P_k(\mathbf{w}) = \sum_{1 \le i_1 < \cdots < i_k \le v} \prod_{\alpha < \beta} w_{i_\alpha i_\beta}.$$

For a Boolean adjacency vector, each summand indicates whether those k vertices form a clique, and since addition in characteristic two is taken modulo 2, P_k is the parity of the number of k-cliques. Each summand contains $D = \binom{k}{2}$ edge weights, so P_k has degree at most D. An *evaluator* is an adversary in Assumption 7.4: on input $\mathbf{w}$, it outputs $P_k(\mathbf{w})$ or $\perp$, meaning no answer, and it outputs a wrong value only with negligible probability.

Intuition for the reduction. The premise concerns worst-case Boolean graphs, whereas an evaluator that breaks Assumption 7.4 only works on uniformly random edge weights in $\mathbb{F}$, and may succeed only with an inverse-polynomial probability, at least λ^{-r} for a fixed constant r. The bridge is that P_k is a low-degree polynomial that agrees with clique parity on Boolean inputs. To evaluate it at a worst-case input x, choose independent uniformly random vectors $R, S \in \mathbb{F}^{\binom{v}{2}}$, and consider the curve $t \mapsto x + tR + t^2 S$. Here the field element t indexes the queries. For each fixed $t \neq 0$, the point on the curve has the uniform distribution regardless of x, since tR is uniform and independent of x and S, so each query is an average-case input. Restricted to the curve, P_k becomes a univariate polynomial $Q(t)$ of degree at most $2D$ whose value at $t = 0$ is the answer $P_k(x)$. A univariate polynomial of degree at most $2D$ is uniquely determined by its values at any $2D + 1$ points, so from $2D + 1$ correct values of Q we can interpolate Q and read off $Q(0)$.

Each query individually looks uniformly random, but this alone does not ensure that sufficiently many queries receive answers: a large expected number of answers could, for instance, come from rare runs in which all of them arrive together. The quadratic curve makes every pair of query points independent, which

lets Chebyshev's inequality bound the probability of receiving too few correct answers. A line $t \mapsto x + tR$ would not give pairwise independence: for fixed x and known $t_i \neq t_j$, the point $x + t_i R$ determines R, and hence also $x + t_j R$.

Finally, because we make only polynomially many queries and each has negligible wrong-answer probability, except with negligible probability every answer other than $\perp$ is correct. We can therefore interpolate from these answers without needing to identify or correct errors. This is where the restriction of Assumption 7.4 to adversaries that rarely give wrong answers is used.

Proposition B.1. *Suppose that for every constant $d > 0$ there is a constant $k \geq 3$ such that, for all sufficiently large v, no randomized Boolean circuit of size at most v^d computes the parity of the number of k-cliques with error at most $1/3$ on every v-vertex graph. Then Assumption 7.4 holds.*

Proof. Fix the size bound s and success exponent r in Assumption 7.4. We first give the reduction for any fixed k, and then choose k using the premise. Put $D := \binom{k}{2}$, the degree of P_k, and $v = \lambda$. Suppose an evaluator of size at most $s(\lambda)$ has correct-answer probability $\rho \geq \lambda^{-r}$ and wrong non-$\perp$ probability η on uniform edge weights over $\mathbb{F} = \mathbb{F}_{2^\lambda}$.

The reduction. Given the Boolean adjacency vector x of a v-vertex graph, sample independent uniform vectors $R, S \in \mathbb{F}^{\binom{v}{2}}$. Choose distinct nonzero $t_1, \ldots, t_M \in \mathbb{F}$, where

$$M := \lceil 48(2D+1)\lambda^r \rceil,$$

and query the evaluator at $x + t_j R + t_j^2 S$ for each j, using fresh evaluator coins. Since each query is answered correctly with probability $\rho \geq \lambda^{-r}$, this many queries give at least $48(2D+1)$ correct answers in expectation, many more than the $2D+1$ needed for interpolation. There are enough distinct nonzero field elements for all sufficiently large λ. If fewer than $2D+1$ answers are non-$\perp$, return 0. Otherwise interpolate a polynomial Q of degree at most $2D$ from the first $2D+1$ such answers, and return 1 if and only if $Q(0) = 1$.

