Unambiguous SNARGs for P from LWE with Applications to PPAD Hardness

Liyan Chen[1], Cody Freitag[2], Zhengzhong Jin[2], and Daniel Wichs[2,3]

[1]MIT
[2]Northeastern University
[3]NTT Research

October 13, 2025

Abstract

We construct the first *unambiguous* succinct non-interactive arguments (SNARGs) for P and incrementally verifiable computation (IVC) for P from the polynomial hardness of learning with errors (LWE). Unambiguity guarantees that it is computationally hard to find two distinct accepting proofs for the same statement.

As an application, we establish the first PPAD hardness result based on the polynomial hardness of LWE combined with a widely believed complexity assumption.

Central to our approach is a new notion of rate-1 *witness-unambiguous* batch arguments for NP, which we give the first construction from the polynomial hardness of LWE. This notion may be of independent interest.

Contents

1 Introduction **1**
- 1.1 Our Results . . . 3

2 Technical Overview **5**
- 2.1 Constructing Non-Trivial Witness-Unambiguous BARGs . . . 6
- 2.2 From Non-Trivial Witness-Unambiguous BARGs to PPAD-Hardness . . . 10
  - 2.2.1 Unambiguous SNARGs from Witness-Unambiguous BARGs. . . . 11
  - 2.2.2 Boosting the Efficiency of Witness-Unambiguous BARGs . . . 13
  - 2.2.3 Unambiguous Incrementally Verifiable Computation . . . 16

3 Preliminaries **17**
- 3.1 Lattice-based Cryptography . . . 18
- 3.2 Linear PCPs . . . 19
- 3.3 Correlation Intractable Hash Function . . . 20
- 3.4 Batch Arguments . . . 21
- 3.5 Hash Trees . . . 23
- 3.6 RAM Computation and Unambiguous RAM Delegation . . . 26

4 Somewhere-Extractable Linearly Homomorphic Commitments **29**
- 4.1 Preliminary: Regev's Public Key Encryption and Ciphertext Errors . . . 29
- 4.2 Definition . . . 30
- 4.3 Construction . . . 32
- 4.4 Analysis . . . 33

5 Witness-Unambiguous Batch Arguments **36**
- 5.1 Intermediate Construction . . . 37
- 5.2 Improving the Dependence on k . . . 44

6 Unambiguous and Flexible RAM Delegation **45**
- 6.1 Flexible RAM Computation and Delegation Definitions . . . 45
- 6.2 Incremental and Unambiguous Computation of Hash Roots . . . 48
- 6.3 RAM Delegation from BARGs . . . 49

7 Boosting Batch Arguments Preserves Unambiguity **54**
- 7.1 Succinct BARGs from Non-Trivial RAM Delegation . . . 55
- 7.2 Rate-1 Unambiguous seBARGs . . . 62

8 Unambiguous Incrementally Verifiable Computation **63**
- 8.1 Unambiguous Rigid Mergeable Delegation . . . 63
- 8.2 Unambiguous Incrementally Verifiable Computation . . . 68

9 Unambiguous IVC with Universal Soundness and PPAD Hardness **70**
9.1 Universal Soundness 70
9.1.1 Additional Definitions 71
9.1.2 The Transform 72
9.2 PPAD Hardness 73

A Rate-1 Fully Local Somewhere Extractable Hash Trees **81**

B Proof of Lemma 3.8 **85**

C Additional Preliminaries **87**
C.1 Fully Homomorphic Encryption 87
C.2 Total Search Problems 88

1 Introduction

While it is known that a mixed Nash equilibrium is guaranteed to exist in any bimatrix game [Nas51], there is no known polynomial-time algorithm for finding one. In his seminal work, Papadimitriou introduced the complexity class PPAD (Polynomial Parity Arguments on Directed graphs) and showed that finding Nash equilibrium is in PPAD [Pap94]. PPAD can be defined through the following complete problem, End-of-the-Line, which asks for an efficient algorithm to find a source or sink in an exponentially large graph represented by a small circuit, where each vertex has an in-degree and out-degree of at most 1. Subsequently, the work [DGP09] and [CDT09] established that computing a Nash equilibrium is PPAD-complete.

The actual hardness of PPAD remains unresolved. As suggested by Papadimitriou, PPAD may contain cryptographic problems. Following this, a long line of work [BPR15; GPS16; HY20; CHK+19a; CHK+19b; EFKP20a; KPY20; LV20; JKKZ21; BCH+22; KLV23] has focused on basing PPAD hardness on cryptographic assumptions. Despite substantial progress, the current landscape remains unsatisfactory: existing results either rely on the super-polynomial hardness of the assumptions [KPY20; JKKZ21; KLV23] by incurring a super-polynomial time overhead in the reduction, or combine multiple cryptographic assumptions [GPS16; BCH+22]. A central objective of this long line of work is to achieve a clean result that bases PPAD on a single, standard cryptographic assumption with polynomial-time hardness.

In this work, we ask the following question:

Can we base the hardness of PPAD on the polynomial hardness of the learning with errors problem?

The learning with errors (LWE) assumption is a standard cryptographic assumption whose hardness can be based on worst-case lattice problems [Reg05]. It is conjectured to be secure against quantum adversaries and is versatile enough to be the cornerstone for a wide range of cryptographic primitives. So far, we know how to build encryption schemes [Reg05], digital signatures [GPV08], and numerous advanced encryption schemes from LWE [Gen09; BV11; GVW13; GKPVZ13; GSW13; BGG+14], making LWE one of the most promising assumptions for post-quantum cryptography. Basing PPAD hardness on LWE would additionally provide post-quantum hardness for PPAD problems. In contrast, the previous works based on multiple well-studied polynomial-time assumptions [GPS16; BCH+22] both relied on some assumptions known to be quantum insecure.

Due to their close connection to the End-of-the-Line problem, building unambiguous (i.e., computationally unique) succinct non-interactive arguments (SNARGs) and incrementally verifiable computation (IVC) have been core components in constructing hard PPAD instances, as shown in a series of prior works [CHK+19a; CHK+19b; EFKP20a; KPY20; LV20; JKKZ21; BCH+22; KLV23].

SNARGs, IVC, and Unambiguity. SNARGs allow a prover to convince a verifier of the correctness of a computation via a short, single message, where the soundness holds only against polynomial-time bounded adversaries. Importantly, the proof size and verification time do not grow with the number of computation steps. SNARGs are constructed in the common reference string (CRS) model, where a trusted setup algorithm generates a CRS and shares it with the prover and the verifier. Since Micali's seminal work [Mic94] on computationally-sound proofs, there has been tremendous progress in basing SNARGs on minimal cryptographic assumptions. To date, we have SNARGs for P from various standard cryptographic assumptions [KVZ21; CJJ22; HJKS22;

WW22; CGJJZ23], for certain subclasses of NP from standard assumptions [CJJ21; BBKLP23; JKLV24], and for NP from indistinguishability obfuscation (iO) and one-way functions [SW14; WW24b; WZ24; WW24a], which in turn can be based on standard assumptions. *Incrementally verifiable computation* (IVC) [Val08] is a type of SNARG for P where the proof can be efficiently updated with each new computational step, using the current proof and the latest computation. Recently, IVC for deterministic computations has been constructed from standard cryptographic assumptions [DGKV22; PP22]. SNARGs and IVC have a broad range of applications such as zero-knowledge [Kiy22], mitigating misinformation [NT16], blockchains [BCG+14], and many more.

From a foundational perspective, we emphasize that the complexity class P has unique proofs; the computation trace of a deterministic computation is uniquely determined given its input. So philosophically, it is natural to wonder if SNARGs for P can also be made unique. Towards answering this question, we consider unambiguous SNARGs or IVC where the accepting proof is at least *computationally* unique. That is, no polynomial-time adversary can find two different accepting proofs. Most of the existing SNARG constructions are *not* unambiguous. To the best of our knowledge, the only unambiguous SNARGs for P require sub-exponential hardness of iO [SW14; WW24b; WZ24; WW24a], non-falsifiable knowledge assumptions [BBHMR19], a non-standard quasi-polynomial hardness assumption on bilinear maps [KPY20], or only support bounded depth computations [CHK+19a; JKKZ21; KLV23] or specific problems in P [CHK+19b; EFKP20a; LV20; BCH+22] (notably the repeated squaring function). Furthermore, only the works of [BBHMR19] and [KPY20] yield unambiguous IVC for P.

As such, we ask the following question:

Can we build unambiguous SNARGs for P and IVC from any *falsifiable polynomial-time hardness assumptions? In particular, from the polynomial hardness of LWE?*

As mentioned above, unambiguous SNARGs and IVC have been extensively studied in recent years due to their applications for constructing hard PPAD instances via the connection to the End-of-the-Line problem. For these applications, unambiguity is essential to ensure that the represented graph in the (relaxed) End-of-the-Line instance has in- and out-degrees of at most one. The work of [CLP15] leveraged the (perfect) uniqueness of the particular iO-based SNARG of [SW14] to build constant-round concurrent zero-knowledge arguments from iO. More broadly, unambiguity has been extensively studied in both complexity theory and other areas of cryptography. Examples include the complexity class of unambiguous computation UP [Val76], unambiguous interactive proof systems [RRR16], unique signatures [GO93; Lys02], and unique (non-interactive) zero-knowledge proofs [LMs05; AFGJM22].

PPAD Hardness via Unambiguous IVC for P. The recent work of [BPSS23] shows how to base PPAD hardness on any unambiguous IVC for P and *a widely believed complexity assumption*.[1] Roughly speaking, the complexity assumption essentially assumes the existence of a constant c such that $\mathsf{P} \cap \mathsf{DSPACE}(n^c) \not\subset \mathsf{ioSIZE}(n^d)$ for any constant d. We emphasize that this is a rather mild assumption, since it does not appear to imply one-way functions or even $\mathsf{NP} \neq \mathsf{P}$. Indeed, the assumption can be viewed as a natural extension of the time hierarchy theorem to the non-uniform and space-bounded setting.

Specifically, the PPAD instance is generated by computing a hard puzzle and using an unambiguous IVC to certify the correctness of this computation. The main challenge is to prove the

[1] The complexity assumption can be a worst-case assumption if fully homomorphic encryption is assumed, or alternatively, an average-case complexity assumption.

hardness of the instance while relying only on the polynomial security of the underlying IVC. This is difficult because, intuitively, the reduction needs to compute the hard puzzle by itself to verify whether the computation is correct, which already takes super-polynomial time. The non-interactive universal argument [BPSS23] resolved this issue by assuming the aforementioned widely believed complexity assumption, and then they showed how to lift any unambiguous IVC for P to achieve an additional soundness property called *universal soundness*. This notion guarantees soundness for any *exponential-time* computation while relying only on polynomial hardness of LWE and the above complexity assumption.

1.1 Our Results

In this work, we answer the above two questions positively by constructing unambiguous IVCs from the polynomial hardness of LWE.

Theorem 1.1 (Unambiguous IVC, Informal). Assuming the polynomial hardness of LWE, there exist unambiguous IVCs for all deterministic computations. ◇

We again emphasize that this is the first such construction from *any* falsifiable, polynomial-time hardness assumptions as the previous constructions of [BBHMR19] required non-falsifiable knowledge assumptions and [KPY20] required quasi-polynomial hardness of a non-standard assumption on bilinear maps.

As observed in [PP22], IVC immediately gives *complexity-preserving* SNARGs for P; these are extremely efficient SNARGs where proving deterministic time-T and space-S computation takes essentially the same time $\tilde{O}(T)$ and space $\tilde{O}(S)$, up to multiplicative polynomial factors in the security parameter. As unambiguous IVC gives unambiguous SNARGs, we get the following corollary to Theorem 1.1.

Theorem 1.2 (Unambiguous Complexity-Preserving SNARGs for P, Informal). Let λ denote the security parameter. Assuming the polynomial hardness of LWE, there exists an unambiguous SNARG for P such that proving any deterministic RAM computation M that on input x runs in time T and space S such that $T, S \leq 2^{\lambda}$ has the following efficiency:

- CRS size: $\mathsf{poly}(\lambda)$
- Prover complexity: time $T \cdot \mathsf{poly}(\lambda, |x|)$ and space $S \cdot \mathsf{poly}(\lambda, |x|)$
- Proof size: $\mathsf{poly}(\lambda)$
- Verification time: $|x| \cdot \mathsf{poly}(\lambda)$ ◇

Our result for PPAD hardness follows from the framework of [BPSS23] as an immediate corollary to Theorem 1.1. In addition to the polynomial hardness of LWE, we require the following complexity assumption.

Assumption 1.3 (Average-case Complexity Assumption, Informal). There exists a constant c such that, for any constant d, there exists an $O(n^c)$-space polynomial-time function that is hard-on-average against functions in $\mathsf{ioSIZE}(n^d)$.

Theorem 1.4 (PPAD Hardness, Informal). Assuming the polynomial hardness of LWE and Assumption 1.3, there is a problem in PPAD that can't be solved by any non-uniform probabilistic polynomial time with non-negligible probability. ◇

We remark that the above assumption is relatively mild. By further assuming fully homomorphic encryption, the average-case complexity assumption 1.3 can be weakened to the worst-case assumption that $\mathsf{P} \cap \mathsf{DSPACE}(n^c) \not\subset \mathsf{ioSIZE}(n^d)$ for any constant d [BPSS23]. Hence, both the average-case and the worst-case assumptions are widely believed to hold.

Witness-unambiguous BARGs. Our construction of unambiguous IVC follows the blueprint of [DGKV22; PP22] that goes through rate-1 batch arguments (BARGs) for NP. However, we are unlikely to achieve unambiguous BARGs for NP as witnesses for NP aren't even unique. As such, we introduce and construct a new notion of "witness-unambiguous" BARGs for NP, which is a key component for achieving our main results.

A BARG is a proof system in the CRS model where the prover convinces the verifier that a batch of instances $x_1, \ldots, x_k$ are in an NP language L with communication sub-linear in the length of the k NP witnesses. At a high level, we say that a BARG is *witness-unambiguous* if any adversary that produces two distinct accepting proofs must have used two different sets of witnesses to generate these proofs. We formalize this using "somewhere extractable" BARGs, where there is a way to program the CRS at a hidden index $i \in [k]$ such that, given a trapdoor for the CRS and accepting BARG proof, you can extract a witness for the i-th instance. Witness-unambiguity guarantees that if an adversary can produce two distinct accepting proofs, there must exist some hidden index i for which you can extract two accepting witnesses for x_i.

At the heart of our unambiguous IVC construction is the *first* witness-unambiguous BARG based on the polynomial hardness of LWE. Moreover, our construction is rate-1, meaning the proof size only grows with the length of one of the NP witnesses, up to a sub-linear additive overhead.

Theorem 1.5 (Rate-1 Witness-Unambiguous BARGs, Informal). Let λ denote the security parameter. Assuming the polynomial hardness of LWE, there exists a rate-1 witness-unambiguous BARG for NP such that proving a batch of k NP statements with witness length m and NP-verification time T for $k, m, T \leq 2^\lambda$ has the following efficiency:

- CRS size: $\mathsf{poly}(\lambda, T)$
- Prover time: $\mathsf{poly}(\lambda, k, T)$
- Proof size: $m + O(m/\lambda) + \mathsf{poly}(\lambda)$
- Verification time: $\mathsf{poly}(\lambda, T)$ ◇

Paper Organization and Blueprint for Obtaining Our Results

In Section 2, we give an overview of the constructions and techniques to achieve our main results. We formalize the necessary preliminaries for the paper in Section 3. In the rest of the paper, we give the technical details behind our results. At a high level, our main results follow from the following blueprint:

- First, in Section 5, we give a construction of a non-trivially succinct witness-unambiguous BARG from LWE. This construction relies on a new somewhere-extractable linearly homomorphic commitment from LWE that we build in Section 4.

- In Section 6, we give the transformation from a witness-unambiguous BARG to an unambiguous SNARG for P. Combined with the non-trivial BARG from Section 5, this gives a non-trivially succinct SNARG for P.
- Then, in Section 7, we build rate-1 witness-unambiguous BARGs from non-trivially succinct SNARGs for P.
- In Section 8, we give the construction of unambiguous IVC from rate-1 witness-unambiguous BARGs.
- Finally, in Section 9, we formalize the application of unambiguous IVC to PPAD hardness.

2 Technical Overview

In this section, we give an overview for the main techniques underlying our construction of unambiguous (i.e., computationally unique) succinct non-interactive arguments (SNARGs) for P with applications to PPAD hardness.

Ignoring uniqueness, the current approach to building SNARGs for P (under minimal assumptions) goes through the route of *batch arguments for NP* (BARGs). As such, it would be natural to first try to construct BARGs with unique proofs, and then show that these give SNARGs for P with unique proofs as well. Unfortunately, it seems very unlikely to build BARGs with unique proofs since BARGs are defined for NP, where the witnesses may not even be unique![2] It turns out, however, that the existing constructions of SNARGs for P rely on BARGs for very particular NP languages, where the NP witnesses are at least *computationally unique*, meaning that no polynomially-bounded adversary can find two valid witnesses. So, there is hope that we can preserve this computational uniqueness property in the BARG-to-SNARG transformation. We define a new uniqueness property for BARGs—which we call *witness unambiguity*—that captures this property.

Defining witness-unambiguity for BARGs. Recall that for a BARG, the prover is trying to efficiently convince the verifier that a batch of k statements $x_1, \ldots, x_k$ are in some NP language L. At a high level, we want witness-unambiguity to guarantee that if a prover comes up with two convincing proofs $\pi \neq \pi'$ for the batched statement, then it must "know" two distinct valid witnesses $w_i \neq w_i'$ for some input x_i.

We formalize this intuition using the notion of "somewhere extractability," which most existing BARGs already satisfy. Specifically, in a somewhere extractable BARG (seBARG), the setup algorithm is programmed at a hidden index i and outputs a trapdoor for i. Then, there is an extraction algorithm that given this trapdoor and an accepting proof π for the batched statement, extracts a witness w_i for x_i.

Using this somewhere extraction property, we can formalize our intuition for witness unambiguity. Namely, we require that if an adversary can come up with two distinct proofs $\pi \neq \pi'$ that are accepting for a batched statement $(x_1, \ldots, x_k)$, then there must exist an index i such that if we had programmed the setup to be extractable on i, we would extract out distinct valid witnesses $w_i \neq w_i'$ for x_i with comparable probability. In essence, this captures the notion that to generate

[2] Technically, [CPW20] show that constructing BARGs for NP with unique proofs would imply a witness encryption scheme, which we currently don't know how to construct from LWE or other polynomial hardness assumptions.

two distinct BARG proofs for the same inputs, you must use a different set of witnesses. So, if those particular inputs $x_1, \ldots, x_k$ have (computationally) unique witnesses, then this preserves the (computational) uniqueness of the BARG proof.

In the remainder of this overview, we focus on (1) how to construct witness-unambiguous seBARGs, and (2) how to use such BARGs to generically build incrementally and uniquely verifiable computation following the various transformations of [CJJ22; DGKV22; PP22; KLVW23]. Furthermore, following the works of [CHK+19a; KPY20; BPSS23], this implies hard-on-average instances for problems in PPAD.

2.1 Constructing Non-Trivial Witness-Unambiguous BARGs

The BARG construction of [CJJ22] heavily relies on probabilistically checkable proofs (PCPs) in order to achieve poly-logarithmic communication. PCPs are long, information-theoretic proofs that a verifier can efficiently check by probing the proof at a few random locations. PCPs, however, are far from being unique. Since each individual bit is very unlikely to be queried, modifying a valid proof at a single location likely yields another distinct valid proof.

Starting point: BARGs from linear PCPs. We follow the approach of [CJJ21] that achieves BARGs with non-trivial (but not poly-logarithmic) communication by relying on a highly structured variant of PCPs, known as *linear PCPs*. To see why linear PCPs may be useful for achieving unique proofs, we start by describing the linear PCP implicit in the Hadamard-based PCP of [ALMSS98].

Let (x, w) be an instance-witness pair such that $C(x, w) = 1$ for some NP circuit C of size t. This linear PCP computes the wire values s corresponding to every gate in $C(x, w)$ and gives out s as well as all pair-wise products for values in s:

$$\pi_{\mathsf{LPCP}} = (s, s \otimes s).$$

The verifier checks π_{LPCP} by specifying queries of the form $q \in \{0,1\}^{t+t^2}$ and receives $\langle \pi_{\mathsf{LPCP}}, q \rangle$. Using very few queries, the verifier first checks that π_{LPCP} is of the correct form $(s, s \otimes s)$ for some wire assignment s, and then checks random linear combinations of the gate constraints imposed by C to ensure that s is indeed a satisfying assignment.

As a plain PCP, the above proof is woefully inefficient: it has length $2^{O(t^2)}$, whereas state-of-the-art PCPs have length $t \cdot \mathsf{polylog}(t)$. However, because this size $O(t^2)$ representation is highly structured, it yields a distinct *linear PCP proof* per accepting gate assignment. Specifically, given any two distinct accepting proofs $\pi_{\mathsf{LPCP}} \neq \pi'_{\mathsf{LPCP}}$, we can read off the distinct witnesses $w \neq w'$ that the gate assignments correspond to.

For this reason, we first focus on the approach of [CJJ21] that achieves BARGs with non-trivial efficiency by proving batches of structured PCPs as above. In particular, they rely on succinct linear homomorphic commitments to aggregate the linear PCP proofs, meaning for any vectors $\mathbf{x}$, $\mathbf{y}$,

$$\mathsf{Commit}(\mathbf{x}) + \mathsf{Commit}(\mathbf{y}) = \mathsf{Commit}(\mathbf{x} + \mathbf{y}).$$

We note that such commitments are *deterministic*, so in particular are not hiding.

We describe their BARG for the index language where the inputs are simply indices $1, \ldots, k$ (this is known to be equivalent to general BARGs, so we focus on this case for simplicity). In this setting, the prover on input a machine M and witnesses $w_1, \ldots, w_k$ is trying to convince the

verifier that $M(i, w_i) = 1$ for all $i \in [k]$. We let t denote the running time of M such that a linear PCP for the claim that $M(i, w_i) = 1$ has length $\ell \in O(t^2)$. We give a high-level overview of their construction below:

The Prover Algorithm.

1. For each $i \in [k]$, the prover first computes a linear PCP π^i_{LPCP} for the claim that $M(i, w_i) = 1$. The prover arranges the proofs $\left\{\pi^i_{\mathsf{LPCP}}\right\}_{i\in[k]}$ in rows, and uses a linear homomorphic commitment to aggregate the columns: $C = (\mathsf{c}_1, \ldots, \mathsf{c}_\ell)$ such that c_j for $j \in [\ell]$ is a commitment to bits $\pi^i_{\mathsf{LPCP}}[j]$ for each $i \in [k]$.
2. The prover uses a correlation intractable (CI) hash function on C to sample the randomness to generate the linear PCP queries Q. For each query $q \in Q$, the prover computes the responses for all k linear PCPs: $\left\langle \pi^i_{\mathsf{LPCP}}, q \right\rangle$ for each $i \in [k]$ and stores the values in a length k vector $\mathbf{v}_q$.
3. The final proof π consists of the commitment C and query responses $\mathbf{v}_q$ for every $q \in Q$. Note that, up to polynomial factors in the security parameter, C has length $|\pi_{\mathsf{LPCP}}| = \ell \in O(t^2)$ and each vector $\mathbf{v}_q$ consists of k entries, so the proof has length $|\pi| \in O(k + t^2)$.

The Verifier Algorithm.

1. The verifier uses the CI hash on C to generate the same linear PCP queries Q and verifies:
   - **Consistent responses.** For each query $q \in Q$,
   $$\left\langle C, q \right\rangle = \sum_{j\in[\ell]} q_j \mathsf{c}_j = \mathsf{Commit}(\mathbf{v}_q),$$
   meaning that the "query response" for the committed columns $\left\langle C, q \right\rangle$ should be equal to the "commitment" to query responses $\mathsf{Commit}(\mathbf{v}_q)$.
   - **Linear PCP checks.** For each $i \in [k]$, the linear PCP verifier accepts statement i given responses $(\mathbf{v}_q)_i$ for each $q \in Q$.

To prove soundness for the above protocol, we want to argue that the only way it can pass the consistency checks and linear PCP checks is if it actually "knows" accepting linear PCPs for each statement $i \in [k]$—and hence the batched claim was actually true. To formalize this argument, [CJJ21] uses commitments that satisfy additional linearly homomorphic extraction properties as follows:

- **Somewhere Extractable.** The setup algorithm produces a trapdoor td programmed at a hidden index i^* such
$$\mathbf{x}_{i^*} = \mathsf{Ext}(\mathsf{td}, \mathsf{Commit}(\mathbf{x})).$$
- **Linearly Homomorphic Extraction.** For (even maliciously chosen) commitments $\mathsf{c}_1, \mathsf{c}_2$ it holds that
$$\mathsf{Ext}(\mathsf{td}, \mathsf{c}_1) + \mathsf{Ext}(\mathsf{td}, \mathsf{c}_2) = \mathsf{Ext}(\mathsf{td}, \mathsf{c}_1 + \mathsf{c}_2).$$

Then, if we set the commitment to be extractable at index i^*, we can extract a linear PCP $\pi^{i^*}_{\mathsf{LPCP}}$ from C. Furthermore, the homomorphic properties of the commitment guarantee that

$$(\mathbf{v}_q)_{i^*} = \left\langle \pi^{i^*}_{\mathsf{LPCP}}, q \right\rangle$$

for each $q \in Q$. By using an appropriate CI hash function (such as the one of [HLR21] from LWE), it follows that $\pi^{i^*}_{\mathsf{LPCP}}$ must be a valid linear PCP proof, so statement i^* must actually be true.

The work of [CJJ21] constructs such extractable linear homomorphic commitments based on the quadratic residuosity (QR) assumption. Also, their proof of soundness requires particular CI hash functions known from either LWE or sub-exponential DDH assumptions. Our goal in this work is twofold:

- First, to minimize the necessary assumptions, ideally relying only on the polynomial hardness of the learning-with-errors (LWE) assumption. To do so, we construct suitable extractable linear homomorphic commitments from LWE.
- Second, to prove that this construction satisfies witness-unambiguity. This requires several careful modifications to the protocol based on the LWE-based commitment we use to ensure that the commitment values C and query responses $\mathbf{v}_q$ are uniquely determined by the witnesses used.

Somewhere-extractable linearly homomorphic commitments from LWE. To start, consider any public-key encryption (PKE) scheme that supports linear homomorphic computations. This gives a natural candidate for a commitment as follows. The commitment key consists of k ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_k)$, and to commit a vector $\mathbf{x} = (x_1, \ldots, x_k)$, we compute $\mathsf{c} = \sum_{i\in[k]} x_i \mathsf{ct}_i$. To extract the index i^*, we can set ct_{i^*} in the commitment key to be an encryption of 1 and set all others to be encryptions of 0, and give the secret key sk out as a trapdoor. It follows that

$$\mathsf{Ext}\left(\mathsf{td} = \mathsf{sk}, \mathsf{c} = \sum_{i\in[k]} x_i \mathsf{ct}_i\right) = \mathsf{Dec}\left(\mathsf{sk}, \sum_{i\in[k]} x_i \mathsf{ct}_i\right) = x_{i^*}.$$

Unfortunately, this doesn't quite work for the BARG construction above. The issue is that the extraction argument only holds for *well-formed* ciphertexts and may not hold for maliciously chosen ciphertexts.

Indeed, if we instantiate the above approach with Regev's LWE-based PKE scheme,[3] the above argument is false when the commitment is a ciphertext with large noise. One way to circumvent this issue is not to work with such commitments in our BARG construction! Specifically, when the prover sends the initial commitment values $C = (\mathsf{c}_1, \ldots, \mathsf{c}_\ell)$, we have it additionally prove that the commitments all have "mild" noise (namely, small enough to work for linear homomorphic extraction). We do this by deriving random linear queries of the form $q \in \{0,1\}^\ell$ using a CI hash function on C, and for each one, we have the prover give a short vector $\mathbf{y}$ opening $\left\langle C, q \right\rangle$ such that $\mathsf{Commit}(\mathbf{y}) = \sum_{j\in[\ell]} q_j \mathsf{c}_j = \left\langle C, q \right\rangle$. If such a $\mathbf{y}$ exists, then $\left\langle C, q \right\rangle$ must have mild noise, so all c_j must have mild noise. We briefly argue why this additional proof is complete and sound:

[3] The ciphertexts ct_i in Regev encryption have a large modulus. For technical reasons, we will require that commitment vectors have short entries. To make every entry of $\mathsf{Commit}(\mathbf{x})$ small, we apply lattice gadget to the ciphertexts ct_i, so we will instead compute $\mathsf{Commit}(\mathbf{x}) = \sum_{i\in[k]} x_i \mathbf{G}^{-1}[\mathsf{ct}_i]$.

- **Completeness.** Observe that an honest prover can compute $\mathbf{y}$ from the pre-images of $\mathsf{c}_1, \ldots, \mathsf{c}_\ell$: if $\mathsf{c}_j = \mathsf{Commit}(\mathbf{x}_j)$ for each $j \in [\ell]$, then $\mathbf{y} = \sum_{j\in[\ell]} q_j \mathbf{x}_j$ is short and satisfies $\mathsf{Commit}(\mathbf{y}) = \sum_{j\in[\ell]} q_j \mathsf{c}_j = \langle C, q \rangle$, as required.

- **Soundness.** We claim that for each query q, $\langle C, q \rangle$ will have large noise with probability at least $1/2$ if one of $\mathsf{c}_1, \ldots, \mathsf{c}_\ell$ have large noise. This is because if c_j has large noise, for instance, then one of $\langle C, q \rangle$ or $\langle C, q \rangle \pm \mathsf{c}_j$ must have large noise. Furthermore, we can instantiate this CI hash for product relations using the LWE-based hash function of [HLR21].

Using the commitment as above with the associated proof of mild noise, it follows that the linear PCP based BARG is sound assuming only LWE.

Towards a proof of witness-unambiguity. Let's try to argue that the current construction satisfies witness-unambiguity. Suppose that an adversary outputs two distinct valid proofs $\pi \neq \pi'$. Recall that π consists of two parts: column-wise commitments of linear PCPs $C = (\mathsf{c}_1, \ldots, \mathsf{c}_\ell)$, and row-wise product vectors $\mathbf{v}_q$ for each query $q \in Q$ (including both the linear PCP queries and the queries for proving the commitments have mild noise). Let $\pi = (C, \{\mathbf{v}_q : q \in Q\})$ and $\pi' = (C', \{\mathbf{v}'_q : q \in Q'\})$. If $\pi \neq \pi'$, we want to argue that we must be able to extract distinct valid witnesses $w_i \neq w'_i$. We consider the following two cases:

- If $C = C'$, then note that the set of queries Q and Q' are the same as they are derived by a CI hash of the same values. It remains to argue that for any query q, $\mathbf{v}_q = \mathbf{v}'_q$. As the verifier accepts each proof, it follows that

  $$\mathsf{Commit}(\mathbf{v}_q) = \langle C, q \rangle = \langle C', q \rangle = \mathsf{Commit}(\mathbf{v}'_q)$$

  as $C' = C$ and the verifier checks that $\mathsf{Commit}(\mathbf{v}_q) = \langle C, q \rangle$ and $\mathsf{Commit}(\mathbf{v}'_q) = \langle C', q \rangle$.

  If we could extract from these commitments, $\mathbf{v}_q = \mathbf{v}'_q$ would follow from index hiding, but extraction may fail for maliciously formed vectors with large entries. Fortunately, it's easy to check if vectors have large entries, and honestly generated responses should have entries at most ℓ since $\mathbf{v}_q = \langle \pi^i_{\mathsf{LPCP}}, q \rangle$ where $\pi^i_{\mathsf{LPCP}}, q \in \{0,1\}^\ell$ are binary strings. So, by forcing the verifier to reject responses $\mathbf{v}_q$ with entries greater than ℓ, $\mathsf{Commit}(\cdot)$ is collision-resistant, so $\mathbf{v}_q = \mathbf{v}'_q$.

- Otherwise, it must be the case that $C \neq C'$. Since C, C' are column-wise commitments to the k linear PCPs for each statement, then intuitively, we would hope to argue that we can extract out distinct linear PCPs $\pi_{\mathsf{LPCP}} \neq \pi'_{\mathsf{LPCP}}$ at some index $i \in [k]$, which suffices for witness-unambiguity.

  To argue this, let's imagine there was a "common query" q that was included in both π and π' that gave responses $\mathbf{v}_q, \mathbf{v}'_q$. If we can't extract distinct linear PCPs at any index $i \in [k]$, then it must be the case that $\mathbf{v}_q = \mathbf{v}'_q$. It follows that

  $$\langle C, q \rangle = \mathsf{Commit}(\mathbf{v}_q) = \mathsf{Commit}(\mathbf{v}'_q) = \langle C', q \rangle .$$

  Now, we observe if $\langle C, q \rangle = \langle C', q \rangle$ for many different random queries q, it likely should be the case that C is actually equal to C'. Technically, if the set of common queries Q^* forms a linear collision-resistant hash function, then $C \cdot Q^* = C' \cdot Q^*$ would imply that $C = C'$, leading to a contradiction.

Can we ensure that π and π' contain such common queries? The answer is yes! Our key insight is that we can just put such queries in the CRS, forcing every valid proof to provide such answers with associated consistency checks.

In summary, our final witness-unambiguous BARG construction follows the linear PCP based framework of [CJJ21] with the following modifications:

1. We use an LWE-based, somewhere-extractable linear homomorphic commitment based on Regev encryption. We have the prover respond to additional queries (determined by a CI hash) to ensure that the commitments C have mild noise, so we can argue that they satisfy the required linearly homomorphic extraction property necessary for soundness.
2. For all query response vectors $\mathbf{v}_q$, the verifier checks that the vector is short—has entries at most the length of the linear PCP ℓ. This ensures proper extraction from the response vectors to argue that they must be unique in the case that $C = C'$.
3. Lastly, we add common queries to the CRS making up a binary linear CRHF Q^*. This is the key modification that guarantees the witness-unambiguity property. It allows us to argue that if two proofs give different responses to these common queries, we can extract out distinct linear PCPs and hence witnesses for some statement i.

Improving the dependence on k. For our BARG construction, the verifier still needs to perform k linear PCP checks given the responses $\mathbf{v}_q$ for each query $q \in Q$, of which there are only polynomially many in the security parameter λ. By the efficiency of the linear PCPs, each check is essentially independent of the complexity t for directly verifying each statement. This gives total verification time of $(k + \mathsf{poly}(t)) \cdot \mathsf{poly}(\lambda)$. Specifically, let c be a constant such that the verification time is at most $(k + t^c) \cdot \lambda^c$.

Our goal is to achieve non-trivial efficiency (better than $k \cdot t$), which unfortunately may not be the case when t is very small and $k \cdot \lambda^c > k \cdot t$. However, we observe that we can group $k^{1/(c+1)}$ statements together and use the above BARG for $k' = k^{c/(c+1)}$ new grouped statements each with complexity $t' = t \cdot k^{1/(c+1)}$. It follows that the resulting BARG has complexity

$$(k' + (t')^c) \cdot \lambda^c \leq k^{c/(c+1)} \cdot (t^c + 1) \cdot \lambda^c,$$

which is non-trivial for large enough k given any fixed t, λ.

2.2 From Non-Trivial Witness-Unambiguous BARGs to PPAD-Hardness

We next show how to use the non-trivial witness-unambiguous BARG above to build incrementally and uniquely verifiable computation, which yields hard-on-average instances for PPAD. Our construction follows three main steps:

1. witness-unambiguous BARGs to unambiguous SNARGs following [CJJ22; KLVW23],
2. non-trivially succinct unambiguous SNARGs to rate-1 witness-unambiguous BARGs following [PP22; DGKV22; KLVW23], and
3. rate-1 witness-unambiguous BARGs to incrementally and uniquely verifiable computation following [DGKV22; PP22].

We primarily focus on the first two steps in this overview since they require new techniques to preserve uniqueness. The third step follows with minor modifications over the existing.

We emphasize that these transformations are generic in the underlying witness-unambiguous BARG. The only additional assumptions we need are rate-1 string oblivious transfer (OT) protocols, which we use to build rate-1 somewhere extractable hash trees. Furthermore, rate-1 string OT protocols are known from any one of the following cryptographic assumptions: LWE, QR, DCR, DDH, or $O(1)$-LIN.

2.2.1 Unambiguous SNARGs from Witness-Unambiguous BARGs.

We show how to modify the construction of BARGs to SNARGs from [CJJ22] to preserve (computational) uniqueness. We start by defining the notion of RAM computation we consider in this work in order to define the notion of RAM delegation we achieve.

A RAM computation is defined by its "configuration" consisting of its local state and working memory. RAM computation proceeds via steps, where in each step the machine reads a bit from memory, updates its local state, and then potentially writes a bit to memory. In a RAM delegation scheme, a prover tries to convince a verifier that from the initial configuration cf, the computation reaches a new configuration cf' after t steps of computation. To handle RAM computations with potentially large memory (that might even be longer than the computation itself), the verifier takes as input only a short digest d of the configuration at each step.

At a very high level, BARGs seem like they should be useful to efficiently prove t-step RAM computations: simply prove that each step was performed correctly. This intuition is a little misleading, however, since naively BARGs are only used to prove independent statement. So even if the prover comes up with a convincing proof that the initial digest d_0 transitions to d_1 and d_1 transitions to d_2, it might be the case that a malicious cheating prover uses different values for d_1 in each claim. To circumvent this issue, [CJJ22] suggested to first use a hash tree to hash all of the digests resulting in a root rt_d, and prove in each step i that d_{i-1} transitions to d_i *and* $\mathsf{d}_{i-1}, \mathsf{d}_i$ are consistent with rt_d.

As a result, the proof π of a RAM computation consists only of two parts: the hash root of the digests rt_d, and the BARG proof π_{barg}. Let's try to prove that this proof already is computationally unique assuming the BARG proof π_{barg} satisfies witness-unambiguity. Suppose that an adversary comes up with two different proofs $\pi \neq \pi'$ for a claim that digest d_0 transitions to digest d_t after t steps. Let $\pi = (\mathsf{rt}_\mathsf{d}, \pi_{\mathsf{barg}})$ and $\pi' = (\mathsf{rt}'_\mathsf{d}, \pi'_{\mathsf{barg}})$. We consider two different cases:

- If $\mathsf{rt}_\mathsf{d} = \mathsf{rt}'_\mathsf{d}$ but $\pi_{\mathsf{barg}} \neq \pi'_{\mathsf{barg}}$, then the distinct BARG proofs must be for the same statement. In this case, witness-unambiguity implies that π_{barg} and π'_{barg} must extract to different BARG witnesses at some step i. These BARG witnesses correspond to digested states d_{i-1} and d_{i-1}, each authenticated w.r.t. rt_d, as well as authenticated values for verifying the state transition. Using hash trees for these authentication proofs guarantees that finding any two distinct valid witnesses yields a collision in the underlying hash, which leads to a contradiction in this case.

- Otherwise, it must be the case that $\mathsf{rt}_\mathsf{d} \neq \mathsf{rt}'_\mathsf{d}$. Unfortunately, there is nothing stopping this from being the case in current constructions. For all we know, rt_d may include extraneous bits that aren't checked when verifying authentication proofs. To solve this, our plan is to simultaneously—and uniquely—*prove* that rt_d must be the correctly computed hash root corresponding to digests $\mathsf{d}_0, \ldots, \mathsf{d}_t$. However, we're currently trying to build unique proofs,

so we're seemingly back to square one! At a high level, we avoid this circularity by proving the correct computation of rt_d *relative to* rt_d *itself*. In this way, we won't need an additional hash root that may lead to the same issue again.

Incremental and uniquely verifiable computation of hash roots. We leverage the structure of hash trees to incrementally compute the hash root rt_d while verifying the computation relative to rt_d. To explain this approach, we first recall how hash trees are constructed.

To hash a sequence of values $(x_1, \ldots, x_k)$, you hash each pair of values $(x_1, x_2), \ldots, (x_{k-1}, x_k)$ resulting in a new sequence of values $x'_1, \ldots, x'_{k/2}$. You then recursively hash this new sequence until you're left with only a single value, the hash "root." This naturally yields a binary-tree structure, where the leaf values are the initial inputs $(x_1, \ldots, x_k)$ and the internal nodes are computed by hashing its two children. In this sense, each node in a hash tree is a commitment to all of the leaf values below it. So, to authenticate a particular leaf value or node in a tree, it suffices to give a succinct commitment to all other leaf values, which corresponds to the siblings along the path to the root. For this reason, we often refer to these as *authentication paths*.

Recall that our goal is to use a BARG to prove the correct computation of a hash root rt over values $x_1, \ldots, x_k$, relative to the same root rt. To do this, we will compute rt incrementally and in low space, while simultaneously verifying the state at each step relative to the eventual output rt. So, let $\mathsf{st}_0 = \bot$ denote the initial state. At each step i, we will read the next input x_i and update the previous state st_{i-1} to a new state st_i. We describe our construction as follows:

- **The states.** Each state st_i will be a commitment to the values $(x_1, \ldots, x_i)$ seen so far. Specifically, it will contain the minimal set of nodes in the hash tree that "cover" the leaf values $x_1, \ldots, x_i$. In particular, this will consist of a single node at each level of the binary tree, so st_i will consist of at most $\log k$ nodes at any point in time.

- **Incremental updates.** Starting with a state st_i as above, we compute the next state st_{i+1} given the next value x_{i+1} as follows. We first add x_{i+1} as the next leaf node for the partial tree of st_i. If the sibling of the new leaf node for x_{i+1} is in st_i, we "merge" x_{i+1} into the current state by hashing it with its sibling to compute its parent. We continue to merge nodes until we are left only with the minimal set of nodes covering $x_1, \ldots, x_{i+1}$ in the tree, which is exactly st_{i+1}.

- **Unique proofs w.r.t.** rt**.** As each state st_i simply consists of nodes in the eventual hash tree computing rt, we can prove st_i is the correct state by giving authentication paths for each node in st_i for the eventual full tree with hash root rt. By collision-resistance, this ensures that both the states st_i and their associated proofs π_i are computationally unique.

Finally, we observe that the final state st_k is the minimal node covering inputs $x_1, \ldots, x_k$, which is simply the hash root rt by definition.

Unambiguous SNARG construction. Using the above ideas, we modify our RAM delegation protocol to ensure that $\mathsf{rt}_\mathsf{d} = \mathsf{rt}'_\mathsf{d}$ for any two accepting proofs π, π'.

In our full construction, we additionally compute the incremental states $\mathsf{st}_0, \ldots, \mathsf{st}_t$ for computing rt_d over digests $\mathsf{d}_0, \ldots, \mathsf{d}_t$ corresponding to each step of the computation. For each such state st_i, we also compute the associated proof π_i relative to rt_d that st_i is the correct state on the way to computing rt_d. We then add to the BARG statement at each step i the claim that st_{i-1} transitions

to st_i given the next input d_i on the way to computing rt_d, and use proofs π_{i-1}, π_i to verify that $\mathsf{st}_{i-1}, \mathsf{st}_i$ are the correct states relative to rt_d. In the last step, we verify that st_t is additionally equal to the root rt_d.

The overall proof π still consists of two parts $(\mathsf{rt}_\mathsf{d}, \pi_{\mathsf{barg}})$, where π_{barg} is now for the updated statement as above. To finish the proof of unambiguity from before, suppose that an adversary comes up with two different proofs $\pi \neq \pi'$ for our new construction. We consider the same two cases:

- If $\mathsf{rt}_\mathsf{d} = \mathsf{rt}'_\mathsf{d}$ but $\pi_{\mathsf{barg}} \neq \pi'_{\mathsf{barg}}$, then the BARG proofs are for the same statement, so witness-unambiguity guarantees that we must be able to extract out two different witnesses for some step i. The witnesses now additionally contain states $\mathsf{st}_{i-1}, \mathsf{st}_i$ with proofs π_{i-1}, π_i relative to rt_d, but we argued that these incremental states and proofs are computationally unique relative to rt_d by collision-resistance, leading to a contradiction.
- Otherwise, if $\mathsf{rt}_\mathsf{d} \neq \mathsf{rt}'_\mathsf{d}$, we aren't even guaranteed that π_{barg} and π'_{barg} are for the same statement. However, we will instead rely on soundness of the BARG to argue that, individually, both rt_d and rt'_d must be equal to the hash root rt^*_d corresponding to digests $\mathsf{d}^*_0, \ldots, \mathsf{d}^*_t$ corresponding to the correct computation state at each step starting from d_0. At a high level, we leverage the current proof of soundness that already argues that the digests $\mathsf{d}_i = \mathsf{d}^*_i$ for each statement of the BARG, so it remains to argue that rt^*_d must also be correct. For this, we use a similar argument to show that the incremental states st_i at each step must be equal to the true states st^*_i at each step given that $\mathsf{d}_i = \mathsf{d}^*_i$. It follows that $\mathsf{st}^*_t = \mathsf{rt}^*_\mathsf{d}$, which is also verified to be equal to rt_d. So, rt_d and rt'_d must be equal since they are both equal to rt^*_d, in contradiction.

Unambiguous flexible RAM delegation. We actually show that the above construction can be modified to give a *flexible* RAM delegation scheme as introduced by [KLVW23]. In this setting, the RAM computation has access to an external read-only memory in addition to its working memory. At a high level, the verifier checks computations that read from this external memory given only an extractable hash root rt over the memory. The RAM delegation scheme in this setting is required to satisfy *partial-input soundness*, which stipulates that soundness holds for computations that only read from a subset of external memory relative to the extracted memory values at these locations.

2.2.2 Boosting the Efficiency of Witness-Unambiguous BARGs

The construction of flexible RAM delegation from BARGs above inherits the efficiency—or inefficiency—of the underlying BARG. So, if the underlying BARG verifier runs in time $f(k)$ for batches of k statements, then the corresponding SNARG verifier runs in time $f(t)$ for t-time computations (up to polynomial factors in the security parameter). Following the approach of [KLVW23], we next show how to construct witness-unambiguous BARGs with $\mathsf{polylog}(k)$-efficiency from any unambiguous flexible RAM delegation scheme with sub-linear $o(t)$-efficiency. Furthermore, with slight modifications following the works of [PP22; DGKV22], this transformation gives essentially optimal rate-1 (somewhere extractable) BARGs, where the BARG proof grows with the size of a single witness up to only additive sub-linear terms.

We focus on the case of constructing BARGs for index languages. So, given a RAM program M and witnesses $w_1, \ldots, w_k$, the prover tries to convince the verifier that $M(i, w_i) = 1$ for all $i \in [k]$. The construction of [KLVW23] uses a non-trivial flexible RAM delegation to reduce the above task to a simpler BARG claim as follows:

1. The prover first computes a hash root rt for the external memory consisting of the witnesses $(w_1, \ldots, w_k)$ such that rt is extractable on a single witness w_i.
2. Next, the prover uses the flexible RAM delegation scheme to compute a proof π_i that $M(i, w_i) = 1$ for each $i \in [k]$.
3. The prover defines $k/2$ new witnesses $w'_1, \ldots, w'_{k/2}$ where w'_i consists of two proofs (π_{2i-1}, π_{2i}). The prover then recursively proves the claim that $M'(i, w'_i) = 1$ for each $i \in [k/2]$ where M' verifies both proofs π_{2i-1} and π_{2i}.

In terms of efficiency, it suffices to verify a single witness after $\log k$ levels of recursion. As long as the time to verify two proofs at each level is less than the time to verify a single witness at the previous level, the verification time doesn't blow up, which gives the desired efficiency. Furthermore, using an appropriate rate-1 somewhere extractable hash tree over the memory (as constructed in [PP22; DGKV22]), this transformation yields a rate-1 BARG when starting with an efficient enough RAM delegation scheme (which is in turn implied by the original transformation).

Focusing on one step of the recursion, let's try to prove that the construction satisfies witness-unambiguity when using an unambiguous flexible RAM delegation scheme. The reduction starts with a claim about k witnesses $w_1, \ldots, w_k$ and computes a hash root rt over $(w_1, \ldots, w_k)$ and computes k proofs $\pi_1, \ldots, \pi_k$. For witness-unambiguity, we want to argue that if an adversary computes two distinct sets of values $(\mathsf{rt}, \pi_1, \ldots, \pi_k)$ and $(\mathsf{rt}', \pi'_1, \ldots, \pi'_k)$ that satisfy the recursive claims, then we must be able to extract distinct valid witnesses $w_i \neq w'_i$. We consider the following two cases:

- If $\mathsf{rt} = \mathsf{rt}'$ but there exists a proof $\pi_i \neq \pi'_i$, then it must be the case that π_i and π'_i are accepting for the same statement. However, this can't happen since the proofs are computationally unique, leading to a contradiction.
- Otherwise, it must be the case that $\mathsf{rt} \neq \mathsf{rt}'$. Intuitively, this should imply that there exists some hashed inputs $w_i \neq w'_i$, which we'd extract from $\mathsf{rt}, \mathsf{rt}'$ and be done. However, this is only the case when each rt is actually the correct hash of the external memory $w_1, \ldots, w_k$, so we could simply add a unique proof that this is the case. Unfortunately, we don't have unambiguous RAM delegation with good efficiency, as that's exactly what we're trying to build! Since computing rt takes time $k \cdot |w_i|$, the associated proof would be too large. At a high level, we overcome this issue by devising a new way to prove that rt is computed correctly that fits into the current recursive protocol.

Recursive-friendly proofs for computing hash roots. Recall that flexible RAM delegation gives a way to prove RAM computation relative to a working memory hashed down to some digest d and an external memory hashed down to some root rt. To prove that rt is computed correctly, our key insight is to use *the same hash tree* for the working memory and the external memory. So, if we can prove that a RAM computation has working memory $w_1, \ldots, w_k$ with digest d relative to rt, and rt is computed using the same hash tree over $w_1, \ldots, w_k$, then it must be the case that $\mathsf{d} = \mathsf{rt}$!

Towards this end, consider the RAM computation R_i that reads w_i from memory and puts it into its working memory at the same location by writing each bit of w_i per step. Then, we'll prove that rt is the correct hash root of $w_1, \ldots, w_k$ as follows:

- Initially, let d_0 be the hash root for the all 0s memory $0^{|w_1|}, \ldots, 0^{|w_k|}$. We note that this initial value can be efficiently computed in time logarithmic in k and $|w_i|$ by leveraging the fact

that the leaf values are all 0, so we can save computation in computing the root since the nodes at every level of the tree are the same.

- Assume that d_{i-1} is the hash root over the memory $w_1, \ldots, w_{i-1}, 0^{|w_i|}, \ldots, 0^{|w_k|}$. Compute a flexible RAM delegation proof τ_i that the machine R_i starting with digest d_{i-1} results in d_i after $|w_i|$ steps.
- Given digests $\mathsf{d}_0, \ldots, \mathsf{d}_k$ and proofs $\tau_1, \ldots, \tau_k$, the verifier checks that each proof τ_i accepts for computation R_i transitioning from d_{i-1} to d_i and checks that $\mathsf{d}_k = \mathsf{rt}$.

We've reduced the problem of verifying rt is computed correctly to verifying k different update proofs $\tau_1, \ldots, \tau_k$. However, these proofs are no longer independent as before since the statement $(\mathsf{d}_{i-1}, \mathsf{d}_i)$ for τ_i overlaps with the statement $(\mathsf{d}_i, \mathsf{d}_{i+1})$ for τ_{i+1}. This turns out not to be an issue by having the RAM delegation protocol be extractable on two consecutive witnesses at a time to ensure consistency between consecutive update proofs.

Witness-unambiguous boosting BARG construction. We proceed to describe our witness-unambiguous BARG construction using the method for verifying the hash root above. For simplicity, we just focus on one step of the recursion. So, given a RAM program M and witnesses $w_1, \ldots, w_k$, the prover does the following:

1. The prover computes a hash root rt for the external memory $(w_1, \ldots, w_k)$ such that rt is extractable on *two* witnesses.
2. Next, the prover computes proofs π_i such that $M(i, (w_{i-1}, w_i)) = 1$ for each $i \in [k]$ (setting $w_{i-1} = \perp$ when $i = 1$). In the base case, M simply ignores w_{i-1} and just checks that $M(w_i) = 1$.
3. The prove then computes digests $\mathsf{d}_0, \ldots, \mathsf{d}_k$ and update proofs $\tau_1, \ldots, \tau_k$ such that $\mathsf{d}_k = \mathsf{rt}$ as described above.
4. The prover defines $k/2$ new witnesses $w'_1, \ldots, w'_{k/2}$ where w'_i consists of

$$(\pi_{2i-1}, \pi_{2i}, \mathsf{d}_{2i-1}, \mathsf{d}_{2i}, \tau_{2i-1}, \tau_{2i}).$$

   The prover recursively proves the claim that $M'(i, (w'_{i-1}, w'_i)) = 1$ for each $i \in [k/2]$ relative to external memory $(w'_1, \ldots, w'_{k/2})$ where M' verifies proofs π_{2i-1}, π_{2i} as well as τ_{2i-1}, τ_{2i} relative to $\mathsf{d}_{2i-2}, \mathsf{d}_{2i-1}, \mathsf{d}_{2i}$. Note that M' needs to read from witness w'_{2i-1} to get the value for d_{2i-2} (or directly computes d_0 when $i = 1$).

Now, to prove witness-unambiguity, suppose that the adversary comes up with two distinct sets of values $(\mathsf{rt}, \pi_1, \ldots, \pi_k, \mathsf{d}_0, \ldots, \mathsf{d}_k, \tau_1, \ldots, \tau_k)$ and $(\mathsf{rt}', \pi'_1, \ldots, \pi'_k, \mathsf{d}'_0, \ldots, \mathsf{d}'_k, \tau'_1, \ldots, \tau'_k)$ that satisfy the recursive claims. We show that either we must be able to extract two distinct valid witnesses $w_i \neq w'_i$ via the following case analysis:

- First, if $\mathsf{rt} = \mathsf{rt}'$, then π_i must be equal to π'_i since the proofs are for the same statements with respect to the same external memory. Next, note that $\mathsf{d}_0 = \mathsf{d}'_0$ by construction since they are both independent of the external memory. By unambiguity of the RAM delegation scheme, it follows that τ_1 and τ'_1 prove the same statements according to the same rt value, so the proofs must be the same and result in the same digest $\mathsf{d}_1 = \mathsf{d}'_1$. By the same logic, it continues to hold that $\mathsf{d}_i = \mathsf{d}'_i$ and $\tau_i = \tau'_i$ for all $i \in [k]$, in contradiction.

- Otherwise, it must be the case that $\mathsf{rt} \neq \mathsf{rt}'$. Again, it must be the case that $\mathsf{d}_0 = \mathsf{d}'_0$ since they are independent of external memory. Since $\mathsf{rt} \neq \mathsf{rt}'$ yet $\mathsf{d}_k = \mathsf{rt}$ and $\mathsf{d}'_k = \mathsf{rt}'$ by soundness, it follows that $\mathsf{d}_k \neq \mathsf{d}'_k$. So there must exist some step where $\mathsf{d}_{i-1} = \mathsf{d}'_{i-1}$ yet $\mathsf{d}_i \neq \mathsf{d}'_i$. By partial-input soundness of the RAM delegation scheme, this means that rt and rt′ must extract different witness values $w_i \neq w'_i$ at location i in memory, which are valid by soundness given proofs π_i and π'_i. This establishes witness-unambiguity, as required.

In terms of efficiency, the same argument holds as in the original construction by repeating the above recursive step $\log k$ times. Now, as long as the time to verify *four* proofs at each level is less than the time to verify a single witness at the previous level, the verification time doesn't blow up. To make the construction rate-1, we again rely on a rate-1 somewhere extractable hash tree for computing rt. We observe that in the first level of recursion, we only need to be extractable on one witness. In subsequent levels, we need to be extractable on two witnesses, but we can ensure the witnesses at these levels are much smaller than the original witnesses by using a very succinct RAM delegation scheme (which we just constructed).

2.2.3 Unambiguous Incrementally Verifiable Computation

Armed with rate-1 witness-unambiguous BARGs and unambiguous RAM delegation, we show how to build incrementally verifiable computation (IVC) with computationally unique proofs following the approach of [PP22; DGKV22].

In RAM delegation, recall that the prover's goal is to compute an efficiently verifiable proof $\pi_{0\to t}$ that some initial configuration cf_0 transitions to a new configuration cf_t at t steps. Anyone can verify this proof given access only to digests $\mathsf{d}_0, \mathsf{d}_t$ for the associated configurations.

Suppose someone has spent a long time in order to come up with a proof $\pi_{0\to t}$ that $\mathsf{cf}_0 \to \mathsf{cf}_t$, but you actually want to prove that $\mathsf{cf}_0 \to \mathsf{cf}_{t+1}$. IVC gives a way to "update" proofs efficiently, so given a proof $\pi_{0\to t}$ for some partial computation, you can compute a new proof $\pi_{0\to t+t'}$ for an extension of the computation without re-doing the whole computation. Naively, you could accomplish the above separately computing $\pi_{t\to t+t'}$ proving that $\mathsf{cf}_t \to \mathsf{cf}_{t+t'}$ and setting the new proof $\pi_{0\to t+t'} = (\pi_{0\to t}, \pi_{t\to t+t'})$. This approach, however, does not scale well and results in non-succinct proofs. Suppose, for example, that $\pi_{0\to t}$ was computed one step at a time. Then $|\pi_{0\to t}| \geq t$, and we would have gained nothing.

At a high level, IVC imposes the efficiency constraint that the size of the resulting proof π grows at most poly-logarithmically with the number of updates. For simplicity, we'll assume that the number of updates is at most 2^λ in the security parameter, which stipulates that $|\pi|$ is always bounded by a fixed polynomial in λ. In this work, we consider *unambiguous IVC*, which imposes the additional requirement that it is hard to find two distinct valid proofs $\pi \neq \pi'$ for the same statement. Namely, this is the same unambiguity property as for RAM delegation, just with the efficiency properties of IVC.

Constructing unambiguous IVC. The main idea for building IVC from BARGs is to use the BARG to "merge" proofs together. So, given RAM delegation proofs $\pi_{0\to 1}, \pi_{0\to 2}$, you can use the BARG to compute a new proof $\pi_{0\to 2}$ that proves that there exist accepting proofs $\pi_{0\to 1}$ and $\pi_{1\to 2}$. Furthermore, if the BARG is rate-1, $|\pi_{0\to 2}|$ is essentially the same length as one of the underlying proofs, with some fixed additive sub-linear term. If we just naively merged proofs at every step, this additive term would grow with the number of updates. Instead, we merge

proofs in a tree-like structure so the additive terms only grow logarithmically with the number of updates. Furthermore, to do so, we only need to keep track of a logarithmic number of proofs at any given time.

To formalize this tree-based construction, we define a "level-ℓ" proof as one that proves 2^ℓ steps of computation. Then, a proof $\pi_{0\to t}$ will consist of a single level-ℓ proof for each bit in the binary decomposition of t. To update $\pi_{0\to t}$ one step, we compute a level-0 proof $\pi_{t\to t+1}$ and then merge $\pi_{t\to t+1}$ with the other level-0 proof in $\pi_{0\to t}$, if one exists. We continue to merge the new level-ℓ proof until there are no proofs to merge. Note that this suffices to compute arbitrary t'-size updates to the proof by just performing the single proof update t' times.

In terms of uniqueness, we first observe that the set of proofs in the tree for a proof $\pi_{0\to t}$ is uniquely determined since they correspond to the binary decomposition of t, which is unique. It remains to argue that each such proof is also computationally unique, which we argue inductively on the level ℓ of the proof.

In the base case at level 0, unambiguity follows immediately from the unambiguity of the RAM delegation scheme. For general $\ell > 0$, we inductively assume that level-$(\ell-1)$ proofs are unique. Since a level-ℓ proof is a rate-1 witness-unambiguous BARG of two level-$(\ell-1)$ proofs, it follows that if you could come up with two distinct valid level-ℓ proofs, you must be able to extract out two distinct valid level-$(\ell-1)$ proofs, which contradicts our inductive assumption.

3 Preliminaries

We let $\mathbb{N} = \{1, 2, 3, \ldots\}$ denote the set of natural numbers, and for any $n \in \mathbb{N}$, we write $[n]$ to denote the set $[n] = \{1, \ldots, n\}$. For integers $a, b \in \mathbb{Z}$ with $a \le b$, we write $[a, b]$ to denote the set $\{a, a+1, \ldots, b\}$. For a set Σ, referred to as the *alphabet*, we denote Σ^* the set of strings consisting of 0 or more elements from Σ. We let Σ^n denote the set of n-character strings from Σ. For a string $s \in \Sigma^*$, we let $|s|$ denote the length of s. For any string $s \in \Sigma^*$ and $i \in [|s|]$, let $s[i]$ denote the ith character of s. Unless specified otherwise, we assume that a string s is defined over the binary alphabet $\{0, 1\}$.

We write $\lambda \in \mathbb{N}$ to denote the security parameter. We say that a function $p : \mathbb{N} \to \mathbb{N}$ is in the set $\mathsf{poly}(\lambda)$ and is *polynomially-bounded* if there exists a constant c and an index $i \in \mathbb{N}$ such that $p(\lambda) \le \lambda^c$ for all $\lambda \ge i$. We say that a function $\mu : \mathbb{N} \to [0, 1]$ is in the set $\mathsf{negl}(\lambda)$ and is negligible if for every constant $c > 0$ there exists $i \in \mathbb{N}$ such that $\mu(\lambda) \le \lambda^{-c}$ for all $\lambda \ge i$.

We use PPT to denote the acronym *probabilistic, polynomial time*. A uniform algorithm A is implemented by a program with a fixed (constant-size) description length. A non-uniform algorithm A consists of a sequence of algorithms $\{A_\lambda\}_{\lambda\in\mathbb{N}}$, one for each security parameter λ; we assume for simplicity that A_λ always receives 1^λ as its first input. When the security parameter is clear from context, we may write $A(\cdot)$ instead of $A_\lambda(\cdot)$ for simplicity. A non-uniform PPT algorithm is one where the description size of A_λ as a function of λ is in $\mathsf{poly}(\lambda)$.

For a distribution X, we write $x \leftarrow X$ to denote the process of sampling a value x from the distribution X. For a set $\mathcal{X}$, we use $x \leftarrow \mathcal{X}$ to denote the process of sampling a value x from the uniform distribution over $\mathcal{X}$. We use $x = A(\cdot)$ to denote the output of a deterministic algorithm and $x \leftarrow A(\cdot)$ to denote the output of a randomized algorithm. For a randomized algorithm, we write $x = A(\cdot; r)$ to denote the deterministic output given sequential access to the random coins r. We write $x := y$ to denote the assignment of value y to x. For a distribution D, we define $\mathsf{supp}(D)$ to denote the support of the distribution D.

3.1 Lattice-based Cryptography

We next overview the necessary preliminaries from lattice-based cryptography that we use in this work.

Assumptions. We formally define the learning-with-error (LWE) assumption and short integer solution (SIS) problem.

Definition 3.1 (Learning-With-Errors Assumption). The (decisional) learning with errors (LWE) assumption with parameters $n = n(\lambda), m = m(\lambda), q = q(\lambda), \chi \in \mathcal{D}(\mathbb{Z}_q)$, denoted by $\mathsf{LWE}_{n,m,q,\chi}$, states that the distribution ensembles $\{(\mathbf{A}, \mathbf{b})\}_\lambda$ and $\{(\mathbf{A}, \mathbf{r})\}_\lambda$ are computationally indistinguishable, where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{b}^T = s^T\mathbf{A} + \mathbf{e}^T \pmod q, \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \chi^m$ and $\mathbf{r} \leftarrow \mathbb{Z}_q^m$. ◇

Various parameter settings are commonly used for LWE, but we note that the typical parameters for LWE have $q = \mathsf{poly}(n)$, $m = \Theta(n \log q)$, and χ defined to be the uniform distribution on $[-B, B] \subseteq \mathbb{Z}_q$ for $B = \mathsf{poly}(\lambda)$ but smaller than q.

Definition 3.2 (Short Integer Solution Problem). The $\mathsf{SIS}_{n,m,q,\beta}$ problem is: given a uniformly random matrix $A \in \mathbb{Z}_q^{n \times m}$, find a non-zero integral vector $\mathbf{z} \in \mathbb{Z}^m$ such that $A \cdot \mathbf{z} \equiv 0 \mod q$ and $||\mathbf{z}|| \leq \beta$. ◇

The typical parameter setting for SIS is $m = \Theta(n \log q), \beta = \mathsf{poly}(\lambda)$ and $q \geq \beta \cdot \tilde{O}(\sqrt{n})$.

Lattice gadgets. We overview some notation for typical gadgets used in lattice-based cryptography. We note that these are taken primarily from [PS19].

For a positive integer modulus q, let $\ell = \lceil \log q \rceil$. The gadget vector [MP12] over $\mathbb{Z}_q$ is defined as

$$\mathbf{g}^T = (1, 2, 4, \ldots, 2^{\ell-1}) \in \mathbb{Z}_q^\ell.$$

For every $u \in \mathbb{Z}_q$, there is an efficiently computable binary vector $\mathbf{g}^{-1}[u] \in \{0,1\}^\ell$ such that $\langle \mathbf{g}, \mathbf{g}^{-1}[u] \rangle = u \pmod q$. Specifically, $\mathbf{g}^{-1}[u]$ corresponds to the binary representation of the distinguished representative of u in $\{0, 1, \ldots, q-1\}$.

For a dimension n, the gadget matrix is defined as

$$\mathbf{G}_n = \mathbf{I}_n \otimes \mathbf{g}^T \in \mathbb{Z}_q^{n \times m},$$

where $m = n\ell$. We often drop the subscript n when it is clear from context. Similarly to above, we define the function $\mathbf{G}^{-1} = (\mathbf{I} \otimes \mathbf{g}^{-1}) : \mathbb{Z}_q^n \to \{0,1\}^m$, which applies $\mathbf{g}^{-1}$ to each coordinate and appends the results. We note that for every $\mathbf{u} \in \mathbb{Z}_q^n$,

$$\mathbf{G} \cdot \mathbf{G}^{-1}[\mathbf{u}] = \mathbf{u}.$$

Linear CRHF. Our construction requires linear collision-resistant hash functions (CRHF), whose existence can be implied by standard lattice assumptions.

Definition 3.3 (Linear CRHF). We say a hash family $\mathcal{H} = \{\mathcal{H}_\lambda\}_\lambda$ parameterized by value bounds M, s and input length n is a linear collision-resistant hash function (CRHF) family if every $h_\lambda \in \mathcal{H}_\lambda$ is a linear function, i.e. it can be represented as a matrix in $\mathbb{Z}^{\mathsf{poly}(\lambda) \times n}$, where each entry is in $[0, M]$ for some bound $M < 2^\lambda$, and $\mathcal{H}$ is collision resistant for vectors in $[0, s]^n$. Furthermore, a random hash key can be generated in time $\mathsf{poly}(\lambda, n, \log s)$. ◇

The existence of linear CRHF is immediately implied by the hardness of the SIS problem.

Fact 3.4. For parameters $n(\lambda), q(\lambda)$ and $s(\lambda)$, if $\mathsf{SIS}_{\mathsf{poly}(\lambda),n,q,s\sqrt{n}}$ holds, then there exists a linear CRHF with matrix representation in $[0, q-1]^{\mathsf{poly}(\lambda)\times n}$ and is collision resistant for vectors in $[0, s]^n$. ◇

Furthermore, we can always transform a linear CRHF into a binary linear CRHF by applying the lattice gadget $\mathbf{G}^{-1}$ to it, which gives the following lemma.

Lemma 3.5. Assume that $M \leq 2^\lambda$. If CRHF family $\mathcal{H} = \left\{h \in [0, M]^{p(\lambda)\times n}\right\}$ is a linear CRHF family collision resistant for input vectors in $[0, s]^n$, then the linear CRHF family $\mathcal{H}' = \left\{\mathbf{G}^{-1}[h] \in [0,1]^{p(\lambda)\lambda\times n} : h \in \mathcal{H}\right\}$ is also collision resistant for input vectors in $[0, s]^n$. ◇

3.2 Linear PCPs

We next define linear probabilistically checkable proofs (PCPs), which were introduced by [ALMSS98] and further formalized and studied in [IKO07; BCIOP13; GGPR13]. At a high level, linear PCPs are proofs specified by a linear function. In this paper, we focus on linear PCPs defined over the integers $\mathbb{Z}$ for simplicity.

Definition 3.6 (Linear PCP). Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. A linear PCP for $\mathcal{R}$ over $\mathbb{Z}$ with proof length ℓ, knowledge soundness error ε, and query complexity t is a pair of algorithms $(\mathcal{P}_{\mathsf{LPCP}}, \mathcal{V}_{\mathsf{LPCP}})$ with the following syntax and efficiency.

- $\mathcal{P}^{\mathsf{LPCP}}(x, w) \to \pi$. This is a deterministic polynomial-time algorithm that on input $(x, w) \in \mathcal{R}$ outputs a proof $\pi \in \mathbb{Z}^\ell$.
- $\mathcal{V}^{\mathsf{LPCP}}(x, \pi, r) \to b$. This is a deterministic algorithm that takes as input an instance x, a proof π, and randomness r, and outputs a bit b indicating whether to accept or reject the proof.

We require that $(\mathcal{P}_{\mathsf{LPCP}}, \mathcal{V}_{\mathsf{LPCP}})$ satisfy the following properties:

- **Completeness.** For any $(x, w) \in \mathcal{R}$, take $\pi = \mathcal{P}^{\mathsf{LPCP}}(x, w)$, we have
$$\Pr_r[\mathcal{V}^{\mathsf{LPCP}}(x, \pi, r) = 1] = 1.$$
- **Proof of Knowledge.** There is a knowledge extraction algorithm Ext such that for any (x, π) satisfying
$$\Pr_r[\mathcal{V}^{\mathsf{LPCP}}(x, \pi, r) = 1] \geq \varepsilon,$$
it holds that $\mathsf{Ext}(x, \pi)$ computes a witness w such that $(x, w) \in \mathcal{R}$ in expected time $\mathsf{poly}(|x|, 1/\varepsilon)$.
- **Non-adaptive Queries.** The verifier $\mathcal{V}^{\mathsf{LPCP}}(x, \pi, r)$ consists of a query algorithm Q^{LPCP} and a decision algorithm D^{LPCP}. The query algorithm Q^{LPCP} only takes randomness r and outputs queries $q_1, \ldots, q_t \in \mathbb{Z}^\ell$, which are independent of x, as well as a short state information u. The decision algorithm D^{LPCP} takes as input the statement x, the answers $\langle \pi, q_1 \rangle, \ldots, \langle \pi, q_t \rangle \in \mathbb{Z}$ to Q^{LPCP}'s queries, and the state information. D^{LPCP} outputs the one-bit verdict.

- **Efficient Verification.** The query algorithm Q^{LPCP} only uses $O(\ell)$ bits of randomness, and only makes a constant number of queries. ◇

We use the fact that there is a linear PCP for the universal language for Turing machine satisfiability, which we define as follows.

Definition 3.7. The language TMSAT consists of instances of the form $x = (M, z, T)$, where M is the description of a Turing machine, z is an input string, and T is a running time. An instance $x = (M, z, T)$ is in TMSAT if there exists a string w such that $M(z, w)$ accepts within T steps. We assume that $T \geq |x|$ if x is in TMSAT. ◇

Let $\mathcal{R}$ be the corresponding relation for TMSAT, i.e. $(x = (M, z, T), w) \in \mathcal{R}$ if $M(z, w)$ accepts within T steps. We leverage the following particular linear PCP for $\mathcal{R}$, which follows from the Hadamard-based PCP of [ALMSS98]. We formalize this construction in Appendix B for completeness.

Lemma 3.8. There exists a linear PCP system for TMSAT over $\mathbb{Z}$, with proof length $\mathsf{poly}(T)$, state information length $\mathsf{poly}(|x|)$ and constant knowledge soundness error ε. Furthermore, it satisfies these additional properties:

- **Proof Uniqueness w.r.t. Witness.** For any instance x and two accepting proofs π, π' such that
$$\Pr_r[\mathcal{V}^{\mathsf{LPCP}}(x, \pi, r) = 1] \geq \varepsilon, \quad \Pr_r[\mathcal{V}^{\mathsf{LPCP}}(x, \pi', r) = 1] \geq \varepsilon,$$
it holds that $w \neq w'$ for $w = \mathsf{Ext}(\pi)$ and $w' = \mathsf{Ext}(\pi')$.
- **Binary Accepting Proofs.** For any instance x, if a proof π satisfies
$$\Pr_r[\mathcal{V}^{\mathsf{LPCP}}(x, \pi, r) = 1] \geq \varepsilon,$$
then it must be the case that $\pi \in \{0,1\}^\ell$.
- **Binary Queries.** For any randomness r, $Q^{\mathsf{LPCP}}(r)$ specifies only binary queries in $\{0,1\}^\ell$. ◇

3.3 Correlation Intractable Hash Function

The following preliminaries on correlation intractable hash functions are due to [HLR21]. Throughout this section, let $R \subseteq X \times Y^t$ be a binary relation.

Definition 3.9 (Product Relation). We say that R is a product relation if for every x, the set $R(x) = \{y : (x, y) \in R\} \subseteq Y^t$ has a decomposition
$$R(x) = S_1 \times S_2 \times \ldots \times S_t,$$
(where $S_1, \ldots, S_t$ may depend on x). ◇

Definition 3.10 (Efficient Product Verifiability). We say that a relation R is efficiently product verifiable if there is a polynomial-size circuit C such that, on every input x with some corresponding sets $(S_1, \ldots, S_t)$ corresponding to x, it holds that $C(x, y, i) = 1$ if and only if $y \in S_i$. ◇

Definition 3.11 (Product Sparsity). We say that a product relation R has product sparsity ρ if for every input x, the sets $S_1, \ldots, S_t$ have size at most $\rho|Y|$. ◇

Definition 3.12 (Hash Family). A hash family $\mathcal{H}$ with input set X_λ, output set Y_λ, is associated with two algorithms $(\mathcal{H}.\mathcal{G}, \mathcal{H}.\mathsf{Hash})$, such that:

- $\mathcal{H}.\mathcal{G}$ is a PPT algorithm that takes as input a security parameter 1^λ and outputs a key k.
- $\mathcal{H}.\mathsf{Hash}$ is a polynomial time computable (deterministic) algorithm that takes as input a key $k \leftarrow \mathcal{H}.\mathcal{G}(1^\lambda)$ and an element $x \in X_\lambda$ and outputs an element $y \in Y_\lambda$. ◇

Definition 3.13 (Correlation-Intractability). For a hash family $\mathcal{H} : \{h_\lambda : X_\lambda \to Y_\lambda\}_\lambda$ and a relation ensemble $R = \{R_\lambda \subseteq X_\lambda \in Y_\lambda\}$, we say $\mathcal{H}$ is correlation intractable for R if for every non-uniform polynomial-size adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$ such that for every $R_\lambda \in R$,

$$\Pr\left[(x, \mathcal{H}.\mathsf{Hash}(k,x)) \in R_\lambda \;:\; \begin{array}{l} k \leftarrow \mathcal{H}.\mathcal{G}(1^\lambda) \\ x \leftarrow \mathcal{A}(k) \end{array} \right] \leq \mu(\lambda).$$

◇

Theorem 3.14 (Theorem 5.5 of [HLR21]). Let $R = R_\lambda \subseteq X_\lambda \times Y_\lambda^{t_\lambda}$ be an ensemble of product relations that are time-$T(\lambda)$ product-verifiable as in Definition 3.10 (where $\log|X_\lambda|, T(\lambda), t_\lambda \in \mathsf{poly}(\lambda)$) with product sparsity at most $1 - \varepsilon$ for $\varepsilon \geq \lambda^{-O(1)}$. Then, if $t_\lambda \geq \lambda/\varepsilon$, there exists a hash family $\mathcal{H} = \left\{\mathcal{H}_\lambda : X_\lambda \to Y_\lambda^{t_\lambda}\right\}_{\lambda \in \mathbb{N}}$ that is correlation intractable for R under the LWE assumption. Moreover, $\mathcal{H}$ depends only on $(X_\lambda, Y_\lambda, T_\lambda, t_\lambda, \varepsilon)$ and can be evaluated in time $\mathsf{poly}(\log|X|, t, T)$. ◇

3.4 Batch Arguments

A batch argument (BARG) system for an NP language L enables proving that k statements are true with communication sub-linear in k. There are many BARG variants which are known to be existentially equivalent under mild computational assumptions (see, e.g., [CJJ22; KVZ21; KLVW23]). In this work, for simplicity, we make use of (publicly verifiable and non-interactive) somewhere extractable batch arguments for the index language where the batched statements of size k are defined relative to a fixed machine M and each statement takes as input only an index $i \in [k]$. For a Turing or RAM machine M, let $\mathcal{U}_M$ be the language that contains (x, T) if and only if the machine M accepts x in T steps. Without loss of generality, we assume for $(x, T) \in \mathcal{U}_M$, we must have $T \geq |M|$.

Definition 3.15 (Somewhere Exctractable Batch Arguments for NP). Let $L : \mathbb{N}^2 \to \mathbb{N}$. An (publicly verifiable and non-interactive) L-succint somewhere extractable batch argument (seBARG) for the index language is a tuple of algorithms $(\mathcal{G}_{\mathsf{barg}}, \mathcal{P}_{\mathsf{barg}}, \mathcal{V}_{\mathsf{barg}}, \mathsf{Ext}_{\mathsf{barg}})$ with the following syntax:

- $\mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^m, 1^T, k, i^*) \to (\mathsf{crs}, \mathsf{td})$. This is a randomized algorithm that takes as input a security parameter 1^λ, a bound on the witness length 1^m, a bound on the running time 1^T, a number of statement k, and an index $i^* \in [k]$, and outputs a common reference string crs along with a trapdoor td.
- $\mathcal{P}_{\mathsf{barg}}(\mathsf{crs}, M, (w_1, \ldots, w_k)) \to \pi$. This is a deterministic algorithm that takes as input a common reference string crs, a machine M, and a sequence of witnesses $(w_1, \ldots, w_k)$, and outputs a proof π.

- $\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) \to b$. This is a deterministic algorithm that takes as input a common reference string crs, a machine M, and a proof π, and outputs a bit b indicating whether to accept or reject the proof.
- $\mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi) \to w^*$. This is a deterministic algorithm that takes as input a trapdoor td, a machine M, and a proof π, and outputs a single witness w^*.

We require that $(\mathcal{G}_{\mathsf{barg}}, \mathcal{P}_{\mathsf{barg}}, \mathcal{V}_{\mathsf{barg}}, \mathsf{Ext}_{\mathsf{barg}})$ satisfy the following properties:

- **Completeness.** For any $\lambda, k, m, T \in \mathbb{N}$, $i^* \in [k]$, machine M, and sequence of witnesses $(w_1, \ldots, w_k) \in (\{0,1\}^m)^k$ such that $M(i, w_i) = 1$ within T steps for every $i \in [k]$, it holds that

$$\Pr\left[\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 \;:\; \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^m, 1^T, k, i^*), \\ \pi = \mathcal{P}_{\mathsf{barg}}(\mathsf{crs}, M, (w_1, \ldots, w_k)) \end{array} \right] = 1.$$

- **L-Succinctness.** In the completeness experiment, there exists a polynomial q such that the following holds for $k, m, T \leq 2^\lambda$:
  - $\mathcal{G}_{\mathsf{barg}}$ runs in time $L(k, \lambda) \cdot q(T)$ and outputs a common reference string of size $|\mathsf{crs}| \leq L(k, \lambda) \cdot q(T)$.
  - $\mathcal{P}_{\mathsf{barg}}$ runs in time $q(\lambda + k + T)$ and outputs a proof of size $|\pi| \leq L(k, \lambda) \cdot q(T)$.
  - $\mathcal{V}_{\mathsf{barg}}$ tuns in time $L(k, \lambda) \cdot q(T)$.
- **Index hiding.** For any non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$ and polynomials k, m, T and sequence of indices $\{(i_{0,\lambda}, i_{1,\lambda}) \in [k(\lambda)]^2\}_{\lambda \in \mathbb{N}}$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[A_\lambda(\mathsf{crs}) = b \;:\; \begin{array}{l} b \leftarrow \{0,1\}, \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^{m(\lambda)}, 1^{T(\lambda)}, k(\lambda), i_{b,\lambda}) \end{array} \right] \leq \frac{1}{2} + \mu(\lambda).$$

- **Somewhere argument of knowledge.** For any non-uniform algorithm $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$, polynomials k, m, T, and sequence of indices $\{i^*_\lambda \in [k(\lambda)]\}_{\lambda \in \mathbb{N}}$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 \\ \wedge\, M(i^*, w^*) \neq 1 \text{ within } T(\lambda) \text{ steps} \end{array} \;:\; \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^{m(\lambda)}, 1^{T(\lambda)}, k(\lambda), i^*_\lambda) \\ (M, \pi) = A_\lambda(\mathsf{crs}) \\ w^* \leftarrow \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi) \end{array} \right] \leq \mu(\lambda).$$

$\diamond$

We define the additive overhead of a BARG, and as a special case, rate-1 BARGs, as follows.

Definition 3.16 (Additive Overhead of BARGs). Let $\alpha \colon \mathbb{N}^2 \to \mathbb{N}$. We say that a BARG has additive overhead α if in the completeness experiment $|\pi| - m \leq \alpha(\lambda, m)$. In particular, we say that a BARG is rate-1 if there exists a polynomial q and a constant $c \in \mathbb{N}$ such that for any $\lambda \in \mathbb{N}$, it has additive overhead $\alpha(\lambda, m) \leq c \cdot m/\lambda + q(\lambda)$. $\diamond$

Remark 1 (Extracting BARGs on Multiple Indices). Given an seBARG, one can naturally extend the definition of the key generation algorithm $\mathcal{G}_{\mathsf{barg}}$ to take as input an index *set* $I \subset [k]$, as opposed to a single index. $\mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^m, 1^T, k, I)$ will simply run $\mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^m, 1^T, k, i)$ for every $i \in I$. The

prover algorithm $\mathcal{P}_{\mathsf{barg}}$, given a crs that encodes the $|I|$ indices, will simply generate $|I|$ proofs (one for each crs), and the verifier will check these $|I|$ proofs independently. In this situation, the seBARG trapdoor $\mathsf{td} = (\mathsf{td}_1, \ldots, \mathsf{td}_{|I|})$ is written in $|I|$ components, and the scheme inherits a strong form of index hiding: given the CRS crs, the jth index i_j remains hidden even given all $|I| - 1$ other trapdoors $\mathsf{td}_{i_{j'}}$. ◇

3.5 Hash Trees

We define an unambiguous hash tree following [KP16]. At a high level, a hash tree is a hash function that provides verifiable local openings, which can be instantiated from any collision-resistant hash function [Mer90]. For such constructions, it also holds that the opening proofs themselves are computationally unique, so we refer to this as an unambiguous hash tree.

Definition 3.17. An *unambiguous hash tree* consists of a tuple of algorithms $(\mathcal{G}_{\mathsf{ht}}, \mathcal{H}_{\mathsf{ht}}, \mathcal{P}_{\mathsf{ht}}, \mathcal{V}_{\mathsf{ht}})$ with the following syntax:

- $\mathcal{G}_{\mathsf{ht}}(1^\lambda, S) \to \mathsf{hk}$. This is a randomized algorithm that takes as input a security parameter λ, and a space bound S, and outputs a hash key hk. We implicitly assume that hk includes $1^\lambda, S$.
- $\mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, D) \to \mathsf{rt}$. This is a deterministic algorithm that on input a hash key hk and a database $D \in \{0,1\}^S$ outputs a root rt.
- $\mathcal{P}_{\mathsf{ht}}(\mathsf{hk}, D, i) \to \rho$. This is a deterministic algorithm that on input a hash key hk, a database $D \in \{0,1\}^S$, an index $i \in [S]$, outputs an opening proof ρ. When a set $I \subseteq [S]$ is given as input, we interpret this as providing proofs for (i, j) for all $j \in [m]$.
- $\mathcal{V}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{rt}, i, v, \rho) \to b$. This is a deterministic algorithm that on input a hash key hk, a hash tree root rt, an index $i \in [S]$, a bit $v \in \{0,1\}$, and an opening proof ρ, outputs a bit $b \in \{0,1\}$ indicating whether to accept or reject the opening proof. When a set $I \subseteq [S]$ and value $v \in \{0,1\}^{|I|}$ is given as input, we interpret this as verifying proofs for $v[j]$ and index i for all $j \in [|I|]$ and $i \in I$.

We require that $(\mathcal{G}_{\mathsf{ht}}, \mathcal{H}_{\mathsf{ht}}, \mathcal{P}_{\mathsf{ht}}, \mathcal{V}_{\mathsf{ht}})$ satisfy the following properties:

- **Opening Completeness.** For any $\lambda, S \in \mathbb{N}$, $i \in [S]$, database $D \in \{0,1\}^S$, it holds that

$$\Pr\left[\mathcal{V}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{rt}, i, D[i], \rho) = 1 \;:\; \begin{array}{l} \mathsf{hk} \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, S) \\ \mathsf{rt} = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, D) \\ \rho = \mathcal{P}_{\mathsf{ht}}(\mathsf{hk}, D, i) \end{array}\right] = 1.$$

- **Efficiency.** There exists a polynomial q such that the following holds in the opening completeness and extraction correctness experiment above for $S \leq 2^\lambda$:
  - $\mathcal{G}_{\mathsf{ht}}$ runs in time $q(\lambda)$.
  - $\mathcal{H}_{\mathsf{ht}}$ runs in time $S \cdot q(\lambda)$. Additionally, if $D = x||0^{S-|x|}$, there exists an algorithm that computes $\mathcal{H}_{\mathsf{ht}}$ in time $|x| \cdot q(\lambda)$.
  - $\mathcal{P}_{\mathsf{ht}}$ runs in time $S \cdot q(\lambda)$ and $|\rho| \leq q(\lambda)$.
  - $\mathcal{V}_{\mathsf{ht}}$ runs in time $q(\lambda)$. ◇

- **Opening Unambiguity.** For any non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda\in\mathbb{N}}$ and polynomials S, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l}(\rho, v) \neq (\rho', v') \\ \wedge\ 1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{rt}, i, v, \rho) \\ \wedge\ 1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{rt}, i, v', \rho')\end{array} : \begin{array}{l}\mathsf{hk} \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, S(\lambda)) \\ (\mathsf{rt}, i, v, v', \rho, \rho') \leftarrow A_\lambda(\mathsf{hk})\end{array}\right] \leq \mu(\lambda).$$

Locally updatable hash tree. A hash tree is locally updatable if there is an efficient RAM program that can compute verifiable local updates to a hash tree in time that scales only with the size of the updates and not the whole tree. We model this by associating a hash tree as above with additional stateful RAM algorithms for hashing, reading from, and writing to the hash tree. We refer to [KP16; EFKP20b] for constructions of hash trees from any collision-resistant hash function with the properties below.

Definition 3.18. A *locally updatable hash tree* is a hash tree $(\mathcal{G}_{\mathsf{ht}}, \mathcal{H}_{\mathsf{ht}}, \mathcal{P}_{\mathsf{ht}}, \mathcal{V}_{\mathsf{ht}})$ with additional algorithms $(\mathsf{Rd}_{\mathsf{ht}}, \mathsf{Wrt}_{\mathsf{ht}}, \mathsf{VerWrt}_{\mathsf{ht}})$ with the following syntax:

- $\mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, D) \to (\mathsf{rt}, \mathsf{tree})$. The hashing algorithm optionally outputs memory tree for a RAM program.
- $\mathsf{Rd}_{\mathsf{ht}}^{\mathsf{tree}}(i) \to \rho$. This is a deterministic read-only RAM program that reads from tree in memory, takes as input an index $i \in S$, and outputs and opening proof ρ.
- $\mathsf{Wrt}_{\mathsf{ht}}^{\mathsf{tree}}(i, v) \to (\mathsf{rt}', \tau)$. This is a deterministic RAM program that accesses tree in memory, takes as input an index $i \in S$ and a bit v, and outputs a new root rt' with update proof τ. When a set $I \subseteq [S]$ is given as input with value $v \in \{0,1\}^{|I|}$, we interpret this as sequentially updating locations $i \in I$ with values $v[j]$ for each $j \in [|I|]$.
- $\mathsf{VerWrt}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{rt}, i, v, \mathsf{rt}', \tau) \to b$. This is a deterministic algorithm that on input a hash key hk, an old hash tree root rt, an index $i \in [S]$, a bit v, a new hash tree root rt', and an update proof τ, outputs a bit $b \in \{0,1\}$ indicating whether to accept or reject the update proof. When a subset $I \subseteq [S]$ is given as input, we interpret this as verifying $v[j] \in \{0,1\}$ for each $j \in [|I|]$.

We require that the algorithms above satisfy the following additional properties:

- **Completeness of Reads and Writes.** For any $\lambda, S \in \mathbb{N}$, $i \in [S]$, database $D \in \{0,1\}^S$, the following holds with probability 1 over $\mathsf{hk} \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, S)$:
  - For $(\cdot, \mathsf{tree}) = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, D)$, it holds that $\mathsf{Rd}_{\mathsf{ht}}^{\mathsf{tree}}(i) = \mathcal{P}_{\mathsf{ht}}(\mathsf{hk}, D, i)$.
  - For $(\cdot, \mathsf{tree}) = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, D)$, let $(\mathsf{rt}', \tau) = \mathsf{Wrt}_{\mathsf{ht}}^{\mathsf{tree}}(i, v)$ and tree' be the new RAM memory. It holds that $\mathsf{VerWrt}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{rt}, \mathsf{rt}', i, v, \tau) = 1$ and $(\mathsf{rt}', \mathsf{tree}') = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, D')$ where D' is the database D with value v written at location $i \in [S]$. ◇
- **Efficiency.** There exists a polynomial q such that in the completeness of reads and writes experiment above, for $S \leq 2^\lambda$, it holds that $\mathsf{Rd}_{\mathsf{ht}}^{\mathsf{tree}}$, $\mathsf{Wrt}_{\mathsf{ht}}^{\mathsf{tree}}$, and $\mathsf{VerWrt}_{\mathsf{ht}}$ run in time $q(\lambda)$ and $|\tau| \leq q(\lambda)$. Furthermore, on databases of the form $D = x \| 0^{S-|x|}$, the size of memory $|\mathsf{tree}| \leq |x| \cdot q(\lambda)$.

- **Update Unambiguity.** For any non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda\in\mathbb{N}}$ and functions S such that $S(\lambda) \le 2^\lambda$ for all $\lambda \in \mathbb{N}$, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} (\mathsf{rt},\tau) \neq (\mathsf{rt}',\tau') \\ \wedge\ 1 = \mathsf{VerWrt}_{\mathsf{ht}}(\mathsf{hk},\mathsf{rt}^*,i,v,\mathsf{rt},\tau) \\ \wedge\ 1 = \mathsf{VerWrt}_{\mathsf{ht}}(\mathsf{hk},\mathsf{rt}^*,i,v,\mathsf{rt}',\tau') \end{array} : \begin{array}{l} \mathsf{hk} \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, S(\lambda)) \\ (\mathsf{rt}^*,i,v,\mathsf{rt},\mathsf{rt}',\tau,\tau') \leftarrow A_\lambda(\mathsf{hk}) \end{array}\right] \le \mu(\lambda).$$

Fully local somewhere extractable hash trees. We next define somewhere extractable hash tree (SEH) adopted from [DGKV22]. At a high level, an SEH is a hash tree where the root rt is extractable on a set of input values. For notational convenience, we think of the database $D \in \{0,1\}^S$ as consisting of k inputs $(x_1,\ldots,x_k)$ of size m, where the SEH is extractable at some input $i^* \in [k]$. Note that this extraction requirement necessitates that $|\mathsf{rt}| \ge m$.

To account for this inefficiency, [DGKV22] define a fully local SEH (flSEH) as one where $\mathcal{H}_{\mathsf{ht}}$ computes an extractable hash value h and then an additional algorithm $\mathcal{D}_{\mathsf{ht}}$ produces the root rt of fixed $\mathsf{poly}(\lambda)$ size. For simplicity of presentation, we only define an flSEH below and note that a (non-fully local) SEH is one where $\mathsf{h} = \mathsf{rt}$ and $|\mathsf{rt}|$, $|\rho|$ and the running time of $\mathcal{V}_{\mathsf{seh}}$ are in $\mathsf{poly}(\lambda + m)$.

Definition 3.19. A *fully local somewhere extractable hash tree (flSEH)* is a hash tree $(\mathcal{G}_{\mathsf{ht}},\mathcal{H}_{\mathsf{ht}},\mathcal{P}_{\mathsf{ht}},\mathcal{V}_{\mathsf{ht}})$ with additional algorithms $(\mathcal{D}_{\mathsf{ht}},\mathcal{E}_{\mathsf{ht}})$ with the following modified syntax:

- $\mathcal{G}_{\mathsf{ht}}(1^\lambda, m, k, i^*) \to (\mathsf{hk},\mathsf{td})$. The key generation algorithm additionally takes as input an index i^* and outputs a trapdoor td. We implicitly assume that td includes hk and i^*.
- $\mathcal{H}_{\mathsf{ht}}(\mathsf{hk},(x_1,\ldots,x_k)) \to \mathsf{h}$. The hashing algorithm outputs an intermediate extractable hash value h rather than the root rt.
- $\mathcal{D}_{\mathsf{ht}}(\mathsf{hk},\mathsf{h}) \to \mathsf{rt}$. The digest algorithm produces the root rt given the hash key hk and a hash value h.
- $\mathcal{E}_{\mathsf{ht}}(\mathsf{td},\mathsf{h}) \to x$. The extraction algorithm is a deterministic algorithm that on input a trapdoor td and a hash value h outputs a string $x \in \{0,1\}^m$.

We require that $(\mathcal{G}_{\mathsf{ht}},\mathcal{H}_{\mathsf{ht}},\mathcal{D}_{\mathsf{ht}},\mathcal{P}_{\mathsf{ht}},\mathcal{V}_{\mathsf{ht}},\mathcal{E}_{\mathsf{ht}})$ satisfy the following additional properties:

- **Extraction Correctness.** For any $\lambda, m, k \in \mathbb{N}$, $i^*, i \in [k]$, $j \in [m]$, inputs $(x_1,\ldots,x_k) \in (\{0,1\}^n)^k$, it holds that

$$\Pr\left[\mathcal{E}_{\mathsf{ht}}(\mathsf{td},\mathsf{h}) = x_{i^*} \ :\ \begin{array}{l} (\mathsf{hk},\mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, m, k, i^*) \\ \mathsf{h} = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk},(x_1,\ldots,x_k)) \\ \mathsf{rt} = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk},\mathsf{h}) \end{array}\right] = 1.$$

- **Efficiency.** There exists a polynomial q such that the following holds in the extraction correctness experiment above for $m, k \le 2^\lambda$:
  - $\mathcal{H}_{\mathsf{ht}}$ outputs a hash value $|\mathsf{h}| \le q(\lambda + m)$.
  - $\mathcal{D}_{\mathsf{ht}}$ runs in time $q(\lambda + m)$ and output a root of size $|\mathsf{rt}| \le q(\lambda)$.

– $\mathcal{E}_{\mathsf{ht}}$ runs in time $q(\lambda + m)$. ◇

- **Index Hiding.** For any non-uniform PPT algorithm $A = \{A_\lambda\}_{\lambda\in\mathbb{N}}$, polynomials m, k, and sequences of indices $\{(i_{0,\lambda}, i_{1,\lambda}) \in [k(\lambda)]^2\}_{\lambda\in\mathbb{N}}$, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{ll} & b \leftarrow \{0,1\} \\ b' = b \ : & (\mathsf{hk}_b, \cdot) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, m(\lambda), k(\lambda), i_{b,\lambda}) \\ & b' \leftarrow A_\lambda(\mathsf{hk}_b) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

- **Somewhere Statistically Binding w.r.t. Opening.** For any non-uniform (potentially unbounded) algorithm $A = \{A_\lambda\}_{\lambda\in\mathbb{N}}$, polynomials m, k, and sequences of indices $\{i^*_\lambda \in [k(\lambda)]\}_{\lambda\in\mathbb{N}}$, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} 1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{rt}, i^*_\lambda \cdot m(\lambda) + j, v, \rho) \\ \wedge\ j \in [m(\lambda)] \\ \wedge\ v \neq x[j] \end{array} : \begin{array}{l} (\mathsf{hk}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, m(\lambda), k(\lambda), i^*_\lambda) \\ (\mathsf{h}, v, j, \rho) \leftarrow A_\lambda(\mathsf{hk}) \\ \mathsf{rt} = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{h}) \\ x = \mathcal{E}_{\mathsf{ht}}(\mathsf{h}, \mathsf{td}) \end{array}\right] \leq \mu(\lambda).$$

We define the additive overhead of an SEH as the overhead of h compared to the input length m, where a rate-1 SEH has sub-linear additive overhead for large enough m. We formalize this as follows.