Correctness. The query points are pairwise independent and uniform: for $i \neq j$, the linear map from (R, S) to the two query points has determinant $t_i t_j (t_j - t_i) \neq 0$. With independent evaluator coins, the indicators of correct answers are therefore also pairwise independent. If X is the number of correct answers and $\mu := M\rho$, then

$$\mu \geq 48(2D+1), \qquad \mathrm{Var}(X) \leq \mu.$$

Chebyshev's inequality gives $\Pr[X < 2D+1] \leq 1/12$: since $2D+1 \leq \mu/2$, the event $X < 2D+1$ implies $|X - \mu| > \mu/2$, so its probability is at most $\mathrm{Var}(X)/(\mu/2)^2 \leq 4/\mu \leq 1/12$. A union bound gives probability at most $M\eta$ that any query receives a wrong non-$\perp$ answer.

When neither event occurs, the answers used for interpolation all lie on

$$Q(t) = P_k(x + tR + t^2 S),$$

which has degree at most $2D$. Interpolation therefore recovers Q exactly, and $Q(0) = P_k(x)$ is the clique-parity bit of the input graph. For every x, the reduction's error probability is at most $1/12 + M\eta$.

Choosing k. The resulting circuit has size $O_k(\lambda^{c_{s,r}})$ for an exponent $c_{s,r}$ independent of k. Indeed, field arithmetic has polynomial bit cost, the number of queries is $O_k(\lambda^r)$, and interpolation uses $2D+1$ points. Dependence on k enters only through constants. In other words, the exponent of the reduction is determined by the evaluator's size s and success exponent r, and does not grow with k, whereas the premise lets us choose k so that the worst-case problem needs a larger exponent d.

Choose $d > c_{s,r}$, and then take k from the proposition's premise. If Assumption 7.4 failed for this k, some evaluator family would have $\rho \geq \lambda^{-r}$ at infinitely many lengths. Its wrong-answer probability η is negligible, so $M\eta$ is negligible as well. At all sufficiently large such lengths, the reduction would have size at most v^d and error less than $1/3$ on every v-vertex graph, contradicting the premise. □

C Hardness of UEOPL

The class UEOPL of Fearnley, Gordon, Mehta and Savani [FGMS19] is a subclass of CLS whose canonical problem adds solutions witnessing that the line from the start vertex is not unique. Prior hardness results for UEOPL were obtained from unambiguous succinct arguments [BCH+22, CFJW25]. In this section, we show that every hard distribution of RSVL in this paper also gives a hard distribution in UEOPL. As in [BCH+22], we work with the following problem, which has no predecessor circuit.

Definition C.1 (Unique-Forward-End-of-Metered-Line [FGMS19, BCH+22])**.** *An instance of* UFEOML *consists of a successor circuit* $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$*, a meter circuit* $\mathsf{M} : \{0,1\}^m \to [0, 2^m - 1]$ *and a string* $v_0 \in \{0,1\}^m$*, with the guarantee that* $\mathsf{S}(v_0) \neq v_0$ *and* $\mathsf{M}(v_0) = 0$*. A solution is one of the following:*

1. sink: *a vertex* v *with* $\mathsf{S}(v) \neq v$ *and* $\mathsf{S}(\mathsf{S}(v)) = \mathsf{S}(v)$*;*
2. meter violation: *a vertex* v *with* $\mathsf{S}(v) \neq v$ *and* $\mathsf{M}(\mathsf{S}(v)) \neq \mathsf{M}(v) + 1$*;*
3. meter collision: *two vertices* $u \neq v$ *with* $\mathsf{S}(u) \neq u$*,* $\mathsf{S}(v) \neq v$ *and* $\mathsf{M}(u) = \mathsf{M}(v)$*.*

UFEOML is total. Follow the successor from v_0. Either some step is a meter violation, or the meter increases by one at every step. In the latter case, no vertex repeats, and since the meter is bounded, the path reaches a vertex fixed by S, whose predecessor is a sink. Meter collisions witness that the line from v_0 is not unique. UEOPL is the class of total search problems that are Karp-reducible to UFEOML. This is equivalent to the definition of [FGMS19] through their problem Unique-End-of-Potential-Line [FGMS19, BCH+22]. Moreover, UEOPL ⊆ CLS ⊆ PPAD [FGMS19]. Hard distributions of UFEOML are defined as for EOML.