Definition 3.20. Let $\alpha\colon \mathbb{N}^2 \to \mathbb{N}$. We say that an SEH has *additive overhead* $\alpha(\lambda, m)$ if for any $\lambda, m, k \in \mathbb{N}$, $i^* \in [k]$, and inputs $(x_1, \ldots, x_k) \in (\{0,1\}^m)^k$, it holds that

$$\Pr\left[|\mathsf{h}| - m \leq \alpha(\lambda, m) : \begin{array}{l} \mathsf{hk} \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, m, k, i^*) \\ \mathsf{h} = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k)) \end{array}\right] = 1.$$

If there exists a polynomial q and a constant $c \in \mathbb{N}$ such that an SEH has additive overhead $\alpha(\lambda, m) \leq c \cdot m/\lambda + q(\lambda)$, we say that it is a *rate-1* SEH. ◇

3.6 RAM Computation and Unambiguous RAM Delegation

We define our model of RAM computation and then define RAM delegation, primarily following the work of [PP22].

RAM Computation. We consider RAM computations with word size λ (set to the security parameter for simplicity) specified by a machine M that has random access to a working read-write memory D of size at most 2^λ. The computation of M is carried out one step at a time by a polynomial-time CPU algorithm step that takes as input a description of a program M, a local RAM state rst of size $O(\lambda)$, and a bit b^{mem} read from D. It then outputs a tuple

$$(\mathsf{rst}_{\mathsf{new}}, \ell^{\mathsf{mem}}, \ell^{\mathsf{wrt}}, b^{\mathsf{wrt}}) = \mathsf{step}(M, \mathsf{rst}, b^{\mathsf{mem}}),$$

where $\mathsf{rst}_{\mathsf{new}}$ is the updated state, ℓ^{mem} is the next location to read from D, ℓ^{wrt} is the location in D to write next, and b^{wrt} is the bit to be written.

A RAM computation $M^D(w)$ with initial memory D starts with some initial state rst_0 and initial memory and witness read locations $\ell^{\mathsf{mem}}_0 = \ell^{\mathsf{wit}}_0 = 1$. Starting from $j = 1$, the jth execution step proceeds as follows:

1. Read the memory bit $b_{j-1}^{\mathsf{mem}} := D[\ell_{j-1}^{\mathsf{mem}}]$.
2. Compute $(\mathsf{rst}_j, \ell_j^{\mathsf{mem}}, \ell_j^{\mathsf{wrt}}, b_j^{\mathsf{wrt}}) = \mathsf{step}(M, \mathsf{rst}_{j-1}, b_{j-1}^{\mathsf{mem}})$.
3. If $\ell_j^{\mathsf{wrt}} \neq \perp$, write a bit to memory $D[i_j^{\mathsf{wrt}}] := b_j^{\mathsf{wrt}}$.

The execution terminates when step outputs a special terminating state for rst_t, which specifies whether the computation is accepting and outputs 1 or rejecting and outputs 0.

We write $M(x, w)$ to denote the computation $M^{D_x}(w)$ where D_x is initialized with the input x followed by all zeroes. We define the language $\mathcal{L}_M$ that contains x if $M(x) = 1$.

In this work, we primarily focus on the language of partial computations $\mathcal{T}_M$ that specifies transitions between RAM configurations, which captures the language $\mathcal{L}_M$. For a machine M, we define a RAM configuration $\mathsf{cf} = (\mathsf{rst}, \ell^{\mathsf{mem}}, D)$ consisting of its current local state rst, location ℓ^{mem}, and its current local working memory D. We define the language of partial computations $\mathcal{T}_M$ for a machine M that contains $(\mathsf{cf}, \mathsf{cf}', t)$ if and only if the machine M starting at configuration cf transitions to configuration cf′ after t steps. Observe that $x \in \mathcal{L}_M$ if and only if there exists $t \in \mathbb{N}$ such that $(\mathsf{cf}_x, \mathsf{cf}', t) \in \mathcal{T}_M$ where cf_x consists of initial configuration with memory D_x and the resulting configuration cf′ contains an accepting state rst_t. We'll assume for simplicity that the accepting state always corresponds to a configuration where $\ell^{\mathsf{mem}} = \perp$ and D consists of all 0s; this is without loss of generality up to the time it takes to zero out the memory.

Unambiguous RAM Delegation We use a definition of RAM delegation following [PP22] with the additional property of *unambiguity*.

At a high level, a prover for a RAM delegation scheme convinces a verifier that $(\mathsf{cf}, \mathsf{cf}', t) \in \mathcal{T}_M$ for some RAM computation M. Rather than read the full configuration, the verifier instead receives a digest of the above statement relative to some hash tree HT. Namely, it receives as input $(\mathsf{d}, \mathsf{d}', t)$ where d, d' are short digests of $\mathsf{cf}, \mathsf{cf}'$, respectively.

Definition 3.21. Let $L\colon \mathbb{N}^2 \to \mathbb{N}$ and $\mathsf{HT} = (\mathcal{G}_{\mathsf{ht}}, \mathcal{H}_{\mathsf{ht}}, \mathcal{P}_{\mathsf{ht}}, \mathcal{V}_{\mathsf{ht}})$ be a hash tree. An *L-succinct unambiguous RAM delegation scheme for* HT consists of a tuple of algorithms $(\mathcal{G}_{\mathsf{del}}, \mathcal{D}_{\mathsf{del}}, \mathcal{P}_{\mathsf{del}}, \mathcal{V}_{\mathsf{del}})$ with the following syntax:

- $\mathcal{G}_{\mathsf{del}}(1^\lambda, T) \to \mathsf{crs}$. This is a randomized algorithm that takes as input a security parameter λ and a time bound T, and outputs a common reference string crs. We implicitly assume that crs includes 1^λ and T.
- $\mathcal{D}_{\mathsf{del}}(\mathsf{hk}, \mathsf{cf}) \to \mathsf{d}$. This is a deterministic algorithm that takes as input a hash key hk and a configuration cf and outputs a digest d. We assume for simplicity that on input a configuration $\mathsf{cf} = (\mathsf{rst}, \ell^{\mathsf{mem}}, D)$, $\mathcal{D}_{\mathsf{del}}$ outputs $\mathsf{d} = (\mathsf{rst}, \ell^{\mathsf{mem}}, \mathsf{rt}_D)$ for $\mathsf{rt}_D = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, D)$.
- $\mathcal{P}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, M, (\mathsf{cf}, \mathsf{cf}', t)) \to \pi$. This is a deterministic algorithm that takes as input a common reference string crs, a hash key hk, a machine M, and a statement $(\mathsf{cf}, \mathsf{cf}', t)$ where $\mathsf{cf}, \mathsf{cf}'$ are configurations and t is a number of steps, and outputs a proof π.
- $\mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, M, (\mathsf{d}, \mathsf{d}', t), \pi) \to b$. This is a deterministic algorithm that takes as input the common reference string crs, a hash key hk, a machine M, a hashed statement $(\mathsf{d}, \mathsf{d}', t)$ where d, d' are digests and t is a number of steps, and a proof π, and outputs a bit $b \in \{0, 1\}$ indicating whether to accept or reject the proof.

We require that $(\mathcal{G}_{\mathsf{del}}, \mathcal{D}_{\mathsf{del}}, \mathcal{P}_{\mathsf{del}}, \mathcal{V}_{\mathsf{del}})$ satisfy the following properties for a valid hash tree HT:

- **Completeness.** For any $\lambda, S, T \in \mathbb{N}$, machine M, configurations $\mathsf{cf}, \mathsf{cf}'$, number of steps $t \leq T$ such that $(\mathsf{cf}, \mathsf{cf}', t) \in \mathcal{T}_M$, it holds that

$$\Pr\left[\mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, M, (\mathsf{d}, \mathsf{d}', t), \pi) = 1 \;:\; \begin{array}{l} \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda, T) \\ (\mathsf{hk}, \cdot) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, S) \\ \pi = \mathcal{P}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, M, (\mathsf{cf}, \mathsf{cf}', t)) \\ \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}, \mathsf{cf}), \mathsf{d}' = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{cf}') \end{array}\right] = 1.$$

- **Efficiency and L-Succinctness** There exists a polynomial q such that the following hold in the completeness experiment above for $|M|, S \leq 2^\lambda$ such that M actually uses a working memory of size s and halts within t steps for $s, t \leq 2^\lambda$:
  - $\mathcal{G}_{\mathsf{del}}$ runs in time $L(t, \lambda)$.
  - $\mathcal{P}_{\mathsf{del}}$ runs in time $q(\lambda + t + s)$ and outputs a proof of size $L(t, \lambda)$.
  - $\mathcal{V}_{\mathsf{del}}$ runs in time $L(t, \lambda) \cdot q(|M|)$. ◊

- **Soundness.** For any pair of non-uniform polynomial-time adversaries $(A_1, A_2) = \{(A_{1,\lambda}, A_{2,\lambda})\}_{\lambda \in \mathbb{N}}$ and polynomials T, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} \mathsf{d}' \neq \mathcal{D}_{\mathsf{del}}(\mathsf{hk}, \mathsf{cf}') \\ \wedge\, \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, M, z, \pi) = 1 \\ \wedge\, t \leq T(\lambda) \end{array} \;:\; \begin{array}{l} \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda, T(\lambda)) \\ 1^S \leftarrow A_{1,\lambda}(\mathsf{crs}) \\ (\mathsf{hk}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, S) \\ (M, \mathsf{cf}, \mathsf{d}', t, \pi) \leftarrow A_{2,\lambda}(\mathsf{crs}, \mathsf{hk}) \\ \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}, \mathsf{cf}) \\ z = (\mathsf{d}, \mathsf{d}', t) \\ (\mathsf{cf}, \mathsf{cf}', t) \in \mathcal{T}_M \end{array}\right] \leq \mu(\lambda).$$

- **Unambiguity.** For any pair of non-uniform polynomial-time adversaries $(A_1, A_2) = \{(A_{1,\lambda}, A_{2,\lambda})\}_{\lambda \in \mathbb{N}}$ and polynomials T, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} \pi \neq \pi' \\ \wedge\, \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, M, z, \pi) = 1 \\ \wedge\, \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, M, z, \pi') = 1 \\ \wedge\, t \leq T(\lambda) \end{array} \;:\; \begin{array}{l} \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda, T(\lambda)) \\ 1^S \leftarrow A_{1,\lambda}(\mathsf{crs}) \\ (\mathsf{hk}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, S) \\ (M, \mathsf{cf}, \mathsf{d}', t, \pi, \pi') \leftarrow A_{2,\lambda}(\mathsf{crs}, \mathsf{hk}) \\ \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}, \mathsf{cf}) \\ z = (\mathsf{d}, \mathsf{d}', t) \end{array}\right] \leq \mu(\lambda).$$

The above definition assumes that the CRS generation algorithm $\mathcal{G}_{\mathsf{del}}$ is independent of the machine M. We consider a weaker notion of RAM delegation for a particular machine M, where $\mathcal{G}_{\mathsf{del}}$ and the associated effiency polynomial is allowed to depend on M (we still include M explicitly in the prover and verifier algorithms for consistency).

Remark 2. We also consider L-succinct unambiguous RAM delegation schemes with the hash tree HT not explicitly given. In this case, HT can be any unambiguous hash tree (Definition 3.17) and we assume the $\mathcal{G}_{\mathsf{del}}$ algorithm implicitly generates hk and put it in crs. All other algorithms do not take hk but take crs instead. ◊

4 Somewhere-Extractable Linearly Homomorphic Commitments

In this section, we define and construct a somewhere-extractable linearly homomorphic commitment (seLHC) that can succinctly commit a vector over $\mathbb{Z}$[4] and is extractable on one coordinate of the committed vector (via a trapdoor). We require a strong notion of linear homomorphism: the extraction is linearly homomorphic even for maliciously crafted commitments. In a nutshell, our construction is based on Regev's PKE, which is linearly homomorphic for well-formed ciphertexts, and a proof of "well-formedness".

In the following, we recall Regev's PKE and discuss its linear homomorphism in Section 4.1, define the commitment scheme in Section 4.2, give the construction and the analysis in Section 4.3 and Section 4.4 respectively.

4.1 Preliminary: Regev's Public Key Encryption and Ciphertext Errors

In this section, we recall the construction of Regev's public key encryption. This PKE scheme satisfies a limited notion of linear homomorphism for well-formed ciphertexts, namely those with small decryption errors (as we will define later).

Construction 1 (Regev's PKE). The public key encryption scheme $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ for message modulus $M = M(\lambda)$ is associated with parameters $n = n(\lambda), m = m(\lambda), B = B(\lambda), q = q(\lambda)$ such that $n = \Omega(m \log q)$, $nBM < \frac{q}{2}$.

- $\mathsf{KeyGen}(1^\lambda)$ generates $\mathsf{pk} = (A, \mathbf{v} = A\mathbf{s} + \mathbf{e}) \in \mathbb{Z}_q^{n \times m} \times \mathbb{Z}_q^n$, where $A \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{s} \leftarrow \mathbb{Z}_q^m$ and $\mathbf{e} \leftarrow [-B, B]^n$, and $\mathsf{sk} = \mathbf{s}$.
- $\mathsf{Enc}(\mathsf{pk}, i)$ takes as input the public key pk and a message $i \in [0, M-1]$. It samples $w \in \{0,1\}^n$, and outputs the ciphertext $\mathsf{ct} = (w^T A, \langle w, v \rangle + i \cdot q/M) \in \mathbb{Z}_q^{m+1}$.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ takes as input the secret key sk and a ciphertext ct. It parses $\mathsf{ct} = (\mathsf{ct}^0, \mathsf{ct}^1) \in \mathbb{Z}_q^m \times \mathbb{Z}_q$, and compute the output by rounding $\mathsf{ct}^1 - \langle \mathsf{ct}^0, \mathsf{sk} \rangle$ to the nearest multiple of q/M. ◇

We note that the semantic security of Regev's PKE is implied by hardness of $\mathsf{LWE}_{m,n,q,[-B,B]}$.

Regev's PKE is linearly homomorphic mod M for ciphertexts *with small decryption error*. In the following, we formally define the decryption errors of the ciphertexts.

Definition 4.1. For any ciphertext $\mathsf{ct} = (\mathsf{ct}^0, \mathsf{ct}^1) \in \mathbb{Z}_q^m \times \mathbb{Z}_q$ of Regev's PKE, we define its rounding error as the distance between $\mathsf{ct}^1 - \langle \mathsf{ct}^0, \mathsf{sk} \rangle$ and the nearest multiple of q/M. That is, we define $\mathsf{Error}(\mathsf{sk}, \mathsf{ct}) := \min_{i \in \mathbb{Z}} |\mathsf{ct}^1 - \langle \mathsf{ct}^0, \mathsf{sk} \rangle - i \cdot q/M|$. ◇

Remark 3. We define the decryption error as a function of both the secret key and the ciphertext to make it efficiently computable. ◇

As one immediate observation, for ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$, and vector $(x_1, \ldots, x_n) \in \mathbb{Z}^n$, linear homomorphism mod M holds for $\sum_{i=1}^n \mathsf{ct}_i \cdot x_i$ (i.e. $\mathsf{Dec}(\mathsf{sk}, \sum_{i=1}^n \mathsf{ct}_i \cdot x_i) \equiv \sum_{i=1}^n \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_i) \cdot x_i \pmod M$) as long as $\sum_{i=1}^n |x_i| \cdot \mathsf{Error}(\mathsf{sk}, \mathsf{ct}_i) < \frac{q}{2M}$.

[4] It is also interesting to consider finite fields other than $\mathbb{Z}$. For our application to witness-unambiguous BARG, we only consider $\mathbb{Z}$.

Fact 4.2. For any $(\mathsf{pk}, \mathsf{sk})$ of Regev's PKE, ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$, and vector $(x_1, \ldots, x_n) \in \mathbb{Z}^n$, if $\sum_{i=1}^{n} |x_i| \cdot \mathsf{Error}(\mathsf{sk}, \mathsf{ct}_i) < \frac{q}{2M}$, then

$$\mathsf{Dec}(\mathsf{sk}, \sum_{i=1}^{n} \mathsf{ct}_i \cdot x_i) \equiv \sum_{i=1}^{n} \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_i) \cdot x_i \pmod{M}.$$

♢

4.2 Definition

In this section, we define the somewhere-extractable linearly homomorphic commitment scheme we need as follows.

Definition 4.3 (Somewhere-Extractable Linearly Homomorphic Commitments with Proofs of Well-Formedness). A *somewhere extractable linearly homomorphic commitment (seLHC) with proofs of well-formedness* is a tuple of algorithms $\mathsf{LHC} = (\mathcal{G}, \mathsf{Commit}, \mathsf{Ext}, \mathcal{P}, \mathcal{V})$ with the following syntax:

- $\mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \to (\mathsf{ck}, \mathsf{crs}, \mathsf{td})$. This is a randomized algorithm that takes as input a security parameter λ, input length 1^n, an upper bound on the number of commitments 1^m, an upper bound on committed values 1^s, and an index $i^* \in [n]$. It outputs a commitment key ck, a common reference string crs, and a trapdoor td.
- $\mathsf{Commit}(\mathsf{ck}, (x_1, \ldots, x_n)) \to \mathsf{c}$. This is a deterministic algorithm that takes as input a commitment key ck and a vector $(x_1, \ldots, x_n) \in \mathbb{Z}^n$, and outputs a commitment value c.
- $\mathsf{Ext}(\mathsf{td}, \mathsf{c}) \to x$. This is a deterministic algorithm that takes as input a trapdoor td and a commitment value c, and outputs an integer x.
- $\mathcal{P}(\mathsf{ck}, \mathsf{crs}, \{(x_{1,j}, \ldots, x_{n,j})\}_{j \in [m]}) \to \pi$. This is a deterministic algorithm takes as input a commitment key ck, a common reference string crs, and *binary* vectors $\{(x_{1,j}, \ldots, x_{n,j})\}_{j \in [m]} \in \{0,1\}^{n \cdot m}$. It outputs a proof $\pi \in \{0,1\}^{n \cdot \mathsf{poly}(\lambda)}$.
- $\mathcal{V}(\mathsf{ck}, \mathsf{crs}, (\mathsf{c}_1, \ldots, \mathsf{c}_m), \pi) \to b$. This is a deterministic algorithm that takes as input a commitment key ck, a common reference string crs, a sequence of commitments $(\mathsf{c}_1, \ldots, \mathsf{c}_m)$, and a proof π. It outputs a bit b indicating whether to accept or reject the proof.

We require that LHC satisfy the following commitment properties:

- **Commitment efficiency.** The algorithms $\mathcal{G}$ and Commit satisfy the following efficiency properties for $n, m, s \leq 2^\lambda$.
  - $\mathcal{G}$ runs in time $n \cdot \mathsf{poly}(\lambda)$ to generate ck and time $\mathsf{poly}(\lambda, m)$ to generate crs. The length of commitment key ck is in $n \cdot \mathsf{poly}(\lambda)$, and the length of crs is in $\mathsf{poly}(\lambda, m)$.
  - Commit runs in time $n \cdot \mathsf{poly}(\lambda)$ for any input vector $\mathbf{x} = (x_1, \ldots, x_n) \in [0, s]^n$.
- **Short vector commitments.** For any $\lambda, n, m, s \in \mathbb{N}$ such that $n, m, s \leq 2^\lambda$, $i^* \in [n]$, and any $\mathsf{ck} \in \mathsf{supp}(\mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*))$, it holds that if $(x_1, \ldots, x_n) \in \{0,1\}^n$, then $\mathsf{Commit}(\mathsf{ck}, (x_1, \ldots, x_n)) \in [0, n]^{\mathsf{poly}(\lambda)}$.

- **Index hiding.** For any non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda\in\mathbb{N}}$, polynomial n, m, s, and sequence of indices $\{(i_{0,\lambda}, i_{1,\lambda}) \in [n(\lambda)]^2\}_{\lambda\in\mathbb{N}}$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} A_\lambda(\mathsf{ck},\mathsf{crs}) = b \\ \\ \end{array} : \begin{array}{l} b \leftarrow \{0,1\} \\ (\mathsf{ck},\mathsf{crs},\mathsf{td}) \leftarrow \mathcal{G}(1^\lambda, 1^{n(\lambda)}, 1^{m(\lambda)}, 1^{s(\lambda)}, i_{b,\lambda}) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

- **Linear homomorphism.** For any commitment key ck, $a, b \in \mathbb{Z}$, and any integer vectors $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^n$, we have

$$a \cdot \mathsf{Commit}(\mathsf{ck}, \mathbf{x}) + b \cdot \mathsf{Commit}(\mathsf{ck}, \mathbf{y}) = \mathsf{Commit}(\mathsf{ck}, a \cdot \mathbf{x} + b \cdot \mathbf{y}).$$

- **Somewhere extraction correctness.** For any $\lambda, n, m, s \in \mathbb{N}$ such that $n, m, s \leq 2^\lambda$, and any $i^* \in [n]$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $\mathbf{x} \in [0, s]^n$, it holds that

$$\Pr\left[\begin{array}{l} x_{i^*} \neq \mathsf{Ext}(\mathsf{td}, \mathsf{c}) \\ \\ \end{array} : \begin{array}{l} (\mathsf{ck},\mathsf{crs},\mathsf{td}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \\ \mathsf{c} = \mathsf{Commit}(\mathsf{ck}, \mathbf{x}) \end{array}\right] \leq \mu(\lambda).$$

We also require that LHC satisfy the following augmented proof system properties:

- **Verification efficiency.** $\mathcal{V}$ runs in time $n \cdot \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda, m)$, except performing range check over its input.

- **Prover completeness.** For any $\lambda, n, m, s \in \mathbb{N}$ such that $n, m, s \leq 2^\lambda$, $\mathbf{x}_j \in \{0,1\}^n$ for every $j \in [m]$ and any $i^* \in [n]$, it holds that

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{ck},\mathsf{crs},(\mathsf{c}_1,\ldots,\mathsf{c}_m),\pi) = 1 \\ \\ \end{array} : \begin{array}{l} (\mathsf{ck},\mathsf{crs},\mathsf{td}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \\ \forall j \in [m], \mathsf{c}_j = \mathsf{Commit}(\mathsf{ck}, \mathbf{x}_j) \\ \pi = \mathcal{P}(\mathsf{ck},\mathsf{crs},\{\mathbf{x}_j\}_{j\in[m]}) \end{array}\right] = 1.$$

- **Linearly homomorphic extraction with proof.** For any non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda\in\mathbb{N}}$, polynomially bounded values $n = n(\lambda), m = m(\lambda), s = s(\lambda)$, and any sequence of indices $i^* = i^*(\lambda) \in [n(\lambda)]$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{ck},\mathsf{crs},(\mathsf{c}_1^*,\ldots,\mathsf{c}_m^*),\pi^*) = 1 \\ \wedge\ \exists \mathbf{q} \in \{0,1\}^m, \\ \quad \mathsf{Ext}(\mathsf{td}, C^* \cdot \mathbf{q}) \neq \mathsf{Ext}(\mathsf{td}, C^*) \cdot \mathbf{q} \end{array} : \begin{array}{l} (\mathsf{ck},\mathsf{crs},\mathsf{td}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \\ (C^* = (\mathsf{c}_1^*,\ldots,\mathsf{c}_m^*), \pi^*) \leftarrow A_\lambda(\mathsf{ck},\mathsf{crs}) \end{array}\right] \leq \mu(\lambda),$$

  where $C^* \cdot \mathbf{q} := \sum_i q_i \cdot \mathsf{c}_i^*$, and $\mathsf{Ext}(\mathsf{td}, C^*) := (\mathsf{Ext}(\mathsf{td}, \mathsf{c}_1^*), \ldots, \mathsf{Ext}(\mathsf{td}, \mathsf{c}_m^*))$.

- **Proof unambiguity.** For any non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda\in\mathbb{N}}$, any polynomially bounded values $n = n(\lambda), m = m(\lambda), s = s(\lambda)$, and $i^* = i^*(\lambda) \in [n(\lambda)]$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} \pi \neq \pi' \\ \wedge\ \mathcal{V}(\mathsf{ck},\mathsf{crs},C^*,\pi) = 1 \\ \wedge\ \mathcal{V}(\mathsf{ck},\mathsf{crs},C^*,\pi') = 1 \end{array} : \begin{array}{l} (\mathsf{ck},\mathsf{crs},\mathsf{td}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \\ (C^*, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck},\mathsf{crs}) \end{array}\right] \leq \mu(\lambda).$$

◇

We observe that Commit is implied to be collision-resistant for certain inputs given the above properties.

Remark 4 (Collision-Resistance of Commit). We note that index hiding and somewhere extraction correctness together imply that the commitment $\mathsf{Commit}(\cdot)$ is collision-resistant on inputs in $[0, s]^n$. ◇

4.3 Construction

We construct a seLHC with proofs of well-formedness from LWE, where the LWE modulus-to-noise ratio polynomially depends on the input parameters.

Theorem 4.4. There exists a somewhere extractable linearly homomorphic commitment scheme with proofs of well-formedness assuming polynomial hardness of LWE.

Our construction of LHC uses the following building blocks:

- The linearly homomorphic Regev's public key encryption (see Section 4.1). We will use this PKE scheme in a white-box way.
- A CI hash family $\mathcal{H}$ for the ensemble of product relations $R = R_\lambda \subseteq [0,n]^{\mathsf{poly}(\lambda)\times m} \times \{0,1\}^{m\times t_\lambda}$ that are time-$\mathsf{poly}(\lambda, m, \log n)$ product-verifiable with product sparsity at most $1-\varepsilon$, where $t_\lambda = \lambda/\varepsilon$ and we take $\varepsilon = 1/2$. Such an $\mathcal{H}$ is guaranteed to exist by Theorem 3.14. Let $\mathsf{hk} \leftarrow \mathcal{H}.\mathcal{G}(1^\lambda, 1^m, n)$ denote the hash key generation algorithm and $\mathcal{H}.\mathsf{Hash}(\mathsf{hk}, \cdot)$ denote the evaluation algorithm.

For our construction, we assume without loss of generality that $s \geq m^2$, otherwise we can take s to be $\max(s, m^2)$. We define the seLHC algorithms $(\mathcal{G}, \mathsf{Commit}, \mathsf{Ext}, \mathcal{P}, \mathcal{V})$ for our construction as follows:

- $(\mathsf{ck}, \mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*)$:
  1. Set up PKE parameters $n', m', B \in \mathsf{poly}(\lambda)$ as fixed polynomials. Take lattice modulus $q = \mathsf{poly}(\lambda, n, m, s)$ such that $q \geq 20n'B\max(ns^2, nm^2s)$. Looking ahead, we will base the security on $\mathsf{LWE}_{n',m',q,[-B,B]}$.
  2. Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}$ using parameters decided in the previous step, i.e. take $\mathsf{pk} = (A, \mathbf{v} = A\mathbf{s} + \mathbf{e}) \in \mathbb{Z}_q^{n'\times m'} \times \mathbb{Z}_q^{n'}$, where $A \leftarrow \mathbb{Z}_q^{n'\times m'}, \mathbf{s} \leftarrow \mathbb{Z}_q^{m'}, \mathbf{e} \leftarrow [-B, B]^{n'}$, and $\mathsf{sk} = \mathbf{s}$.
  3. For each $i \in [n]$, let $b_i = \mathbf{1}[i = i^*]$, and compute the ciphertext that encodes $b_i \in [0, s]$ as $\mathsf{ct}_i \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, b_i) \in \mathbb{Z}_q^{m'+1}$, i.e. sample $w \in \{0,1\}^{n'}$, and set $\mathsf{ct}_i = (w^T A, \langle w, v\rangle + b_i \cdot q/(s+1)) \in \mathbb{Z}_q^{m'+1}$.
  4. Sample CI hash key $\mathsf{hk} \leftarrow \mathcal{H}.\mathcal{G}(1^\lambda, 1^m, n)$.
  5. The commitment key is
  $$\mathsf{ck} = (\mathbf{G}^{-1}(\mathsf{ct}_1), \ldots, \mathbf{G}^{-1}(\mathsf{ct}_n)) \in \{0,1\}^{((m'+1)\lceil \log q\rceil)\times n},$$
  the prover common reference string is $\mathsf{crs} = \mathsf{hk}$, and the trapdoor is $\mathsf{td} = \mathsf{sk}$.
- $\mathsf{c} = \mathsf{Commit}(\mathsf{ck}, (x_1, x_2, \ldots, x_n))$:
  1. Parse ck as a binary matrix in $\{0,1\}^{((m'+1)\lceil \log q\rceil)\times n}$, and let $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Z}^n$.
  2. Output $\mathsf{c} = \mathsf{ck} \cdot \mathbf{x} \in \mathbb{Z}^{(m'+1)\lceil \log q\rceil}$. (Equivalently, $\mathsf{c} = \sum_{i=1}^n x_i \mathbf{G}^{-1}(\mathsf{ct}_i)$.)
- $x = \mathsf{Ext}(\mathsf{td}, \mathsf{c})$:
  1. Parse $\mathsf{td} = \mathsf{sk}$.

2. Compute $\mathsf{ct} = \mathbb{G}_{m'+1} \cdot \mathsf{c} \pmod q \in \mathbb{Z}_q^{m'+1}$.
3. Output $x = \mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct})$, i.e. parse $\mathsf{ct} = (\mathsf{ct}^0, \mathsf{ct}^1) \in \mathbb{Z}_q^{m'} \times \mathbb{Z}_q$, and compute x by rounding $\mathsf{ct}^1 - \langle \mathsf{ct}^0, \mathsf{sk} \rangle$ to the nearest multiple of $q/(s+1)$.

- $\pi = \mathcal{P}(\mathsf{ck}, \mathsf{crs}, \{\mathbf{x}_j \in \{0,1\}^n\}_{j \in [m]})$:
  1. Parse $\mathsf{crs} = \mathsf{hk}$.
  2. For $j \in [m]$, compute $\mathsf{c}_j = \mathsf{Commit}(\mathsf{ck}, \mathbf{x}_j)$. Assemble them by column into $C = (\mathsf{c}_1, \ldots, \mathsf{c}_m) \in \mathbb{Z}^{(m'+1)\lceil \log q \rceil \times m}$. Note that our construction guarantees that $C \in [0,n]^{(m'+1)\lceil \log q \rceil \times m}$.
     Equivalently, taking $X = (\mathbf{x}_1, \ldots, \mathbf{x}_m) \in \{0,1\}^{n \times m}$, we compute C as $\mathsf{ck} \cdot X$.
  3. Compute $(\mathbf{q}_1, \ldots, \mathbf{q}_{t_\lambda}) \leftarrow \mathcal{H}.\mathsf{Hash}(\mathsf{hk}, C)$, where $\mathbf{q}_i \in \{0,1\}^m$ for every $i \in [t_\lambda]$.
  4. For each $i \in [t_\lambda]$, compute $\mathbf{v}_i = X\mathbf{q}_i = \sum_{j=1}^m (\mathbf{q}_i)_j \cdot \mathbf{x}_j \in [0,m]^n$.
  5. Output $\pi = \{\mathbf{v}_i\}_{i \in [t_\lambda]}$.
- $b = \mathcal{V}(\mathsf{ck}, \mathsf{crs}, C = (\mathsf{c}_1, \ldots, \mathsf{c}_m), \pi)$:
  1. Parse $\pi = \{\mathbf{v}_i\}_{i \in [t_\lambda]}$, reject if $\mathbf{v}_i \notin [0,m]^n$ for any $i \in [t_\lambda]$.
  2. Reject if $C \notin [0,n]^{(m'+1)\lceil \log q \rceil \times m}$.
  3. Compute $(\mathbf{q}_1, \ldots, \mathbf{q}_{t_\lambda}) \leftarrow \mathcal{H}.\mathsf{Hash}(\mathsf{hk}, C)$.
  4. For all $i \in [t_\lambda]$, check if $\mathsf{Commit}(\mathsf{ck}, \mathbf{v}_i) = C \cdot \mathbf{q}_i$, reject otherwise.

4.4 Analysis

We now prove that our construction satisfies all the required properties as in Definition 4.3.

Efficiency. The algorithms satisfy the following efficiency properties:

- The size of ck is $n \cdot O(m' \log q) = n \cdot \mathsf{poly}(\lambda)$, and the size of crs is $\mathsf{poly}(\lambda \cdot m \log n, t_\lambda) = \mathsf{poly}(m, \lambda)$. Furthermore, ck and crs are generated in time $n \cdot \mathsf{poly}(\lambda), \mathsf{poly}(m, \lambda)$, respectively.
- Algorithm Commit only does a matrix multiplication over $\mathbb{Z}$ with all values bounded by $s \cdot n < 2^{O(\lambda)}$, thus runs in time $n \cdot \mathsf{poly}(\lambda)$.
- $\mathcal{V}$ runs in time $\mathsf{poly}(m, \lambda, \log n) = \mathsf{poly}(m, \lambda)$ to compute the CI hash, and runs in time $t_\lambda \cdot (n+m)\mathsf{poly}(\lambda)$ to verify $\mathsf{Commit}(\mathsf{ck}, \mathbf{v}_i) = C \cdot \mathbf{q}_i$ for all $i \in [t_\lambda]$. Therefore, its total running time is $n \cdot \mathsf{poly}(\lambda) + \mathsf{poly}(m, \lambda)$.

Short vector commitments. For $\mathbf{x} \in \{0,1\}^n$, $\mathsf{Commit}(\mathsf{ck}, \mathbf{x})$ is the product of a binary matrix in $\{0,1\}^{\mathsf{poly}(\lambda) \times n}$ and $\mathbf{x}$, thus is in $[0,n]^{\mathsf{poly}(\lambda)}$.

Index hiding. This directly follows from semantic security of the Regev's PKE scheme, which holds under hardness of $\mathsf{LWE}_{n',m',q,[-B,B]}$.

Linear homomorphism. This directly follows from our construction, as $\mathsf{Commit}(\mathsf{ck}, \cdot)$ is a linear function of vectors over $\mathbb{Z}$.

Somewhere extraction correctness. Fix $\mathsf{ck} \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*)$, and $\mathbf{x} \in [0, s]^n$. Let $c = \mathsf{Commit}(\mathsf{ck}, \mathbf{x})$, and $\mathsf{ct} = \mathbb{G} \cdot \mathsf{c} \pmod q \in \mathbb{Z}_q^{m'+1}$.

It follows from our construction that $\mathsf{ct} = \sum_{i=1}^n x_i \cdot \mathsf{ct}_i$ where $\mathsf{ct}_i \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, \mathbf{1}[i = i^*])$ are the forming components of ck. By Fact 4.2, it suffices to bound the rounding errors (Definition 4.1) to show that $\mathsf{PKE.Dec}(\mathsf{sk}, ct) = \sum_{i=1}^n x_i \cdot \mathbf{1}[i = i^*] \bmod (s+1) = x_{i^*}$.

We observe that for each $i \in [n]$, $\mathsf{Error}(\mathsf{sk}, \mathsf{ct}_i) \le n'B$. Therefore, we choose $q \ge 10ns^2 \cdot n'B$ such that $\sum_{i=1}^n x_i \mathsf{Error}(\mathsf{sk}, \mathsf{ct}_i) \le ns \cdot n'B \ll q/(s+1)$, so linear homomorphism holds.

Prover Completeness. It suffices to show that, for any $\mathbf{q} \in \{0,1\}^m$, let $\mathbf{v} = X \cdot \mathbf{q}$, we have $\mathsf{Commit}(\mathsf{ck}, \mathbf{v}) = C \cdot \mathbf{q}$, where $C = \mathsf{ck} \cdot X$. This equation holds by the associativity of matrix multiplication:

$$\mathsf{Commit}(\mathsf{ck}, \mathbf{v}) = \mathsf{ck}(X\mathbf{q}) = (\mathsf{ck}X)\mathbf{q} = C \cdot \mathbf{q}.$$

Linearly homomorphic extraction with proof. Linear homomorphism of the commitments, i.e, $\mathsf{Ext}(\mathsf{td}, C \cdot \mathbf{q}) = \mathsf{Ext}(\mathsf{td}, C) \cdot \mathbf{q}$ where $C = (\mathsf{c}_1, \ldots, \mathsf{c}_m)$ and $\mathbf{q} \in \{0,1\}^m$, is implied by linear homomorphism of Regev's PKE, as the commitments are essentially Regev's PKE ciphertexts. The high-level idea of this proof is to ensure the latter, by ruling out the following two cases:

1. Ciphertexts $(\mathsf{c}_1, \ldots, \mathsf{c}_m)$ have a large decryption error.
2. Ciphertexts $(\mathsf{c}_1, \ldots, \mathsf{c}_m)$ encode large numbers. Note that Regev's PKE used in our construction is only linearly homomorphic mod $(s+1)$, but we intend to support linear homomorphism over $\mathbb{Z}$. Therefore, we need to prevent overflow.

In the following, we show that an accepting proof π^* rules out these two cases, as indicated by Lemma 4.5 and Lemma 4.8. By combining them we prove the desired linearly homomorphic extraction.

We first show that an accepting proof ensures the commitments are all PKE ciphertexts with small decryption error. In the following, we write $\mathsf{Error}(\mathsf{sk}, \mathsf{c}) := \mathsf{Error}(\mathsf{sk}, \mathbf{G} \cdot \mathsf{c} \bmod q)$ for $\mathsf{c} \in \mathbb{Z}^{(m'+1)\lceil \log q \rceil}$, treating commitments as PKE ciphertexts by applying the lattice gadget matrix.

Lemma 4.5. For any non-uniform polynomial-size adversary $\mathcal{A}$, any polynomials $n(\lambda), m(\lambda), s(\lambda)$ and any $i^* = i^*(\lambda) \in [n]$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathsf{ck}, \mathsf{crs}, C^*, \pi^*) = 1 \wedge & (\mathsf{ck}, \mathsf{crs}, \mathsf{td} = \mathsf{sk}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \\ \exists j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}_j^*) > 2nm \cdot n'B & (C^* = (\mathsf{c}_1^*, \ldots, \mathsf{c}_m^*), \pi^*) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{crs}) \end{array}\right] \le \mu(\lambda). \qquad \diamond$$

To prove the lemma, consider the bad relation $R^{err}_{\lambda,\mathsf{sk}}$ defined as

$$R^{err}_{\lambda,\mathsf{sk}} = \left\{ (C = (\mathsf{c}_1, \ldots, \mathsf{c}_m), (\mathbf{q}_1, \ldots, \mathbf{q}_{t_\lambda})) \;\middle|\; \begin{array}{c} \exists j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}_j) > 2nm \cdot n'B \\ \wedge \forall i \in [t_\lambda], \mathsf{Error}(\mathsf{sk}, C \cdot \mathbf{q}_i) \le nm \cdot n'B \end{array} \right\}.$$

It is easy to see that $R^{err}_{\lambda,\mathsf{sk}}$ can be product-verified in $\mathsf{poly}(\lambda, m)$ time, by a circuit with sk hard-coded.

Claim 4.6. $R^{err}_{\lambda,\mathsf{sk}}$ has product sparsity at most $1/2$. ◇

Proof. Fix any $C = (\mathsf{c}_1, \ldots, \mathsf{c}_m)$, then either $R^{err}_{\lambda,\mathsf{sk}}(C) = \emptyset$ or $\exists j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}_j) > 2nm \cdot n'B$. We fix any such $j \in [m]$ in the latter case.

The observation is, for any $\mathbf{q} \in \{0,1\}^m$, let $\bar{\mathbf{q}} = \mathbf{q} \oplus \mathbf{e}_j$, i.e. $\bar{\mathbf{q}}$ equals $\mathbf{q}$ at all coordinates except the j-th. Then it is impossible to have both $\mathsf{Error}(\mathsf{sk}, C \cdot \mathbf{q}) \le nm \cdot n'B$ and $\mathsf{Error}(\mathsf{sk}, C \cdot \bar{\mathbf{q}}) \le nm \cdot n'B$, as which would imply $\mathsf{Error}(\mathsf{sk}, \mathsf{c}_j) = \mathsf{Error}(\mathsf{sk}, C \cdot \mathbf{e}_j) \le 2nm \cdot n'B$. □

Proof (Proof of Lemma 4.5). By the above discussion on bad relation $R^{err}_{\lambda,\mathsf{sk}}$, $\mathcal{H}$ is correlation intractable for it. Therefore,

$$\Pr\left[\begin{array}{l|ll} \exists j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}^*_j) > 2nm \cdot n'B & : & (\mathsf{ck}, \mathsf{crs} = \mathsf{hk}, \mathsf{td} = \mathsf{sk}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \\ \wedge \forall i \in [t_\lambda], \mathsf{Error}(\mathsf{sk}, C^* \cdot \mathbf{q}_i) \le nm \cdot n'B & & (C^* = (\mathsf{c}^*_1, \ldots, \mathsf{c}^*_m), \pi^*) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{crs}) \\ & & (\mathbf{q}_1, \ldots, \mathbf{q}_{t_\lambda}) = \mathcal{H}.\mathsf{Hash}(\mathsf{hk}, C^*) \end{array}\right] \le \mathsf{negl}(\lambda).$$

Note that $\mathcal{V}(\mathsf{ck}, \mathsf{crs}, C^*, \pi^*) = 1$ implies, for every $i \in [t_\lambda]$, we have $C^* \cdot \mathbf{q}_i = \mathsf{Commit}(\mathsf{ck}, \mathbf{v}^*_i)$ for some $\mathbf{v}^*_i \in \{0, \ldots, m\}^n$, this further means $\mathsf{Error}(\mathsf{sk}, C^* \cdot \mathbf{q}_i) = \mathsf{Error}(\mathsf{sk}, \mathsf{Commit}(\mathsf{ck}, \mathbf{v}^*_i)) \le nm \cdot n'B$. Therefore,

$$\begin{aligned} &\Pr\left[\begin{array}{l|ll} \mathcal{V}(\mathsf{ck}, \mathsf{crs}, C^*, \pi^*) = 1 \wedge & : & (\mathsf{ck}, \mathsf{crs}, \mathsf{td} = \mathsf{sk}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \\ \exists j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}^*_j) > 2nm \cdot n'B & & (C^* = (\mathsf{c}^*_1, \ldots, \mathsf{c}^*_m), \pi^*) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{crs}) \end{array}\right] \\ \le &\Pr\left[\begin{array}{l|ll} \forall i \in [t_\lambda], \mathsf{Error}(\mathsf{sk}, C^* \cdot \mathbf{q}_i) \le nm \cdot n'B & & (\mathsf{ck}, \mathsf{crs} = \mathsf{hk}, \mathsf{td} = \mathsf{sk}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \\ \wedge \exists j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}^*_j) > 2nm \cdot n'B & : & (C^* = (\mathsf{c}^*_1, \ldots, \mathsf{c}^*_m), \pi^*) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{crs}) \\ & & (\mathbf{q}_1, \ldots, \mathbf{q}_{t_\lambda}) = \mathcal{H}.\mathsf{Hash}(\mathsf{hk}, C^*) \end{array}\right] \\ \le &\mathsf{negl}(\lambda). \end{aligned}$$

□

By ruling out commitments of large decryption error, we establish linearly homomorphic extraction with modulus $(s+1)$.

Lemma 4.7. For any non-uniform polynomial-size adversary $\mathcal{A}$, any polynomials $n(\lambda), m(\lambda), s(\lambda)$, and any $i^* = i^*(\lambda) \in [n]$, assume $q/(s+1) \ge 10nm^2 \cdot n'B$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l|ll} \mathcal{V}(\mathsf{ck}, \mathsf{crs}, (\mathsf{c}^*_1, \ldots, \mathsf{c}^*_m), \pi^*) = 1 & & (\mathsf{ck}, \mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \\ \wedge \exists \mathbf{q} \in \{0,1\}^m, & : & (C^* = (\mathsf{c}^*_1, \ldots, \mathsf{c}^*_m), \pi^*) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{crs}) \\ \mathsf{Ext}(\mathsf{td}, C^* \cdot \mathbf{q}) \not\equiv \mathsf{Ext}(\mathsf{td}, C^*) \cdot \mathbf{q} \pmod{s+1} & & \end{array}\right] \le \mu(\lambda).$$

◇

Proof. It directly follows from Lemma 4.5 and Fact 4.2. Note that we have $\mathsf{Error}(\mathsf{sk}, C^* \cdot \mathbf{q}) \le 2nm^2 \cdot n'B \ll q/(s+1)$. □

To achieve linearly homomorphic extraction over $\mathbb{Z}$, we further show the commitments always encode small numbers, such that the modulus $(s+1)$ can be ignored.

Lemma 4.8. For any non-uniform polynomial-size adversary $\mathcal{A}$, any polynomials $n(\lambda), m(\lambda), s(\lambda)$ and any $i^* = i^*(\lambda) \in [n]$, assume $q/(s+1) \ge 10nm^2 \cdot n'B$, there exists a negligible function $\mu(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|ll} \mathcal{V}(\mathsf{ck}, \mathsf{crs}, C^*, \pi^*) = 1 \wedge & & (\mathsf{ck}, \mathsf{crs}, \mathsf{td} = \mathsf{sk}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \\ \exists j \in [m], \mathsf{Ext}(\mathsf{td}, \mathsf{c}^*_j) > m & : & (C^* = (\mathsf{c}^*_1, \ldots, \mathsf{c}^*_m), \pi^*) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{crs}) \end{array}\right] \le \mu(\lambda).$$

◇

Proof. Consider the bad relation $R^{\mathsf{Ext}}_{\lambda,\mathsf{td}}$ defined as

$$R^{\mathsf{Ext}}_{\lambda,\mathsf{td}} = \left\{ (C = (\mathsf{c}_1, \ldots, \mathsf{c}_m), (\mathbf{q}_1, \ldots, \mathbf{q}_{t_\lambda})) \;\middle|\; \begin{array}{c} \exists j \in [m], \mathsf{Ext}(\mathsf{td}, \mathsf{c}_j) > m \\ \wedge \forall j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}_j) \leq 2nm \cdot n'B \\ \wedge \forall i \in [t_\lambda], \mathsf{Ext}(\mathsf{td}, C \cdot \mathbf{q}_i) \leq m \end{array} \right\}.$$

It is easy to see that $R^{\mathsf{Ext}}_{\lambda,\mathsf{td}}$ can be product-verified in $\mathsf{poly}(\lambda, m)$ time, and it has product sparsity at most $1/2$. Therefore, $\mathcal{H}$ is correlation intractable for $R^{\mathsf{Ext}}_{\lambda,\mathsf{td}}$, so

$$\Pr\left[\begin{array}{l} \exists j \in [m], \mathsf{Ext}(\mathsf{td}, \mathsf{c}^*_j) > m \\ \wedge \forall j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}^*_j) \leq 2nm \cdot n'B \\ \wedge \forall i \in [t_\lambda], \mathsf{Ext}(\mathsf{td}, C^* \cdot \mathbf{q}_i) \leq m \end{array} : \begin{array}{l} (\mathsf{ck}, \mathsf{crs}, \mathsf{td} = \mathsf{sk}) \leftarrow \mathcal{G}(1^\lambda, 1^n, 1^m, 1^s, i^*) \\ (C^* = (\mathsf{c}^*_1, \ldots, \mathsf{c}^*_m), \pi^*) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{crs}) \end{array} \right] \leq \mathsf{negl}(\lambda). \quad (1)$$

By the verification algorithm, $\mathcal{V}(\mathsf{ck}, \mathsf{crs}, C^*, \pi^*) = 1$ implies for all $i \in [t_\lambda]$, $\mathsf{Commit}(\mathsf{ck}, \mathbf{v}^*_i) = C^* \cdot \mathbf{q}_i$, and all entries in $\mathbf{v}^*_i$ are in $[0, m]$. Since we take $s \geq m^2 > m$, by somewhere extraction correctness $\mathsf{Ext}(\mathsf{td}, C^* \cdot \mathbf{q}_i) = \mathsf{Ext}(\mathsf{td}, \mathsf{Commit}(\mathsf{ck}, \mathbf{v}^*_i))$ equals to a certain coordinate in $\mathbf{v}^*_i$ and is therefore in $[0, m]$. Therefore,

$$\begin{aligned} &\left\{ \begin{array}{c} \mathcal{V}(\mathsf{ck}, \mathsf{crs}, C^*, \pi^*) = 1 \wedge \\ \exists j \in [m], \mathsf{Ext}(\mathsf{td}, \mathsf{c}^*_j) > m \end{array} \right\} \\ \subseteq &\left\{ \begin{array}{c} \mathcal{V}(\mathsf{ck}, \mathsf{crs}, C^*, \pi^*) = 1 \wedge \\ \exists j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}^*_j) > 2nm \cdot n'B \end{array} \right\} \bigcup \left\{ \begin{array}{c} \mathcal{V}(\mathsf{ck}, \mathsf{crs}, C^*, \pi^*) = 1 \wedge \\ \forall j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}^*_j) \leq 2nm \cdot n'B \wedge \\ \exists j \in [m], \mathsf{Ext}(\mathsf{td}, \mathsf{c}^*_j) > m \end{array} \right\} \\ \subseteq &\left\{ \begin{array}{c} \mathcal{V}(\mathsf{ck}, \mathsf{crs}, C^*, \pi^*) = 1 \wedge \\ \exists j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}^*_j) > 2nm \cdot n'B \end{array} \right\} \bigcup \left\{ \begin{array}{c} \forall j \in [m], \mathsf{Error}(\mathsf{sk}, \mathsf{c}^*_j) \leq 2nm \cdot n'B \wedge \\ \exists j \in [m], \mathsf{Ext}(\mathsf{td}, \mathsf{c}^*_j) > m \wedge \\ \forall i \in [t_\lambda], \mathsf{Ext}(\mathsf{td}, C^* \cdot \mathbf{q}_i) \leq m \end{array} \right\}. \end{aligned}$$

We finish the proof as the probability of the two events in the last line are bounded by Lemma 4.5 and Equation 1, respectively. □

Proof (Linearly homomorphic extraction with proof). It follows from Lemma 4.7 and Lemma 4.8, as we have $0 \leq \mathsf{Ext}(\mathsf{td}, C^* \cdot \mathbf{q}), \mathsf{Ext}(\mathsf{td}, C^*) \cdot \mathbf{q} \leq m^2 \leq s$, which avoids overflow. (Recall we assume $s \geq m^2$.) □

Proof unambiguity. Let the output of the adversary be $C^*, \pi = \{\mathbf{v}_i\}_{i \in [t_\lambda]}, \pi' = \{\mathbf{v}'_i\}_{i \in [t_\lambda]}$. If both proofs are accepting, we have $\mathsf{Commit}(\mathsf{ck}, \mathbf{v}_i) = C \cdot \mathbf{q}_i = \mathsf{Commit}(\mathsf{ck}, \mathbf{v}'_i)$ for all $i \in [t_\lambda]$, and $\mathbf{v}_i, \mathbf{v}'_i \in \{0, \ldots, m\}^n \subseteq \{0, \ldots, s\}^n$. However, since $\mathsf{Commit}(\mathsf{ck}, \cdot)$ is a CRHF for inputs in $\{0, \ldots, s\}^n$ (implied by index hiding and somewhere extraction correctness, see Remark 4), no efficient adversary can found $\mathbf{v}_i \neq \mathbf{v}'_i$ for any $i \in [t_\lambda]$. Therefore, we conclude that $\pi = \pi'$ with all but negligible probability.

5 Witness-Unambiguous Batch Arguments

In this section, we define and construct witness-unambiguous somewhere extractable BARGs. The construction relies on the seLHC with well-formed proofs from Section 4.

At a high level, witness-unambiguity captures the idea that if an adversary generates two distinct valid proofs for a seBARG, then it must have used two different sets of witnesses. We formalize this as follows.

Definition 5.1 (Witness-Unambiguity). We say an seBARG scheme $\mathsf{BARG} = (\mathcal{G}_{\mathsf{barg}}, \mathcal{P}_{\mathsf{barg}}, \mathcal{V}_{\mathsf{barg}}, \mathsf{Ext}_{\mathsf{barg}})$ satisfies *witness-unambiguity* if for any non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda\in\mathbb{N}}$, polynomials k, m, T, and sequence of indices $\{i^*_\lambda \in [k(\lambda)]\}_{\lambda\in\mathbb{N}}$, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} \pi \neq \pi' \\ \wedge\ \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 \\ \wedge\ \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathcal{G}_{i^*_\lambda} \\ (M, \pi, \pi') \leftarrow A_\lambda(\mathsf{crs}) \end{array}\right]$$

$$\leq \sum_{i\in[k(\lambda)]} \Pr\left[\begin{array}{l} w_i \neq w'_i \\ \wedge\ \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 \\ \wedge\ \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 \end{array} : \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_i \\ (M, \pi, \pi') \leftarrow A_\lambda(\mathsf{crs}) \\ w_i = \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi) \\ w'_i = \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi') \end{array}\right] + \mu(\lambda), \qquad \diamond$$

where we define $\mathcal{G}_i := \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^{m(\lambda)}, 1^{T(\lambda)}, k(\lambda), i)$.

Remark 5 (On Weak Witness-Unambiguity). In this work, we define and achieve the above notion of witness-unambiguity. However, in some cases, it may be sufficient to satisfy the following weaker notion: If, for every index $i \in [k(\lambda)]$, the experiment above extracts out $w_i \neq w'_i$ for accepting proofs π, π' with at most negligible probability, then it must also be the case that $\pi \neq \pi'$ with at most negligible probability. At a high level, this definition is weaker since it potentially allows for multiplicative polynomial factors in the upper bound above. $\diamond$

In the rest of this seciton, we construct a witness-unambiguous seBARG from LWE with non-trivial efficiency, resulting in the following theorem.

Theorem 5.2. Assuming polynomial hardness of LWE, there exists a constant $\delta > 0$ and a $(k^{1-\delta} \cdot \mathsf{poly}(\lambda))$-succinct witness-unambiguous somewhere extractable batch argument for index languages.

To prove the theorem, we first give an intermediate construction with the following guarantees.

Lemma 5.3. Assuming polynomial hardness of LWE, there exists a witness-unambiguous somewhere extractable batch argument for index languages that on security parameter λ and batches of k statements with time bound T has CRS length, proof size, and verification time bounded by $(k + \mathsf{poly}(T)) \cdot \mathsf{poly}(\lambda)$. $\diamond$

We define the intermediate construction in Section 5.1 and then prove the necessary properties. Then we give a simple post-processing step in Section 5.2 that improves the dependence on k as required for Theorem 5.2.

5.1 Intermediate Construction

Our intermediate seBARG scheme relies on the following building blocks:

- A linear PCP scheme LPCP as specified in Lemma 3.8. We recall that the verifier $\mathcal{V}^{\mathsf{LPCP}}(x, \pi, r)$ consists of a query algorithm Q^{LPCP} and a decision algorithm D^{LPCP}.
- A somewhere extractable linearly homomorphic commitment with proofs of well-formedness scheme LHC (Definition 4.3).
- A linear collision resistant hash family CRHF (Definition 3.3) where the hash function can be represented as a binary matrix. The existence of such a CRHF is implied by LHC.

  We call its hash key generation algorithm as $\mathbf{Q} \leftarrow \mathsf{CRHF}.\mathcal{G}(1^\lambda, 1^\ell, k)$, where ℓ, k denotes input length and input value bound respectively (so inputs are vectors in $[0,k]^\ell$), and $\mathbf{Q} \in \{0,1\}^{\ell \times \mathsf{poly}(\lambda)}$. The hash evaluation is simply $\mathbf{Q} \cdot \mathbf{x}$.
- A CI hash family $\mathcal{H}$ for the ensemble of product relations $R = R_\lambda \subseteq [0,n]^{\mathsf{poly}(\lambda) \times \ell} \times \{0,1\}^{O(\ell) \times t_\lambda}$ that are time-$\mathsf{poly}(\lambda, \ell, \log n)$ product-verifiable with product sparsity at most $1 - \varepsilon$, where $t_\lambda = \lambda/\varepsilon$, and we take ε as the soundness error of the LPCP scheme. Existence of $\mathcal{H}$ is guaranteed by Theorem 3.14.

  Semantically, $\mathcal{H}$ takes commitments $C = (\mathsf{c}_1, \ldots, \mathsf{c}_\ell)$ as input and outputs t_λ-size lists of LPCP verification randomness. Let $\mathsf{hk} \leftarrow \mathcal{H}.\mathcal{G}(1^\lambda, 1^\ell, n)$ denote the hash key generation algorithm and $\mathcal{H}.\mathsf{Hash}(\mathsf{hk}, \cdot)$ the hash evaluation algorithm.

We note that all of the above building blocks can be built from the polynomial hardness of LWE. We construct our initial seBARG $\mathsf{BARG} = (\mathcal{G}_{\mathsf{barg}}, \mathcal{P}_{\mathsf{barg}}, \mathcal{V}_{\mathsf{barg}}, \mathsf{Ext}_{\mathsf{barg}})$ as follows:

- $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^m, 1^T, k, i^*)$:
  1. Take $\ell = \mathsf{poly}(T)$ as the proof length of the LPCP scheme for TMSAT instance with T steps.
  2. Sample $(\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, \mathsf{td}^{\mathsf{LHC}}) \leftarrow \mathsf{LHC}.\mathcal{G}(1^\lambda, 1^k, 1^\ell, 1^\ell, i^*)$.
  3. Sample $\mathsf{hk} \leftarrow \mathcal{H}.\mathcal{G}(1^\lambda, 1^\ell, k)$.
  4. Sample $\mathbf{Q} \leftarrow \mathsf{CRHF}.\mathcal{G}(1^\lambda, 1^\ell, k)$ and note that $\mathbf{Q} \in \{0,1\}^{\ell \times \mathsf{poly}(\lambda)}$.
  5. Out $(\mathsf{crs}, \mathsf{td})$ where $\mathsf{crs} = (\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, \mathsf{hk}, \mathbf{Q})$ and $\mathsf{td} = \mathsf{td}^{\mathsf{LHC}}$.
- $\pi = \mathcal{P}_{\mathsf{barg}}(\mathsf{crs}, M, w_1, \ldots, w_k)$:
  1. Parse crs as $(\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, \mathsf{hk}, \mathbf{Q})$.
  2. For every $i \in [k]$, prepare the i-th TMSAT instance $x_i = (M, i, T)$, and compute $\pi_i^{\mathsf{LPCP}} = \mathcal{P}^{\mathsf{LPCP}}(x_i, w_i)$.
  3. Assemble $\{\pi_i^{\mathsf{LPCP}}\}_{i \in [k]}$ row-by-row into a matrix $\Pi \in \{0,1\}^{k \times \ell}$. Compute the commitment of Π column by column: for all $j \in [\ell]$, $\mathsf{c}_j \leftarrow \mathsf{LHC}.\mathsf{Commit}(\mathsf{ck}, \Pi_{*,j})$ where $\Pi_{*,j}$ denotes the column vector $(\Pi_{1,j}, \ldots, \Pi_{k,j})$. We write $C = (\mathsf{c}_1, \ldots, \mathsf{c}_\ell) \in [0,k]^{\mathsf{poly}(\lambda) \times \ell}$.
  4. Compute the proof of well-formedness of C: $\pi^{\mathsf{LHC}} \leftarrow \mathsf{LHC}.\mathcal{P}(\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, \{\Pi_{*,j}\}_{j \in [\ell]})$.
  5. Compute LPCP verification randomness using hash key hk: $(r_1, \ldots, r_{t_\lambda}) \leftarrow \mathsf{H}.\mathsf{Hash}(\mathsf{hk}, C)$. For each $i \in [t_\lambda]$, run $Q^{\mathsf{LPCP}}(r_i)$ to generate queries $(\mathbf{q}_{i,1}, \ldots, \mathbf{q}_{i,s})$ where recall that t_λ is a parameter of the CI hash Hash and we use s to denote the query complexity of the LPCP scheme.
  6. Prepare the set of queries as $\mathcal{Q} = \{\mathbf{q}_{i,j}\}_{i \in [t_\lambda], j \in [s]} \cup \{\mathbf{q} \in \{0,1\}^\ell : \mathbf{q} \text{ is a column of } \mathbf{Q}\}$.

7. For each query $\mathbf{q} \in Q$: compute answer $\mathbf{v}_\mathbf{q} = \Pi \cdot \mathbf{q}$.
8. Output the proof $\pi = (C, \pi^{\mathsf{LHC}}, \{\mathbf{v}_\mathbf{q}\}_{\mathbf{q} \in Q})$.

- $b = \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi)$:
  1. Parse crs into $(\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, \mathsf{hk}, \mathbf{Q})$.
  2. Parse π into $(C, \pi^{\mathsf{LHC}}, \{\mathbf{v}_\mathbf{q}\}_{\mathbf{q} \in \tilde{Q}})$.
  3. Check $C \in [0, k]^{\mathsf{poly}(\lambda) \times \ell}$.
  4. Check $\mathsf{LHC}.\mathcal{V}(\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, C, \pi^{\mathsf{LHC}}) = 1$.
  5. Compute the set of queries Q in the same way as the honest prover from $\mathsf{hk}, C, \mathbf{Q}$. Check that $\tilde{Q} = Q$.
  6. For each query $\mathbf{q} \in Q$: check $\mathsf{Commit}(\mathsf{ck}, \mathbf{v}_\mathbf{q}) = C \cdot \mathbf{q}$, and $\mathbf{v}_\mathbf{q} \in [0, \ell]^k$.
  7. Compute LPCP verification randomness using hash key hk: $(r_1, \ldots, r_{t_\lambda}) \leftarrow \mathcal{H}.\mathsf{Hash}(\mathsf{hk}, C)$.
  8. For every $i \in [k]$, prepare the i-th TMSAT instance $x_i = (M, i, T)$. For each $j \in [t_\lambda]$, compute queries $(\mathbf{q}_{j,1}, \ldots, \mathbf{q}_{j,s}) \subseteq Q$ by executing $Q^{\mathsf{LPCP}}(r_j)$, and compute the state information u_j. For every $i \in [k], j \in [t_\lambda]$, check $D^{\mathsf{LPCP}}(x_i, (\mathbf{v}_{\mathbf{q}_{j,1}})_i, \ldots, (\mathbf{v}_{\mathbf{q}_{j,s}})_i, u_j) = 1$.
  9. Output 1 if all checks pass and 0 otherwise.
- $w = \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi)$:
  1. Parse $\mathsf{td} = \mathsf{td}^{\mathsf{LHC}}$.
  2. Parse π into $(C, \pi^{\mathsf{LHC}}, \{\mathbf{v}_\mathbf{q}\}_{\mathbf{q} \in \tilde{Q}})$.
  3. Compute $\pi^{\mathsf{LPCP}} = \mathsf{LHC}.\mathsf{Ext}(\mathsf{td}^{\mathsf{LHC}}, C)$.
  4. Output $w = \mathsf{LPCP}.\mathsf{Ext}(\pi^{\mathsf{LPCP}})$.

We proceed to argue completeness, efficiency, index hiding, somewhere argument of knowledge, and witness unambiguity as required for Lemma 5.3.

Completeness. We go over all checks performed by $\mathcal{V}_{\mathsf{barg}}$ to verify the completeness of our construction.

- $C \in [0, k]^{\mathsf{poly}(\lambda) \times \ell}$. This is by the small value commitment property of LHC.
- $\mathsf{LHC}.\mathcal{V}(\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, C, \pi^{\mathsf{LHC}}) = 1$. This follows from prover completeness of LHC.
- $\tilde{Q} = Q$. This step is merely recomputing query set Q from crs and C. An honest proof always passes this check.
- For each query $\mathbf{q} \in Q$, $\mathsf{Commit}(\mathsf{ck}, \mathbf{v}_\mathbf{q}) = C \cdot \mathbf{q}$, $\mathbf{v}_\mathbf{q} \in [0, \ell]^k$.

  Let $\Pi \in \{0, 1\}^{k \times \ell}$ be the matrix consists of all LPCP proofs constructed by the prover algorithm, we know
  $$C = (\mathsf{Commit}(\mathsf{ck}, \Pi_{*,1}), \ldots, \mathsf{Commit}(\mathsf{ck}, \Pi_{*,\ell}))$$

is computed on columns of Π, and $\mathbf{v}_{\mathbf{q}} = \Pi \cdot \mathbf{q}$. Since $\mathsf{Commit}(\mathsf{ck}, \cdot)$ is linear, we have

$$\mathsf{Commit}(\mathsf{ck}, \mathbf{v}_{\mathbf{q}}) = \mathsf{Commit}(\mathsf{ck}, \Pi \cdot \mathbf{q}) = \sum_{i \in [\ell]} \mathbf{q}_i \mathsf{Commit}(\mathsf{ck}, \Pi_{*,i}) = C \cdot \mathbf{q}.$$

Also, since $\Pi \in \{0,1\}^{k \times \ell}$ and $\mathbf{q} \in \{0,1\}^{\ell}$, we have $\mathbf{v}_{\mathbf{q}} \in [0, \ell]^k$.

- For every $i \in [k], j \in [t_\lambda]$, the LPCP verifier accepts with randomness r_j: for LPCP queries $(\mathbf{q}_{j,1}, \ldots, \mathbf{q}_{j,s})$ and state information u_j, we claim that $D^{\mathsf{LPCP}}(x_i, (\mathbf{v}_{\mathbf{q}_{j,1}})_i, \ldots, (\mathbf{v}_{\mathbf{q}_{j,s}})_i, u_j) = 1$.

  This is because in an honest proof, we have $(\mathbf{v}_{\mathbf{q}_{j,1}})_i = \left\langle \pi_i^{\mathsf{LPCP}}, \mathbf{q}_{j,1} \right\rangle, \ldots, (\mathbf{v}_{\mathbf{q}_{j,s}})_i = \left\langle \pi_i^{\mathsf{LPCP}}, \mathbf{q}_{j,s} \right\rangle$, where π_i^{LPCP} is the corresponding LPCP proof for x_i. Therefore, the inputs to LPCP decider are just the correct linear query answers, and it accepts by completeness of the LPCP scheme.

Efficiency. In the efficiency analysis below, we recall that ℓ is the proof length for the linear PCP scheme, so $\ell \in \mathsf{poly}(T)$.

- For the efficiency of $\mathcal{G}_{\mathsf{barg}}$, we note that it generates $(\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, \mathsf{hk}, \mathbf{Q})$ and $\mathsf{td}^{\mathsf{LHC}}$. By efficiency of LHC (Definition 4.3), $\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, \mathsf{td}^{\mathsf{LHC}}$ are generated in time $k \cdot \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda, \ell)$. By our construction, hk and $\mathbf{Q}$ are generated in time $\mathsf{poly}(\lambda, \ell)$. Therefore, $\mathcal{G}_{\mathsf{barg}}$ runs in time $k \cdot \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda, \ell) = k \cdot \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda, T)$, which also bounds $|\mathsf{crs}|$.
- For the length of π, we have that π consists of $(C, \pi^{\mathsf{LHC}}, \{\mathbf{v}_{\mathbf{q}}\}_{\mathbf{q} \in Q})$, which are of length $\ell \cdot \mathsf{poly}(\lambda), k \cdot \mathsf{poly}(\lambda), k \cdot \mathsf{poly}(\lambda)$, respectively. So $|\pi| \leq k \cdot \mathsf{poly}(\lambda) + \ell \cdot \mathsf{poly}(\lambda) = k \cdot \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda, T)$.
- For efficiency of $\mathcal{V}_{\mathsf{barg}}$, we note that:
  - Running $\mathsf{LHC}.\mathcal{V}(\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, C, \pi^{\mathsf{LHC}})$ takes time $k \cdot \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda, \ell)$.
  - Evaluating $\mathcal{H}.\mathsf{Hash}(\mathsf{hk}, C)$ takes time $\mathsf{poly}(\lambda, \ell)$, and computing Q also takes time $\mathsf{poly}(\lambda, \ell)$.
  - There are $\mathsf{poly}(\lambda)$ queries in Q, and to check $\mathbf{v}_{\mathbf{q}}$ for each $\mathbf{q} \in Q$ takes time $k \cdot \mathsf{poly}(\lambda)$.
  - Finally, $\mathcal{V}_{\mathsf{barg}}$ evaluates D^{LPCP} for $k \cdot \mathsf{poly}(\lambda)$ times, each time takes $\mathsf{poly}(\lambda)$ steps.

  So, the total running time is $k \cdot \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda, \ell) = k \cdot \mathsf{poly}(\lambda) + \mathsf{poly}(\lambda, T)$.

Index hiding. This follows immediately from the index hiding property of the LHC scheme.

Somewhere argument of knowledge. Let $\mathsf{CORRECT}(\pi)$ be the event that π is an accepting LPCP proof for $x_{i^*} = (M, i^*, T)$, where i^* is the index on which seBARG is knowledge-extractable. Our goal is to prove that

$$\Pr\left[\begin{array}{l|l} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi^*) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_{i^*} \\ \wedge \neg \mathsf{CORRECT}(\pi_{i^*}^{\mathsf{LPCP}}) & (M, \pi^*) = \mathcal{A}(\mathsf{crs}) \\ & (C^*, \pi^{\mathsf{LHC}*}, \{\mathbf{v}_{\mathbf{q}}^*\}_{\mathbf{q} \in \tilde{Q}}) = \pi^* \\ & \pi_{i^*}^{\mathsf{LPCP}} = \mathsf{LHC.Ext}(\mathsf{td}, C^*) \end{array}\right] \leq \mathsf{negl}(\lambda). \tag{2}$$

Consider the bad relation $R_{\lambda,\mathsf{td}}^{\mathsf{LPCP}}$ defined as

$$R^{\mathsf{LPCP}}_{\lambda,\mathsf{td}} = \left\{ (C = (\mathsf{c}_1, \ldots, \mathsf{c}_\ell), (r_1, \ldots, r_{t_\lambda})) \;\middle|\; \begin{array}{c} \pi^{\mathsf{LPCP}}_{i^*} = \mathsf{LHC.Ext}(\mathsf{td}, C) \\ \neg \mathsf{CORRECT}(\pi^{\mathsf{LPCP}}_{i^*}) \wedge \\ \forall i \in [t_\lambda], \mathcal{V}^{\mathsf{LPCP}}(x_{i^*}, \pi^{\mathsf{LPCP}}_{i^*}, r_i) = 1 \end{array} \right\}.$$

Note that $R^{\mathsf{LPCP}}_{\lambda,\mathsf{td}}$ can be product-verified in $\mathsf{poly}(\lambda, \ell)$ time with td, x_{i^*} hard-coded in, as both LHC extraction and LPCP verification are efficient; and the product-sparsity of $R^{\mathsf{LPCP}}_{\lambda,\mathsf{td}}$ is implied by the soundness of the LPCP scheme.

Therefore, the CI hash $\mathcal{H}$ is correlation intractable for $R^{\mathsf{LPCP}}_{\lambda,\mathsf{td}}$. Then we have

$$\Pr\left[\begin{array}{l} \neg \mathsf{CORRECT}(\pi^{\mathsf{LPCP}}_{i^*}) \wedge \\ \forall i \in [t_\lambda], \mathcal{V}^{\mathsf{LPCP}}(x_{i^*}, \pi^{\mathsf{LPCP}}_{i^*}, r_i) = 1 \end{array} : \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_{i^*} \\ (M, \pi^*) = \mathcal{A}(\mathsf{crs}) \\ (C^*, \pi^{\mathsf{LHC}*}, \{\mathbf{v}^*_{\mathbf{q}}\}_{\mathbf{q} \in \tilde{Q}}) = \pi^* \\ \pi^{\mathsf{LPCP}}_{i^*} = \mathsf{LHC.Ext}(\mathsf{td}, C^*) \\ (r_1, \ldots, r_{t_\lambda}) \leftarrow \mathcal{H}.\mathsf{Hash}(\mathsf{hk}, C^*) \end{array} \right] \leq \mathsf{negl}(\lambda). \tag{3}$$

Inequality (3) means that if we extract an incorrect proof from C^*, then it cannot pass the LPCP verifications using randomness computed from $\mathcal{H}.\mathsf{Hash}(\mathsf{hk}, C^*)$. In our protocol, LPCP verification is emulated by using $(\mathbf{v}_{\mathbf{q}})_{i^*}$ to replace LPCP query answer $\pi^{\mathsf{LPCP}}_{i^*} \cdot \mathbf{q}$. In the following claim, we show the replacement is valid, by proving that even for a malicious prover, we have $(\mathbf{v}_{\mathbf{q}})_{i^*} = \pi^{\mathsf{LPCP}}_{i^*} \cdot \mathbf{q}$.

Claim 5.4. For any non-uniform polynomial-size adversary $\mathcal{A}$ and polynomially bounded $k = k(\lambda), m = m(\lambda), T = T(\lambda)$, assuming LHC is secure for input parameters at most $\max(k, \ell)$, then for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi^*) = 1 \\ \wedge \exists \mathbf{q} \in Q, (\mathbf{v}^*_{\mathbf{q}})_{i^*} \neq \pi^{\mathsf{LPCP}}_{i^*} \cdot \mathbf{q} \end{array} : \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_{i^*} \\ (M, \pi^*) = \mathcal{A}(\mathsf{crs}) \\ (C^*, \pi^{\mathsf{LHC}*}, \{\mathbf{v}^*_{\mathbf{q}}\}_{\mathbf{q} \in \tilde{Q}}) = \pi^* \\ \pi^{\mathsf{LPCP}}_{i^*} = \mathsf{LHC.Ext}(\mathsf{td}, C^*) \end{array} \right] \leq \mathsf{negl}(\lambda). \qquad \diamond$$

Proof. Over the same probability space, we show that

$$\begin{aligned} &\Pr\left[\begin{array}{l} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi^*) = 1 \\ \wedge \exists \mathbf{q} \in Q, (\mathbf{v}^*_{\mathbf{q}})_{i^*} \neq \pi^{\mathsf{LPCP}}_{i^*} \cdot \mathbf{q} \end{array} \right] \\ \leq &\Pr\left[\begin{array}{l} \mathsf{LHC}.\mathcal{V}(\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, C^*, \pi^{\mathsf{LHC}*}) = 1 \\ \wedge \forall \mathbf{q} \in Q, \mathsf{LHC.Commit}(\mathsf{ck}, \mathbf{v}^*_{\mathbf{q}}) = C^* \cdot \mathbf{q}, \mathbf{v}^*_{\mathbf{q}} \in \{0, \ldots, \ell\}^k \\ \wedge \exists \mathbf{q} \in Q, (\mathbf{v}^*_{\mathbf{q}})_{i^*} \neq \pi^{\mathsf{LPCP}}_{i^*} \cdot \mathbf{q} \end{array} \right] \\ \leq &\Pr\left[\begin{array}{l} \forall \mathbf{q} \in Q, \mathsf{LHC.Ext}(\mathsf{td}, C^* \cdot \mathbf{q}) = \pi^{\mathsf{LPCP}}_{i^*} \cdot \mathbf{q} \\ \wedge \forall \mathbf{q} \in Q, \mathsf{LHC.Commit}(\mathsf{ck}, \mathbf{v}^*_{\mathbf{q}}) = C^* \cdot \mathbf{q}, \mathbf{v}^*_{\mathbf{q}} \in \{0, \ldots, \ell\}^k \\ \wedge \exists \mathbf{q} \in Q, (\mathbf{v}^*_{\mathbf{q}})_{i^*} \neq \pi^{\mathsf{LPCP}}_{i^*} \cdot \mathbf{q} \end{array} \right] + \mathsf{negl}(\lambda) \\ \leq &\Pr\left[\begin{array}{l} \forall \mathbf{q} \in Q, (\mathbf{v}^*_{\mathbf{q}})_{i^*} = \pi^{\mathsf{LPCP}}_{i^*} \cdot \mathbf{q} \\ \wedge \exists \mathbf{q} \in Q, (\mathbf{v}^*_{\mathbf{q}})_{i^*} \neq \pi^{\mathsf{LPCP}}_{i^*} \cdot \mathbf{q} \end{array} \right] + \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda). \end{aligned}$$

Here the first inequality follows from the construction of $\mathcal{V}_{\mathsf{barg}}$; the second inequality follows from the linearly homomorphic extraction with proof property of LHC; the third inequality follows

from somewhere extraction correctness of LHC, i.e. we have $\pi_{i^*}^{\mathsf{LPCP}} \cdot \mathbf{q} = \mathsf{LHC.Ext}(\mathsf{td}, C^* \cdot \mathbf{q}) = \mathsf{LHC.Ext}(\mathsf{td}, \mathsf{LHC.Commit}(\mathsf{ck}, \mathbf{v}_{\mathbf{q}}^*)) = (\mathbf{v}_{\mathbf{q}}^*)_{i^*}$. □

Now to analyze Equation (2), we have that

$$\begin{aligned}
&\Pr\left[\begin{array}{l}\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi^*) = 1 \\ \wedge \neg \mathsf{CORRECT}(\pi_{i^*}^{\mathsf{LPCP}})\end{array}\right] \\
&\leq \Pr\left[\begin{array}{l}\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi^*) = 1 \\ \wedge \exists i \in [t_\lambda], \mathcal{V}^{\mathsf{LPCP}}(x_{i^*}, \pi_{i^*}^{\mathsf{LPCP}}, r_i) = 0\end{array}\right] + \mathsf{negl}(\lambda) \quad \text{(By inequality 3).}
\end{aligned}$$

Then by Claim 5.4, we know that for every query $\mathbf{q} \in Q$, except with negligible probability it holds that $(\mathbf{v}_{\mathbf{q}}^*)_{i^*} = \pi_{i^*}^{\mathsf{LPCP}} \cdot \mathbf{q}$. Using this, we upper bound the above for any such $j \in [t_\lambda]$ such that $\mathcal{V}^{\mathsf{LPCP}}(x_{i^*}, \pi_{i^*}^{\mathsf{LPCP}}, r_j) = 0$ and letting $(\mathbf{q}_{j,1}, \ldots, \mathbf{q}_{j,s}, u_j) = Q^{\mathsf{LPCP}}(r_j)$.

$$\begin{aligned}
&\leq \Pr\left[\begin{array}{l}\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi^*) = 1 \\ \wedge D^{\mathsf{LPCP}}(x_{i^*}, \pi_{i^*}^{\mathsf{LPCP}} \cdot \mathbf{q}_{j,1}, \ldots, \pi_{i^*}^{\mathsf{LPCP}} \cdot \mathbf{q}_{j,s}, u_j) = 0 \\ \wedge D^{\mathsf{LPCP}}(x_{i^*}, (\mathbf{v}_{\mathbf{q}_{j,1}}^*)_{i^*}, \ldots, (\mathbf{v}_{\mathbf{q}_{j,s}}^*)_{i^*}, u_j) = 1\end{array}\right] + \mathsf{negl}(\lambda) \\
&\leq \Pr\left[\begin{array}{l}\forall \mathbf{q} \in Q, (\mathbf{v}_{\mathbf{q}}^*)_{i^*} = \pi_{i^*}^{\mathsf{LPCP}} \cdot \mathbf{q} \\ \wedge D^{\mathsf{LPCP}}(x_{i^*}, \pi_{i^*}^{\mathsf{LPCP}} \cdot \mathbf{q}_{j,1}, \ldots, \pi_{i^*}^{\mathsf{LPCP}} \cdot \mathbf{q}_{j,s}, u_j) = 0 \\ \wedge D^{\mathsf{LPCP}}(x_{i^*}, (\mathbf{v}_{\mathbf{q}_{j,1}}^*)_{i^*}, \ldots, (\mathbf{v}_{\mathbf{q}_{j,s}}^*)_{i^*}, u_j) = 1\end{array}\right] + \mathsf{negl}(\lambda) \\
&\leq \mathsf{negl}(\lambda),
\end{aligned}$$

as required for somewhere argument of knowledge.

Witness-unambiguity. We write the two proofs generated by the adversary as $\pi = (C, \pi^{\mathsf{LHC}}, \{\mathbf{v}_{\mathbf{q}}\}_{\mathbf{q} \in Q})$ and $\pi' = (C', \pi^{\mathsf{LHC}\prime}, \{\mathbf{v}'_{\mathbf{q}}\}_{\mathbf{q} \in Q'})$. (Note that Q is uniquely determined by crs and C, and Q' is uniquely determined by crs and C'.) In the following, we prove that

$$\begin{aligned}
&\Pr\left[\begin{array}{l}\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 \\ \wedge \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 \\ \wedge \pi \neq \pi'\end{array} : \begin{array}{l}\mathsf{crs} \leftarrow \mathcal{G}_{i^*} \\ (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs})\end{array}\right] \\
&\leq \sum_{i=1}^{k} \Pr\left[\begin{array}{l}\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 \\ \wedge \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 \\ \wedge \pi_i^{\mathsf{LPCP}} \neq \pi_i^{\mathsf{LPCP}\prime}\end{array} : \begin{array}{l}(\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_i \\ (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \\ \pi_i^{\mathsf{LPCP}} \leftarrow \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi) \\ \pi_i^{\mathsf{LPCP}\prime} \leftarrow \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi')\end{array}\right] + \mathsf{negl}(\lambda).
\end{aligned}$$

where we define $\mathcal{G}_i := \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^m, 1^T, k, i)$ for any $i \in [k]$ for simplicity. The rest follows from inequality (2) and the fact that in our LPCP scheme, each witness corresponds to at most one correct LPCP proof (see Lemma 3.8).

We first show that if $C = C'$, then the remaining parts of the proof should be the same.

Claim 5.5. For any non-uniform polynomial-time adversary $\mathcal{A}$, polynomially bounded values $k = k(\lambda), m = m(\lambda), T = T(\lambda)$ and $i^* = i^*(\lambda) \in [k(\lambda)]$, there exists a negligible function $\mu(\cdot)$ such that for any $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l}\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 \\ \wedge \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 \\ \wedge C = C' \\ \wedge \pi \neq \pi'\end{array} : \begin{array}{l}\mathsf{crs} \leftarrow \mathcal{G}_{i^*} \\ (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs})\end{array}\right] \leq \mu(\lambda).$$

◇

Proof. The seBARG proof π consists of three parts: $(C, \pi^{\mathsf{LHC}}, \{\mathbf{v_q}\}_{\mathbf{q}\in Q})$. The second part of the two proofs is the same with high probability as the LHC proof satisfies unambiguity, so

$$\begin{aligned}
&\Pr\left[\begin{array}{l|ll} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 & & \\ \wedge \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 & : & \mathsf{crs} \leftarrow \mathcal{G}_{i^*} \\ \wedge C = C' & & (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge \pi^{\mathsf{LHC}} \neq \pi^{\mathsf{LHC}\prime} & & \end{array}\right] \\
\leq &\Pr\left[\begin{array}{l|ll} \mathsf{LHC}.\mathcal{V}(\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, C, \pi^{\mathsf{LHC}}) = 1 & & \\ \wedge \mathsf{LHC}.\mathcal{V}(\mathsf{ck}, \mathsf{crs}^{\mathsf{LHC}}, C, \pi^{\mathsf{LHC}\prime}) = 1 & : & \mathsf{crs} \leftarrow \mathcal{G}_{i^*} \\ \wedge \pi^{\mathsf{LHC}} \neq \pi^{\mathsf{LHC}\prime} & & (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).
\end{aligned}$$

For the third part, we rely on the fact that $\mathsf{Commit}(\mathsf{ck}, \cdot)$ is collision-resistant for short vectors.

$$\begin{aligned}
&\Pr\left[\begin{array}{l|ll} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 & & \\ \wedge \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 & : & \mathsf{crs} \leftarrow \mathcal{G}_{i^*} \\ \wedge C = C' & & (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge \left\{\mathbf{v_q}\right\}_{\mathbf{q}\in Q} \neq \left\{\mathbf{v'_q}\right\}_{\mathbf{q}\in Q} & & \end{array}\right] \\
\leq &\Pr\left[\begin{array}{l|ll} \forall \mathbf{q} \in Q, \mathsf{Commit}(\mathsf{ck}, \mathbf{v_q}) = C \cdot \mathbf{q} = \mathsf{Commit}(\mathsf{ck}, \mathbf{v'_q}) & & \\ \wedge \forall \mathbf{q} \in Q, \mathbf{v_q}, \mathbf{v'_q} \in \{0, \ldots, \ell\}^k & : & \mathsf{crs} \leftarrow \mathcal{G}_{i^*} \\ \wedge \left\{\mathbf{v_q}\right\}_{\mathbf{q}\in Q} \neq \left\{\mathbf{v'_q}\right\}_{\mathbf{q}\in Q} & & (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \\
\leq &\mathsf{negl}(\lambda).
\end{aligned}$$

As we assumed $C = C'$ in the event for the statement of the claim, this covers all cases where $\pi \neq \pi'$, so the claim follows. □

Given the above claim, it suffices to argue that C, C' must be equal unless different witnesses are extracted. Recall that in our construction, the crs contains $\mathbf{Q} \in \{0,1\}^{\ell \times \mathsf{poly}(\lambda)}$ that is a linear CRHF for short vectors, and we force both Q and Q' to contain all column vectors in $\mathbf{Q}$. By collision resistance of $\mathbf{Q}$, as $C, C' \in [0,k]^{\mathsf{poly}(\lambda)\times\ell}$, it is computationally hard to have both $C \neq C'$ and $C \cdot \mathbf{Q} = C' \cdot \mathbf{Q}$, and therefore having $C \neq C'$ implies the existence of some $\mathbf{q} \in Q \cap Q'$ as a column vector of $\mathbf{Q}$, that $C \cdot \mathbf{q} \neq C' \cdot \mathbf{q}$. This implies that, by the correctness of proofs, $\mathsf{Commit}(\mathsf{ck}, \mathbf{v_q}) = C \cdot \mathbf{q} \neq C' \cdot \mathbf{q} = \mathsf{Commit}(\mathsf{ck}, \mathbf{v'_q})$, and furthermore $\mathbf{v_q} \neq \mathbf{v'_q}$. We use this to give a bound in the case where $C \neq C'$ in the following claim.

Claim 5.6. For any non-uniform polynomial-time adversary $\mathcal{A}$, polynomially bounded $k = k(\lambda), m = m(\lambda), T = T(\lambda)$, and index $i^* = i^*(\lambda) \in [k]$, there exists a negligible function μ such that for any $\lambda \in \mathbb{N}$, it holds that

$$\begin{aligned}
&\Pr\left[\begin{array}{l|ll} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 & & \\ \wedge \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 & : & \mathsf{crs} \leftarrow \mathcal{G}_{i^*} \\ \wedge C \neq C' & & (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \\
\leq &\Pr\left[\begin{array}{l|ll} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 & & \\ \wedge \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 & : & \mathsf{crs} \leftarrow \mathcal{G}_{i^*} \\ \wedge \exists \mathbf{q} \in Q \cap Q', \mathbf{v_q} \neq \mathbf{v'_q} & & (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] + \mu(\lambda). \qquad \diamond
\end{aligned}$$

Proof. As $\mathbf{Q}$ is a linear CRHF for vectors in $\{0,\ldots,k\}^{\ell}$, we have

$$\Pr\left[\; C \neq C' \wedge C\cdot\mathbf{Q} = C'\cdot\mathbf{Q} \;:\; \begin{array}{l}\mathsf{crs}\leftarrow\mathcal{G}_{i^*}\\ (M,\pi,\pi')\leftarrow\mathcal{A}(\mathsf{crs})\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Therefore,

$$\begin{aligned}
&\Pr\left[\begin{array}{l}\mathcal{V}_{\mathsf{barg}}(\mathsf{crs},M,\pi)=1\\ \wedge\mathcal{V}_{\mathsf{barg}}(\mathsf{crs},M,\pi')=1\\ \wedge C\neq C'\end{array} : \begin{array}{l}\mathsf{crs}\leftarrow\mathcal{G}_{i^*}\\ (M,\pi,\pi')\leftarrow\mathcal{A}(\mathsf{crs})\end{array}\right]\\
\leq\ &\Pr\left[\begin{array}{l}\mathcal{V}_{\mathsf{barg}}(\mathsf{crs},M,\pi)=1\\ \wedge\mathcal{V}_{\mathsf{barg}}(\mathsf{crs},M,\pi')=1\\ \wedge C\cdot\mathbf{Q}\neq C'\cdot\mathbf{Q}\end{array} : \begin{array}{l}\mathsf{crs}\leftarrow\mathcal{G}_{i^*}\\ (M,\pi,\pi')\leftarrow\mathcal{A}(\mathsf{crs})\end{array}\right] + \mathsf{negl}(\lambda)\\
\leq\ &\Pr\left[\begin{array}{l}\mathcal{V}_{\mathsf{barg}}(\mathsf{crs},M,\pi)=1\\ \wedge\mathcal{V}_{\mathsf{barg}}(\mathsf{crs},M,\pi')=1\\ \exists\mathbf{q}\in Q\cap Q', C\cdot\mathbf{q}\neq C'\cdot\mathbf{q}\end{array} : \begin{array}{l}\mathsf{crs}\leftarrow\mathcal{G}_{i^*}\\ (M,\pi,\pi')\leftarrow\mathcal{A}(\mathsf{crs})\end{array}\right] + \mathsf{negl}(\lambda).
\end{aligned}$$

The last step follows from the fact that, when π,π' are both accepted, $\mathbf{v}_{\mathbf{q}} = \mathbf{v}'_{\mathbf{q}}$ implies $C\cdot\mathbf{q} = \mathsf{Commit}(\mathsf{ck},\mathbf{v}_{\mathbf{q}}) = \mathsf{Commit}(\mathsf{ck},\mathbf{v}'_{\mathbf{q}}) = C'\cdot\mathbf{q}$. □

Given $\mathbf{v}_{\mathbf{q}} \neq \mathbf{v}'_{\mathbf{q}}$, we check the event that $(\mathbf{v}_{\mathbf{q}})_{i^*} \neq (\mathbf{v}'_{\mathbf{q}})_{i^*}$ for every possible index i. It follows from the index hiding that

$$\begin{aligned}
&\Pr\left[\begin{array}{l}\mathcal{V}_{\mathsf{barg}}(\mathsf{crs},M,\pi)=1\\ \wedge\mathcal{V}_{\mathsf{barg}}(\mathsf{crs},M,\pi')=1\\ \wedge\exists\mathbf{q}\in Q\cap Q', \mathbf{v}_{\mathbf{q}}\neq\mathbf{v}'_{\mathbf{q}}\end{array} : \begin{array}{l}\mathsf{crs}\leftarrow\mathcal{G}_{i^*}\\ (M,\pi,\pi')\leftarrow\mathcal{A}(\mathsf{crs})\end{array}\right]\\
\leq\ &\sum_{i=1}^{k}\Pr\left[\begin{array}{l}\mathcal{V}_{\mathsf{barg}}(\mathsf{crs},M,\pi)=1\\ \wedge\mathcal{V}_{\mathsf{barg}}(\mathsf{crs},M,\pi')=1\\ \wedge\exists\mathbf{q}\in Q\cap Q', (\mathbf{v}_{\mathbf{q}})_i\neq(\mathbf{v}'_{\mathbf{q}})_i\end{array} : \begin{array}{l}\mathsf{crs}\leftarrow\mathcal{G}_{i}\\ (M,\pi,\pi')\leftarrow\mathcal{A}(\mathsf{crs})\end{array}\right] + \mathsf{negl}(\lambda).
\end{aligned}$$

Finally, by Claim 5.4, as the two proofs π,π' are accepted, we have $\pi_i^{\mathsf{LPCP}}\cdot\mathbf{q} = (\mathbf{v}_{\mathbf{q}})_i$ and $\pi_i^{\mathsf{LPCP}'}\cdot\mathbf{q} = (\mathbf{v}'_{\mathbf{q}})_i$. From $(\mathbf{v}_{\mathbf{q}})_i \neq (\mathbf{v}'_{\mathbf{q}})_i$, we have $\pi_i^{\mathsf{LPCP}}\cdot\mathbf{q} \neq \pi_i^{\mathsf{LPCP}'}\cdot\mathbf{q}$, so $\pi_i^{\mathsf{LPCP}} \neq \pi_i^{\mathsf{LPCP}'}$. Recall this suffices for witness unambiguity since each witness corresponds to at most one correct proof for the linear PCP.

5.2 Improving the Dependence on k

To finish the proof of Theorem 5.2, we show how to post-process the intermediate seBARG of Lemma 5.3 to obtain verification efficiency with sub-linear dependence on k. The high-level idea is to use the underlying seBARG on $k^{1-\delta}$ many new statements each consisting of a group of k^{δ} original statements, for some constant $\delta > 0$.

Proof (Theorem 5.2). Recall that we want to prove that there exist witness $w_1,\ldots,w_k$ such that $M(i,w_i) = 1$ for all $i \in [k]$. Let m be a bound on the witness size and T be a bound on the running time of M.

For the full construction, we use our intermediate construction in a black-box way by grouping k^{δ} claims together for some $\delta > 0$. Namely, the BARG proves $k^{1-\delta}$ new statements where each

statement is composed of k^δ original claims. After the transform, the number of new statements is reduced to $k^{1-\delta}$ each of witness size $m \cdot k^\delta$ and verification time $T \cdot k^\delta$. Recall that in our intermediate construction, there exists a constant $c > 0$ such that the various succinctness parameters gros as $(k + T^c) \cdot \lambda^c$. If we choose $\delta = 1/(c+1)$, then it holds that the resulting seBARG with groups statements has efficiency

$$(k^{1-1/(c+1)} + (k^{1/(c+1)}T)^c) \cdot \lambda^c = k^{c/(c+1)} \cdot T^c \cdot \lambda^c = k^{1-\delta} \cdot \mathsf{poly}(\lambda) \cdot \mathsf{poly}(T),$$

as required.

It is straightforward to see that this transformation preserves completeness, index hiding, and somewhere argument of knowledge. It remains to show that witness-unambiguity is preserved after batching any number $b = b(\lambda)$ statements together as follows.

$$\begin{aligned}
&\Pr\left[\begin{array}{l|ll} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 & & \mathsf{crs} \leftarrow \mathcal{G}_{i^*} \\ \wedge \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 & : & (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge \pi \neq \pi' & & \end{array}\right] \\
&\leq \sum_{i \in [k/b]} \Pr\left[\begin{array}{l|ll} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 & & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_i \\ \wedge \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 & : & (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge w_i \neq w'_i & & w_i = (w_{i,1}, \ldots, w_{i,b}) \leftarrow \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi) \\ & & w'_i = (w'_{i,1}, \ldots, w'_{i,b}) \leftarrow \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi') \end{array}\right] + \mathsf{negl}(\lambda) \\
&\leq \sum_{i \in [k/b], j \in [b]} \Pr\left[\begin{array}{l|ll} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 & & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_i \\ \wedge \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 & : & (M, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge w_{i,j} \neq w'_{i,j} & & w_i = (w_{i,1}, \ldots, w_{i,b}) \leftarrow \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi) \\ & & w'_i = (w'_{i,1}, \ldots, w'_{i,b}) \leftarrow \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi') \end{array}\right] + \mathsf{negl}(\lambda),
\end{aligned}$$

where the first inequality follows from witness-unambiguity of the original BARG scheme before batching, and the second inequality is by the fact that if $w_i \neq w'_i$, then for at least one $j \in [b]$, $w_{i,j} \neq w'_{i,j}$.

Thus, all required properties hold for the final construction, completing the proof of the theorem. □

6 Unambiguous and Flexible RAM Delegation

In this section, we prove unambiguity for a variant of the construction of flexible RAM delegation with partial input soundness from [KLVW23]. In Section 6.1, we first define the new notion of RAM computation we use to fomalize the flexible RAM delegation properties we achieve. Then in Section 6.2, we define and prove a necessary property of hash trees that we use in our construction. Specifically, we show that hash roots can be computed in an incremental and unambiguous way. Finally, we give the construction and analysis of our flexible RAM delegation scheme in Section 6.3.