The reduction. Given an RSVL instance $I = (\mathsf{S}, \mathsf{V}, L, v_0)$ with $L \geq 1$, we augment each vertex with an index and let the meter read it. The instance $I' = (\mathsf{S}', \mathsf{M}', v_0')$ has vertices $(v, i) \in \{0,1\}^m \times [0, 2^m - 1]$, encoded with $2m$ bits, so it is an instance of Definition C.1 with vertex length $2m$, whose meter takes values in $[0, 2^m - 1] \subseteq [0, 2^{2m} - 1]$, and

$$\mathsf{S}'(v,i) := \begin{cases} (\mathsf{S}(v), i+1) & \text{if } \mathsf{V}(v,i) = 1 \text{ and } i < L, \\ (v,i) & \text{otherwise,} \end{cases} \qquad \mathsf{M}'(v,i) := i, \qquad v_0' := (v_0, 0).$$

These circuits have size polynomial in $|I|$. If I satisfies the promise, then $\mathsf{V}(v_0, 0) = 1$ and $L \geq 1$, so $\mathsf{S}'(v_0') \neq v_0'$, and $\mathsf{M}'(v_0') = 0$. Thus I' satisfies the guarantee. The vertices with $\mathsf{S}'(v,i) \neq (v,i)$ are exactly the pairs with $\mathsf{V}(v,i) = 1$ and $i < L$. The honest line is $(\mathsf{S}^i(v_0), i)$ for $i \in [0, L]$.

Lemma C.2 (Solutions of the reduced instance)**.** *Let* I *satisfy the promise of* RSVL*. Given a solution of* I'*, one can compute in polynomial time an* RSVL *solution of* I*, except for a meter collision, from which one can compute two candidates, at least one of which is an* RSVL *solution of* I*.*

Proof. We consider the three types of solutions in turn.

Meter violations. There are none. If $\mathsf{S}'(v,i) \neq (v,i)$, then $\mathsf{S}'(v,i) = (\mathsf{S}(v), i+1)$, so $\mathsf{M}'(\mathsf{S}'(v,i)) = i + 1 = \mathsf{M}'(v,i) + 1$.

Sinks. Let (v, i) be a sink. Then $\mathsf{V}(v, i) = 1$ and $i < L$, and $w := \mathsf{S}(v)$ satisfies $\mathsf{S}'(w, i+1) = (w, i+1)$. Thus $i + 1 = L$ or $\mathsf{V}(w, i + 1) = 0$. If $i + 1 = L$ and $\mathsf{V}(w, L) = 1$, then w is a sink of I. Otherwise $\mathsf{V}(w, i + 1) = 0$. If v were the honest vertex $\mathsf{S}^i(v_0)$, then $w = \mathsf{S}^{i+1}(v_0)$ would be accepted at index $i + 1$ by the promise. Hence $v \neq \mathsf{S}^i(v_0)$, and (v, i) is a false positive. The reduction evaluates $\mathsf{V}(w, L)$ to decide which case holds.

Meter collisions. Let $(u, i) \neq (v, i)$ be a collision. Both pairs leave their vertex, so $\mathsf{V}(u, i) = \mathsf{V}(v, i) = 1$, and $u \neq v$ since the indices agree. At most one of them equals $\mathsf{S}^i(v_0)$, so (u, i) or (v, i) is a false positive. □

Lemma C.2 cannot decide which of the two candidates is a false positive, since that requires computing $\mathsf{S}^i(v_0)$. Choosing one at random suffices for hardness.

Theorem C.3 (Hardness of UEOPL)**.** *If $\mathcal{D}$ is a hard distribution of* RSVL *instances with $L \geq 1$, then the distribution of I' for $I \leftarrow \mathcal{D}(1^\lambda)$ is a hard distribution of* UFEOML*. In particular, the hard distributions of* RSVL *in Theorems 4.5 and 4.7 give hard distributions in* UEOPL*, under the same assumptions as Corollaries 4.6 and 4.8.*

Proof. Let $\mathcal{A}$ output a solution of I' with probability ϵ. The adversary $\mathcal{B}$ runs $\mathcal{A}$ on I' and applies Lemma C.2, choosing one of the two candidates uniformly at random for a meter collision. Since $\mathcal{D}$ is supported on instances satisfying the promise, $\mathcal{B}$ outputs an RSVL solution of I with probability at least $\epsilon/2$, and it has size polynomial in that of $\mathcal{A}$. Since $\mathcal{D}$ is hard, ϵ is negligible. In Fig. 7, $L = 2^H \geq 1$. □

The worst-case result of Section A transfers in the same way: a circuit that solves UFEOML on all instances yields an RSVL solution of each promised instance with probability at least $1/2$, which can replace the guarantee of Theorem 3.3 used there.