6.1 Flexible RAM Computation and Delegation Definitions

We adopt a definition of flexible RAM delegation with partial input soundness following [KLVW23; PP22] with the additional property of *unambiguity*. In particular, we modify the

definition of [KLVW23] to handle potentially large space RAM computation as in [PP22], which is based on RAM delegation techniques introduced by [KP16]. We start by defining our model of flexible RAM computations.

We now consider flexible RAM computations where the machine M has random access to two different memory tapes: a working read-write memory D, and an external read-only witness w. In each step, the RAM computation also reads a bit b^{wit} from the witness w and specifies a next location ℓ^{wit} to read, so a single step now has the syntax

$$(\mathsf{rst}_{\mathsf{new}}, \ell^{\mathsf{mem}}, \ell^{\mathsf{wrt}}, b^{\mathsf{wrt}}, \ell^{\mathsf{wit}}) = \mathsf{step}(M, \mathsf{rst}, b^{\mathsf{mem}}, b^{\mathsf{wit}}).$$

A flexible RAM computation $M^D(w)$ takes as input the read-only witness w and uses step to evolve the computation, and we write $M(x, w)$ to denote the computation $M^{D_x}(w)$ where D_x is initialized with the input x followed by all zeroes. For a machine M, we define a flexible RAM configuration $\mathsf{cf} = (\mathsf{rst}, \ell^{\mathsf{mem}}, \ell^{\mathsf{wit}}, D)$ to include the witness location ℓ^{wit} at each step. The language of partial computations $\mathcal{T}_M$ contains $(\mathsf{cf}, \mathsf{cf}', w, t)$ if and only if the machine M starting at configuration cf transitions to configuration cf′ after t steps when reading from witness w.

At a high level, a prover for a RAM delegation scheme convinces a verifier that $(\mathsf{cf}, \mathsf{cf}', w, t) \in \mathcal{T}_M$ for some RAM computation M. Rather than read the full configuration, the verifier instead receives a digest of the above statement relative to some hash tree HT. Namely, it receives as input $(\mathsf{d}, \mathsf{d}', \mathsf{rt}, t)$ where $\mathsf{d}, \mathsf{d}', \mathsf{rt}$ are short digests of $\mathsf{cf}, \mathsf{cf}', w$, respectively.

In a flexible RAM delegation scheme, the prover has access to a set of *potential* inputs $x_1, \ldots, x_k$ that a machine M may access via its external read-only witness tape. For a given subset $I \subseteq [k]$, it proves to the verifier that $(\mathsf{cf}, \mathsf{cf}', x_I, t) \in \mathcal{T}_M$ where $x_I = (x_i)_{i \in I}$. The verifier checks the proof given only a root of a fully local somewhere extractable hash tree over $(x_1, \ldots, x_k)$, and soundness is guaranteed relative to extracted inputs x_I.

We define flexible RAM delegation below relative to a fixed machine M rather than allowing the prover to specify M as in Definition 3.21. Our construction only achieves this weaker property, but this will suffice for our later applications.

Definition 6.1. Let $L \colon \mathbb{N}^2 \to \mathbb{N}$ and $\mathsf{HT} = (\mathcal{G}_{\mathsf{ht}}, \mathcal{H}_{\mathsf{ht}}, \mathcal{D}_{\mathsf{ht}}, \mathcal{P}_{\mathsf{ht}}, \mathcal{V}_{\mathsf{ht}}, \mathcal{E}_{\mathsf{ht}})$ be a fully local somewhere extractable hash tree and M a RAM program. An *L-succinct unambiguous flexible RAM delegation scheme for an flSEH* HT *and machine M* consists of a tuple of algorithms $(\mathcal{G}_{\mathsf{del}}, \mathcal{D}_{\mathsf{del}}, \mathcal{P}_{\mathsf{del}}, \mathcal{V}_{\mathsf{del}})$ with the following syntax:

- $\mathcal{G}_{\mathsf{del}}(1^\lambda, T) \to \mathsf{crs}$. This is a randomized algorithm that takes as input a security parameter λ and a time bound T, and outputs a common reference string crs. We implicitly assume that crs includes 1^λ and T.

- $\mathcal{D}_{\mathsf{del}}(\mathsf{hk}, \mathsf{cf}) \to \mathsf{d}$. This is a deterministic algorithm that takes as input a hash key hk and a configuration cf and outputs a digest d. We assume for simplicity that on input a configuration $\mathsf{cf} = (\mathsf{rst}, \ell^{\mathsf{mem}}, \ell^{\mathsf{wit}}, D)$, $\mathcal{D}_{\mathsf{del}}$ outputs $\mathsf{d} = (\mathsf{rst}, \ell^{\mathsf{mem}}, \ell^{\mathsf{wit}}, \mathsf{rt}_D)$ for $\mathsf{rt}_D = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, D))$.

- $\mathcal{P}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, (\mathsf{cf}, \mathsf{cf}', t), (x_1, \ldots, x_k), I) \to \pi$. This is a deterministic algorithm that takes as input a common reference string crs, a hash key hk, a statement $(\mathsf{cf}, \mathsf{cf}', t)$ where cf, cf′ are configurations for M and t is a number of steps, inputs $(x_1, \ldots, x_k)$, and a set of indices $I \subseteq [k]$, and outputs a proof π.

- $\mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, (\mathsf{d}, \mathsf{d}', t), \mathsf{rt}, I, \pi) \to b$. This is a deterministic algorithm that takes as input the common reference string crs, a hash key hk, a hashed statement $(\mathsf{d}, \mathsf{d}', t)$ where d, d′ are digests

and t is a number of steps, a hash tree root rt, a set of indices $I \subseteq [k]$, and a proof π, and outputs a bit $b \in \{0,1\}$ indicating whether to accept or reject the proof.

We require that $(\mathcal{G}_{\mathsf{del}}, \mathsf{del}, \mathcal{P}_{\mathsf{del}}, \mathcal{V}_{\mathsf{del}})$ satisfy the following properties for a valid fully local somewhere extractable hash tree HT:

- **Completeness.** For any $\lambda, m, k, T \in \mathbb{N}$, subsets of indices $I, I^* \subseteq [k]$, configurations $\mathsf{cf}, \mathsf{cf}'$, number of steps $t \leq T$, and inputs $(x_1, \ldots, x_k) \in (\{0,1\}^m)^k$ such that $(\mathsf{cf}, \mathsf{cf}', x_I, t) \in \mathcal{T}_M$, it holds that

$$\Pr\left[\mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, (M, \mathsf{d}, \mathsf{d}', t), \mathsf{rt}, I, \pi) = 1 \;:\; \begin{array}{l} \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda, T) \\ (\mathsf{hk}, \cdot) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, m, k, I^*) \\ \pi = \mathcal{P}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, (\mathsf{cf}, \mathsf{cf}', t), (x_1, \ldots, x_k), I) \\ \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}, \mathsf{cf}), \mathsf{d}' = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{cf}') \\ \mathsf{rt} = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k))) \end{array}\right] = 1.$$

- **Efficiency and L-Succinctness** There exists a polynomial q such that the following hold in the completeness experiment above for $m, k \leq 2^\lambda$ such that M actually uses a working memory of size s and halts within t steps for $s, t \leq 2^\lambda$:
  - $\mathcal{G}_{\mathsf{del}}$ runs in time $L(t, \lambda)$.
  - $\mathcal{P}_{\mathsf{del}}$ runs in time $q(\lambda + k + t + s)$ and outputs a proof of size $|I| \cdot L(t, \lambda)$.
  - $\mathcal{V}_{\mathsf{del}}$ runs in time $|I| \cdot L(t, \lambda)$. ◊

- **Partial-Input Soundness.** For any pair of non-uniform polynomial-time adversaries $(A_1, A_2) = \{(A_{1,\lambda}, A_{2,\lambda})\}_{\lambda \in \mathbb{N}}$ and polynomials T, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} \mathsf{d}' \neq \mathcal{D}_{\mathsf{del}}(\mathsf{hk}, \mathsf{cf}') \\ \wedge\, \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi) = 1 \\ \wedge\, t \leq T(\lambda) \end{array} \;:\; \begin{array}{l} \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda, T(\lambda)) \\ (1^m, 1^k, I) \leftarrow A_{1,\lambda}(\mathsf{crs}) \\ (\mathsf{hk}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, m, k, I) \\ (\mathsf{cf}, \mathsf{d}', t, \mathsf{h}, \pi) \leftarrow A_{2,\lambda}(\mathsf{crs}, \mathsf{hk}) \\ \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}, \mathsf{cf}) \\ z = (M, \mathsf{d}, \mathsf{d}', t) \\ \mathsf{rt} = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{h}) \\ x_I = \mathcal{E}_{\mathsf{ht}}(\mathsf{td}, \mathsf{h}) \\ (\mathsf{cf}, \mathsf{cf}', x_I, t) \in \mathcal{T}_M \end{array}\right] \leq \mu(\lambda).$$

- **Unambiguity.** For any pair of non-uniform polynomial-time adversaries $(A_1, A_2) = \{(A_{1,\lambda}, A_{2,\lambda})\}_{\lambda \in \mathbb{N}}$ and polynomials T, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} \pi \neq \pi' \\ \wedge\, \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi) = 1 \\ \wedge\, \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi') = 1 \\ \wedge\, t \leq T(\lambda) \end{array} \;:\; \begin{array}{l} \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda, T(\lambda)) \\ (1^m, 1^k, I) \leftarrow A_{1,\lambda}(\mathsf{crs}) \\ (\mathsf{hk}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, m, k, I) \\ (\mathsf{cf}, \mathsf{d}', t, \mathsf{h}, \pi, \pi') \leftarrow A_{2,\lambda}(\mathsf{crs}, \mathsf{hk}) \\ \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}, \mathsf{cf}) \\ z = (M, \mathsf{d}, \mathsf{d}', t) \\ \mathsf{rt} = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{h}) \end{array}\right] \leq \mu(\lambda).$$

We note that a flexible RAM delegation scheme implies a RAM delegation scheme by considering empty external memory.

6.2 Incremental and Unambiguous Computation of Hash Roots

We show that there are hash trees where the roots can be computed in an incremental and unambiguous way, which we formalize below.

Definition 6.2. Let $(\mathsf{Gen}_f, \mathsf{Eval}_f)$ be an efficiently computable multi-input function in the CRS model. We say that $(\mathsf{Gen}_f, \mathsf{Eval}_f)$ can be computed incrementally and unambiguously if there exist a tuple of algorithms $(\mathsf{Upd}_f, \mathsf{Prove}_f, \mathsf{Ver}_f)$ with the following syntax:

- $\mathsf{Upd}_f(\mathsf{crs}, \mathsf{st}, x) \to \mathsf{st}'$. This is a deterministic algorithm that takes as input a common reference string crs, a state st, and an input x and outputs a new state st'.
- $\mathsf{Prove}_f(\mathsf{crs}, (x_1, \ldots, x_k), \mathsf{st}, i) \to \pi$. This is a deterministic algorithm that takes as input a common reference string crs, a sequence of inputs $(x_1, \ldots, x_k)$, a state st, an index i, and outputs a proof π.
- $\mathsf{Ver}_f(\mathsf{crs}, y, \mathsf{st}, i, \pi) \to b$. This is a deterministic algorithm that takes as input a common reference string crs, an output y, a state st, an index i, and a proof π, and outputs a bit b indicating whether to accept or reject the current state.

We require that $(\mathsf{Upd}_f, \mathsf{Prove}_f, \mathsf{Ver}_f)$ satisfy the following properties:

- **Correctness and Completeness.** For any $\lambda, k \in \mathbb{N}$, sequence of inputs $(x_1, \ldots, x_k) \in (\{0,1\}^m)^k$, it holds that

$$\Pr\left[\begin{array}{l|l} \forall i \in [0,k], \mathsf{Ver}_f(\mathsf{crs}, y, \mathsf{st}_i, i, \pi_i) = 1 & \mathsf{crs} \leftarrow \mathsf{Gen}_f(1^\lambda) \\ \wedge\ \mathsf{st}_k = y & \mathsf{st}_0 = \mathsf{crs}, \pi_0 = \bot \\ & y = \mathsf{Eval}_f(\mathsf{crs}, (x_1, \ldots, x_k)) \\ & \text{for } i = 1, \ldots, k: \\ & \quad \mathsf{st}_i = \mathsf{Upd}_f(\mathsf{crs}, \mathsf{st}_{i-1}, x_i) \\ & \quad \pi_i = \mathsf{Prove}_f(\mathsf{crs}, (x_1, \ldots, x_k), \mathsf{st}, i) \end{array}\right] = 1.$$

  Note that for index $i = 0$, π_0 can be empty without loss of generality and Ver_f simply checks that $\mathsf{st}_0 = \mathsf{crs}$. Similarly for $i = k$, π_k can be empty and Ver_f can check that $\mathsf{st}_k = y$.

- **Efficiency.** There exists a polynomial q such that in the correctness and completeness experiment above, the following holds for $k \leq 2^\lambda$ and all $i \in [k]$:
  - $\mathsf{Upd}_f(\mathsf{crs}, \mathsf{st}_{i-1}, x_i)$ runs in time $q(\lambda + m)$ and outputs a new state of size at most $q(\lambda + m)$,
  - $\mathsf{Prove}_f(\mathsf{crs}, (x_1, \ldots, x_k), \mathsf{st}, i)$ runs in time $q(\lambda + m + k)$ and outputs a proof of size $q(\lambda + m)$, and
  - $\mathsf{Ver}_f(\mathsf{crs}, y, \mathsf{st}_i, i, \pi_i)$ runs in time at most $q(\lambda + m)$. ◇

- **Unambiguity.** For any non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l|l} \mathsf{Ver}_f(\mathsf{crs}, y, \mathsf{st}', j, \pi') = 1 & \mathsf{crs} \leftarrow \mathsf{Gen}_f(1^\lambda) \\ \wedge\ (\mathsf{st}', \pi') \neq (\mathsf{st}_j, \pi_j) & \mathsf{st}_0 = \mathsf{crs}, \pi_0 = \bot \\ & (x_1, \ldots, x_k, \mathsf{st}', j, \pi') \leftarrow A_\lambda(\mathsf{crs}) \\ & y = \mathsf{Eval}_f(\mathsf{crs}, (x_1, \ldots, x_k)) \\ & \text{for } i = 1, \ldots, k: \\ & \quad \mathsf{st}_i = \mathsf{Upd}_f(\mathsf{crs}, \mathsf{st}_{i-1}, x_i) \\ & \quad \pi_i = \mathsf{Prove}_f(\mathsf{crs}, (x_1, \ldots, x_k), \mathsf{st}, i) \end{array}\right] \leq \mu(\lambda).$$

We prove in Appendix A that the hash tree root in our rate-1 flSEH construction of Theorem A.1 can be computed in an unambiguous and incremental way, which gives the following theorem. Specifically, we prove that the function $f(\mathsf{hk}, (x_1, \ldots, x_k)) = \mathsf{rt} = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k))$ can be computed with the properties given by the definition above.

Theorem 6.3. Assuming a rate-1 string OT protocol, there exists a rate-1 fully local somewhere extractable hash tree with associated algorithms $(\mathsf{Upd}_f, \mathsf{Prove}_f, \mathsf{Ver}_f)$ that compute the hash root incrementally and unambiguously as in Definition 6.2. ◇

6.3 RAM Delegation from BARGs

The following theorem follows the high level construction of flexible RAM delegation from [KLVW23]. We modify the construction to preserve unambiguity and to handle arbitrary space RAM computations. To achieve unambiguity, we leverage the fact that hash tree roots can be computed incrementally and unambiguously as in Definition 6.2 to ensure that the hash tree root associated with the BARG in the construction of [KLVW23] is unique.

Theorem 6.4. Let $L\colon \mathbb{N}^2 \to \mathbb{N}$ and M a RAM program. Assume the existence of an $L(k, \lambda)$-succinct witness-unambiguous seBARG and an SEH HT whose hash root can be computed unambiguously and incrementally as in Definition 6.2, then there exists an $L(T, \lambda) \cdot \mathsf{poly}(\lambda, \log T)$-succinct unambiguous flexible RAM delegation scheme for the machine M relative to HT. ◇

Proof. Let $\mathsf{seBARG} = (\mathcal{G}_{\mathsf{barg}}, \mathcal{P}_{\mathsf{barg}}, \mathcal{V}_{\mathsf{barg}}, \mathsf{Ext}_{\mathsf{barg}})$ be an seBARG extractable at two locations. Let $\mathsf{HT} = (\mathcal{G}_{\mathsf{ht}}, \mathcal{H}_{\mathsf{ht}}, \mathcal{P}_{\mathsf{ht}}, \mathcal{V}_{\mathsf{ht}}, \mathcal{E}_{\mathsf{ht}})$ be an SEH with associated algorithms $(\mathsf{Upd}_f, \mathsf{Prove}_f, \mathsf{Ver}_f)$ for unambiguously and incrementally computing the function $f(x_1, \ldots, x_k) = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k))$. To prove the theorem, we construct a flexible RAM delegation scheme given by algorithms $(\mathcal{G}_{\mathsf{del}}, \mathcal{P}_{\mathsf{del}}, \mathcal{V}_{\mathsf{del}})$.

At a high level, we use seBARG to prove the T steps of RAM computation are computed correctly relative to a hash tree over each computation state. Along the way, we additionally use the BARG to prove that the root of the hash tree is uniquely determined whenever the BARG is valid. We additionally leverage hash trees to verify that the inputs to the computation are correct at each step, corresponding either to the machines working memory or external read-only memory. We formalize this with the machine M_{ver} in Figure 1 for verifying RAM computation.

We define the flexible RAM delegation scheme $(\mathcal{G}_{\mathsf{del}}, \mathcal{P}_{\mathsf{del}}, \mathcal{V}_{\mathsf{del}})$ as follows.

- $\mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda, T)$:
  - Sample $(\mathsf{crs}_{\mathsf{barg}}, \mathsf{td}_{\mathsf{barg}}) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^{m_{\mathsf{ver}}}, 1^{T_{\mathsf{ver}}}, T, \{1, 1\})$ where $m_{\mathsf{ver}}, T_{\mathsf{ver}}$ are the witness length and running time, respectively, of the machine M_{ver} described in Figure 1, and the pair of indices $\{1, 1\}$ that seBARG is extractable on is arbitrary.
  - Sample $(\mathsf{hk}_{\mathsf{st}}, \mathsf{td}_{\mathsf{st}}) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, m_{\mathsf{st}}, T + 1, 1)$ where $m_{\mathsf{st}} \in O(\lambda)$ is a bound on the RAM computation state size for working memory of size at most 2^λ.
  - Sample $\mathsf{hk}_{\mathsf{mem}} = \mathcal{G}_{\mathsf{ht}}(1^\lambda, 2^\lambda)$. Note that we don't need this instantiation of the hash tree to be extractable.
  - Output $\mathsf{crs} = (\mathsf{crs}_{\mathsf{barg}}, \mathsf{hk}_{\mathsf{st}}, \mathsf{hk}_{\mathsf{mem}})$.
- $\pi = \mathcal{P}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, (\mathsf{cf}, \mathsf{cf}', t), (x_1, \ldots, x_k), I)$:
  - Let $\mathsf{crs} = (\mathsf{crs}_{\mathsf{barg}}, \mathsf{hk}_{\mathsf{st}}, \mathsf{hk}_{\mathsf{mem}})$ and $\mathsf{rt} = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k))$.

Machine M_{ver} for Verifying RAM Computation

Hardcoded Values: $(M, \mathsf{d}, \mathsf{d}', t, \mathsf{hk}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}, \mathsf{hk}_{\mathsf{mem}}, \mathsf{hk}, \mathsf{rt}, I)$ such that M is the input RAM machine, d, d' are hashed states, t is the number of steps, $\mathsf{hk}_{\mathsf{st}}$ is a hash key for the tree over computation states with root $\mathsf{rt}_{\mathsf{st}}$, $\mathsf{hk}_{\mathsf{mem}}$ is a hash key for working memory, hk is a hash key for external read-only memory with root rt, and I specifies the inputs to read from external memory.

Inputs: M_{ver} takes as input an index i and a witness

$$w_i = (\mathsf{st}_{i-1}, \mathsf{st}_i, \rho_{i-1}, \rho_i, b_{i-1}^{\mathsf{mem}}, \rho_{i-1}^{\mathsf{mem}}, b_{i-1}^{\mathsf{wit}}, \rho_{i-1}^{\mathsf{wit}}, \tau_i^{\mathsf{mem}}, \hat{\mathsf{st}}_{i-1}, \hat{\mathsf{st}}_i, \hat{\pi}_{i-1}, \hat{\pi}_i)$$

consisting of hashed states $\mathsf{st}_{i-1} = (\mathsf{rst}_{i-1}, \ell_{i-1}^{\mathsf{mem}}, \ell_{i-1}^{\mathsf{wit}}, \mathsf{d}_{i-1}), \mathsf{st}_i = (\mathsf{rst}_i, \ell_i^{\mathsf{mem}}, \ell_i^{\mathsf{wit}}, \mathsf{d}_i)$ with opening proofs ρ_{i-1}, ρ_i, memory bit b_{i-1}^{mem} with opening proof $\rho_{i-1}^{\mathsf{mem}}$, a witness bit b_{i-1}^{wit} with opening proof $\rho_{i-1}^{\mathsf{wit}}$, an updated digest proof τ_i^{mem}, and incremental root states $\hat{\mathsf{st}}_{i-1}, \hat{\mathsf{st}}_i$ with authentication proofs $\hat{\pi}_{i-1}, \hat{\pi}_i$.

Computation: The machine $M_{\mathsf{ver}}(i, w_i)$ outputs 1 if and only if the following conditions hold:

- The computation states are valid w.r.t. $\mathsf{rt}_{\mathsf{st}}$: $1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}, i, \mathsf{st}_{i-1}, \rho_{i-1})$ and $1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}, i+1, \mathsf{st}_i, \rho_i)$.
- The memory bit b_{i-1}^{mem} is valid: $1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{mem}}, \mathsf{d}_{i-1}, \ell_{i-1}^{\mathsf{mem}}, b_{i-1}^{\mathsf{mem}}, \rho_{i-1}^{\mathsf{mem}})$.
- The witness bit b_{i-1}^{wit} is valid at location $\ell = I[\ell_{i-1}^{\mathsf{wit}}]$: $1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{rt}, \ell, b_{i-1}^{\mathsf{wit}}, \rho_{i-1}^{\mathsf{wit}})$.
- Compute $(\mathsf{rst}_i', \ell'^{\mathsf{mem}}_i, \ell'^{\mathsf{wit}}_i, \ell_i^{\mathsf{wrt}}, b_i^{\mathsf{wrt}}) = \mathsf{step}(M, \mathsf{rst}_{i-1}, b_{i-1}^{\mathsf{mem}}, b_{i-1}^{\mathsf{wit}})$ and verify that $\mathsf{rst}_i' = \mathsf{rst}_i$, $\ell'^{\mathsf{mem}}_i = \ell_i^{\mathsf{mem}}$, $\ell'^{\mathsf{wit}}_i = \ell_i^{\mathsf{wit}}$.
- If $\ell_i^{\mathsf{wrt}} = b_i^{\mathsf{wrt}} = \perp$, the digest over memory doesn't change: $\mathsf{d}_i = \mathsf{d}_{i-1}$.
  Otherwise, the new digest d_i is valid: $1 = \mathsf{VerWrt}_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{mem}}, \mathsf{d}_{i-1}, \ell_i^{\mathsf{wrt}}, b_i^{\mathsf{wrt}}, \mathsf{d}_i, \tau_i^{\mathsf{mem}})$.
- The incremental root states are valid w.r.t. $\mathsf{rt}_{\mathsf{st}}$: $1 = \mathsf{Ver}_f(\mathsf{hk}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}, \hat{\mathsf{st}}_{i-1}, i, \hat{\pi}_{i-1})$ and $1 = \mathsf{Ver}_f(\mathsf{hk}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}, \hat{\mathsf{st}}_i, i+1, \hat{\pi}_i)$.
- The root states are updated correctly: $\hat{\mathsf{st}}_i = \mathsf{Upd}_f(\mathsf{hk}_{\mathsf{st}}, \hat{\mathsf{st}}_{i-1}, \mathsf{st}_i)$.
- If $i = 1$, verify that $\mathsf{d} = \mathsf{st}_0$, $\hat{\mathsf{st}}_0 = \mathsf{hk}_{\mathsf{st}}$, and $\hat{\pi}_0 = \perp$.
- If $i = t$, verify that $\mathsf{d}' = \mathsf{st}_t$ and $\hat{\mathsf{st}}_t = \mathsf{rt}_{\mathsf{st}}$.

Figure 1: Machine that checks correctness of each step of RAM computation consistent with associated hash tree roots.

  - Parse $\mathsf{cf} = (\mathsf{rst}_0, \ell_0^{\mathsf{mem}}, \ell_0^{\mathsf{wit}}, D)$ and set the initial hashed state $\mathsf{st}_0 = (\mathsf{rst}_0, \ell_0^{\mathsf{mem}}, \ell_0^{\mathsf{wit}}, \mathsf{d}_0)$ for $(\mathsf{d}_0, \mathsf{tree}) = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{mem}}, D)$.
  - For $j = 1, \ldots, t$, compute the values $(b_{j-1}^{\mathsf{mem}}, \rho_{j-1}^{\mathsf{mem}}, b_{j-1}^{\mathsf{wit}}, \rho_{j-1}^{\mathsf{wit}}, \mathsf{rst}_j, \ell_j^{\mathsf{mem}}, \ell_j^{\mathsf{wit}}, \mathsf{d}_j, \tau_j^{\mathsf{mem}}, \mathsf{st}_j)$ as follows:

* Let b^{mem}_{j-1} be the bit at location $\ell^{\mathsf{mem}}_{j-1}$ for the database associated with tree and compute $\rho^{\mathsf{mem}}_{j-1} = \mathsf{Rd}^{\mathsf{tree}}_{\mathsf{ht}}(\ell^{\mathsf{mem}}_{j-1})$.
* Let $\ell \in [m \cdot k]$ be the value in memory corresponding to location $I[\ell^{\mathsf{wit}}_{j-1}]$, set b^{wit}_{j-1} to be the ℓth bit in memory, and compute $\rho^{\mathsf{wit}}_{j-1} = \mathcal{P}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k), \ell)$.
* Compute $(\mathsf{rst}_j, \ell^{\mathsf{mem}}_j, \ell^{\mathsf{wit}}_j, \ell^{\mathsf{wrt}}_j, b^{\mathsf{wrt}}_j) = \mathsf{step}(M, \mathsf{rst}_{j-1}, b^{\mathsf{mem}}_{j-1}, b^{\mathsf{wit}}_{j-1})$.
* If $\ell^{\mathsf{wrt}}_j = b^{\mathsf{wrt}}_j = \bot$, set $\mathsf{d}_j = \mathsf{d}_{j-1}$. Otherwise, compute $(\mathsf{d}_j, \tau^{\mathsf{mem}}_j) = \mathsf{Wrt}^{\mathsf{tree}}_{\mathsf{ht}}(\ell^{\mathsf{wrt}}_j, b^{\mathsf{wrt}}_j)$.
* Set $\mathsf{st}_j = (\mathsf{rst}_j, \ell^{\mathsf{mem}}_j, \ell^{\mathsf{wit}}_j, \mathsf{d}_j)$.

– Compute the hash tree root $\mathsf{rt}_{\mathsf{st}} = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{st}}, (\mathsf{st}_0, \ldots, \mathsf{st}_t))$ over the hashed states.
– For every $j \in [0, t]$, compute the opening proof $\rho_j = \mathcal{P}_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{st}}, (\mathsf{st}_0, \ldots, \mathsf{st}_t), j+1)$ for each hashed state.
– Initialize $\hat{\mathsf{st}}_0 = \mathsf{hk}_{\mathsf{st}}$ and $\hat{\pi}_0 = \bot$. For each $j \in [t]$, compute the next root state $\hat{\mathsf{st}}_j = \mathsf{Upd}_f(\mathsf{hk}_{\mathsf{st}}, \hat{\mathsf{st}}_{j-1}, \mathsf{st}_j)$ with associated proof $\hat{\pi}_j = \mathsf{Prove}_f(\mathsf{hk}_{\mathsf{st}}, (\mathsf{st}_0, \ldots, \mathsf{st}_t), \hat{\mathsf{st}}_j, j)$.
– Compute a BARG proof $\pi_{\mathsf{barg}} = \mathcal{P}_{\mathsf{barg}}(\mathsf{crs}_{\mathsf{barg}}, M_{\mathsf{ver}}, (w_1, \ldots, w_t))$ for the machine M_{ver} of Figure 1 using witnesses

$$w_j = (\mathsf{st}_{j-1}, \mathsf{st}_j, \rho_{j-1}, \rho_j, b^{\mathsf{mem}}_{j-1}, \rho^{\mathsf{mem}}_{j-1}, b^{\mathsf{wit}}_{j-1}, \rho^{\mathsf{wit}}_{j-1}, \tau^{\mathsf{mem}}_j, \hat{\mathsf{st}}_{j-1}, \hat{\mathsf{st}}_j, \hat{\pi}_{j-1}, \hat{\pi}_j)$$

for each $j \in [t]$.
– Output $\pi = (\mathsf{rt}_{\mathsf{st}}, \pi_{\mathsf{barg}})$.

- $b = \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, (\mathsf{d}, \mathsf{d}', t), \mathsf{rt}, I, \pi)$:

– Let $\mathsf{crs} = (\mathsf{crs}_{\mathsf{barg}}, \mathsf{hk}_{\mathsf{st}}, \mathsf{hk}_{\mathsf{mem}})$ and $\pi = (\mathsf{rt}_{\mathsf{st}}, \pi_{\mathsf{barg}})$.
– Construct the machine M_{ver} using hardcoded values $(M, \mathsf{d}, \mathsf{d}', t, \mathsf{hk}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}, \mathsf{hk}_{\mathsf{mem}}, \mathsf{hk}, \mathsf{rt}, I)$. Output 1 if and only if

$$1 = \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}_{\mathsf{barg}}, M_{\mathsf{ver}}, \pi_{\mathsf{barg}}).$$

We defer to [KLVW23] for the proofs of completeness, efficiency and L-succinctness, and partial-input soundness. We make two main modifications to their protocol.

- First, we prove arbitrary space RAM computation (rather than bounded space) using the BARG following a standard technique for RAM delegation introduced by [KP16]. This modification doesn't impact (partial-input) soundness or succinctness and was first used by [CJJ22] in their construction of (non-flexible) RAM delegation.

- Second, we add the incremental root values $\hat{\mathsf{st}}_j$ with authentication proofs $\hat{\pi}_j$ to the witness of the seBARG. Completeness still holds by inspection given the correctness and completeness of $(\mathsf{Upd}_f, \mathsf{Prove}_f, \mathsf{Ver}_f)$. Furthermore, this doesn't affect efficiency or succinctness as $\hat{\mathsf{st}}_j, \hat{\pi}_j \in \mathsf{poly}(|\mathsf{st}_j|) = \mathsf{poly}(\lambda)$ by assumption. Soundness and partial-input soundness remain unchanged since we've only added additional checks for the machine M_{ver} checked by the seBARG verifier.

As such, we proceed to prove that the protocol satisfies unambiguity.

Proof of Unambiguity. Suppose by way of contradiction that there exists a pair of non-uniform polynomial-time adversaries $(A_1, A_2) = \{(A_{1,\lambda}, A_{2,\lambda})\}_{\lambda\in\mathbb{N}}$ and polynomials T, p such that for infinitely many $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} \\ \pi \neq \pi' \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi) = 1 \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi') = 1 \\ \wedge\ t \leq T(\lambda) \\ \\ \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda) \\ (1^m, 1^k, I) \leftarrow A_{1,\lambda}(\mathsf{crs}, T(\lambda)) \\ (\mathsf{hk}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, m, k, I) \\ (\mathsf{cf}, \mathsf{d}', t, \mathsf{h}, \pi, \pi') \leftarrow A_{2,\lambda}(\mathsf{crs}, \mathsf{hk}) \\ \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}, \mathsf{cf}) \\ z = (M, \mathsf{d}, \mathsf{d}', t) \\ \mathsf{rt} = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{h}) \end{array}\right] > 1/p(\lambda).$$

Assume without loss of generality that $t \leq T(\lambda)$. Let $\pi = (\mathsf{rt}_{\mathsf{st}}, \pi_{\mathsf{barg}})$ and $\pi' = (\mathsf{rt}'_{\mathsf{st}}, \pi'_{\mathsf{barg}})$ be the proofs output in the experiment above. So, if $\pi \neq \pi'$, then either $\mathsf{rt}_{\mathsf{st}} = \mathsf{rt}'_{\mathsf{st}}$ yet $\pi_{\mathsf{barg}} \neq \pi'_{\mathsf{barg}}$, or $\mathsf{rt}_{\mathsf{st}} \neq \mathsf{rt}'_{\mathsf{st}}$. Define events (E1) and (E2) as follows:

$$\text{(E1)} = \left\{\begin{array}{l} \mathsf{rt}_{\mathsf{st}} = \mathsf{rt}'_{\mathsf{st}} \wedge \pi_{\mathsf{barg}} \neq \pi'_{\mathsf{barg}} \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi) = 1 \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi') = 1 \end{array}\right\}, \quad \text{(E2)} = \left\{\begin{array}{l} \mathsf{rt}_{\mathsf{st}} \neq \mathsf{rt}'_{\mathsf{st}} \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi) = 1 \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi') = 1 \end{array}\right\}.$$

By assumption, one of (E1) or (E2) must hold with probability at least $1/(2p(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$. In the following claims, we argue that either case leads to a contradiction with our assumptions.

Claim 6.5. There exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, $\Pr[\text{(E1)}] \leq \mu(\lambda)$. ◊

Proof. Given event (E1), the values $(M, \mathsf{d}, \mathsf{d}', t, \mathsf{hk}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}, \mathsf{hk}_{\mathsf{mem}}, \mathsf{hk}, \mathsf{rt}, I)$ used to construct M_{ver} are identical when $\mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi)$ and $\mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi')$ both accept, so the proofs π_{barg} and π'_{barg} are for the same statement. By construction, $1 = \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}_{\mathsf{barg}}, M_{\mathsf{ver}}, \pi_{\mathsf{barg}})$ whenever $1 = \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi)$. So, by the witness-unambiguity of seBARG, there exists a negligible function ν_1 such that $\Pr[\text{(E1)}]$ is at most

$$\sum_{i=1}^{T} \cdot \Pr\left[\begin{array}{l} w_i \neq w'_i \\ \wedge\ 1 = \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}_{\mathsf{barg}}, M_{\mathsf{ver}}, \pi_{\mathsf{barg}}) \\ \wedge\ 1 = \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}_{\mathsf{barg}}, M_{\mathsf{ver}}, \pi'_{\mathsf{barg}}) \end{array}\right] + \nu_1(\lambda),$$

where the probability is over $(\mathsf{crs}_{\mathsf{barg}}, \mathsf{td}_{\mathsf{barg}}) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^{m_{\mathsf{ver}}}, 1^{T_{\mathsf{ver}}}, T, \{i, i\})$.

By the somewhere argument of knowledge property of seBARG, there exists a negligible function ν_2 such that

$$\Pr\left[\begin{array}{l} w_i \neq w'_i \\ \wedge\ 1 = \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}_{\mathsf{barg}}, M_{\mathsf{ver}}, \pi_{\mathsf{barg}}) \\ \wedge\ 1 = \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}_{\mathsf{barg}}, M_{\mathsf{ver}}, \pi'_{\mathsf{barg}}) \end{array}\right] \leq \Pr\left[\begin{array}{l} w_i \neq w'_i \\ \wedge\ M_{\mathsf{ver}}(i, w_i) = 1 \\ \wedge\ M_{\mathsf{ver}}(i, w'_i) = 1 \end{array}\right] + \nu_2(\lambda).$$

Recall that w_i and w'_i consists of values

$$(\mathsf{st}_{i-1}, \mathsf{st}_i, \rho_{i-1}, \rho_i, b^{\mathsf{mem}}_{i-1},\ \rho^{\mathsf{mem}}_{i-1},\ b^{\mathsf{wit}}_{i-1},\ \rho^{\mathsf{wit}}_{i-1},\ \tau^{\mathsf{mem}}_i,\ \hat{\mathsf{st}}_{i-1}, \hat{\mathsf{st}}_i, \hat{\pi}_{i-1}, \hat{\pi}_i) \text{ and}$$
$$(\mathsf{st}'_{i-1}, \mathsf{st}'_i, \rho'_{i-1}, \rho'_i, b'^{\mathsf{mem}}_{i-1}, \rho'^{\mathsf{mem}}_{i-1}, b'^{\mathsf{wit}}_{i-1}, \rho'^{\mathsf{wit}}_{i-1}, \tau'^{\mathsf{mem}}_i, \hat{\mathsf{st}}'_{i-1}, \hat{\mathsf{st}}'_i, \hat{\pi}'_{i-1}, \hat{\pi}'_i),$$

respectively. Because $M_{\mathsf{ver}}(w_i) = 1$ and $M_{\mathsf{ver}}(w'_i) = 1$, it holds that all verification checks on the above values pass in M_{ver}. Opening unambiguity of HT implies that $(\mathsf{st}_{i-1}, \rho_{i-1}) = (\mathsf{st}'_{i-1}, \rho'_{i-1})$, $(\mathsf{st}_i, \rho_i) = (\mathsf{st}'_i, \rho'_i)$, $(b^{\mathsf{mem}}_{i-1}, \rho^{\mathsf{mem}}_{i-1}) = (b'^{\mathsf{mem}}_{i-1}, \rho'^{\mathsf{mem}}_{i-1})$, and $(b^{\mathsf{wit}}_{i-1}, \rho^{\mathsf{wit}}_{i-1}) = (b'^{\mathsf{wit}}_{i-1}, \rho'^{\mathsf{wit}}_{i-1})$ except with some negligible probability $\nu_3(\lambda)$. As the inputs to step are the same, it follows that the outputs ℓ^{wrt}_i and b^{wrt}_i computed during M_{ver} are the same on inputs w_i, w'_i. It follows from update unambiguity of HT that $\tau^{\mathsf{mem}}_i = \tau'^{\mathsf{mem}}_i$ except with some negligible probability $\nu_4(\lambda)$. Similarly, unambiguity of $(\mathsf{Upd}_f, \mathsf{Prove}_f, \mathsf{Ver}_f)$ guarantees that $(\hat{\mathsf{st}}_{i-1}, \hat{\pi}_{i-1}) = (\hat{\mathsf{st}}'_{i-1}, \hat{\pi}'_{i-1})$, $(\hat{\mathsf{st}}_i, \hat{\pi}_i) = (\hat{\mathsf{st}}'_i, \hat{\pi}'_i)$, except with some negligible probability $\nu_5(\lambda)$.

Combining the above, we conclude that

$$\Pr[(\text{E1})] \leq T \cdot (\nu_2(\lambda) + \nu_3(\lambda) + \nu_4(\lambda) + \nu_5(\lambda)) + \nu_1(\lambda),$$

which is negligible as T, m_{ver}, T_{ver} are polynomially bounded in λ. □

Claim 6.6. There exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, $\Pr[(\text{E2})] \leq \mu(\lambda)$. ◇

Proof. At a high level, this follows by a similar argument to partial-input soundness from [KLVW23]. Namely, the proof of partial-input soundness argues that the final hashed state $\mathsf{st}_t = \mathsf{d}'$ if π_{barg} is accepting. This is shown by inductively arguing that the experiment that programs $\mathsf{hk}_{\mathsf{st}}$ and $\mathsf{crs}_{\mathsf{barg}}$ to be extractable at index i extracts a value st_i from $\mathsf{rt}_{\mathsf{st}}$ such that st_i is equal to the actual hashed state st^*_i (when reading the appropriate values from $(x_1, \ldots, x_k)$). For unambiguity, we give a similar inductive argument relying on the above to show that the final updated state $\hat{\mathsf{st}}_t$ is equal to the actual root $\mathsf{rt}^* = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{st}}, (\mathsf{st}^*_0, \ldots, \mathsf{st}^*_t))$ with overwhelming probability whenever π_{barg} is accepting. A similar argument will hold for π'_{barg}, so we argue this for the values associated with π_{barg} without loss of generality.

Towards the above, let $\mathsf{st}^*_0, \ldots, \mathsf{st}^*_t$ denote the actual hashed states when reading external memory from $(x_1, \ldots, x_k)$. Then, define $\hat{\mathsf{st}}^*_0 = \mathsf{hk}_{\mathsf{st}}$, and for $i = 1, \ldots, t$, define $\hat{\mathsf{st}}^*_i = \mathsf{Upd}_f(\mathsf{hk}_{\mathsf{st}}, \hat{\mathsf{st}}^*_{i-1}, \mathsf{st}^*_i)$. Consider the experiment where $(\mathsf{hk}_{\mathsf{st}}, \mathsf{td}_{\mathsf{st}})$ and $(\mathsf{crs}_{\mathsf{barg}}, \mathsf{td}_{\mathsf{barg}})$ are programmed at index i and let $\mathsf{st}_i = \mathcal{E}_{\mathsf{ht}}(\mathsf{td}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}})$ the extracted RAM state in SEH and $w_i = \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}_{\mathsf{barg}}, M_{\mathsf{ver}}, \pi_{\mathsf{barg}})$ be the extracted witness with corresponding computation state st'_i and root state $\hat{\mathsf{st}}_i$. Following [KLVW23], we define the event BAD_i according to the above probability space:

$$\mathsf{BAD}_i = \left\{ \begin{array}{l} \mathsf{st}_i \neq \mathsf{st}^*_i \\ \wedge\, \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi) = 1 \end{array} \right\}.$$

In the same probability space, we define a new event $\widehat{\mathsf{BAD}}_i$ that captures the event that $\hat{\mathsf{st}}_i \neq \hat{\mathsf{st}}^*_i$:

$$\widehat{\mathsf{BAD}}_i = \left\{ \begin{array}{l} \hat{\mathsf{st}}_i \neq \hat{\mathsf{st}}^*_i \\ \wedge\, \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathsf{hk}, z, \mathsf{rt}, I, \pi) = 1 \end{array} \right\}.$$

Leveraging the claim from [KLVW23] that there exists a negligible function μ' such that $\Pr[\mathsf{BAD}_i] \leq \mu'(\lambda)$ for all $\lambda \in \mathbb{N}$, we will show that for each $i \in [t]$, there exists a negligible function μ_i such that $\Pr[\widehat{\mathsf{BAD}}_i] \leq \mu_i(\lambda)$ for all $\lambda \in \mathbb{N}$. From this, it follows that $\Pr[\widehat{\mathsf{BAD}}_t] \leq \mu_t(\lambda)$, which bounds the probability that $\mathsf{rt}_{\mathsf{st}} \neq \mathsf{rt}'_{\mathsf{st}}$ for event (E2) as required for the claim.

In the base case, we want to argue that there is a negligible function μ_1 such that $\Pr[\widehat{\mathsf{BAD}}_1] \leq \mu_1(\lambda)$. This follows from the following sub-claims:

- First, by somewhere argument of knowledge of seBARG, it holds that $M_{\mathsf{ver}}(1, w_1) = 1$ except with some negligible probability $\nu_1(\lambda)$. It follows that $\hat{\mathsf{st}}_1 = \mathsf{Upd}_f(\mathsf{hk}_{\mathsf{st}}, \hat{\mathsf{st}}_0, \mathsf{st}'_1)$, $\hat{\mathsf{st}}_0 = \mathsf{hk}_{\mathsf{st}} = \hat{\mathsf{st}}^*_0$, and $1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}, 2, \mathsf{st}'_1, \rho_1)$.

- Next, by the opening unambiguity of HT, it holds that $\mathsf{st}'_1 = \mathsf{st}_1$ except with some negligible probability $\nu_2(\lambda)$.
- Finally, $\mathsf{st}_1 = \mathsf{st}^*_1$ except with probability $\Pr[\mathsf{BAD}_1] \leq \mu'(\lambda)$ by assumption.

It follows that $\mathsf{st}_1 = \mathsf{st}^*_1 = \mathsf{Upd}_f(\mathsf{hk}_{\mathsf{st}}, \hat{\mathsf{st}}^*_0, \mathsf{st}^*_1)$ except with probability

$$\Pr[\widehat{\mathsf{BAD}}_i] \leq \nu_1(\lambda) + \nu_2(\lambda) + \mu'(\lambda) = \mu_i(\lambda),$$

which is negligible as required.

Let $i \in [2, t]$ and inductively assume that $\Pr[\widehat{\mathsf{BAD}}_{i-1}] \leq \mu_{i-1}(\lambda)$ for some negligible function μ_{i-1}. We want to show there is a negligible function μ_i such that $\Pr[\widehat{\mathsf{BAD}}_i] \leq \mu_i(\lambda)$. This follows from the following sub-claims:

- First, by somewhere argument of knowledge of seBARG, it holds that $M_{\mathsf{ver}}(i, w_i) = 1$ except with negligible probability $\nu_1(\lambda)$, so $\hat{\mathsf{st}}_i = \mathsf{Upd}_f(\mathsf{hk}_{\mathsf{st}}, \hat{\mathsf{st}}_{i-1}, \mathsf{st}'_i)$, $1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}, i+1, \mathsf{st}'_i, \rho_i)$, and $1 = \mathsf{Ver}_f(\mathsf{hk}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}, \hat{\mathsf{st}}_{i-1}, i, \hat{\pi}_{i-1})$.
- Next, by opening unambiguity of HT, it again follows that $\mathsf{st}'_i = \mathsf{st}_i$ except with negligible probability $\nu_2(\lambda)$.
- It holds that $\mathsf{st}_i = \mathsf{st}^*_i$ except with probability $\Pr[\mathsf{BAD}_i] \leq \mu'(\lambda)$ by assumption.
- Next, by the index hiding of seBARG, the above still holds for the extracted witnesses w_{i-1} and w_i when $(\mathsf{crs}_{\mathsf{barg}}, \mathsf{td}_{\mathsf{barg}})$ is programmed to be extractable on $\{i-1, i\}$ except with some negligible probability $\nu_3(\lambda)$.
- Now, let $\hat{\mathsf{st}}'_{i-1}$ be the value corresponding to extracting witness w_{i-1} from π_{barg}. Note that since $M_{\mathsf{ver}}(i-1, w_{i-1}) = 1$, it holds that $1 = \mathsf{Ver}_f(\mathsf{hk}_{\mathsf{st}}, \mathsf{rt}_{\mathsf{st}}, \hat{\mathsf{st}}'_{i-1}, i, \hat{\pi}_{i-1})$. By unambiguity of $(\mathsf{Upd}_f, \mathsf{Prove}_f, \mathsf{Ver}_f)$, it follows that $\hat{\mathsf{st}}'_{i-1} = \hat{\mathsf{st}}_{i-1}$ except with negligible probability $\nu_4(\lambda)$.
- Finally, it follows from our inductive assumption that $\hat{\mathsf{st}}'_{i-1} = \hat{\mathsf{st}}^*_{i-1}$ except with probability $\Pr[\widehat{\mathsf{BAD}}_{i-1}] \leq \mu_{i-1}(\lambda)$.

It follows that $\mathsf{st}_i = \mathsf{st}^*_i = \mathsf{Upd}_f(\mathsf{hk}_{\mathsf{st}}, \hat{\mathsf{st}}^*_{i-1}, \mathsf{st}^*_i)$ except with probability

$$\Pr[\widehat{\mathsf{BAD}}_i] \leq \nu_1(\lambda) + \nu_2(\lambda) + \mu'(\lambda) + \nu_3(\lambda) + \nu_4(\lambda) + \mu_{i-1}(\lambda) = \mu_i(\lambda),$$

which is negligible since $i \leq T(\lambda)$ is polynomially bounded as required. □

This completes the proof of Theorem 6.4. □

7 Boosting Batch Arguments Preserves Unambiguity

In this section, we construct $\mathsf{poly}(\log k, \lambda)$-succinct BARGs and then rate-1 BARGs given the non-trivial unambiguous RAM delegation scheme of Theorem 6.4.

7.1 Succinct BARGs from Non-Trivial RAM Delegation

The following theorem follows the high level approach for the construction of BARGs from non-trivial RAM delegation from [KLVW23]. However, we make significant changes to the recursive step in order to preserve unambiguity. In addition to using an unambiguous RAM delegation scheme, we prove that the hash value computed at each recursion level is computed correctly and uniquely in a recursive-friendly manner. For simplicity, we state the following theorem assuming a RAM delegation scheme for any machine M, but in the construction we will instantiate it for fixed machines described during the protocol.

Theorem 7.1. Let $L\colon \mathbb{N}^2 \to \mathbb{N}$ and M a RAM program. Let HT be an unambiguous, locally updatable flSEH and DEL an $L(T,\lambda)$-succinct unambiguous flexible RAM delegation scheme for M relative to HT. If there exists a polynomial q such that $L(T,\lambda) < \max(T/(16\lambda), q(\lambda))$, then there exists a $\mathsf{poly}(\log k, \lambda)$-succinct witness-unambiguous seBARG. ◇

Proof. Let $\mathsf{HT} = (\mathcal{G}_{\mathsf{ht}}, \mathcal{H}_{\mathsf{ht}}, \mathcal{D}_{\mathsf{ht}}, \mathcal{P}_{\mathsf{ht}}, \mathcal{V}_{\mathsf{ht}}, \mathcal{E}_{\mathsf{ht}})$ be an unambiguous, locally updatable flSEH with local update algorithms $(\mathsf{Rd}_{\mathsf{ht}}, \mathsf{Wrt}_{\mathsf{ht}}, \mathsf{VerWrt}_{\mathsf{ht}})$. Let $\mathsf{DEL} = (\mathcal{G}_{\mathsf{del}}, \mathcal{P}_{\mathsf{del}}, \mathcal{V}_{\mathsf{del}})$ be an unambiguous flexible RAM delegation scheme for HT. We will instantiate this for fixed machines below, but use DEL for simplicity. We construct an seBARG $(\mathcal{G}_{\mathsf{barg}}, \mathcal{P}_{\mathsf{barg}}, \mathcal{V}_{\mathsf{barg}}, \mathsf{Ext}_{\mathsf{barg}})$ as follows:

- $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^m, 1^T, k, i)$:
  - Assume without loss of generality that k is a power of 2 and define $k_\ell = k/2^{\ell-1}$ for notational convenience.
  - Define $t_\ell = \max(T/(2\lambda)^{\ell-1}, 8q(\lambda))$.
  - For each $\ell \in [\log k]$, let $(\mathsf{hk}_\ell, \mathsf{td}_\ell) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, t_\ell, k_\ell + 1, I_\ell)$ where $I_1 = \{i+1, i+1, i+1\}$ and $I_\ell = \{1,1,1\}$ for $\ell > 1$.
  - Define R^1_π to be the RAM program that on input i with external memory (R, w) and outputs 1 if and only if $R(i,w) = 1$. For $\ell > 1$, define R^ℓ_π to be the RAM program that on input i with external memory (R, w, w') and outputs 1 if and only if $R(i,(w,w')) = 1$.
  - For each $\ell = 1, \ldots, \log k$, let M_ℓ be the machine define in $\mathcal{P}_{\mathsf{barg}}$ below and define R^ℓ_τ to be the RAM program that on input i with external memory w inserts the jth bit $w[j]$ to memory at positions $|M_1| + i \cdot t_1 + j$ for each $j \in [|w|]$.
  - For each $\ell \in [\log k]$, let $\mathsf{crs}_\ell \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda, t_\ell)$ be a CRS from machines R^ℓ_π and R^ℓ_τ (we use the same CRS for simplicity since they take the same time bound).
  - Output $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_{\log k}, \mathsf{hk}_1, \ldots, \mathsf{hk}_{\log k})$ and $\mathsf{td} = (\mathsf{td}_1, \ldots, \mathsf{td}_{\log k})$.
- $\pi = \mathcal{P}_{\mathsf{barg}}(\mathsf{crs}, M, (w_1, \ldots, w_k))$:
  - Let $w^1_i = w_i$ for each $i \in [k]$.
  - Define the machine M_1 be the RAM program that takes as input an index i with external memory (w^1_{i-1}, w^1_i) (where $w^1_{i-1} = \perp$ when $i = 0$) and outputs 1 if and only if $M(w^1_i) = 1$.
  - For $\ell = 1, \ldots, \log k$, do the following:
    - Compute a hash value $\mathsf{h}_\ell = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}_\ell, (M_\ell, w^\ell_1, \ldots, w^\ell_k))$ with associated root $\mathsf{rt}_\ell = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}_\ell, \mathsf{h}_\ell)$.
    - Compute an opening proof $\rho_\ell = \mathcal{P}_{\mathsf{ht}}(\mathsf{hk}_\ell, (M_\ell, w^\ell_1, \ldots, w^\ell_{k_\ell}), \{1, \ldots, |M_\ell|\})$.

* For $i \in [k_\ell]$, compute

$$\pi_i^\ell = \mathcal{P}_{\mathsf{del}}(\mathsf{crs}_\ell, \mathsf{hk}_\ell, (R_\pi^\ell, \mathsf{cf}_i^{\mathsf{init}}, \mathsf{cf}^{\mathsf{acc}}, t_\ell), (M_\ell, w_1^\ell, \ldots, w_k^\ell), \{1, i, i+1\})$$

where for computation $R_\pi^\ell(i, (M_\ell, w_{i-1}^\ell, w_i^\ell))$ it holds that $\mathsf{cf}_i^{\mathsf{init}}$ is the initial configuration for input i, $\mathsf{cf}^{\mathsf{acc}}$ is the final accepting configuration, and the program halts and accepts within t_ℓ steps. Let $\mathsf{d}_{\ell,i}^{\mathsf{init}} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}_\ell, \mathsf{cf}_i^{\mathsf{init}})$ and $\mathsf{d}_\ell^{\mathsf{acc}} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}_\ell, \mathsf{cf}^{\mathsf{acc}})$ be the associated digests for the initial and final configurations.

* Let cf_0^ℓ denote the initial configuration for machine R_τ^ℓ with initial memory $M_\ell || 0^{t_\ell \cdot k_\ell}$ and compute $\mathsf{d}_0^\ell = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}_\ell, \mathsf{cf}_0^\ell)$. For $i = 1, \ldots, k_\ell$, let cf_i^ℓ be the configuration such that $(\mathsf{cf}_{i-1}^\ell, \mathsf{cf}_i^\ell, t_\ell, w_i^\ell) \in \mathcal{T}_{R_\tau^\ell}$ for $t_\ell \geq |w_i^\ell|$ and compute

$$\mathsf{d}_i^\ell = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}_\ell, \mathsf{cf}_i) \text{ and}$$

$$\tau_i^\ell = \mathcal{P}_{\mathsf{del}}(\mathsf{crs}_\ell, \mathsf{hk}_\ell, (R_\tau^\ell, \mathsf{cf}_{i-1}, \mathsf{cf}_i, t_\ell), (M_\ell, w_1^\ell, \ldots, w_{k_\ell}^\ell), i+1).$$

* For each $i \in [k_{\ell+1}]$, set $w_i^{\ell+1} = (\pi_{2i-1}^\ell, \pi_{2i}^\ell, \mathsf{d}_{2i-1}^\ell, \mathsf{d}_{2i}^\ell, \tau_{2i-1}^\ell, \tau_{2i}^\ell)$.
* Define the machine $M_{\ell+1}$ to be the RAM program with hardcoded values

$$(\mathsf{crs}_\ell, \mathsf{hk}_\ell, \mathsf{rt}_\ell, t_\ell, \mathsf{d}_\ell^{\mathsf{acc}}, \mathsf{d}_0^\ell)$$

that takes as input an index i with external memory $(w_{i-1}^{\ell+1}, w_i^{\ell+1})$ defined according to the above, computes digest $\mathsf{d}_{\ell,2i-1}^{\mathsf{init}} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}_\ell, \mathsf{cf}_{2i-1}^{\mathsf{init}})$ and $\mathsf{d}_{\ell,2i}^{\mathsf{init}} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}_\ell, \mathsf{cf}_{2i}^{\mathsf{init}})$ and outputs 1 if the following hold:

$$\begin{aligned}
&1 = \mathcal{V}_{\mathsf{del}}(\mathsf{crs}_\ell, \mathsf{hk}_\ell, (R_\pi^\ell, \mathsf{d}_{\ell,2i-1}^{\mathsf{init}}, \mathsf{d}_\ell^{\mathsf{acc}}, t_\ell), \mathsf{rt}_\ell, \{1, 2i-1, 2i\}, \pi_{2i-1}^\ell),\\
&1 = \mathcal{V}_{\mathsf{del}}(\mathsf{crs}_\ell, \mathsf{hk}_\ell, (R_\pi^\ell, \mathsf{d}_{\ell,2i}^{\mathsf{init}}, \mathsf{d}_\ell^{\mathsf{acc}}, t_\ell), \mathsf{rt}_\ell, \{1, 2i, 2i+1\}, \pi_{2i}^\ell),\\
&1 = \mathcal{V}_{\mathsf{del}}(\mathsf{crs}_\ell, \mathsf{hk}_\ell, (R_\tau^\ell, \mathsf{d}_{2i-2}^\ell, \mathsf{d}_{2i-1}^\ell, t_\ell), \mathsf{rt}_\ell, \{2i\}, \tau_{2i-1}^\ell),\\
&1 = \mathcal{V}_{\mathsf{del}}(\mathsf{crs}_\ell, \mathsf{hk}_\ell, (R_\tau^\ell, \mathsf{d}_{2i-1}^\ell, \mathsf{d}_{2i}^\ell, t_\ell), \mathsf{rt}_\ell, \{2i+1\}, \tau_{2i}^\ell), \text{ and}\\
&\mathsf{d}_{2i}^\ell \text{ corresponds to memory root } \mathsf{rt}_\ell \text{ if } 2i = k_\ell.
\end{aligned}$$

Note that all of the non-hardcoded values come from $w_i^{\ell+1}$ except for d_{2i-2}^ℓ which comes from $w_{i-1}^{\ell+1}$. For $i = 1$, $M_{\ell+1}$ uses the value d_0^ℓ hardcoded.

– Output $\pi = (\mathsf{h}_1, \rho_1, \ldots, \mathsf{h}_{\log k}, \rho_{\log k}, w_1^{\log k+1})$.

- $b = \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi)$:

– Let $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_{\log k}, \mathsf{hk}_1, \ldots, \mathsf{hk}_{\log k})$ and $\pi = (\mathsf{h}_1, \rho_1, \ldots, \mathsf{h}_{\log k}, \rho_{\log k}, w_1^{\log k+1})$.

– Let M_1 be the level 1 RAM program defined in $\mathcal{P}_{\mathsf{barg}}$ as a function of the input machine M. For each $\ell = 1, \ldots, \log k$, construct the machine $M_{\ell+1}$ by computing the required hardcoded values as follows:

* The values $\mathsf{crs}_\ell, \mathsf{hk}_\ell$ come from crs.
* Compute $\mathsf{rt}_\ell = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}_\ell, \mathsf{h}_\ell)$.
* Compute $t_\ell = T/(2\lambda)^{\ell-1}$.
* Compute $\mathsf{d}_\ell^{\mathsf{acc}} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}_\ell, \mathsf{cf}^{\mathsf{acc}})$ where $\mathsf{cf}^{\mathsf{acc}}$ is the accepting state.
* Compute $\mathsf{d}_0^\ell = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}_\ell, \mathsf{cf}_0^\ell)$ where cf_0^ℓ is the initial configuration for R_τ^ℓ with memory set to $M_\ell || 0^{t_\ell \cdot k_\ell}$.

- For each $\ell \in [\log k]$, verify that $1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}_\ell, \mathsf{rt}_\ell, \{1, \ldots, |M_\ell|\}, M_\ell, \rho_\ell)$.
- Output 1 if and only if $M_{\log k+1}(1, w_1^{\log k+1}) = 1$.

- $w = \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, \pi)$:
  - Let $\mathsf{td} = (\mathsf{td}_1, \ldots, \mathsf{td}_{\log k})$ and $\pi = (\mathsf{h}_1, \rho_1, \ldots, \mathsf{h}_{\log k}, \rho_{\log k}, w_1^{\log k+1})$.
  - Output $w = \mathcal{E}_{\mathsf{ht}}(\mathsf{td}_1, \mathsf{h}_1)$.

We prove each of the required properties of the construction, with an emphasis on witness-unambiguity.

Completeness. In terms of completeness, fix values $\lambda, n, m, T, k \in \mathbb{N}$, $i \in [k]$, and let $(\mathsf{crs}, \cdot) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^n, 1^m, 1^T, k, i)$. Let M be a machine and witnesses $(w_1, \ldots, w_k)$ such that $M(i, w_i) = 1$ for all $i \in [k]$. Let $\pi = \mathcal{P}_{\mathsf{barg}}(\mathsf{crs}, M, (w_1, \ldots, w_k))$ for $\pi = (\mathsf{h}_1, \rho_1, \ldots, \mathsf{h}_{\log k}, \rho_{\log k}, w_1^{\log k+1})$. We want to show that $1 = \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi)$.

First, note that $\mathcal{V}_{\mathsf{barg}}$ constructs the same machines $M_1, \ldots, M_{\log k+1}$ as in $\mathcal{P}_{\mathsf{barg}}$. By opening completeness of HT, it is the case that $1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}_\ell, \mathsf{rt}_\ell, \{1, \ldots, |M_\ell|\}, M_\ell, \rho_\ell)$ for all $\ell \in [\log k]$.

It remains to argue that $M_{\log k+1}(1, w_1^{\log k+1}) = 1$. We argue at each level $\ell \in [\log k]$, if $M_\ell(i, (w_{i-1}^\ell, w_i^\ell)) = 1$ for all $i \in [k_\ell]$, then $M_{\ell+1}(i, (w_{i-1}^{\ell+1}, w_i^{\ell+1})) = 1$ for all $i \in [k_{\ell+1}]$. This suffices since in the base case, $M_1(i, (w_{i-1}, w_i)) = M(i, w_i) = 1$ for all $i \in [k]$ by assumption. Now assume that for some $\ell \in [k_\ell]$ that $M_\ell(i, (w_{i-1}^\ell, w_i^\ell)) = 1$ for all $i \in [k_\ell]$. By completeness of DEL, it follows that the proofs π_i^ℓ are accepting for all $i \in [k_\ell]$. Similarly, the update proofs τ_i^ℓ are also all accepting by construction. It follows that all checks required for $M_{\ell+1}$ pass, so $M_{\ell+1}(i, (w_{i-1}^{\ell+1}, w_i^{\ell+1})) = 1$ for all $i \in [k_{\ell+1}]$ as required.

Efficiency. In terms of efficiency, we first argue that $t_\ell = \max(T/(2\lambda)^{\ell-1}, 8q(\lambda))$ bounds the size of w_i^ℓ and running time of $M_\ell(i, (w_{i-1}^\ell, w_i^\ell))$ at each level $\ell \in [\log k]$. In the base case of $\ell = 1$, this holds by assumption given the inputs to $\mathcal{G}_{\mathsf{barg}}$. Assume it holds for level ℓ, and we will argue that the same must hold for level $\ell + 1$. Witness $w_i^{\ell+1}$ contains proofs π_{2i-1}^ℓ, π_{2i}^ℓ, τ_{2i-1}^ℓ, τ_{2i}^ℓ, each of which is for RAM computation consisting of t_ℓ steps by assumption. So $L(t_\ell, \lambda) \leq \max(t_\ell/(16\lambda), q(\lambda))$ bounds the size of each proof and the runtime of $\mathcal{V}_{\mathsf{del}}$ for such a proof. Witness $w_i^{\ell+1}$ also contains digests $\mathsf{d}_{2i-1}^\ell, \mathsf{d}_{2i}^\ell$ and computes $\mathsf{d}_{\ell,2i-1}^{\mathsf{init}}$ and $\mathsf{d}_{\ell,2i}^{\mathsf{init}}$ which takes some fixed polynomial time in λ. Assume without loss of generality each of these are of size and take time at most $q(\lambda)$ since it is independent of T (otherwise we could choose q to be larger), so it follows that these are also bounded by $L(t_\ell, \lambda)$. It follows that

$$t_{\ell+1} \leq 8 \cdot L(t_\ell, \lambda) \leq \max(T/(2\lambda)^\ell, 8q(\lambda)),$$

as required.

Using the above, we proceed to bound the efficiency of $\mathcal{G}_{\mathsf{barg}}$, $\mathcal{P}_{\mathsf{barg}}$, $\mathcal{V}_{\mathsf{barg}}$, and $\mathsf{Ext}_{\mathsf{barg}}$ as follows:

- *Efficiency of $\mathcal{G}_{\mathsf{barg}}$.* $\mathcal{G}_{\mathsf{barg}}$ samples $\log k$ hash keys for HT and CRS values for $\mathcal{G}_{\mathsf{del}}$. The hash keys are of size at most $\mathsf{poly}(\lambda)$ and each crs_ℓ is of size at most $L(t_\ell, \lambda) \cdot \mathsf{poly}(\lambda)$. Thus, there exists a polynomial q' such that the overall running time and size of crs is at most

$$|\mathsf{crs}| \leq \log k \cdot L(T, \lambda) \cdot q(\lambda) \in \mathsf{poly}(\log k, \lambda) \cdot \mathsf{poly}(T),$$

as required.

- *Efficiency of* $\mathcal{P}_{\mathsf{barg}}$. The proof π output by $\mathcal{P}_{\mathsf{barg}}$ consists of values $\mathsf{h}_\ell, \rho_\ell$ for each $\ell \in [\log k]$ and a witness $w_1^{\log k+1}$. Each h_ℓ is of size at most $3 \cdot t_\ell$ since HT is set to be extractable at 3 locations, and ρ_ℓ has size $|M_\ell| \cdot \mathsf{poly}(\lambda)$ and $|M_\ell| \leq t_\ell$. Lastly, $|w_1^{\log k+1}| \leq t_{\log k+1} = \max(T/(2\lambda)^{\log k}, 8q(\lambda))$. As $t_{\ell+1}$ for $\ell > 1$ is either bounded by a fixed polynomial $8 \cdot q(\lambda)$ or is at most $t_\ell/(2\lambda)$, it follows that there exists a polynomial q' such that

$$|\pi| \leq \log k \cdot q'(\lambda) + 3T \cdot \sum_{\ell=1}^{\log k+1} \frac{1}{(2\lambda)^{\ell-1}} \leq \log k \cdot q'(\lambda) + 3T \cdot (1 + 1/\lambda).$$

- *Efficiency of* $\mathcal{V}_{\mathsf{barg}}$. The same analysis bounding the size of the proof bounds the run time of the verifier since we only used the fact that t_ℓ bounds the size at each level, which also bounds the proof. So efficiency of $\mathcal{V}_{\mathsf{barg}}$ immediately follows.

- *Efficiency of* $\mathsf{Ext}_{\mathsf{barg}}$. $\mathsf{Ext}_{\mathsf{barg}}$ simply outputs $\mathcal{E}_{\mathsf{ht}}(\mathsf{td}_1, \mathsf{h}_1)$, which takes time at most $3 \cdot T \cdot \mathsf{poly}(\lambda)$ by the efficiecny of HT, as required.

Index Hiding. Index hiding follows immediately from the index hiding of HT by a standard hybrid argument.

Somewhere argument of knowledge. This follows similarly to [KLVW23] since we've only added additional checks at each level of recursion, but we give the proof for completeness as we use some of the intermediate claims to prove witness-unambiguity.

Suppose by way of contradiction there exists a non-uniform polynomial-time adversary $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$, polynomials m, T, k, a sequence of indices $\{i^*_\lambda \in [k(\lambda)]\}_{\lambda \in \mathbb{N}}$, and a polynomial p such that for infinitely many $\lambda \in \mathbb{N}$, it hold that

$$\Pr\left[\begin{array}{l|l} \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^{m(\lambda)}, T(\lambda), k(\lambda), i^*_\lambda) \\ \wedge\ M(i^*, w_{i^*}) \neq 1 & (M, \pi) \leftarrow A_\lambda(\mathsf{crs}) \\ & w_{i^*} = \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi) \end{array}\right] > \frac{1}{p(\lambda)}.$$

Define $i^*_\ell = \lceil i^*/2^{\ell-1} \rceil$. We first define an alternative experiment where we sample $\mathcal{G}_{\mathsf{barg}}$ in the following way, which we denote by $\mathcal{G}^*_{\mathsf{barg}}$. For each $\ell \in [\log k]$, we instead sample $(\mathsf{hk}_\ell, \mathsf{td}_\ell) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, t_\ell, k_\ell + 1, I^*_\ell)$ for $I^*_\ell = \{1, i^*_\ell, i^*_\ell + 1\}$. By index hiding of HT, the success probability of the experiments when sampling $(\mathsf{crs}, \mathsf{td})$ from $\mathcal{G}^*_{\mathsf{barg}}$ as opposed to $\mathcal{G}_{\mathsf{barg}}$ differ with at most negligible probability $\nu_1(\lambda)$.

Now, for each $\ell \in [\log k]$, let h_ℓ, crs_ℓ, hk_ℓ, td_ℓ be the corresponding values from π, crs, and td, and define

$$(M^*_\ell, w^\ell_{i^*_\ell - 1}, w^\ell_{i^*_\ell}) = \mathcal{E}_{\mathsf{ht}}(\mathsf{hk}_\ell, \mathsf{h}_\ell).$$

Note that w_{i^*} corresponds to $w^1_{i^*}$ in this experiment.

Whenever $\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi)$ accepts, it holds that $1 = \mathcal{V}_{\mathsf{ht}}(\mathsf{hk}_\ell, \mathsf{rt}_\ell, \{1, \ldots, |M_\ell|\}, M_\ell, \rho_\ell)$, so it follows from the somewhere statistically binding property of HT that $M^*_\ell = M_\ell$ except with negligible probability $\nu_2(\lambda)$. Since $M_{\log k+1}(1, w_1^{\log k+1}) = 1$ when $\mathcal{V}_{\mathsf{barg}}$ accepts, it follows that for infinitely

many $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \forall \ell \in [\log k], M_\ell = M^*_\ell \\ \wedge\ M_{\log k+1}(1, w_1^{\log k+1}) = 1 \\ \wedge\ M_1(i^*, (w^1_{i^*-1}, w^1_{i^*})) \neq 1 \end{array}\right] > \frac{1}{3p(\lambda)}.$$

By a hybrid argument over each $\ell \in [\log k]$, there must exist an index ℓ such that

$$\Pr\left[\begin{array}{l} \forall \ell \in [\log k], M_\ell = M^*_\ell \\ \wedge\ M_{\ell+1}(i^*_{\ell+1}, (w^{\ell+1}_{i^*_{\ell+1}}, w^{\ell+1}_{i^*_{\ell+1}})) = 1 \\ \wedge\ M_\ell(i^*_\ell, (w^\ell_{i^*_\ell-1}, w^\ell_{i^*_\ell})) \neq 1 \end{array}\right] > \frac{1}{3p(\lambda)\cdot \log k}.$$

Let $w^{\ell+1}_{i^*_\ell} = (\pi^\ell_{2i-1}, \pi^\ell_{2i}, \mathsf{d}^\ell_{2i-1}, \mathsf{d}^\ell_{2i}, \tau^\ell_{2i-1}, \tau^\ell_{2i})$ and let d^ℓ_{2i-2} be the corresponding value from $w^{\ell+1}_{i^*_\ell-1}$. Whenever $M_{\ell+1}(i^*_{\ell+1}, (w^{\ell+1}_{i^*_{\ell+1}}, w^{\ell+1}_{i^*_{\ell+1}})) = 1$, it holds that

$$\begin{aligned}
1 &= \mathcal{V}_{\mathsf{del}}(\mathsf{crs}_\ell, \mathsf{hk}_\ell, (R^\ell_\pi, \mathsf{d}^{\mathsf{init}}_{\ell,2i-1}, \mathsf{d}^{\mathsf{acc}}_\ell, t_\ell), \mathsf{rt}_\ell, \{1, 2i-1, 2i\}, \pi^\ell_{2i-1}), \\
1 &= \mathcal{V}_{\mathsf{del}}(\mathsf{crs}_\ell, \mathsf{hk}_\ell, (R^\ell_\pi, \mathsf{d}^{\mathsf{init}}_{\ell,2i}, \mathsf{d}^{\mathsf{acc}}_\ell, t_\ell), \mathsf{rt}_\ell, \{1, 2i, 2i+1\}, \pi^\ell_{2i}), \\
1 &= \mathcal{V}_{\mathsf{del}}(\mathsf{crs}_\ell, \mathsf{hk}_\ell, (R^\ell_\tau, \mathsf{d}^\ell_{2i-2}, \mathsf{d}^\ell_{2i-1}, t_\ell), \mathsf{rt}_\ell, \{2i\}, \tau^\ell_{2i-1}), \text{ and} \\
1 &= \mathcal{V}_{\mathsf{del}}(\mathsf{crs}_\ell, \mathsf{hk}_\ell, (R^\ell_\tau, \mathsf{d}^\ell_{2i-1}, \mathsf{d}^\ell_{2i}, t_\ell), \mathsf{rt}_\ell, \{2i+1\}, \tau^\ell_{2i}),
\end{aligned}$$

for values $\mathsf{d}^{\mathsf{init}}_{\ell,2i-1} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}_\ell, \mathsf{cf}^{\mathsf{init}}_{2i-1})$ and $\mathsf{d}^{\mathsf{init}}_{\ell,2i} = \mathcal{D}_{\mathsf{del}}(\mathsf{hk}_\ell, \mathsf{cf}^{\mathsf{init}}_{2i})$. However, since $M_\ell(i^*_\ell, (w^\ell_{i^*_\ell-1}, w^\ell_{i^*_\ell})) \neq 1$ for values extracted according to h_ℓ, one of the two proofs π^ℓ_{2i-1} or π^ℓ_{2i} must not be accepting. Thus, the event above can occur with at most negligible probability by the partial-input soundness of DEL, in contradiction.

Note that the proof above actually shows a stronger statement: There exists a negligible function $\mu'(\lambda)$ such that for any i^*, the witnesses $w^\ell_{i^*_\ell-1}$ and $w^\ell_{i^*_\ell}$ extracted from $\mathcal{G}^*_{\mathsf{barg}}$ satisfy $M_\ell(i^*_\ell, (w^\ell_{i^*_\ell-1}, w^\ell_{i^*_\ell-1})) = 1$ whenever $\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1$ except with probability $\mu'(\lambda)$.

Witness-Unambiguity. We prove that our construction satisfies witness-unambiguity. So, for any non-uniform polynomial-time adversary $A = \{A_\lambda\}_{\lambda\in\mathbb{N}}$, polynomials m, T, k, a sequence of indices $\{i^*_\lambda \in [k(\lambda)]\}_{\lambda\in\mathbb{N}}$, we show that there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that

$$\begin{aligned}
&\Pr\left[\begin{array}{l|l} \pi \neq \pi' & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^{m(\lambda)}, T(\lambda), k(\lambda), i^*_\lambda) \\ \wedge\ \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 & (M, \pi, \pi') \leftarrow A_\lambda(\mathsf{crs}) \\ \wedge\ \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 & \end{array}\right] \\
&\leq \sum_{i=1}^{k(\lambda)} \Pr[\mathsf{DIFF}_i] + \mu(\lambda),
\end{aligned}$$

where we define the event DIFF_i to be the event where the extracted witnesses differ as follows:

$$\mathsf{DIFF}_i = \left\{\begin{array}{l|l} w_i \neq w'_i & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^{m(\lambda)}, T(\lambda), k(\lambda), i) \\ \wedge\ \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1 & (M, \pi, \pi') \leftarrow A_\lambda(\mathsf{crs}) \\ \wedge\ \mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1 & w_i = \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi) \\ & w'_i = \mathsf{Ext}_{\mathsf{barg}}(\mathsf{td}, M, \pi') \end{array}\right\}.$$

Let $\pi = (\mathsf{h}_1, \rho_1, \ldots, \mathsf{h}_{\log k}, \rho_{\log k}, w_1^{\log k+1})$ and $\pi' = (\mathsf{h}'_1, \rho'_1, \ldots, \mathsf{h}'_{\log k}, \rho'_{\log k}, w'^{\log k+1}_1)$ be the proofs output by the adversary in the experiment above for extracted index i^*_λ. We separately bound the following disjoint events when $\pi \neq \pi'$ yet $\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi) = 1$ and $\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi') = 1$:

- Let (E1) be the event where $(\mathsf{h}_1, \rho_1, \ldots, \mathsf{h}_{\log k}, \rho_{\log k}) = (\mathsf{h}_1, \rho_1, \ldots, \mathsf{h}_{\log k}, \rho_{\log k})$ but $w_1^{\log k+1} \neq w'^{\log k+1}_1$.
- Otherwise, let $\ell \in [\log k]$ be the smallest value where such that $(\mathsf{h}_\ell, \rho_\ell) \neq (\mathsf{h}'_\ell, \rho'_\ell)$. Let (E2) be the event where $\mathsf{h}_\ell = \mathsf{h}'_\ell$ but $\rho_\ell \neq \rho'_\ell$.
- Let (E3) be the event where $\mathsf{h}_\ell \neq \mathsf{h}'_\ell$ for such a smallest ℓ.

Note that these events cover all cases where $\pi \neq \pi'$.

We first observe that if $(\mathsf{h}'_1, \ldots, \mathsf{h}'_\ell) = (\mathsf{h}'_1, \ldots, \mathsf{h}'_\ell)$, then the machines $M_1, \ldots, M_{\ell+1}$ are computed identically by $\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi)$ and $\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}, M, \pi')$. In particular, in event (E1), it holds that the machines $M_{\log k+1}$ are the same in both verifications. Given both verifications are accepting, it holds that $M_{\log k+1}(1, w_1^{\log k+1}) = 1$ and $M_{\log k+1}(1, w'^{\log k+1}_1) = 1$. Let $w_1^{\log k+1} = (\pi_1, \pi_2, \mathsf{d}_1, \mathsf{d}_2, \tau_1, \tau_2)$ and similarly $w'^{\log k+1}_1 = (\pi'_1, \pi'_2, \mathsf{d}'_1, \mathsf{d}'_2, \tau'_1, \tau'_2)$. Since $\mathsf{h}_{\log k} = \mathsf{h}'_{\log k}$, it follows that $\mathsf{rt}_{\log k} = \mathsf{rt}'_{\log k}$. Also, note that $\mathsf{d}^{\mathsf{init}}_{\log k,1}$, $\mathsf{d}^{\mathsf{init}}_{\log k,2}$, $\mathsf{d}^{\mathsf{acc}}_{\log k}$, and d_0 are computed the same in both verifications. It follows that proofs π_1, π'_1 and π_2, π'_2 are both proving the same statements, so these proofs can differ each with at most negligible probability $\nu_1(\lambda)$ by the unambiguity of DEL. Next, by partial-input soundness, τ_1 can only prove that digest d_0 transitions to a unique value d_1 except with negligible probability $\nu_2(\lambda)$. Unambiguity then implies that τ_1 must be unique except with probability $\nu_1(\lambda)$. By a similar argument, τ_2 guarantees that d_2 is uniquely determined, so τ_2 must be unique, both except with $\nu_1(\lambda) + \nu_2(\lambda)$ probability. Thus, it follows that

$$\Pr[(\text{E1})] \leq 4\nu_1(\lambda) + 2\nu_2(\lambda).$$

Next, for event (E2), we have that $\mathsf{h}_\ell = \mathsf{h}'_\ell$ but $\rho_\ell \neq \rho'_\ell$. Since h_ℓ values are the same, they produce the same rt_ℓ values. As we observed above, the machines M_ℓ are also the same, so ρ_ℓ and ρ'_ℓ are both valid openings for M_ℓ whenever $\mathcal{V}_{\mathsf{barg}}$ accepts. It follows by opening unambiguity of HT that $\rho_\ell \neq \rho_\ell$ with at most negligible probability $\nu_3(\lambda)$, so

$$\Pr[(\text{E2})] \leq \nu_3(\lambda).$$

For event (E3), we're given that $\mathsf{h}_\ell \neq \mathsf{h}'_\ell$. However, since ℓ is the first such index, it follows that M_ℓ is computed the same in both verifications. To analyze this case, consider experiments where we sample $(\mathsf{crs}, \mathsf{td})$ using $\mathcal{G}^*_{\mathsf{barg}}$ to extract out witnesses $w_i^{\ell+1}$ for each $i \in [k_{\ell+1}]$.Let w'^{ℓ}_i and w'^{ℓ}_i denote the values extracted from π'. By the proof of somewhere argument of knowledge, we know that $M_{\ell+1}(i, (w_{i-1}^{\ell+1}, w_i^{\ell+1})) = 1$ for each $i \in [k_{\ell+1}]$ except with negligible probability $\mu'(\lambda)$. Also, by partial-input soundness of DEL, it follows that each statement being proved over extracted statements are valid except with some negligible probability $\nu_2(\lambda)$.

Let $\mathsf{d}^\ell_0, \ldots, \mathsf{d}^\ell_{k_\ell}$ be the sequence of extracted digests from the level $\ell+1$ witnesses during this experiment corresponding to root values $\mathsf{rt}^\ell_0, \ldots, \mathsf{rt}^\ell_{k_\ell}$ over the working memory in the configuration for each digest. We claim that the values rt^ℓ_i are equal to the correct values

$$\mathcal{H}_{\mathsf{ht}}(\mathsf{hk}_\ell, M_\ell || w_1^\ell || \ldots || w_i^\ell || 0^{t_\ell \cdot (k_\ell - i)})$$

for each $i \in [0, k_\ell]$ except with negligible probability. In addition, let DIFF_i^ℓ denote the event $w_i^\ell \neq w'^\ell_i$ given $\mathcal{V}_{\mathsf{barg}}$ accepts π and π'. In particular, note that $\mathsf{DIFF}_i^1 = \mathsf{DIFF}_i$ by construction. We claim that the digests d_i^ℓ and roots rt_i^ℓ are the same in the experiments corresponding to extracting from π or π' except with negligible probability plus $\sum_{i=1}^{k_\ell} \Pr[\mathsf{DIFF}_i^\ell]$. We argue the base case and inductive step separately below:

- **Base case.** Recall that the machine M_ℓ extracted is correct by assumption, so it has the correctly computed value d_0^ℓ hardcoded. It follows that $\mathsf{rt}_0^\ell = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}_\ell, M_\ell \| 0^{t_\ell \cdot k_\ell})$ by construction, and the digests d_0^ℓ agree for π and π' since the machines M_ℓ are the same, as required.
- **Inductive step.** Inductively assume that the digests d_i^ℓ and roots rt_i^ℓ are correctly computed and equal except with probability $\sum_{j=1}^{i-1} \Pr[\mathsf{DIFF}_j^\ell]$. Let d_{i+1}^ℓ be the next extracted digest, which is computed from $R_\tau^\ell(w_i^\ell)$ starting with configuration with digest d_i^ℓ. By partial-input soundness, it follows that $\mathsf{rt}_{i+1}^\ell = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}_\ell, M_\ell \| w_1^\ell \| \ldots \| w_{i+1}^\ell \| 0^{t_\ell \cdot (k_\ell - i - 1)})$ except with negligible probability $\nu_2(\lambda)$. Furthermore, if these values differ, it must be the case that $w_{i+1}^\ell \neq w'^\ell_{i+1}$, which occurs with probability at most $\Pr[\mathsf{DIFF}_{i+1}^\ell]$ by assumption.

It follows that the final digest $\mathsf{d}_{k_\ell}^\ell$ and roots $\mathsf{rt}_{k_\ell}^\ell$ correspond to $\mathcal{H}_{\mathsf{ht}}(\mathsf{hk}_\ell, M_\ell \| w_1^\ell \| \ldots \| w_{k_\ell}^\ell)$, which is verified to be equal to rt_ℓ, and are equal except with probability

$$\mathsf{poly}(k_\ell) \cdot (\mu'(\lambda) + \nu_2(\lambda)) + \sum_{i=1}^{k_\ell} \Pr[\mathsf{DIFF}_i^\ell].$$

We next analyze $\Pr[\mathsf{DIFF}_i^\ell]$ when $\ell > 1$. At a high level, this follows a similar logic to event (E1). As $\ell > 1$, we can parse $w_i^\ell = (\pi_{2i-1}^{\ell-1}, \pi_{2i}^{\ell-1}, \mathsf{d}_{2i-1}^{\ell-1}, \mathsf{d}_{2i}^{\ell-1}, \tau_{2i-1}^{\ell-1}, \tau_{2i}^{\ell-1})$. Furthermore, we have that $\mathsf{h}_{\ell-1} = \mathsf{h}'_{\ell-1}$ by assumption, so $\mathsf{rt}_{\ell-1} = \mathsf{rt}'_{\ell-1}$. By somewhere argument of knowledge, $M_\ell(i, (w_{i-1}^\ell, w_i^\ell)) = 1$ in both cases except with probability $\mu'(\lambda)$. It follows that $\pi_{2i-1}^{\ell-1}$ and $\pi_{2i}^{\ell-1}$ are accepting proofs for the same statement between π and π' similar to event (E1). So, by unambiguity of DEL, these values must be equal except with negligible probability $\nu_1(\lambda)$. Next, since the hash values are equal $\mathsf{h}_{\ell-1} = \mathsf{h}'_{\ell-1}$, partial-input soundness ensures that $\mathsf{d}_j^{\ell-1}$ values are equal for all $j \in [k_\ell]$, each except with at most probability $\nu_2(\lambda)$. It follows that the statements are equal for $\tau_{2i-1}^{\ell-1}$ and $\tau_{2i}^{\ell-1}$, so unambiguity again implies that these proofs must be equal except with probability $\nu_1(\lambda)$. It follows that, for $\ell > 1$,

$$\sum_{i=1}^{k_\ell} \Pr[\mathsf{DIFF}_i^\ell] \leq \mathsf{poly}(k_\ell) \cdot (\mu'(\lambda) + \nu_1(\lambda) + \nu_2(\lambda)).$$

It follows that

$$\begin{aligned}
\Pr[(\text{E3})] &\leq \sum_{\ell=1}^{\log k} \left(\mathsf{poly}(k_\ell) \cdot (\mu'(\lambda) + \nu_2(\lambda)) + \sum_{i=1}^{k_\ell} \Pr[\mathsf{DIFF}_i^\ell] \right) \\
&\leq \mathsf{poly}(k) \cdot (\mu'(\lambda) + \nu_1(\lambda) + \nu_2(\lambda)) + \sum_{i=1}^{k} \Pr[\mathsf{DIFF}_i].
\end{aligned}$$

The proof of witness-unambiguity follows by letting $\mu(\lambda)$ be such a function in $\mathsf{poly}(k) \cdot (\mu'(\lambda) + \nu_1(\lambda) + \nu_2(\lambda) + \nu_3(\lambda))$ above. □

7.2 Rate-1 Unambiguous seBARGs

We next show that a slight modification to the construction of Theorem 7.1 yields a rate-1 BARG when using a rate-1 flSEH (given in Theorem A.1) and a $\mathsf{poly}(\log T, \lambda)$-succinct unambiguous RAM delegation scheme (given from using the BARG of Theorem 7.1 in Theorem 6.4).

Theorem 7.2. Let HT be a rate-1 unambiguous, locally updatable flSEH and DEL an $\mathsf{poly}(\log T, \lambda)$-succinct unambiguous RAM delegation scheme for HT. There exists a rate-1 $\mathsf{poly}(\log k, \lambda)$-succinct witness-unambiguous seBARG. ◇

Proof. We make the following modifications to the algorithms $\mathcal{G}_{\mathsf{barg}}$ and $\mathcal{P}_{\mathsf{barg}}$ from Theorem 7.1, leaving everything else unchanged:

- $\mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^m, 1^T, k, i)$:
  - We observe that M_1 can be simply described by the input M. We hardcode M into R^1_π, which on input (i, w) outputs 1 if and only if $M(i, w) = 1$.
  - R^1_τ then can ignore the machine M_1 and simply write each witness w_i to the memory starting at location $(i-1) \cdot m + 1$.
  - For $\ell = 1$, we sample instead sample $(\mathsf{hk}_1, \mathsf{td}_1) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, m, k, i)$ to be over only the k witnesses $w_1, \ldots, w_k$ of size m.
  - We set $t_1 = T$ and $t_\ell = \max(q'(\log T, \lambda), 8q(\lambda))$ for each $\ell > 1$, where q' is the polynomial bounding the succinctness of DEL for all relevant machines.
- $\mathcal{P}_{\mathsf{barg}}(1^\lambda, 1^m, 1^T, k, i)$:
  - For $\ell = 1$, we only hash the witnesses $w_1, \ldots, w_k$ and use $\mathcal{P}_{\mathsf{del}}$ to prove each sub-statement relative to just the index i corresponding to witness w_i.

The analysis of completeness, index hiding, somewhere argument of knowledge, and witness-unambiguity remains unchanged with the above modifications. We proceed to argue that the resulting construction has the required efficiency.

Efficiency and rate-1. First, observe that $t_1 = T$ and t_ℓ for $\ell > 1$ is bounded by a fixed polynomial in λ as long as $T \leq 2^\lambda$. This implies that there exists a polynomial q' such that crs is now of size at most

$$\log k \cdot L(T, \lambda) \cdot q(\lambda) \leq q'(\lambda).$$

In terms of proof size, h_1 is of size $m + m/\lambda + \mathsf{poly}(\lambda)$ by the rate-1 property of HT. All other values in π are bounded by $\mathsf{poly}(\lambda)$ when $k, T \leq 2^\lambda$. So, there exists a polynomial q' such that

$$|\pi| \leq m + m/\lambda + q'(\lambda).$$

In terms of verifier efficiency, observe that the verifier computes $\mathsf{rt}_1 = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}_1, \mathsf{h}_1)$, which requires time $m \cdot \mathsf{poly}(\lambda)$. Everything else runs in fixed $\mathsf{poly}(\lambda)$ time, so the verifier runs in time at most $m \cdot q'(\lambda)$ for some polynomial q'. Similarly, the extractor simply extracts a value from h_1 given td_1, which takes time $q'(\lambda + m)$ by the efficiency of HT. □

8 Unambiguous Incrementally Verifiable Computation

In this section, we use rate-1 witness-unambiguous BARGs to construct unambiguous versions of mergeable delegation (Section 8.1) and incrementally verifiable computation (IVC) (Section 8.2), closely following the approach of [PP22]. We present the full construction for completeness, highlighting our difference to [PP22] necessary to achieve unambiguity.

8.1 Unambiguous Rigid Mergeable Delegation

We define an unambiguous mergeable non-interactive delegation scheme as follows. To achieve unambiguity, we modify the definition of [PP22] by requiring that level-ℓ proofs always verify computations of size 2^ℓ. We refer to this as a *rigid* mergeable delegation scheme, and note that this restriction suffices for applications to IVC. Without this additional contraint, it may be possible to compute a proof for statement $(\mathsf{cf}, \mathsf{cf}', t)$ by merging two sub-proofs for statement $(\mathsf{cf}, \mathsf{cf}^*, k)$ and $(\mathsf{cf}^*, \mathsf{cf}', t-k)$ for any $k \in [t-1]$, which makes arguing uniqueness more challenging.

Definition 8.1 (Unambiguous Rigid Mergeable Delegation). An *unambiguous rigid mergeable non-interactive delegation scheme* is a RAM delegation scheme $(\mathcal{G}_{\mathsf{del}}, \mathcal{D}_{\mathsf{del}}, \mathcal{P}_{\mathsf{del}}, \mathcal{V}_{\mathsf{del}})$ with additional algorithms $(\mathsf{Merge}, \mathsf{Level})$ with the following syntax and modifications:

- $\mathcal{G}_{\mathsf{del}}(1^\lambda) \to \mathsf{crs}$. We assume that the setup algorithm always outputs $\mathcal{G}_{\mathsf{del}}(1^\lambda, 2^\lambda)$ using time bound $T = 2^\lambda$ as input, so we omit the time bound T from the setup algorithm.
- $\mathcal{P}_{\mathsf{del}}(\mathsf{crs}, M, (\mathsf{cf}, \mathsf{cf}', t)) \to \pi$. We assume that the (level-0) prover always receives $t = 1$ as input, and otherwise aborts.
- $\mathcal{V}_{\mathsf{del}}(\mathsf{crs}, M, (\mathsf{d}, \mathsf{d}', t), \pi) \to b$. We assume that the verifier always rejects if $t \neq 2^\ell$ where $\ell = \mathsf{Level}(\pi)$.
- $\mathsf{Merge}(\mathsf{crs}, M, ((\mathsf{d}_i, \mathsf{d}'_i, t_i), \pi_i)_{i\in[2]}) \to \pi$. This is a deterministic algorithm takes as input a crs, a RAM machine M and a pair of digested statements and proofs $((\mathsf{d}_i, \mathsf{d}'_i, t_i), \pi_i)_{i\in[2]}$, and outputs a new proof π.
- $\mathsf{Level}(\pi) \to \ell$. The deterministic level algorithm takes as input a proof π, it outputs a level $\ell \in \mathbb{N}$.

We require that $(\mathcal{G}_{\mathsf{del}}, \mathcal{D}_{\mathsf{del}}, \mathcal{P}_{\mathsf{del}}, \mathcal{V}_{\mathsf{del}})$ and $(\mathsf{Merge}, \mathsf{Level})$ satisfy the following properties:

- **Completeness at Level 0.** For every $\lambda \in \mathbb{N}$, RAM machine M and statement $(\mathsf{cf}, \mathsf{cf}', 1) \in \mathcal{T}_M$, it holds that

$$\Pr\left[\begin{array}{l|l} \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, M, (\mathsf{d}, \mathsf{d}', 1), \pi) = 1 & \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda) \\ \wedge\ \mathsf{Level}(\pi) = 0 & \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}) \\ & \mathsf{d}' = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}') \\ & \pi = \mathcal{P}_{\mathsf{del}}(\mathsf{crs}, M, (\mathsf{cf}, \mathsf{cf}', 1)) \end{array}\right] = 1.$$

- **Completeness of Merge.** For every $\lambda, \ell \in \mathbb{N}$ such that $\ell < \lambda$, RAM machine M and pair of digested statements and proofs $(x_i = (\mathsf{d}_i, \mathsf{d}'_i, 2^\ell), \pi_i)_{i\in[2]}$ such that $\mathsf{d}'_1 = \mathsf{d}_2$, $\mathcal{V}_{\mathsf{del}}(\mathsf{crs}, M, x_i, \pi_i)$, $\mathsf{Level}(\pi_i) = \ell$ for every $i \in [2]$, it holds that

$$\Pr\left[\begin{array}{l|l} \mathcal{V}_{\mathsf{del}}(\mathsf{pk}, M, (\mathsf{d}_1, \mathsf{d}'_2, 2^{\ell+1}), \pi) = 1 & \pi \leftarrow \mathsf{Merge}(\mathsf{crs}, M, (x_i, \pi_i)_{i\in[2]}) \\ \wedge\ \mathsf{Level}(\pi) = \ell + 1 & \end{array}\right] = 1.$$

- **Efficiency.** In the completeness experiments above, there exists a polynomial q such that the following hold for all $t \leq 2^\lambda$:
  - $\mathcal{G}_{\mathsf{del}}$ runs in time $q(\lambda)$.
  - $\mathcal{P}_{\mathsf{del}}$ runs in time $q(\lambda + |M| + t)$ and outputs a proof of length $q(\lambda)$.
  - $\mathcal{V}_{\mathsf{del}}$ runs in time $q(\lambda + |M|)$.
  - Merge runs in time $q(\lambda + |M|)$ and outputs a proof of length $q(\lambda)$.
  - Level runs in time $q(\lambda)$.
- **Soundness for Bounded Level Proofs.** For every non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$ and polynomial T, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l|ll} \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, M, (\mathsf{d}, \mathsf{d}', 2^\ell), \pi) = 1 & : & \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda) \\ \wedge\ \mathsf{d}' \neq \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}') & & (M, \mathsf{cf}, \mathsf{d}', \ell, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge\ \ell \leq \log T(\lambda) & & \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}) \\ & & (\mathsf{cf}, \mathsf{cf}', 2^\ell) \in \mathcal{T}_M \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- **Unambiguity.** For every non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$ and polynomial T, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l|ll} & & \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda) \\ \pi \neq \pi' & & (M, \mathsf{cf}, \mathsf{d}', \ell, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, M, z, \pi) = 1 & : & \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}) \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, M, z, \pi') = 1 & & (\mathsf{cf}, \mathsf{cf}', 2^\ell) \in \mathcal{T}_M \\ \wedge\ \ell \leq \log T(\lambda) & & \mathsf{d}' = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}') \\ & & z = (\mathsf{d}, \mathsf{d}', 2^\ell) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

$\diamond$

We recall that [PP22] constructed a (plain) mergeable delegation scheme from RAM delegation and rate-1 BARGs. Via a similar construction, we construct *unambiguous* rigid mergeable delegation from unambiguous RAM delegation and rate-1 witness-unambiguous BARGs.

Theorem 8.2. Assume the existence of a rate-1 witness-unambiguous BARG and an unambiguous RAM delegation scheme, there exists an unambiguous rigid mergeable RAM delegation scheme.$\diamond$

Proof. Let $\mathsf{BARG} = (\mathcal{G}_{\mathsf{barg}}, \mathcal{P}_{\mathsf{barg}}, \mathcal{V}_{\mathsf{barg}}, \mathsf{Ext}_{\mathsf{barg}})$ be a rate-1 witness-unambiguous BARG and $\mathsf{DEL}' = (\mathsf{del}', \mathcal{D}'_{\mathsf{del}}, \mathcal{P}'_{\mathsf{del}}, \mathcal{V}'_{\mathsf{del}})$ be an unambiguous RAM delegation scheme. We procced to construct an unambiguous rigid mergeable RAM delegation scheme $(\mathcal{G}_{\mathsf{del}}, \mathcal{D}_{\mathsf{del}}, \mathcal{P}_{\mathsf{del}}, \mathcal{V}_{\mathsf{del}}, \mathsf{Merge}, \mathsf{Level})$. We first to define algorithms $(\mathcal{G}_{\mathsf{del}}, \mathcal{D}_{\mathsf{del}}, \mathcal{P}_{\mathsf{del}}, \mathsf{Level})$ as follows:

- $\mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda)$:
  1. Sample $\mathsf{crs}_0 \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda, 1)$.
  2. For $\ell \in [\lambda]$, sample $(\mathsf{crs}_\ell, \mathsf{td}_\ell) \leftarrow \mathcal{G}_{\mathsf{barg}}(1^\lambda, 1^{m'}, 1^{T'}, 2, 1)$ where m' and T' upper bound the proof length and verification time of M^ℓ defined later for all $\ell \in [\lambda]$. As shown by [PP22], m', T' are fixed polynomials in $\mathsf{poly}(\lambda)$.
  3. Output $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\lambda)$.

- $\mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf})$:
  1. Parse $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\lambda)$.
  2. Output $\mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}_0, \mathsf{cf})$.
- $\pi = \mathcal{P}_{\mathsf{del}}(\mathsf{crs}, M, (\mathsf{cf}, \mathsf{cf}', 1))$:
  1. Parse $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\lambda)$.
  2. Compute $\pi = \mathcal{P}_{\mathsf{del}}(\mathsf{crs}_0, (M, \mathsf{cf}, \mathsf{cf}', 1))$.
  3. Output $(0, \pi)$.
- $\ell = \mathsf{Level}(\pi)$:
  1. Parse $\pi = (\ell, \pi')$ and output ℓ.

To define Merge and $\mathcal{V}_{\mathsf{barg}}$, we first define helper verification algorithms. For each $\ell \in [\lambda]$, the algorithm $\mathcal{V}^\ell$ has syntax $\mathcal{V}^\ell((\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell), M, (\mathsf{d}, \mathsf{d}', t), \pi) \to b$ for the relevant values as above. We define $\mathcal{V}^0$ and $\mathcal{V}^\ell$ as follows:

- $b = \mathcal{V}^0(\mathsf{crs}_0, M, (\mathsf{d}, \mathsf{d}', 1), \pi)$:
  1. Parse $\pi = (0, \pi')$ and reject if this is not the case.
  2. Output $b = \mathcal{V}_{\mathsf{del}}(\mathsf{crs}_0, M, (\mathsf{d}, \mathsf{d}', t), \pi')$.
- $b = \mathcal{V}^\ell((\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell), M, (\mathsf{d}, \mathsf{d}', 2^\ell), \pi)$:
  1. Parse $\pi = (\ell, (x_i = (\mathsf{d}_i, \mathsf{d}'_i, 2^{\ell-1}))_{i\in[2]}, \pi')$ and reject if this is not the case.
  2. Check that $\mathsf{d}_1 = \mathsf{d}$, $\mathsf{d}'_1 = \mathsf{d}_2$, and $\mathsf{d}'_2 = \mathsf{d}'$.
  3. Let M^ℓ be a machine with hardcoded values $(\mathsf{crs}, M, x_1, x_2)$ that on input (i, π^*) outputs $\mathcal{V}^{\ell-1}((\mathsf{crs}_0, \ldots, \mathsf{crs}_{\ell-1}), M, x_i, \pi^*)$.
  4. Output $\mathcal{V}_{\mathsf{barg}}(\mathsf{crs}_\ell, M^\ell, \pi')$.

Using the above, we continue to define Merge and $\mathcal{V}_{\mathsf{del}}$ as follows:

- $\pi = \mathsf{Merge}(\mathsf{crs}, M, (x_i = (\mathsf{d}_i, \mathsf{d}'_i, t_i), \pi_i)_{i\in[2]})$:
  1. Parse $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\lambda)$.
  2. For $i \in [2]$, let $\ell_i = \mathsf{Level}(\pi_i)$ and reject if $t_i \neq 2^{\ell_i}$. Reject if $\ell_1 \neq \ell_2$ and set $\ell = \ell_1 + 1$.
  3. Let M^ℓ be the machine with hardcoded values $(\mathsf{crs}, M, x_1, x_2)$ as defined in $\mathcal{V}^\ell$ above.
  4. Compute $\pi = \mathcal{P}_{\mathsf{barg}}(\mathsf{crs}_\ell, M^\ell, (\pi_1, \pi_2))$.
  5. Output $(\ell, (x_1, x_2, \pi))$.
- $b = \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, M, (\mathsf{d}, \mathsf{d}', t), \pi)$:
  1. Parse $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\lambda)$, $\pi = (\ell, \pi')$, and reject if $t \neq 2^\ell$.
  2. Output $b = \mathcal{V}^\ell((\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell), M, (\mathsf{d}, \mathsf{d}', t), \pi')$.

The completeness requirements and the collision resistance requirement follows directly from our construction. We defer to [PP22] for the proofs of efficiency and soundness as the constructions are nearly identical. It remains to prove that this construction satisfies unambiguity.

Proof of unambiguity. For a non-uniform polynomial-size adversary $\mathcal{A}$ with associated time-bound T, we introduce the notation $\mathsf{Suc}(\mathcal{A}, \ell)$ as the probability for $\mathcal{A}$ to outputting two distinct level-ℓ proofs:

$$\mathsf{Suc}(\mathcal{A}, \ell) := \Pr\left[\begin{array}{l} \\ \pi_1 \neq \pi_2 \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, M, z, \pi_1) = 1 \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, M, z, \pi_2) = 1 \\ \\ \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda) \\ (M, \mathsf{cf}, \mathsf{d}', \ell, \pi_1, \pi_2) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}) \\ (\mathsf{cf}, \mathsf{cf}', 2^\ell) \in \mathcal{T}_M \\ \mathsf{d}' = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}') \\ z = (\mathsf{d}, \mathsf{d}', 2^\ell) \end{array}\right].$$

The proof of unambiguity follows by bounding $\mathsf{Suc}(\mathcal{A}, \ell)$ for any $\ell \leq \log T(\lambda)$. Towards this, we give a series of reductions from $\mathsf{Suc}(\mathcal{A}, \ell)$ to $\mathsf{Suc}(\mathcal{A}', \ell - 1)$ for a new adversary $\mathcal{A}'$ of comparable complexity to $\mathcal{A}$. In the base case, we observe that $\mathsf{Suc}(\mathcal{A}^*, 0)$ can be at most negligible for any polynomial-size $\mathcal{A}^*$ by the unambiguity of the underlying RAM delegation scheme DEL. We formalize this reduction in the following claim.

Lemma 8.3. For any polynomial-size adversary $\mathcal{A}$ with associated time bound T and $\ell \in [\log T(\lambda)]$, there exists a negligible function μ and an adversary $\mathcal{A}'$ for the same time bound T that calls $\mathcal{A}$ once, outputs the same machine M as $\mathcal{A}$, and runs an additional $2^\ell \cdot \mathsf{poly}(|M|, \lambda)$ steps, such that

$$\mathsf{Suc}(\mathcal{A}', \ell - 1) \geq \mathsf{Suc}(\mathcal{A}, \ell)/2 - \mu(\lambda). \qquad \diamond$$

Proof. Let π_1, π_2 denote the level-ℓ proofs output by $\mathcal{A}$, and denote Recall that a level-ℓ proof π consists of the following parts:

$$\pi_i = (\ell, (x_i^1, x_i^2, \pi_i')).$$

where $x_i^j = (\mathsf{d}_i^j, \mathsf{d}_i'^j, 2^{\ell-1})$ for $i, j \in [2]$. First, we assume without loss of generality that these statements are "well-formed" for the statement $(\mathsf{d}, \mathsf{d}', 2^\ell)$, e.g. for the case of $i = 1$, meaning that $\mathsf{d}_1^1 = \mathsf{d}$, $\mathsf{d}_1'^1 = \mathsf{d}_1^2$ for some value d^*, and $\mathsf{d}_1'^2 = \mathsf{d}'$. The verifier rejects if this is not the case, so it only restricts the success probability of $\mathcal{A}$. As such, we define $\mathsf{d}_1^{\mathsf{int}}$ and $\mathsf{d}_2^{\mathsf{int}}$ to be the intermediate digest in each case, so $\mathsf{d}_i^{\mathsf{int}} = \mathsf{d}_i'^1 = \mathsf{d}_i^2$.

With this in mind, note that if $\mathsf{d}_1^{\mathsf{int}} = \mathsf{d}_2^{\mathsf{int}}$, then the statements must be equal $(x_1^1, x_1^2) = (x_2^1, x_2^2)$. In this case $\pi_1 \neq \pi_2$ implies that the level $\ell - 1$ proofs $\pi_1' \neq \pi_2'$. We proceed by arguing that (1) $\mathsf{d}_1^{\mathsf{int}} = \mathsf{d}_2^{\mathsf{int}}$ except with some negligible probability $\mu(\lambda)$, and (2) if $\pi_1' \neq \pi_2'$, we can build an adversary $\mathcal{A}'$ that succeeds in breaking unambiguity at level $\ell - 1$.

For (1), this follows from the somewhere argument of knowledge of the underlying BARG as well as soundness at level $\ell - 1$. Namely, for each $i \in [2]$, we program the BARG to extract an accepting proof π_i^{int} from π_i' that proves that d transitions to $\mathsf{d}_j^{\mathsf{int}}$ in $2^{\ell-1}$ steps. This fails with at most $\nu_1(\lambda)$ probability by somewhere argument of knowledge for the BARG. Next, by soundness at level $\ell - 1$, it follows that $\mathsf{d}_j^{\mathsf{int}}$ must be the true digest of the resulting configuration after $2^{\ell-1}$ steps except with some probability $\nu_2(\lambda)$. It follows that $\mathsf{d}_1^{\mathsf{int}} = \mathsf{d}_2^{\mathsf{int}}$ since they must be equal to the true transitioned digest, except with probability $\mu_1(\lambda) = \nu_1(\lambda) + \nu_2(\lambda)$, as required.

For (2), observe that we're not immediately done when $\pi_1' \neq \pi_2'$ since $\mathcal{A}'$ needs to also extract corresponding statements, which is where we'll rely on witness-unambiguity of the underlying BARG. As such, we construct $\mathcal{A}'$ as follows:

1. On input $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\lambda)$, $\mathcal{A}'$ samples a random index $j \leftarrow [2]$ and samples a new BARG CRS with a trapdoor $(\mathsf{crs}'_\ell, \mathsf{td}_\ell) \leftarrow \mathcal{G}_{\mathsf{BARG}}(1^\lambda, 1^{m'}, 1^{T'}, \lambda, i^*_\ell = j)$. $\mathcal{A}'$ sets the new $\mathsf{crs}' = (\mathsf{crs}_0, \ldots, \mathsf{crs}'_\ell, \ldots, \mathsf{crs}_\lambda)$ that replaces crs_ℓ with crs'_ℓ.
2. $\mathcal{A}'$ next runs $\mathcal{A}(\mathsf{crs}')$ to get values $(M, \mathsf{cf}, \mathsf{d}', \ell, \pi_1, \pi_2)$.
3. Set $\mathsf{cf}_1 = \mathsf{cf}$ and compute cf_2 then cf_3 such that $(\mathsf{cf}_i, \mathsf{cf}_{i+1}, 2^{\ell-1}) \in \mathcal{T}_M$ for $i \in [2]$. Then, compute $\mathsf{d}_2 = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}_2)$ and $\mathsf{d}_3 = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}_3)$.
4. Compute $\pi^*_b = \mathsf{Ext}(\mathsf{td}_\ell, \pi'_b)$ for $b \in \{1, 2\}$.
5. Output $(M, \mathsf{cf}_j, \mathsf{d}_{j+1}, \ell - 1, \pi^*_1, \pi^*_2)$.

By index hiding of the underlying BARG, $\mathcal{A}(\mathsf{crs}')$ succeeds with the same probability as $\mathcal{A}(\mathsf{crs})$ up to some negligible probability $\nu_3(\lambda)$. It follows that $\pi'_1 \neq \pi'_2$ with probability $\mathsf{Suc}(\mathcal{A}, \ell) - \mu_1(\lambda) - \nu_3(\lambda)$. By the witness-unambiguity of the BARG, there exists some negligible function ν_4 such that

$$\mathsf{Suc}(\mathcal{A}, \ell) - \mu_1(\lambda) - \nu_3(\lambda) \leq 2\mathsf{Suc}(\mathcal{A}', \ell - 1) + \nu_4(\lambda).$$

Rearranging, there exists a negligible function μ such that

$$\mathsf{Suc}(\mathcal{A}', \ell - 1) \geq \mathsf{Suc}(\mathcal{A}, \ell)/2 - \mu(\lambda).$$

Also, we observe that $\mathcal{A}'$ only calls $\mathcal{A}$ once, and runs additional $2^\ell \mathsf{poly}(\lambda, |\mathsf{cf}|)$ steps, as required in terms of efficiency. This completes the proof of Lemma 8.3. □

To finish the proof of unambiguity, suppose by way of contradiction there exists a non-uniform polynomial-size adversary $\mathcal{A}$ and polynomials T, p such that for infinitely many $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l|l} & \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda) \\ \pi_1 \neq \pi_2 & (M, \mathsf{cf}, \mathsf{d}', \ell, \pi_1, \pi_2) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, M, z, \pi_1) = 1 & \mathsf{d} = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}) \\ \wedge\ \mathcal{V}_{\mathsf{del}}(\mathsf{crs}, M, z, \pi_2) = 1 & (\mathsf{cf}, \mathsf{cf}', 2^\ell) \in \mathcal{T}_M \\ \wedge\ \ell \leq \log T(\lambda) & \mathsf{d}' = \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}') \\ & z = (\mathsf{d}, \mathsf{d}', 2^\ell) \end{array}\right] > 1/p(\lambda).$$

There exists an $\ell \in [\log T(\lambda)]$ such that $\mathcal{A}$ succeeds on ℓ with probability at least $1/(\ell \cdot p(\lambda)) \geq 1/(\log T(\lambda) \cdot p(\lambda))$. Applying the above lemma $\ell \leq \log T(\lambda)$ times for this adversary $\mathcal{A}$ and this value of ℓ results in an adversary $\mathcal{A}^*$ that succeeds for $\ell^* = 0$ with probability

$$\begin{aligned} \mathsf{Suc}(\mathcal{A}^*, 0) &\geq \frac{\mathsf{Suc}(\mathcal{A}, \ell)}{2^\ell} - \ell \cdot \mu(\lambda) \\ &\geq \frac{1}{\log T(\lambda) \cdot p(\lambda) \cdot T(\lambda)} - \mathsf{negl}(\lambda), \end{aligned}$$

which is greater than $1/p'(\lambda)$ for polynomial p' for infinitely many $\lambda \in \mathbb{N}$. This contradicts the unambiguity of the underlying RAM delegation scheme that is used at level 0, in contradiction. □

Remark 6 (Using Weak Witness-Unambiguous BARGs). If we instead started with a weak witness-unambiguous BARG, we can only handle a constant number of levels when merging due to the loss in advantage at each level. To handle this case for polynomially bounded T, it would suffice to instead define a rigid delegation scheme where level ℓ proofs correspond to λ^ℓ steps of computation instead of 2^ℓ. We work with merging two proofs at a time for simplicity following [PP22] and since our BARGs achieve the stronger witness-unambiguity property. ◇

8.2 Unambiguous Incrementally Verifiable Computation

We define IVC following [PP22] with the additional property of unambiguity.

Definition 8.4 (Unambiguous IVC). An *unambiguous incrementally verifiable computation (IVC) scheme* consists of algorithms $(\mathcal{G}, \mathsf{Update}, \mathcal{V})$ with the following syntax:

- $\mathcal{G}_{\mathsf{ivc}}(1^\lambda) \to \mathsf{crs}$. This is a randomized algorithm takes as input a security parameter 1^λ and outputs a common reference string crs. We implicitly assume that crs inlcudes 1^λ.
- $\mathsf{Upd}_{\mathsf{ivc}}(\mathsf{crs}, M, (\mathsf{cf}, \mathsf{cf}', t), \pi) \to \pi'$. This is a deterministic algorithm takes as input the common reference string crs, a machine M, a statement $(\mathsf{cf}, \mathsf{cf}', t)$, and a proof π, and outputs an updated proof π'.
- $\mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, (\mathsf{cf}, \mathsf{cf}', t), \pi) \to b$. This is a deterministic algorithm that takes as input the common reference string crs, a machine M, a statement $(\mathsf{cf}, \mathsf{cf}', t)$, and a proof π, and outputs a bit b indicating whether to accept or reject the proof.

We require that $(\mathcal{G}_{\mathsf{ivc}}, \mathsf{Upd}_{\mathsf{ivc}}, \mathcal{V}_{\mathsf{ivc}})$ satisfy the following properties:

- **Incremental Completeness.** For every $\lambda, t \in \mathbb{N}$, machine M, statements $x = (\mathsf{cf}, \mathsf{cf}', t) \in \mathcal{T}_M$ and $x' = (\mathsf{cf}, \mathsf{cf}'', t+1) \in \mathcal{T}_M$, it holds that

$$\Pr\left[\begin{array}{l|l} \mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, (\mathsf{cf}, \mathsf{cf}, 0), \varepsilon) = 1 & \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{ivc}}(1^\lambda) \\ \wedge\ \mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, x', \pi') = 1 & \pi \text{ s.t. } \mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, x, \pi) = 1 \\ & \pi' = \mathsf{Upd}_{\mathsf{ivc}}(\mathsf{crs}, M, x, \pi) \end{array}\right] = 1.$$

- **Efficiency.** In the incremental completeness experiments above, there exists a polynomial q such that for all $t < 2^\lambda$, it holds that
  - $\mathcal{G}_{\mathsf{ivc}}$ runs in time $q(\lambda)$.
  - $\mathsf{Upd}_{\mathsf{ivc}}$ runs in time $q(\lambda + |M| + |x|)$ and outputs a proof π' of length $q(\lambda)$.
  - $\mathcal{V}_{\mathsf{ivc}}$ runs in time $(|M| + |x|) \cdot q(\lambda)$ and rejects all proofs of length greater than $q(\lambda)$.
- **Soundness.** For every non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$ and polynomial T, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l|l} x \notin \mathcal{T}_M & \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{ivc}}(1^\lambda) \\ \wedge\ \mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, x, \pi) = 1 & (M, x = (\mathsf{cf}, \mathsf{cf}', t), \pi) \leftarrow A_\lambda(\mathsf{crs}) \\ \wedge\ t \leq T(\lambda) & \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- **Unambiguity.** For every non-uniform polynomial-time algorithm $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$ and polynomial T, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l|l} \pi \neq \pi' & \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{ivc}}(1^\lambda) \\ \wedge\ \mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, x, \pi) = 1 & (M, x = (\mathsf{cf}, \mathsf{cf}', t), \pi, \pi') \leftarrow A_\lambda(\mathsf{crs}) \\ \wedge\ \mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, x, \pi') = 1 & \\ \wedge\ t \leq T(\lambda) & \end{array}\right] \leq \mathsf{negl}(\lambda).$$

$\diamond$

We recall that [PP22] constructed IVC from mergeable delegation. Via a similar construction, we construct *unambiguous* IVC from unambiguous rigid mergeable RAM delegation.

Theorem 8.5. Assume the existence of an unambiguous rigid mergeable RAM delegation scheme, there exists an unambiguous IVC scheme. ◇

Proof. Let $\mathsf{DEL} = (\mathcal{G}_{\mathsf{del}}, \mathcal{D}_{\mathsf{del}}, \mathcal{P}_{\mathsf{del}}, \mathcal{V}_{\mathsf{del}}, \mathsf{Merge}_{\mathsf{del}}, \mathsf{Level}_{\mathsf{del}})$ be an unambiguous rigid mergeable RAM delegation scheme. We construct an IVC scheme $(\mathcal{G}_{\mathsf{ivc}}, \mathsf{Upd}_{\mathsf{ivc}}, \mathcal{V}_{\mathsf{ivc}})$ as follows.

For a Turing machine M, let $\mathcal{R}[M]$ denote the RAM machine that emulates M. That is, $\mathcal{T}_M = \mathcal{T}_{\mathcal{R}[M]}$.

- $\mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{ivc}}(1^\lambda)$:
  1. Output $\mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{del}}(1^\lambda)$.
- $b = \mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, (\mathsf{cf}, \mathsf{cf}', t), \pi)$:
  1. If $\mathsf{cf} = \mathsf{cf}', t = 0$ and $\pi = \varepsilon$, then accept.
  2. Otherwise, let $\ell_1, \ldots, \ell_d \in [0, \lambda]$ be the distinct indices such that $t = \sum_{i \in [\lambda]} 2^{\ell_i}$ (i.e., the indices of the 1s in the binary decomposition of t).
  3. Let $\pi = \left\{(x_i = (\mathsf{d}_i, \mathsf{d}'_i, t_i), \pi_i)\right\}_{i \in [d]}$ and reject if π is not of this form.
  4. Check that $\mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}) = \mathsf{d}_1$ and $\mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}') = \mathsf{d}'_d$.
  5. For every $i \in [d-1]$, check that $\mathsf{d}'_i = \mathsf{d}_{i+1}$.
  6. For every $i \in [d]$, check that $t_i = 2^{\ell_i}$.
  7. For every $i \in [d]$, check that $\mathsf{Level}_{\mathsf{del}}(\pi_i) = \ell_i$.
  8. For every $i \in [d]$, check that $\mathcal{V}_{\mathsf{del}}(\mathsf{crs}, \mathcal{R}[M], x_i, \pi_i) = 1$.
  9. Output 1 if and only if all the above checks pass.
- $\pi' = \mathsf{Update}(\mathsf{crs}, M, (\mathsf{cf}, \mathsf{cf}', t), \pi)$:
  1. Let cf'' be the configuration such that $(\mathsf{cf}', \mathsf{cf}'', 1) \in \mathcal{T}_M$. Initialize $\pi' = \pi$.
  2. Let $x^* = (\mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}'), \mathcal{D}_{\mathsf{del}}(\mathsf{crs}, \mathsf{cf}''), 1)$, and compute $\pi^* = \mathcal{P}_{\mathsf{del}}(\mathsf{crs}, \mathcal{R}[M], (\mathsf{cf}', \mathsf{cf}'', 1))$.
  3. Add (x^*, π^*) to π'.
  4. While there exist $\left\{(x_i = (\mathsf{d}_i, \mathsf{d}'_i, t_i), \pi_i)\right\}_{i \in [\lambda]} \subseteq \pi'$ such that $\mathsf{d}'_i = \mathsf{d}_{i+1}$ for $i \in [\lambda - 1]$ and $t_1 = t_2 = \cdots = t_\lambda$:
     - Remove $\left\{(x_i = (\mathsf{d}_i, \mathsf{d}'_i, t_i), \pi_i)\right\}_{i \in [\lambda]}$ from π'.
     - Let $x^* = (\mathsf{d}_1, \mathsf{d}'_\lambda, \sum_{i \in [\lambda]} t_i)$.
     - Compute $\pi^* = \mathsf{Merge}_{\mathsf{del}}(\mathsf{crs}, \mathcal{R}[M], (x_i, \pi_i)_{i \in [\lambda]})$.
     - Add (x^*, π^*) to π'.
  5. Output π'.

We defer to [PP22] for the proofs of incremental completeness, efficiency, and soundness. We focus on the proof of unambiguity.

Proof of Unambiguity. Let $\mathcal{A}$ be a polynomial-size adversary with associated polynomial $T = T(\lambda)$. We want to give a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l|l} \pi_1 \neq \pi_2 & \\ \wedge \mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, x, \pi_1) = 1 & \mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{ivc}}(1^\lambda) \\ \wedge \mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, x, \pi_2) = 1 & (M, x = (\mathsf{cf}, \mathsf{cf}', t), \pi_1, \pi_2) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge t \leq T(\lambda) & \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Let π_1, π_2 be the accepting proofs output by $\mathcal{A}$. For each $j \in [2]$, we can parse each as

$$\pi_j = \left\{(x_j^i = (\mathsf{d}_j^i, \mathsf{d}_j'^i, t_j^i), \pi_j^i)\right\}_{i \in [d]}.$$

□

By construction of $\mathcal{V}_{\mathsf{ivc}}$ and given π_1, π_2 are accepting, the values d_j and $t_j^1, \ldots, t_j^d$ are uniquely determined by the statement $(\mathsf{cf}, \mathsf{cf}', t)$, so are equal for both π_1 and π_2. As such, we simply use d and $t^1, \ldots, t^d$ to refer to these values.

Let cf_1 denote the initial configuration, and let cf_i be the resulting configuration after $\sum_{j=1}^{i} t^j$ steps. For $i = 1, \ldots, d$, we claim that the d^i are the same for both proofs except with some negligible probability $i \cdot \nu_1(\lambda)$. This follows by construction for $i = 1$. For $i > 1$, this follows by a simple reduction to the soundness of the underlying mergeable delegation scheme, which ensures that d'^i is the correct update to d^i after t^i steps except with negligible probability $\nu_1(\lambda)$. Furthermore, the verifier checks that $\mathsf{d}^{i+1} = \mathsf{d}'^i$, which gives the claim.

Next, it follows from the unambiguity of the mergeable delegation that the proofs π^i must be the same except with negligible probability $\nu_2(\lambda)$. Thus $\pi_1 = \pi_2$ except with probability $d \cdot (\nu_1(\lambda) + \nu_2(\lambda))$, which is negligible.

9 Unambiguous IVC with Universal Soundness and PPAD Hardness

Bitansky, Paneth, Shamir, and Solomon [BPSS23] showed that with certain assumptions, the soundness of IVC can be upgraded to *universal soundness*, i.e. the computation time of the statement chosen by adversary is allowed to be super-polynomial. They also claimed that their transform preserves unambiguity, and therefore by applying the transform to our unambiguous IVC we can construct universal unambiguous IVC, which implies hardness of the a sub-class of PPAD, namely UEOPL ⊆ CLS ⊆ PPAD [FGMS19].

In this section, we give a formal exposition for both of the above results.

9.1 Universal Soundness

Definition 9.1. For an incrementally verifiable computation scheme deterministic Turing machines $(\mathcal{G}_{\mathsf{ivc}}, \mathsf{Upd}_{\mathsf{ivc}}, \mathcal{V}_{\mathsf{ivc}})$, we say the scheme has universal soundness if it satisfies:

Universal Soundness. For every $T(\lambda) \leq 2^\lambda$ and poly-size adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$:

$$\Pr\left[\begin{array}{l|l} t \leq T & \\ \wedge \mathcal{V}(\mathsf{crs}, M, x, \pi) = 1 & \mathsf{crs} \leftarrow \mathcal{G}(1^\lambda) \\ \wedge x \notin \mathcal{T}_M & (M, x = (\mathsf{cf}, \mathsf{cf}', t), \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Furthermore, we say the IVC has universal unambiguity if it satisfies:

Universal Unambiguity. For every $T(\lambda) \leq 2^\lambda$ and poly-size adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$:

$$\Pr\left[\begin{array}{l} t \leq T \\ \wedge \mathcal{V}(\mathsf{crs}, M, x, \pi) = 1 \\ \wedge \mathcal{V}(\mathsf{crs}, M, x, \pi') = 1 \\ \wedge \pi \neq \pi' \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathcal{G}(1^\lambda) \\ (M, x = (\mathsf{cf}, \mathsf{cf}', t), \pi, \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We call the IVC $(\mathcal{G}, \mathsf{Update}, \mathcal{V})$ an universal unambiguous IVC, if it has both universal soundness and universal unambiguity. ◊

[BPSS23] proved that with certain assumptions, namely fully homomorphic encryption (FHE) and a worst-case complexity assumption, any IVC can be upgraded to IVC with universal soundness. They also claim that if the original IVC is unambiguous, then after their transform the resulting IVC. For completeness, we present their transform and give a proof sketch why the transform can also upgrade unambiguity. We refer to [BPSS23] for a more detailed proof.

9.1.1 Additional Definitions

We recall the assumptions and definitions from [BPSS23] needed to state our results.

Assumption 9.2 ([BPSS23]). There exists a polynomial $s(\cdot)$ such that for any polynomial $q(\cdot)$ there exists a polynomial Q and a language $\mathcal{L} \in \mathsf{TISP}(Q, s)$ such that any family $C = \{C_\lambda\}$ of size-$q(\lambda)$ circuits fails to decide $\mathcal{L}_\lambda = \mathcal{L} \cap \{0,1\}^\lambda$ for all large enough λ.

Definition 9.3 (Average-Case Puzzles [BPSS23]). An average-case puzzle consists of the following algorithms:

- A deterministic uniform algorithm F that takes as input a difficulty parameter $t \in \mathbb{N}$ and $x \in \{0,1\}^*$ and outputs $y \in \{0,1\}^{m(|x|)}$, for some polynomial $m(\cdot)$.
- A probabilistic uniform sampler D that takes as input a difficulty parameter $t \in \mathbb{N}$ and a security parameter 1^λ and outputs $x \in \{0,1\}^{n(\lambda)}$, for some polynomial $n(\cdot)$.

For a function $t(\cdot)$, we denote by $\mathsf{F}_{t,\lambda}(x)$ the function $\mathsf{F}(t(\lambda), x)$, and by F_t the function ensemble $\{\mathsf{F}_{t,\lambda}\}_{\lambda\in\mathbb{N}}$. Similarly, we denote by $\mathsf{D}_{t,\lambda}$ the distribution $\mathsf{D}(t(\lambda), \lambda)$, and by D_t the distribution ensemble $\{\mathsf{D}(t(\lambda), \lambda)\}_{\lambda\in\mathbb{N}}$.

An average-case puzzle against non-faulty solvers satisfies the following requirements:

Efficiency. We put the following guarantees:

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

We further say that the puzzle is space efficient if $\mathsf{F}(t, x)$ runs in space $\mathsf{poly}(\log t, |x|)$.

Average-case hardness against non-faulty solvers. For every polynomial p, there exists a constant c, such that for any polynomially bounded $t \geq p^c$, $\delta \leq \frac{0.9}{p}$, any adversary $\mathcal{A}$ with size at most p such that for large enough $\lambda \in \mathbb{N}$,

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

Then, for any large enough $\lambda \in \mathbb{N}$,

$$\Pr\left[\ \mathcal{A}(x) = \mathsf{F}_{t,\lambda}(x) \ \ :\ \ x \leftarrow \mathsf{D}_{t,\lambda}\ \right] \leq \frac{1}{p(\lambda)}.$$

◊

Theorem 9.4 (Theorem 4.4 of [BPSS23]). Assuming FHE and assumption 9.2, there exists a space-efficient average-case puzzle against non-faulty solvers. ◊

9.1.2 The Transform

We present the original theorem [BPSS23] and the additional claim that the transform also upgrades unambiguity.

Theorem 9.5 (Theorem 3.2 of [BPSS23]). Assume there exists an incrementally verifiable computation scheme. Additionally, assume that there exists a space-efficient average-case puzzle against non-faulty solvers. Then there exists an incrementally verifiable computation with universal soundness. ◊

Lemma 9.6. If the IVC used in Theorem 9.5 is unambiguous, then the resulting IVC also has universal unambiguity. ◊

By breaking primitives in 9.5 into their underlying assumptions, we obtain the following theorem:

Theorem 9.7. Assume the following:

- The existence of fully homomorphic encryption;
- Assumption 9.2;
- Polynomial hardness of LWE.

Then, there exists an universal unambiguous IVC scheme. ◊

Proof sketch (Lemma 9.6). We write the original IVC scheme as $(\mathcal{G}_{\mathsf{ivc}}, \mathsf{Upd}_{\mathsf{ivc}}, \mathcal{V}_{\mathsf{ivc}})$. To construct an universal unambiguous IVC scheme $(\mathcal{G}'_{\mathsf{ivc}}, \mathsf{Upd}'_{\mathsf{ivc}}, \mathcal{V}'_{\mathsf{ivc}})$, we require that any algorithm claiming some statement $(\mathsf{cf}, \mathsf{cf}', T)$ should prove that its runs for T steps using the average-case puzzle. In particular:

- $\mathcal{G}'_{\mathsf{ivc}}(1^\lambda)$ outputs $\mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{ivc}}(1^\lambda)$, and puzzle instances $Z_0, Z_1, \ldots, Z_\lambda$ with increasing difficulty levels $2^0, 2^1, \ldots, 2^\lambda$.
- The proof π for a t-step computation contains a proof π of the original IVC scheme, and the configuration cf_i for computing $\mathsf{F}(2^i, Z_i)$ for t steps as well as an IVC proof π_i to cf_i.

- $\mathsf{Upd}'_{\mathsf{ivc}}$ parses the proof $\pi' = (\pi, (\mathsf{cf}_i, \pi_i)_{0\le i\le\lambda})$, performs the original $\mathsf{Upd}_{\mathsf{ivc}}$ to all of $\pi, \pi_0, \ldots, \pi_\lambda$, and updates $\mathsf{cf}_i \to \mathsf{cf}_i^+$.
- $\mathcal{V}'_{\mathsf{ivc}}$ reads the proof $\pi' = (\pi, (\mathsf{cf}_i, \pi_i)_{0\le i\le\lambda})$ and checks correctness for all of $\pi, \pi_0, \ldots, \pi_\lambda$ using $\mathcal{V}_{\mathsf{ivc}}$.

[BPSS23] shows the following claim by utilizing the puzzle's average-case hardness and the soundness of the original IVC scheme.

Claim 9.8 (Implicit in [BPSS23]). For every polynomial $q(\lambda)$-size adversary $\mathcal{A}$ that breaks universal soundness with probability $1/q(\lambda)$, there exists a polynomial $p(\lambda)$ such that

$$\Pr\left[\begin{array}{ll} t > p & \mathsf{crs} \leftarrow \mathcal{G}'_{\mathsf{ivc}}(1^\lambda) \\ \wedge \mathcal{V}'_{\mathsf{ivc}}(\mathsf{crs}, M, x, \pi') = 1 & : (M, x = (\mathsf{cf}, \mathsf{cf}', t), \pi') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \le \frac{1}{2q}. \qquad \diamond$$

Therefore, by applying this claim, if the $q(\lambda)$-size $\mathcal{A}$ breaks universal unambiguity, then there exists a polynomial $p(\lambda)$,

$$\Pr\left[\begin{array}{ll} t \le p & \mathsf{crs} \leftarrow \mathcal{G}'_{\mathsf{ivc}}(1^\lambda) \\ \wedge \mathcal{V}'_{\mathsf{ivc}}(\mathsf{crs}, M, x, \pi'_1) = 1 & \\ \wedge \mathcal{V}'_{\mathsf{ivc}}(\mathsf{crs}, M, x, \pi'_2) = 1 & : (M, x = (\mathsf{cf}, \mathsf{cf}', t), \pi'_1, \pi'_2) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \wedge \pi'_1 \neq \pi'_2 & \end{array}\right] \ge \frac{1}{2q}.$$

We claim that this is impossible by the soundness and unambiguity of the original IVC scheme. We write down the format of the two proofs as

$$\pi'_1 = (\pi^1, (\mathsf{cf}_i^1, \pi_i^1)_{0\le i\le\lambda}),\ \pi'_2 = (\pi^2, (\mathsf{cf}_i^2, \pi_i^2)_{0\le i\le\lambda}),$$

For each component, we have:

- As the original IVC is unambiguous, $\pi^1 = \pi^2$.
- For every $0 \le i \le \lambda$, by the soundness of the original IVC scheme, both $\mathsf{cf}_i^1, \mathsf{cf}_i^2$ equals to the correct configuration of running $\mathsf{F}(2^i, Z_i)$ for t steps.
- For every $0 \le i \le \lambda$, given that $\mathsf{cf}_i^1 = \mathsf{cf}_i^2$, we have $\pi_i^1 = \pi_i^2$ as otherwise they would break the unambiguity of the original IVC scheme. $\square$

9.2 PPAD Hardness

We show hardness of UEOPL (Definition C.3), which is a subclass of CLS.

The following theorem shows that a hard search problem and universal unambiguous IVC implies that UEOPL is hard.

Theorem 9.9. Let $\mathcal{R}$ be a search problem that is solvable by a deterministic Turing machine M that runs in time $T(n) = n^{\omega(1)}$ (we assume $T(n) \le 2^n$) and space $S(n) = \mathsf{poly}(n)$, and $\mathcal{R}$ is hard for poly-size algorithms in the average-case for an efficiently samplable distribution (respectively in the worst-case).

If there exists an universal unambiguous incrementally verifiable computation scheme for deterministic Turing machines, then UNIQUEFORWARDEOML is hard for poly-size algorithms in the average-case (respectively in the worst-case). $\diamond$

Proof. We focus on the setting of average-case hardness. The proof for worst-case hardness is similar.

Assume $\mathcal{R}$ is hard with respect to a distribution $\mathcal{D} = \{\mathcal{D}_n\}$, and let $(\mathcal{G}_{\mathsf{ivc}}, \mathsf{Upd}_{\mathsf{ivc}}, \mathcal{V}_{\mathsf{ivc}})$ be the universal unambiguous IVC scheme as in the theorem statement. We take $n(\lambda) = \mathsf{poly}(\lambda)$ such that $T(n) \leq 2^{\lambda}$. We construct a hard distribution of UNIQUEFORWARDEOML as follows:

1. Sample $x \leftarrow \mathcal{D}_n$. Sample $\mathsf{crs} \leftarrow \mathcal{G}_{\mathsf{ivc}}(1^{\lambda})$. Set m to be the sum of $S(n)$ and the proof length of the IVC scheme.
2. Let cf_0 be the initial configuration of the Turing machine M on input x. We assume without loss of generality that at every time step, the configuration of M contains an index $i \in \{0, \ldots, T\}$ corresponding to the current time step. Let $v_0 = (\mathsf{cf}_0, \pi_0 = \varepsilon)$.
3. Let $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$ be the circuit that on input $v = (\mathsf{cf}_i, \pi_i)$, performs the following computation:
   (a) Let i be the current time step contained in cf_i.
   (b) Check that $\mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, (\mathsf{cf}_0, \mathsf{cf}_i, i), \pi_i) = 1$. Otherwise, output v and abort.
   (c) If $i \geq T$ then output v and abort.
   (d) Let cf_{i+1} be the configuration of the next step such that $(\mathsf{cf}_i, \mathsf{cf}_{i+1}, 1) \in \mathcal{T}_M$. Compute $\pi_{i+1} = \mathsf{Upd}_{\mathsf{ivc}}(\mathsf{crs}, M, (\mathsf{cf}_0, \mathsf{cf}_i, i), \pi_i)$.
   (e) Output $(\mathsf{cf}_{i+1}, \pi_{i+1})$.
4. Let $\mathsf{M} : \{0,1\}^m \to \{0, \ldots, 2^m - 1\}$ be the circuit that on input $v = (\mathsf{cf}_i, \pi_i)$, parse and output i as the current time step contained in cf_i.
5. The UNIQUEFORWARDEOML is $(\mathsf{S}, \mathsf{M}, v_0)$.

It is easy to see that by completeness of the IVC scheme, the UNIQUEFORWARDEOML $(\mathsf{S}, \mathsf{M}, v_0)$ satisfies the guarantees. Now we analyze its hardness. In particular, we show that it is computationally hard to find each kind of solution:

- **Sink.** Assume a poly-size adversary $\mathcal{A}$ finds a sink $v = (\mathsf{cf}_{T-1}, \pi_{T-1})$. Write $u = \mathsf{S}(v) = (\mathsf{cf}_T, \pi_T)$. By incremental completeness, we have $\mathcal{V}_{\mathsf{ivc}}(\mathsf{crs}, M, (\mathsf{cf}_0, \mathsf{cf}_T, \mathsf{M}(u)), \pi_T) = 1$ and therefore to have $\mathsf{S}(u) = u$, we have $\mathsf{M}(u) = T$. By universal soundness of the IVC scheme, for any $x \leftarrow \mathcal{D}_n$,

$$\Pr\left[\begin{array}{l|ll} \mathcal{V}(\mathsf{crs}, M, (\mathsf{cf}_0, \mathsf{cf}_T, T), \pi_T) = 1 & & \mathsf{crs} \leftarrow \mathcal{G}(1^{\lambda}) \\ \wedge (\mathsf{cf}_0, \mathsf{cf}_T, T) \notin \mathcal{T}_M & : & v \leftarrow \mathcal{A}(\mathsf{S}, \mathsf{M}, v_0) \\ & & (\mathsf{cf}_T, \pi_T) = u = \mathsf{S}(v) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

  Hence for any poly-size $\mathcal{A}$ that finds sink with noticeable probability, it computes the correct configuration cf_T of M running T steps, and therefore solves the search problem $\mathcal{R}$ on input x with noticeable probability. This contradicts the average-case hardness of $\mathcal{R}$.

- **Violation of meter.** Such solution does not exist. By our construction of S, for any vertex $v = (\mathsf{cf}_i, \pi_i)$ if $\mathsf{S}(v) \neq v$, then we have $\mathsf{S}(v) = (\mathsf{cf}_{i+1}, \pi_{i+1})$ and $\mathsf{M}(\mathsf{S}(v)) = \mathsf{M}(v) + 1$.

- **Collision of meter.** Assume a poly-size adversary $\mathcal{A}$ finds two vertices $u = (\mathsf{cf}_i, \pi_i), v = (\mathsf{cf}'_i, \pi'_i)$ with probability $1/p(\lambda)$ (Denote $i = \mathsf{M}(u) = \mathsf{v}$) and both π_i, π'_i are accepting IVC proofs. By universal soundness of the IVC scheme, for any $x \leftarrow \mathcal{D}_n$,

$$\Pr\left[\begin{array}{ll} & \mathsf{crs} \leftarrow \mathcal{G}(1^\lambda) \\ \mathcal{V}(\mathsf{crs}, M, (\mathsf{cf}_0, \mathsf{cf}_i, i), \pi_i) = 1 & (u, v) \leftarrow \mathcal{A}(\mathsf{S}, \mathsf{M}, v_0) \\ \wedge \mathcal{V}(\mathsf{crs}, M, (\mathsf{cf}_0, \mathsf{cf}'_i, i), \pi'_i) = 1 & : \ (\mathsf{cf}_i, \pi_i) = u \\ \wedge((\mathsf{cf}_0, \mathsf{cf}_i, i) \notin \mathcal{T}_M \vee (\mathsf{cf}_0, \mathsf{cf}'_i, i) \notin \mathcal{T}_M) & (\mathsf{cf}'_i, \pi'_i) = v \end{array}\right] \leq \mathsf{negl}(\lambda).$$

  Therefore, with probability $1/2p(\lambda)$ both $\mathsf{cf}_i, \mathsf{cf}'_i$ are correct configuration of M running for i steps. Since M is deterministic, we have $\mathsf{cf}_i = \mathsf{cf}'_i$, i.e.

$$\Pr\left[\begin{array}{ll} & \mathsf{crs} \leftarrow \mathcal{G}(1^\lambda) \\ i \leq T & \\ \mathcal{V}(\mathsf{crs}, M, (\mathsf{cf}_0, \mathsf{cf}_i, i), \pi_i) = 1 & (u, v) \leftarrow \mathcal{A}(\mathsf{S}, \mathsf{M}, v_0) \\ \wedge \mathcal{V}(\mathsf{crs}, M, (\mathsf{cf}_0, \mathsf{cf}'_i, i), \pi'_i) = 1 & : \ (\mathsf{cf}_i, \pi_i) = u \\ \wedge \mathsf{cf}_i = \mathsf{cf}'_i & (\mathsf{cf}'_i, \pi'_i) = v \\ \wedge \pi_i \neq \pi'_i & \end{array}\right] \geq \frac{1}{2p(\lambda)}.$$

  This contradicts with universal unambiguity of the IVC scheme. □

Acknowledgements

Cody Freitag is supported by the Khoury College Distinguished Postdoctoral Fellowship. Daniel Wichs is supported by NSF CNS-2349972 and CNS-2055510. We also thank Yiding Zhang for the helpful discussion.

References

[Foc] *63rd FOCS*. IEEE Computer Society Press, 2022.

[AFGJM22] Behzad Abdolmaleki, Nils Fleischhacker, Vipul Goyal, Abhishek Jain, and Giulio Malavolta. "Steganography-Free Zero-Knowledge". In: *TCC 2022, Part I*. Ed. by Eike Kiltz and Vinod Vaikuntanathan. Vol. 13747. LNCS. Springer, Cham, Nov. 2022, pp. 143–172. DOI: 10.1007/978-3-031-22318-1_6 (cit. on p. 2).

[ALMSS98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. "Proof Verification and the Hardness of Approximation Problems". In: *J. ACM* 45.3 (1998), pp. 501–555 (cit. on pp. 6, 19, 20, 85).

[BBHMR19] James Bartusek, Liron Bronfman, Justin Holmgren, Fermi Ma, and Ron D. Rothblum. "On the (In)security of Kilian-Based SNARGs". In: *TCC 2019, Part II*. Ed. by Dennis Hofheinz and Alon Rosen. Vol. 11892. LNCS. Springer, Cham, Dec. 2019, pp. 522–551. DOI: 10.1007/978-3-030-36033-7_20 (cit. on pp. 2, 3).

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 459–474. DOI: 10.1109/SP.2014.36 (cit. on p. 2).

[BCIOP13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. "Succinct Non-interactive Arguments via Linear Interactive Proofs". In: *TCC 2013*. Ed. by Amit Sahai. Vol. 7785. LNCS. Springer, Berlin, Heidelberg, Mar. 2013, pp. 315–333. DOI: 10.1007/978-3-642-36594-2_18 (cit. on pp. 19, 85).

[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: *TCC 2022, Part II*. Ed. by Eike Kiltz and Vinod Vaikuntanathan. Vol. 13748. LNCS. Springer, Cham, Nov. 2022, pp. 593–622. DOI: 10.1007/978-3-031-22365-5_21 (cit. on pp. 1, 2, 88).

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. "On the Cryptographic Hardness of Finding a Nash Equilibrium". In: *56th FOCS*. Ed. by Venkatesan Guruswami. IEEE Computer Society Press, Oct. 2015, pp. 1480–1498. DOI: 10.1109/FOCS.2015.94 (cit. on p. 1).

[BPSS23] Nir Bitansky, Omer Paneth, Dana Shamir, and Tomer Solomon. "Non-interactive Universal Arguments". In: *CRYPTO 2023, Part II*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14082. LNCS. Springer, Cham, Aug. 2023, pp. 132–158. DOI: 10.1007/978-3-031-38545-2_5 (cit. on pp. 2, 3, 4, 6, 70, 71, 72, 73).

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. "Fully Key-Homomorphic Encryption, Arithmetic Circuit ABE and Compact Garbled Circuits". In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Berlin, Heidelberg, May 2014, pp. 533–556. DOI: 10.1007/978-3-642-55220-5_30 (cit. on p. 1).

[BRF13] Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, eds. *45th ACM STOC*. ACM Press, June 2013.

[BBKLP23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. "SNARGs for Monotone Policy Batch NP". In: *CRYPTO 2023, Part II*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14082. LNCS. Springer, Cham, Aug. 2023, pp. 252–283. DOI: 10.1007/978-3-031-38545-2_9 (cit. on p. 2).

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. "Efficient Fully Homomorphic Encryption from (Standard) LWE". In: *52nd FOCS*. Ed. by Rafail Ostrovsky. IEEE Computer Society Press, Oct. 2011, pp. 97–106. DOI: 10.1109/FOCS.2011.12 (cit. on p. 1).

[CPW20] Suvradip Chakraborty, Manoj Prabhakaran, and Daniel Wichs. "Witness Maps and Applications". In: *Public Key Cryptography (1)*. Vol. 12110. Lecture Notes in Computer Science. Springer, 2020, pp. 220–246 (cit. on p. 5).

[CDT09] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. "Settling the complexity of computing two-player Nash equilibria". In: *J. ACM* 56.3 (2009), 14:1–14:57 (cit. on p. 1).

[CGJJZ23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. "Correlation Intractability and SNARGs from Sub-exponential DDH". In: *CRYPTO 2023, Part IV*. Ed. by Helena Handschuh and Anna Lysyanskaya. Vol. 14084. LNCS. Springer, Cham, Aug. 2023, pp. 635–668. DOI: 10.1007/978-3-031-38551-3_20 (cit. on p. 2).

[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: *51st ACM STOC*. Ed. by Moses Charikar and Edith Cohen. ACM Press, June 2019, pp. 1103–1114. DOI: 10.1145/3313276.3316400 (cit. on pp. 1, 2, 6, 88).

[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, Paper 2019/667. 2019. URL: https://eprint.iacr.org/2019/667 (cit. on pp. 1, 2).

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. "Non-interactive Batch Arguments for NP from Standard Assumptions". In: *CRYPTO 2021, Part IV*. Ed. by Tal Malkin and Chris Peikert. Vol. 12828. LNCS. Virtual Event: Springer, Cham, Aug. 2021, pp. 394–423. DOI: 10.1007/978-3-030-84259-8_14 (cit. on pp. 2, 6, 7, 8, 10).

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. "SNARGs for $\mathcal{P}$ from LWE". In: *62nd FOCS*. IEEE Computer Society Press, Feb. 2022, pp. 68–79. DOI: 10.1109/FOCS52979.2021.00016 (cit. on pp. 1, 6, 10, 11, 21, 51).

[CLP15] Kai-Min Chung, Huijia Lin, and Rafael Pass. "Constant-Round Concurrent Zero-Knowledge from Indistinguishability Obfuscation". In: *CRYPTO 2015, Part I*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9215. LNCS. Springer, Berlin, Heidelberg, Aug. 2015, pp. 287–307. DOI: 10.1007/978-3-662-47989-6_14 (cit. on p. 2).

[DGP09] Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. "The Complexity of Computing a Nash Equilibrium". In: *SIAM J. Comput.* 39.1 (2009), pp. 195–259 (cit. on p. 1).

[DP11] Constantinos Daskalakis and Christos H. Papadimitriou. "Continuous Local Search". In: *SODA*. SIAM, 2011, pp. 790–804 (cit. on p. 88).

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. "Rate-1 Non-Interactive Arguments for Batch-NP and Applications". In: *63rd FOCS*. IEEE Computer Society Press, 2022, pp. 1057–1068. DOI: 10.1109/FOCS54457.2022.00103 (cit. on pp. 2, 4, 6, 10, 13, 14, 16, 25).

[EFKP20a] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. "Continuous Verifiable Delay Functions". In: *EUROCRYPT 2020, Part III*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12107. LNCS. Springer, Cham, May 2020, pp. 125–154. DOI: 10.1007/978-3-030-45727-3_5 (cit. on pp. 1, 2).

[EFKP20b] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. "SPARKs: Succinct Parallelizable Arguments of Knowledge". In: *EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. LNCS. Springer, Cham, May 2020, pp. 707–737. DOI: 10.1007/978-3-030-45721-1_25 (cit. on p. 24).

[FGHS21] John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. "The complexity of gradient descent:CLS = PPAD ∩ PLS". In: *53rd ACM STOC*. Ed. by Samir Khuller and Virginia Vassilevska Williams. ACM Press, June 2021, pp. 46–59. DOI: 10.1145/3406325.3451052 (cit. on p. 88).

[FGMS19] John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. "Unique End of Potential Line". In: *ICALP*. Vol. 132. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 56:1–56:15 (cit. on pp. 70, 88).

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. "Revisiting the Cryptographic Hardness of Finding a Nash Equilibrium". In: *CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, Berlin, Heidelberg, Aug. 2016, pp. 579–604. DOI: 10.1007/978-3-662-53008-5_20 (cit. on p. 1).

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. "Quadratic Span Programs and Succinct NIZKs without PCPs". In: *EUROCRYPT 2013*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Berlin, Heidelberg, May 2013, pp. 626–645. DOI: 10.1007/978-3-642-38348-9_37 (cit. on p. 19).

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices". In: *41st ACM STOC*. Ed. by Michael Mitzenmacher. ACM Press, 2009, pp. 169–178. DOI: 10.1145/1536414.1536440 (cit. on p. 1).

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. "Trapdoors for hard lattices and new cryptographic constructions". In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 197–206. DOI: 10.1145/1374376.1374407 (cit. on p. 1).

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based". In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Berlin, Heidelberg, Aug. 2013, pp. 75–92. DOI: 10.1007/978-3-642-40041-4_5 (cit. on p. 1).

[GKPVZ13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. "Reusable garbled circuits and succinct functional encryption". In: *45th ACM STOC*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM Press, June 2013, pp. 555–564. DOI: 10.1145/2488608.2488678 (cit. on p. 1).

[GO93] Shafi Goldwasser and Rafail Ostrovsky. "Invariant Signatures and Non-Interactive Zero-Knowledge Proofs are Equivalent (Extended Abstract)". In: *CRYPTO'92*. Ed. by Ernest F. Brickell. Vol. 740. LNCS. Springer, Berlin, Heidelberg, Aug. 1993, pp. 228–245. DOI: 10.1007/3-540-48071-4_16 (cit. on p. 2).

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. "Attribute-based encryption for circuits". In: *45th ACM STOC*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM Press, June 2013, pp. 545–554. DOI: 10.1145/2488608.2488677 (cit. on p. 1).

[HL23] Helena Handschuh and Anna Lysyanskaya, eds. *CRYPTO 2023, Part II*. Vol. 14082. LNCS. Springer, Cham, Aug. 2023.

[HLR21] Justin Holmgren, Alex Lombardi, and Ron D. Rothblum. "Fiat-Shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge)". In: *53rd ACM STOC*. Ed. by Samir Khuller and Virginia Vassilevska Williams. ACM Press, June 2021, pp. 750–760. DOI: 10.1145/3406325.3451116 (cit. on pp. 8, 9, 20, 21).

[HY20] Pavel Hubácek and Eylon Yogev. "Hardness of Continuous Local Search: Query Complexity and Cryptographic Lower Bounds". In: *SIAM J. Comput.* 49.6 (2020), pp. 1128–1172 (cit. on p. 1).

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. "SNARGs for P from Sub-exponential DDH and QR". In: *EUROCRYPT 2022, Part II*. Ed. by Orr Dunkelman and Stefan Dziembowski. Vol. 13276. LNCS. Springer, Cham, 2022, pp. 520–549. DOI: 10.1007/978-3-031-07085-3_18 (cit. on p. 1).

[IKO07] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. "Efficient Arguments without Short PCPs". In: *CCC*. IEEE Computer Society, 2007, pp. 278–291 (cit. on p. 19).

[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: *53rd ACM STOC*. Ed. by Samir Khuller and Virginia Vassilevska Williams. ACM Press, June 2021, pp. 708–721. DOI: 10.1145/3406325.3451055 (cit. on pp. 1, 2).

[JKLV24] Zhengzhong Jin, Yael Kalai, Alex Lombardi, and Vinod Vaikuntanathan. "SNARGs under LWE via Propositional Proofs". In: *56th ACM STOC*. Ed. by Bojan Mohar, Igor Shinkar, and Ryan O'Donnell. ACM Press, June 2024, pp. 1750–1757. DOI: 10.1145/3618260.3649770 (cit. on p. 2).

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. "Boosting Batch Arguments and RAM Delegation". In: *55th ACM STOC*. Ed. by Barna Saha and Rocco A. Servedio. ACM Press, June 2023, pp. 1545–1552. DOI: 10.1145/3564246.3585200 (cit. on pp. 6, 10, 13, 21, 45, 46, 49, 51, 53, 55, 58, 81, 82).

[KLV23] Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. "SNARGs and PPAD Hardness from the Decisional Diffie-Hellman Assumption". In: *EUROCRYPT 2023, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. LNCS. Springer, Cham, Apr. 2023, pp. 470–498. DOI: 10.1007/978-3-031-30617-4_16 (cit. on pp. 1, 2).

[KP16] Yael Tauman Kalai and Omer Paneth. "Delegating RAM Computations". In: *TCC 2016-B, Part II*. Ed. by Martin Hirt and Adam D. Smith. Vol. 9986. LNCS. Springer, Berlin, Heidelberg, 2016, pp. 91–118. DOI: 10.1007/978-3-662-53644-5_4 (cit. on pp. 23, 24, 46, 51).

[KPY20] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. "Delegation with Updatable Unambiguous Proofs and PPAD-Hardness". In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Cham, Aug. 2020, pp. 652–673. DOI: 10.1007/978-3-030-56877-1_23 (cit. on pp. 1, 2, 3, 6).

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. "Somewhere Statistical Soundness, Post-Quantum Security, and SNARGs". In: *TCC 2021, Part I*. Ed. by Kobbi Nissim and Brent Waters. Vol. 13042. LNCS. Springer, Cham, Nov. 2021, pp. 330–368. DOI: 10.1007/978-3-030-90459-3_12 (cit. on pp. 1, 21).

[KW21] Samir Khuller and Virginia Vassilevska Williams, eds. *53rd ACM STOC*. ACM Press, June 2021.

[Kiy22] Susumu Kiyoshima. "Public-Coin 3-Round Zero-Knowledge from Learning with Errors and Keyless Multi-Collision-Resistant Hash". In: *CRYPTO 2022, Part I*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13507. LNCS. Springer, Cham, Aug. 2022, pp. 444–473. DOI: 10.1007/978-3-031-15802-5_16 (cit. on p. 2).

[LMs05] Matt Lepinski, Silvio Micali, and abhi shelat. "Fair-Zero Knowledge". In: *TCC 2005*. Ed. by Joe Kilian. Vol. 3378. LNCS. Springer, Berlin, Heidelberg, Feb. 2005, pp. 245–263. DOI: 10.1007/978-3-540-30576-7_14 (cit. on p. 2).

[LV20] Alex Lombardi and Vinod Vaikuntanathan. "Fiat-Shamir for Repeated Squaring with Applications to PPAD-Hardness and VDFs". In: *CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. LNCS. Springer, Cham, Aug. 2020, pp. 632–651. DOI: 10.1007/978-3-030-56877-1_22 (cit. on pp. 1, 2).

[Lys02] Anna Lysyanskaya. "Unique Signatures and Verifiable Random Functions from the DH-DDH Separation". In: *CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. LNCS. Springer, Berlin, Heidelberg, Aug. 2002, pp. 597–612. DOI: 10.1007/3-540-45708-9_38 (cit. on p. 2).

[Mer90] Ralph C. Merkle. "A Certified Digital Signature". In: *CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. LNCS. Springer, New York, Aug. 1990, pp. 218–238. DOI: 10.1007/0-387-34805-0_21 (cit. on p. 23).

[Mic94] Silvio Micali. "CS Proofs (Extended Abstracts)". In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 436–453. DOI: 10.1109/SFCS.1994.365746 (cit. on p. 1).

[MP12] Daniele Micciancio and Chris Peikert. "Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller". In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Berlin, Heidelberg, Apr. 2012, pp. 700–718. DOI: 10.1007/978-3-642-29011-4_41 (cit. on p. 18).

[MR20] Daniele Micciancio and Thomas Ristenpart, eds. *CRYPTO 2020, Part III*. Vol. 12172. LNCS. Springer, Cham, Aug. 2020.

[MSO24] Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, eds. *56th ACM STOC*. ACM Press, June 2024.

[Nas51] John Nash. "Non-Cooperative Games". In: *The Annals of Mathematics* 54 (1951), pp. 286–295 (cit. on p. 1).

[NT16] Assa Naveh and Eran Tromer. "PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations". In: *2016 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2016, pp. 255–271. DOI: 10.1109/SP.2016.23 (cit. on p. 2).

[PP22] Omer Paneth and Rafael Pass. "Incrementally Verifiable Computation via Rate-1 Batch Arguments". In: *63rd FOCS*. IEEE Computer Society Press, 2022, pp. 1045–1056. DOI: 10.1109/FOCS54457.2022.00102 (cit. on pp. 2, 3, 4, 6, 10, 13, 14, 16, 26, 27, 45, 46, 63, 64, 65, 67, 68, 69, 81, 82).

[Pap94] Christos H. Papadimitriou. "On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence". In: *J. Comput. Syst. Sci.* 48.3 (1994), pp. 498–532 (cit. on p. 1).

[PS19] Chris Peikert and Sina Shiehian. "Noninteractive Zero Knowledge for NP from (Plain) Learning with Errors". In: *CRYPTO 2019, Part I*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. LNCS. Springer, Cham, Aug. 2019, pp. 89–114. DOI: 10.1007/978-3-030-26948-7_4 (cit. on p. 18).

[Reg05] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *37th ACM STOC*. Ed. by Harold N. Gabow and Ronald Fagin. ACM Press, May 2005, pp. 84–93. DOI: 10.1145/1060590.1060603 (cit. on p. 1).

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. "Constant-round interactive proofs for delegating computation". In: *48th ACM STOC*. Ed. by Daniel Wichs and Yishay Mansour. ACM Press, June 2016, pp. 49–62. DOI: 10.1145/2897518.2897652 (cit. on p. 2).

[SW14] Amit Sahai and Brent Waters. "How to use indistinguishability obfuscation: deniable encryption, and more". In: *46th ACM STOC*. Ed. by David B. Shmoys. ACM Press, 2014, pp. 475–484. DOI: 10.1145/2591796.2591825 (cit. on p. 2).

[Val76] Leslie G. Valiant. "Relative Complexity of Checking and Evaluating". In: *Inf. Process. Lett.* 5.1 (1976), pp. 20–23 (cit. on p. 2).

[Val08] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *TCC 2008*. Ed. by Ran Canetti. Vol. 4948. LNCS. Springer, Berlin, Heidelberg, Mar. 2008, pp. 1–18. DOI: 10.1007/978-3-540-78524-8_1 (cit. on p. 2).

[WW22] Brent Waters and David J. Wu. "Batch Arguments for NP and More from Standard Bilinear Group Assumptions". In: *CRYPTO 2022, Part II*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13508. LNCS. Springer, Cham, Aug. 2022, pp. 433–463. DOI: 10.1007/978-3-031-15979-4_15 (cit. on p. 2).

[WW24a] Brent Waters and David J. Wu. *A Pure Indistinguishability Obfuscation Approach to Adaptively-Sound SNARGs for NP*. Cryptology ePrint Archive, Paper 2024/933. 2024. URL: https://eprint.iacr.org/2024/933 (cit. on p. 2).

[WW24b] Brent Waters and David J. Wu. "Adaptively-Sound Succinct Arguments for NP from Indistinguishability Obfuscation". In: *56th ACM STOC*. Ed. by Bojan Mohar, Igor Shinkar, and Ryan O'Donnell. ACM Press, June 2024, pp. 387–398. DOI: 10.1145/3618260.3649671 (cit. on p. 2).

[WZ24] Brent Waters and Mark Zhandry. "Adaptive Security in SNARGs via iO and Lossy Functions". In: *CRYPTO 2024, Part X*. Ed. by Leonid Reyzin and Douglas Stebila. Vol. 14929. LNCS. Springer, Cham, Aug. 2024, pp. 72–104. DOI: 10.1007/978-3-031-68403-6_3 (cit. on p. 2).

A Rate-1 Fully Local Somewhere Extractable Hash Trees

We construct a hash tree that satisfies all of the properties in Section 3.5 following the implicit rate-1 flSEH construction in [PP22; KLVW23] given any rate-1 SEH. [KLVW23] observed that rate-1 SEH can be built from any rate-1 string oblivious transfer (OT) protocol, which is known from any of the following cryptographic assumptions: LWE, QR, DCR, DDH, or $O(1)$-LIN. We observe that such a rate-1 SEH is also unambiguous and locally updatable, which follows follows from

the standard Merkle tree structure given the collision-resistance implicit in the rate-1 string OT protocol.

Theorem A.1 ([PP22; KLVW23]). Assuming a rate-1 string OT protocol, there exists a locally updatable, unambiguous, rate-1 flSEH. ◊

Proof (Proof Sketch). Let $\mathsf{HT} = (\mathcal{G}'_{\mathsf{ht}}, \mathcal{H}'_{\mathsf{ht}}, \mathcal{P}'_{\mathsf{ht}}, \mathcal{V}'_{\mathsf{ht}})$ be algorithms for an unambiguous hash tree with local update algorithms $(\mathsf{Rd}'_{\mathsf{ht}}, \mathsf{Wrt}'_{\mathsf{ht}}, \mathsf{VerWrt}'_{\mathsf{ht}})$. Let $\mathsf{SEH} = (\mathcal{G}_{\mathsf{seh}}, \mathcal{H}_{\mathsf{seh}}, \mathcal{P}_{\mathsf{seh}}, \mathcal{V}_{\mathsf{seh}}, \mathcal{E}_{\mathsf{seh}})$ be a rate-1 unambiguous SEH with additive overhead $m/(2\lambda) + q_{\mathsf{seh}}(\lambda)$ and local update algorithms $(\mathsf{Rd}_{\mathsf{seh}}, \mathsf{Wrt}_{\mathsf{seh}}, \mathsf{VerWrt}_{\mathsf{seh}})$. Define $\ell(\lambda) = 2\lambda \cdot q_{\mathsf{seh}}(\lambda)$. At a high level, we use SEH to create hash trees over ℓ-size blocks of each input x_i, which we then hash to a small root using HT. We proceed to describe the flSEH algorithms $(\mathcal{G}_{\mathsf{ht}}, \mathcal{H}_{\mathsf{ht}}, \mathcal{D}_{\mathsf{ht}}, \mathcal{P}_{\mathsf{ht}}, \mathcal{V}_{\mathsf{ht}}, \mathcal{E}_{\mathsf{ht}})$ and local update algorithms $(\mathsf{Rd}_{\mathsf{ht}}, \mathsf{Wrt}_{\mathsf{ht}}, \mathsf{VerWrt}_{\mathsf{ht}})$ below:

- $\mathcal{G}_{\mathsf{ht}}(1^\lambda, m, k, i^*) \to (\mathsf{hk}, \mathsf{td})$: We first sample a hash key a trapdoor for seh used for blocks of the inputs, so $(\mathsf{hk}_{\mathsf{seh}}, \mathsf{td}_{\mathsf{seh}}) \leftarrow \mathcal{G}_{\mathsf{seh}}(1^\lambda, \ell(\lambda), k, i^*)$. We next sample a hash key for the plain hash tree HT, so $\mathsf{hk}_{\mathsf{ht}} \leftarrow \mathcal{G}'_{\mathsf{ht}}(1^\lambda, m/\ell, \ell + \ell/2\lambda + q_{\mathsf{seh}}(\lambda))$. Output $\mathsf{hk} = (\mathsf{hk}_{\mathsf{seh}}, \mathsf{hk}_{\mathsf{ht}})$ and $\mathsf{td} = \mathsf{td}_{\mathsf{seh}}$.

- $\mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k)) \to \mathsf{h}$. For $j \in [m/\ell]$, let $x_i^j = x_i[(j-1)\cdot\ell + 1, j\cdot\ell]$ and compute $(\mathsf{h}_j, \mathsf{tree}_j) = \mathcal{H}_{\mathsf{seh}}(\mathsf{hk}_{\mathsf{seh}}, (x_1^j, \ldots, x_k^j))$. Output $\mathsf{h} = (\mathsf{h}_1, \ldots, \mathsf{h}_{m/\ell})$.

- $\mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{h}) \to (\mathsf{rt}, \mathsf{tree})$. Let $(\mathsf{rt}, \mathsf{tree}') = \mathcal{H}'_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{h})$. For the local update algorithms, let $\mathsf{tree} = (\mathsf{tree}_1, \ldots, \mathsf{tree}_{m/\ell}, \mathsf{tree}')$ be the RAM memory consisting of all of the above hash tree RAM memories.

- $\mathcal{P}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k), i) \to \rho$. Let $\mathsf{h} = (\mathsf{h}_1, \ldots, \mathsf{h}_{m/\ell})$ be the values computed by $\mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k))$ and $\mathsf{rt} = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{h})$. Compute values $i_1 \in [k], i_2 \in [m/\ell], i_3 \in [\ell]$ such that $i = i_1 \cdot m + i_2 \cdot \ell + i_3$. Compute $\rho_{\mathsf{seh}} = \mathcal{P}_{\mathsf{seh}}(\mathsf{hk}_{\mathsf{seh}}, (x_1^{i_2}, \ldots, x_k^{i_2}), i_1 \cdot m/\ell + i_3)$ and $\rho_{\mathsf{ht}} = \mathcal{P}'_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{ht}}, (\mathsf{h}_1, \ldots, \mathsf{h}_{m/\ell}), I_2)$ for the set $I_2 = [i_2 \cdot (\ell-1) + 1, i_2 \cdot \ell]$. Output $\rho = (\rho_{\mathsf{seh}}, \mathsf{h}_{i_2}, \rho_{\mathsf{ht}})$.

- $\mathsf{Rd}_{\mathsf{ht}}^{\mathsf{tree}}(i) \to \rho$. Let $\mathsf{h}, \mathsf{rt}, i_1, i_2, i_3, I_2$ be the values as in $\mathcal{P}_{\mathsf{ht}}$ and parse $\rho = (\rho_{\mathsf{seh}}, \mathsf{h}_{i_2}, \rho_{\mathsf{ht}})$. Compute $\rho_{\mathsf{seh}} = \mathsf{Rd}_{\mathsf{seh}}^{\mathsf{tree}_{i_2}}(i_1 \cdot m/\ell + i_3)$ and $\rho_{\mathsf{ht}} = \mathsf{Rd}'^{\,\mathsf{tree}'}_{\mathsf{ht}}(I_2)$. Output $(\rho_{\mathsf{seh}}, h_{i_2}, \rho_{\mathsf{ht}})$ as in $\mathcal{P}_{\mathsf{ht}}$.

- $\mathcal{V}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{rt}, i, v, \rho) \to b$. Compute i_1, i_2, i_3, I_2 as in $\mathcal{P}_{\mathsf{ht}}$. Parse $\rho = (\rho_{\mathsf{seh}}, \mathsf{h}_{i_2}, \rho_{\mathsf{ht}})$. Output 1 if and only if $1 = \mathcal{V}_{\mathsf{seh}}(\mathsf{hk}_{\mathsf{seh}}, \mathsf{h}_{i_2}, i_1 \cdot m/\ell + i_3, v, \rho_{\mathsf{seh}})$ and $1 = \mathcal{V}'_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{rt}, I_2, \mathsf{h}_{i_2}, \rho_{\mathsf{ht}})$.

- $\mathsf{Wrt}_{\mathsf{ht}}^{\mathsf{tree}}(i, v) \to (\mathsf{rt}', \tau)$. Let $\mathsf{h}, \mathsf{rt}, i_1, i_2, i_3, I_2$ be the values as in $\mathcal{P}_{\mathsf{ht}}$. Compute $(\mathsf{h}'_{i_2}, \tau_{\mathsf{seh}}) = \mathsf{Wrt}_{\mathsf{seh}}^{\mathsf{tree}_{i_2}}(i_1 \cdot m/\ell + i_3, v)$ and $(\mathsf{rt}', \tau_{\mathsf{ht}}) = \mathsf{Wrt}'^{\,\mathsf{tree}'}_{\mathsf{ht}}(I_2, \mathsf{h}'_{i_2})$. Output (rt', τ) where $\tau = (\tau_{\mathsf{seh}}, \mathsf{h}_{i_2}, \mathsf{h}'_{i_2}, \tau_{\mathsf{ht}})$.

- $\mathsf{VerWrt}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{rt}, i, v, \mathsf{rt}', \tau) \to b$. Let i_1, i_2, i_3, I_2 be the values as in $\mathsf{Wrt}_{\mathsf{ht}}$. Parse $\tau = (\tau_{\mathsf{seh}}, \mathsf{h}_{i_2}, \mathsf{h}'_{i_2}, \tau_{\mathsf{ht}})$. Output 1 if and only if $1 = \mathsf{VerWrt}_{\mathsf{seh}}(\mathsf{hk}_{\mathsf{seh}}, \mathsf{h}_{i_2}, i_1 \cdot m/\ell + i_3, v, \mathsf{h}'_{i_2}, \tau_{\mathsf{seh}})$ and $1 = \mathsf{VerWrt}'_{\mathsf{ht}}(\mathsf{hk}_{\mathsf{ht}}, \mathsf{rt}, I_2, \mathsf{h}'_{i_2}, \mathsf{rt}', \tau_{\mathsf{ht}})$.

- $\mathcal{E}_{\mathsf{ht}}(\mathsf{td}, \mathsf{h}) \to x$. For $j \in [m/\ell]$, set $x^j = \mathcal{E}_{\mathsf{seh}}(\mathsf{td}_{\mathsf{seh}}, \mathsf{h}_j)$ and output $x = (x^1, \ldots, x^{m/\ell})$.

Fix arbitrary values $\lambda, m, k \in \mathbb{N}$, $i^* \in [k]$, and $(x_1, \ldots, x_k)$. Let $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathcal{G}_{\mathsf{ht}}(1^\lambda, m, k, i^*)$, $\mathsf{h} = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k))$, and $(\mathsf{rt}, \mathsf{tree}) = \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{h})$. We sketch arguments for all of the necessary properties as follows:

- **Opening Completeness and Unambiguity:** Let $\rho = (\rho_{\mathsf{seh}}, \mathsf{h}_{i_2}, \rho_{\mathsf{ht}}) = \mathcal{P}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k), i)$ be an opening proof. Both $\mathcal{P}_{\mathsf{ht}}$ and $\mathcal{V}_{\mathsf{ht}}$ compute the same indices i_1, i_2, i_3 such that $i = i_1 \cdot m + i_2 \cdot \ell + i_3$. It follows that ρ_{seh}, ρ_{ht} are valid and unambiguous by the opening completeness and unambiguity of SEH and HT.

- **Completeness of Reads and Writes and Update Unambiguity:** First, observe that $\mathsf{Rd}_{\mathsf{ht}}^{\mathsf{tree}}$ outputs the same values as $\mathcal{P}_{\mathsf{ht}}$ by construction given the completeness of reads for SEH and HT. Next, let $(\mathsf{rt}', \tau) = \mathsf{Wrt}_{\mathsf{ht}}^{\mathsf{tree}}(i, v)$ where $\tau = (\tau_{\mathsf{seh}}, \mathsf{h}_{i_2}, \mathsf{h}'_{i_2}, \tau_{\mathsf{ht}})$. The new memory for tree' only modifies the updates tree values tree_{i_2} and the tree for HT, so the memory values are correct by the guarantees on SEH and HT. Furthermore, the update proofs accept and are unambiguous by the correctness of writes and update unambiguity of SEH and HT.

- **Extraction Correctness:** This follows by the extraction correctness for each individual instance of SEH. Namely, each x^j values for $j \in [m/\ell]$ corresponds to $x^j_{i^*}$, so $(x^1_{i^*}, \ldots, x^{m/\ell}_{i^*}) = x_{i^*}$ as required.

- **Index Hiding:** This follows from the index hiding of SEH by a hybrid argument over the m/ℓ different hash keys generated since m/ℓ is polynomially bounded in λ in the index hiding game.

- **Somewhere Statistically Binding w.r.t. Opening:** This follows from the somewhere statistically binding property of SEH. Namely, any valid opening proof ρ at location i^* consists of three parts $(\rho_{\mathsf{seh}}, \mathsf{h}_{i_2}, \rho_{\mathsf{ht}})$ for values $i_1 \in [k]$, $i_2 \in [m/\ell]$, and $i_3 \in [\ell]$ such that $i^* = i_1 \cdot m + i_2 \cdot \ell + i_3$. ρ_{seh} is guaranteed to be somewhere statistically binding relative to h_{i_2} at the index $i_1 \cdot m/\ell + i_3$, so the only value that can be opened is the extracted value given from $\mathcal{E}_{\mathsf{ht}}(\mathsf{h}_{i_2}, \mathsf{td}_{i_2})$ at the corresponding location.

- **Efficiency and Rate-1:** Efficiency follows in all cases from the efficiency of SEH and HT since the relevant algorithms primarily call the associated algorithms from SEH and HT and otherwise perform simple operations. In terms of rate, the size of h is bounded by

$$\begin{aligned} |\mathsf{h}| &= \sum_{j=1}^{m/\ell} |\mathsf{h}_j| = \frac{m}{\ell} \cdot \left(\ell + \frac{\ell}{2\lambda} + q_{\mathsf{seh}}(\lambda)\right) \\ &= m + \frac{m}{2\lambda} + \frac{m \cdot q_{\mathsf{seh}}(\lambda)}{2\lambda \cdot q_{\mathsf{seh}}(\lambda)} = m + \frac{m}{\lambda}. \end{aligned}$$

  It follows that flSEH is rate-1, as required.

This completes the proof sketch of the theorem. □

We will make use of a hash tree that is extractable on multiple $n > 1$ indices. We note that we can trivially build this from any flSEH that is extractable at one index by repeating the flSEH in parallel for each index, resulting in the following theorem.

Theorem A.2. Assume a rate-1 string OT protocol. For any $n \in \mathbb{N}$, there is a locally updatable unambiguous flSEH that is extractable on n locations simultaneously. All efficiency parameters grow with at most a multiplicative factor of n. ◇

Next, we show that the same construction above allows for incremental and unambiguous computation of the hash root, which we use in our construction of Section 6.3.

Theorem A.3. Let $(\mathcal{G}_{\mathsf{ht}}, \mathcal{H}_{\mathsf{ht}}, \mathcal{D}_{\mathsf{ht}})$ be the associated algorithms in the flSEH construction of Theorem A.1. Define $f(\mathsf{hk}, (x_1, \ldots, x_k)) := \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k)))$. There exists a pair of algorithms $(\mathsf{Upd}_f, \mathsf{Prove}_f, \mathsf{Ver}_f)$ that unambiguously and incrementally compute $(\mathcal{G}_{\mathsf{ht}}, f)$. ◇

Proof (Sketch). Fix $\lambda, m, k \in \mathbb{N}$, $i^* \in [k]$, and a hash key $(\mathsf{hk}, \cdot) \in \mathsf{supp}(\mathcal{G}_{\mathsf{ht}}(1^\lambda, m, k, i^*))$. Recall that $\mathcal{H}_{\mathsf{ht}}(\mathsf{hk}, (x_1, \ldots, x_k))$ computes m/ℓ hash values $\mathsf{h} = (\mathsf{h}_1, \ldots, \mathsf{h}_{m/\ell})$ and then $\mathsf{rt} = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{h})$. First, we claim it suffices to argue that the output of $\mathcal{H}_{\mathsf{seh}}$ can be unambiguously and incrementally computed. This is because Upd_f can aggregate all incrementally updated states for $\mathsf{h}_j = \mathcal{H}_{\mathsf{seh}}(\mathsf{hk}_{\mathsf{seh}}, (x_1^j, \ldots, x_k^j))$ at each step, and then reconstruct h after k steps. In the last step, we can compute $\mathsf{rt} = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{h})$ in time $q(\lambda + m)$ to form the final output. The Prove_f and Ver_f algorithm simply proves and verifies all underlying states in parallel, and Ver_f additionally checks that $\mathsf{st}_k = \mathsf{rt} = \mathcal{D}_{\mathsf{ht}}(\mathsf{hk}, \mathsf{h})$ in the last step.

Fix a hash key $\mathsf{hk}_{\mathsf{seh}}$ and inputs $x_1, \ldots, x_k$ for the underlying SEH hash tree. We recall that $\mathsf{rt} = \mathcal{H}_{\mathsf{seh}}(\mathsf{hk}_{\mathsf{seh}}, (x_1, \ldots, x_k))$ is computed via a binary hash tree for a particular collision-resistant hash function. In each state, we will store a subset of the nodes in the hash tree, and the corresponding proof will be their authentication paths relative to the root rt. Specifically, the ith state st_i will correspond to the "lowest common ancestors" covering only the leaf nodes $(x_1, \ldots, x_i)$ in the tree. So, st_1 consists of the leaf node x_1 and π_i its authentication path. Then, st_2 consists of the parent node of x_1 and x_2, st_3 the parent of x_1 and x_2 as well as the leaf node for x_3, and so on. We give a high level description of the update and verification algorithms for such a hash tree as follows:

- $\mathsf{st}_i = \mathsf{Upd}_f(\mathsf{hk}_{\mathsf{seh}}, \mathsf{st}_{i-1}, x_i)$. st_{i-1} consists of the lowest common ancestors covering only leaf nodes $x_1, \ldots, x_{i-1}$. Given the value for x_i, we add the leaf node x_i to the current set of nodes. If its sibling is included in the set, we remove x_i and its sibling and replace it with their parent node. We continue to merge sibling nodes until we are only left with the lowest common ancestors.
- $\pi_i = \mathsf{Prove}_f(\mathsf{hk}_{\mathsf{seh}}, (x_1, \ldots, x_k), \mathsf{st}_i, i)$. The proof π_i simply consists of the authentication paths for all nodes contained in st_i. Note that this can be computed for each such node given all the leaf nodes in the tree.
- $b = \mathsf{Ver}_f(\mathsf{hk}_{\mathsf{seh}}, \mathsf{rt}, \mathsf{st}_i, i, \pi_i)$. Output 1 if all authentication paths in π_i are valid for the nodes in st_i relative to the root rt.

For correctness, observe that the final state st_k consists of the lowest common ancestor for leaf nodes $(x_1, \ldots, x_k)$ which is simply the root rt, so $\mathsf{st}_k = \mathsf{rt}$, as required. Furthermore, completeness follows by completeness of the hash tree. In terms of efficiency, note that each state st consists of one node per level in the tree, and the associated proof π consists of authentication paths per each node. The depth of the tree is at most $\log k \leq \lambda$, and each node has size $q(\lambda + m)$ by the efficiency of seh. It follows that all the algorithms run in the required efficiency and the proofs have the required size.

Lastly, unambiguity follows since the leaf values are uniquely determined by the inputs $x_1, \ldots, x_k$. Additionally, the root rt uniquely determines the set of nodes that can be opened up in the tree, up to a negligible probability of finding a collision for $\mathsf{hk}_{\mathsf{seh}}$. So, since all nodes in the tree are (computationally) uniquely determined, and the states and proofs simply consist of nodes in the tree, unambiguity follows. □

B Proof of Lemma 3.8

We use the Hadamard-based PCP of [ALMSS98], which is discussed and expressed as a LPCP in Appendix A.1 of [BCIOP13].

As [BCIOP13] shows, we can always transform the relation verification circuit into a poly-size arithmetic circuit over $\mathbb{Z}_2$. In our case, the arithmetic circuit has size $s = \mathsf{poly}(T)$. (We assume $|x| \le T$.) Let z_i denote the value of the i-th wire of the arithmetic circuit given input x and witness w, [BCIOP13] proves that there exists a matrix A and a vector $\mathbf{b}$ such that

- Both A and $\mathbf{b}$ can be efficiently computed from the arithmetic circuit without using x.
- For any $\mathbf{z} = (z_1, \ldots, z_s) \in \{0,1\}^s$, it is a correct circuit assignment that accepts x (with some witness w), if and only if

$$A \cdot \begin{bmatrix} \mathbf{z} & \mathbf{z} \otimes \mathbf{z} \end{bmatrix} \equiv \begin{bmatrix} x \\ \mathbf{b} \end{bmatrix} \pmod 2.$$

  In the following we denote $\mathbf{b}_x = (x, \mathbf{b})$.

In the LPCP system, the honest LPCP proof is $\pi = (\mathbf{z}, \mathbf{z} \otimes \mathbf{z})$. To verify a proof $\pi \in \mathbb{Z}^{s+s^2}$, the verifier checks: 1) there exists some $\mathbf{z} \in \mathbb{Z}^s$ such that $\pi = (\mathbf{z}, \mathbf{z} \otimes \mathbf{z})$; 2) $\mathbf{z}$ in the last step is in $\{0,1\}^s$; and 3) $A \cdot \pi \equiv \mathbf{b}_x \pmod 2$.

LPCP prover algorithm. Given $(x, w) \in \mathcal{R}$, the prover computes the values $\mathbf{z} = (z_1, \ldots, z_s)$ of all wires in the arithmetic verification circuit, and output the proof as

$$\pi = (\mathbf{z}, \mathbf{z} \otimes \mathbf{z}) \in \{0,1\}^{s+s^2}.$$

LPCP verifier algorithm. For readability we split the verifier algorithm into 3 steps, where in each step some queries are made and one property of the proof π is verified. Note that in practice these "steps" are executed in parallel. Every step is repeated for constant times to reduce soundness error.

1. This step aims to ensure the existence of $\mathbf{z} \in \mathbb{Z}^s$ such that $\pi = (\mathbf{z}, \mathbf{z} \otimes \mathbf{z})$.

   The query algorithm samples $r_1, r_2 \leftarrow \{0,1\}^s$. It prepares queries $q_1 = (r_1, 0^{s^2}), q_2 = (r_2, 0^{s^2})$ and $q_3 = (0^s, r_1 \otimes r_2)$. Let the answers be a_1, a_2, a_3 respectively. The decision algorithm checks if $a_3 = a_1 \cdot a_2$.

2. This step aims to ensure that $\mathbf{z}$ in the last step is binary.

   The query algorithm samples $r \leftarrow \{0,1\}^s$. It then computes $r' \in \{0,1\}^{s^2}$ as $r'[i \otimes i] = r[i]$ and $r'[i \otimes j] = 0$ for $i \neq j$. The queries are $q_1 = (r_1, 0^{s^2}), q_2 = (0^s, r')$. Let the answers be a_1, a_2 respectively. The decision algorithm checks if $a_1 = a_2$.

3. This step aims to ensure that $A \cdot \pi \equiv \mathbf{b}_x \pmod 2$.

   The query algorithm has the matrix A and vector $\mathbf{b}$. Let n_A be the number of rows in A. The query algorithm samples $r \leftarrow \{0,1\}^{n_A}$, and sets $q = r^T A \mod 2$. Let r_1 be the first $|x|$ coordinates of r and r_2 be the last $n_A - |x|$ coordinates of r. The state information is $u = (r_1, r_2^T \mathbf{b} \mod 2) \in \{0,1\}^{|x|+1}$.

   Let the answer be a. The decision algorithm checks if $a \equiv (x, 1)^T u \pmod 2$.

It follows by the construction that this LPCP system is complete, and the verification is non-adaptive and efficient. Also, all queries are binary vectors. We delay the verification of all rest properties after proving knowledge soundness.

To prove knowledge soundness, we show that every step of the verifier algorithm achieves its goal.

Analysis: Step 1 Write $\pi = (\mathbf{v}, \mathbf{w})$ where $\mathbf{v} \in \mathbb{Z}^s$, $\mathbf{w} \in \mathbb{Z}^{s^2}$. In this step the verifier checks whether $\mathbf{v} \otimes \mathbf{v} = \mathbf{w}$, by taking $r_1, r_2 \leftarrow \{0,1\}^s$ and checking if $r_1^T(\mathbf{v}\mathbf{v}^T)r_2 = \langle \mathbf{w}, r_1 \otimes r_2 \rangle$. Let $V = \mathbf{v}\mathbf{v}^T$ and $W \in \mathbb{Z}^{s \times s}$ be reorganized from $\mathbf{w}$, it is equivalent to say the verifier checks $r_1^T V r_2 = r_1^T W r_2$.

The following lemma shows that the verifier detects with probability at least $1/4$.

Lemma B.1. For any matrices $V, W \in \mathbb{Z}^{s \times s}$, if $V \neq W$, then

$$\Pr_{r_1, r_2 \leftarrow \{0,1\}^s}[r_1^T V r_2 \neq r_1^T W r_2] \geq \frac{1}{4}. \qquad \diamond$$

Proof. It is easy to show with at least $1/2$ probability over r_1, $r_1^T V \neq r_1^T W$. Fix any such r_1, then with at least $1/2$ probability over r_2, we have $r_1^T V r_2 \neq r_1^T W r_2$. □

After repeating for constant times, any proof π not of the form $(\mathbf{v}, \mathbf{v} \otimes \mathbf{v})$ can pass this step only with probability, say, $1/6$.

Analysis: Step 2 Now write $\pi = (\mathbf{v}, \mathbf{v} \otimes \mathbf{v})$ for some $v \in \mathbb{Z}^s$. In this step the verifier samples $r \leftarrow \{0,1\}^s$ and checks if $\sum_{i \in [s]} r_i \mathbf{v}_i = \sum_{i \in [s]} r_i \mathbf{v}_i^2$, i.e. $\sum_{i \in [s]} r_i(\mathbf{v}_i - \mathbf{v}_i^2) = 0$. It is easy to see that unless $\mathbf{v}_i - \mathbf{v}_i^2 = 0$ for all $i \in [s]$ (which means $\mathbf{v} \in \{0,1\}^s$), the verifier detects with probability at least $1/2$. After repetition we may reduce the error to, say $1/6$.

Analysis: Step 3 Now $\pi = (\mathbf{z}, \mathbf{z} \otimes \mathbf{z})$ for some $\mathbf{z} \in \{0,1\}^s$. In this step the verifier checks if

$$r^T A \cdot \begin{bmatrix} \mathbf{z} & \mathbf{z} \otimes \mathbf{z} \end{bmatrix} \equiv r^T \begin{bmatrix} x \\ \mathbf{b} \end{bmatrix} \pmod 2,$$

for a random $r \leftarrow \{0,1\}^{n_A}$. Again the verifier detects with probability $1/2$ if $A \cdot \pi \not\equiv \mathbf{b}_x \pmod 2$. After repetition the verifier fails to detect by probability $1/6$.

After analyzing these three steps, we conclude that for any proof π that gets accepted with probability $1/6$, it must be of the form $(\mathbf{z}, \mathbf{z} \otimes \mathbf{z}) \in \{0,1\}^{s+s^2}$ for some correct circuit assignment $\mathbf{z} \in \{0,1\}^s$ that accepts x with some witness w. This implies the rest properties:

- Proof of knowledge follows as we can read the witness w from $\mathbf{z}$.
- For any witness w, it is easy to see the only accepting proof that extracts to w is the one determined by the circuit assignment with input (x, w).
- As shown above, accepting proofs π must be binary.

C Additional Preliminaries

C.1 Fully Homomorphic Encryption

Fully homomorphic (public key) encryption (FHE) allows homomorphic computation over ciphertexts. We foramlize this as follows.

Definition C.1 (Fully Homomorphic Encryption). A (public key) fully homomorphic encryption scheme (FHE) consists of algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ with the following syntax and efficiency:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$. This randomized polynomial-time algorithm takes a security parameter 1^λ as input, and outputs a public key pk and a secret key sk.
- $\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$. This randomized polynomial-time algorithm takes as input a public key pk and a message m, and outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to m$. This is a deterministic polynomial-time algorithm takes as input a secret key sk and a ciphertext ct, and outputs a message m.
- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}) \to \mathsf{ct}'$. This is a deterministic polynomial-time algorithm takes as input a public key pk, a circuit C, and a ciphertext ct, and outputs an evaluated ciphertext ct'.

We require that $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ satisfy the following properties:

- **Correctness.** For any polynomial ℓ, $\lambda \in \mathbb{N}$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, circuit C of size at most $\ell(\lambda)$, and message m, we have
$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, C, \mathsf{Enc}(\mathsf{pk}, m))) = C(m).$$
- **Semantic Security.** For any poly-size adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, there exists a negligible function $\mu(\cdot)$ such that for any $\lambda \in \mathbb{N}$,
$$\Pr\left[\mathcal{A}_1(\mathsf{pk}, \mathsf{ct}) = b \;:\; \begin{array}{l}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ b \leftarrow \{0,1\} \\ (m_0, m_1) \leftarrow \mathcal{A}_0(\mathsf{pk}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)\end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$
- **Compactness.** There exists a polynomial $p(\cdot)$, such that for any $\lambda \in \mathbb{N}$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, message $m \in \{0,1\}^*$, and circuit C with input size $|m|$,
$$|\mathsf{Eval}(\mathsf{pk}, C, \mathsf{Enc}(\mathsf{pk}, m))| = |C(m)| \cdot p(\lambda).$$
- **Complexity Preservation.** There exists a polynomial $q(\cdot)$ such that any circuit of size t and width w, can be homomorphically evaluated in time $t \cdot q(\lambda)$ and space $w \cdot q(\lambda)$. ◇

C.2 Total Search Problems

An efficiently-verifiable search problem is described via a pair $(\mathcal{L}, \mathcal{R})$, where $\mathcal{L} \subseteq \{0,1\}^*$ is an efficiently-recognizable set of instances, and $\mathcal{R}$ is an efficiently-computable binary relation. Given $v \in \mathcal{L}$, the task is to find a w such that $\mathcal{R}(v, w)$ — the class that contains all such search problems is known as functional NP (FNP). An efficiently-verifiable search problem is total if for every instance $v \in \mathcal{L}$ there exists a witness w of length $\mathsf{poly}(|v|)$ such that $\mathcal{R}(v, w) = 1$. The class total FNP (TFNP) consists of all efficiently-verifiable search problems that are total.

In this paper, we focus on the TFNP sub-classes UEOPL. UEOPL is a sub-class of CLS $\subseteq$ PPAD [DP11; FGHS21]. Please see [CHK+19a] for a more detailed introduction on these TFNP sub-classes.

Definition C.2 ([FGMS19; BCH+22]). The search problem UNIQUEFORWARDEOML is defined as:

- Instance.
  1. A circuit $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$.
  2. A circuit $\mathsf{M} : \{0,1\}^m \to \{0, \ldots, 2^m - 1\}$.
  3. A string $v_0 \in \{0,1\}^m$.
- Guarantee.
  1. $\mathsf{S}(v_0) \neq v_0$.
  2. $\mathsf{M}(v_0) = 0$.
- Solution. One of the following:
  1. **Sink:** A vertex $v \in \{0,1\}^m$ such that $\mathsf{S}(v) \neq v$ and $\mathsf{S}(\mathsf{S}(v)) = \mathsf{S}(v)$.
  2. **Violation of meter:** A vertex $v \in \{0,1\}^m$ such that $\mathsf{S}(v) \neq v$ and $\mathsf{M}(\mathsf{S}(v)) \neq \mathsf{M}(v) + 1$.
  3. **Collision of meter:** Two vertices $u \neq v \in \{0,1\}^m$ such that $u \neq \mathsf{S}(u), v \neq \mathsf{S}(v)$ and $\mathsf{M}(u) = \mathsf{M}(v)$.

◇

Definition C.3 ([FGMS19]). UEOPL is defined as the set of all total problems that are Karp-reducible to UNIQUEFORWARDEOML. ◇