Public-Coin 3-Round Zero-Knowledge from Learning with Errors and Keyless Multi-Collision-Resistant Hash

Susumu Kiyoshima

NTT Research
susumu.kiyoshima@ntt-research.com

June 23, 2022

Abstract

We construct a public-coin 3-round zero-knowledge argument for NP assuming (i) the sub-exponential hardness of the learning with errors (LWE) problem and (ii) the existence of keyless multi-collision-resistant hash functions against slightly super-polynomial-time adversaries. These assumptions are almost identical to those that were used recently to obtain a private-coin 3-round zero-knowledge argument [Bitansky et al., STOC 2018]. (The difference is that we assume sub-exponential hardness instead of quasi-polynomial hardness for the LWE problem.)

This article is a full version of the following article: Public-Coin 3-Round Zero-Knowledge from Learning with Errors and Keyless Multi-Collision-Resistant Hash, CRYPTO 2022, ©IACR 2022.

Contents

1 Introduction

This paper concerns computational zero-knowledge (ZK) arguments, i.e., ZK proofs where soundness and zero-knowledge are both defined against polynomial-time adversaries.

A central research topic about ZK arguments is 3-round ZK arguments, which are optimal in terms of round complexity due to the impossibility of 2-round ZK arguments [GO94]. Obtaining 3-round ZK arguments is notoriously hard, and obtaining 3-round ZK arguments with black-box simulations is known to be impossible [GK96]. Until recently, 3-round ZK arguments had been obtained only under unfalsifiable knowledge-type assumptions (e.g., [HT98, BP04, CD08, BCC+17, BEP20]) or under weak security definitions (e.g., [Pas03, BCPR14, BBK+16, JKKR17, KS17, BGJ+18, BL18, BKP19, Den20][1]).

Recently, Bitansky, Kalai, and Paneth [BKP18] obtained a 3-round ZK argument by relying on super-polynomial hardness of the learning with errors (LWE) assumption [Reg09] and *keyless multi-collision-resistant hash functions*.[2] Multi-collision resistance [BDRV18, BKP18, KNY18] is a natural relaxation of the standard collision resistance. In the standard keyed setting, *K-collision resistance* ($K \in \mathbb{N}$) of a hash function family $\mathcal{H}$ is defined by requiring that for a random hash function $h \in \mathcal{H}$, any polynomial-time adversary cannot find a *K-collision*, i.e., any $(x_1, \ldots, x_K)$ such that $h(x_1) = \cdots = h(x_K)$. In the keyless setting, multi-collision resistance is defined by allowing K to grow with the adversary's size. That is, K-collision resistance of a keyless hash function h is defined by requiring that any polynomial-time adversary with any polynomial-size non-uniform advice z cannot find a $K(|z|)$-collision. It is unknown whether keyless multi-collision-resistant hash functions can be obtained from more standard cryptographic primitives (including keyed collision-resistant hash function families), but they have a simple falsifiable definition. Recently, they were used to obtain new results about, e.g., ZK proofs/arguments [BKP18, BL18, BP19], succinct arguments [BKP18], and non-malleable commitments [BL18]. (See [BKP18] for more about keyless multi-collision-resistant hash functions.)

Given the result of Bitansky et al. [BKP18], a natural question is whether we can obtain *public-coin* 3-round ZK arguments by relying on the LWE assumption and keyless multi-collision-resistant hash functions. Recall that an interactive argument is called public coin if (i) the verifier only sends the outcome of a coin toss in each round and (ii) the final output of the verifier is deterministically computed from the transcript. (Well-known examples of public-coin ZK proofs/arguments include the classical ZK proofs of Goldreich, Micali, and Wigderson [GMW91] and Blum [Blu86].) In addition to being simple, public-coin ZK arguments have useful properties such as (i) they are *publicly verifiable*, i.e., verifying a proof does not require any secret information, and (ii) they can be used to achieve additional security such as *leakage-soundness* [GJS11] and *resettable soundness* [BGGL01]. The 3-round ZK argument of Bitansky et al. [BKP18] (as well as the subsequent 3-round statistical ZK argument of Bitansky and Paneth [BP19]) is not public coin.

Our result. In this paper, we give a positive result about public-coin 3-round ZK arguments.

Main Theorem (informal). *Assume the existence of polynomially compressing keyless hash functions*[3] *that are multi-collision resistant against slightly super-polynomial-time (e.g., quasi-polynomial-time) adversaries, and additionally assume the sub-exponential hardness of the LWE assumption. Then, there exists a public-coin 3-round zero-knowledge argument for NP.*

(The formal description of this theorem is given in Section 7 as Theorem 3.) The assumptions that we use are similar to those that are used by Bitansky et al. [BKP18] for their (private-coin) 3-round ZK argument. The difference is that we assume the sub-exponential hardness of the LWE assumption whereas Bitansky et al. [BKP18] assume the quasi-polynomial hardness of the LWE assumption.

2 Overview of Our Techniques

Our starting point is the (private-coin) 3-round ZK argument of Bitansky, Kalai, and Paneth [BKP18]. Thus, we first recall their techniques before explaining our techniques.

[1] Some of these works constructed even 2-round or non-interactive ZK arguments under weak security definitions.

[2] More precisely, they obtained it by relying on various cryptographic primitives that can be based on these assumptions.

[3] e.g., those that hash length-λ^2 strings to length-λ strings.

2.1 Techniques of Bitansky et al. [BKP18]

The main component of the 3-round ZK argument of Bitansky et al. [BKP18] is a memory delegation scheme [CKLR11, BBK+16]. In the setting considered in [BBK+16, BKP18], memory delegation schemes proceed in 3 rounds.

1. The prover sends the verifier a short digest of a long memory string. (Importantly, the digest is created non-interactively.)

2. The verifier chooses a computation to be executed on the memory, and sends the prover the description of the computation (e.g., a Turing machine) and a challenge string.

3. The prover responds with the computation output and a proof of correctness.

It is required that the verifier runs in polynomial time even when the length of the memory (and the running time of the computation to be evaluated on it) is slightly super-polynomial, such as $\lambda^{\log \lambda}$ for the security parameter λ. For security, the soundness notion given by Bitansky et al. [BKP18] requires that no prover can generate an accepting proof for a randomly sampled output (which is sampled after a digest and a computation to be evaluated on the memory are fixed).

Bitansky et al. [BKP18] obtained a memory delegation scheme by using a keyless multi-collision-resistant hash function and the 2-round delegation scheme of Kalai, Raz, and Rothblum [KRR14] (the KRR delegating scheme in short). Specifically, their scheme was obtained based on the following two observations.

1. The first observation is that the KRR delegating scheme can be converted to a memory delegation scheme if there exists a keyless multi-collision-resistant hash function with a local opening property (i.e., a property that any location of a hashed string can be opened without revealing the entire string). In the KRR delegating scheme, for an input x and a Turing machine M, the verifier sends a challenge string to the prover, and the prover responds with the output $y := M(x)$ and a proof of the correctness of y. A nice property of the KRR delegating scheme is that soundness holds even when the verifier only has oracle access to (an encoding of) the input x, where the verifier only makes a small number of non-adaptive queries. Given this property, the KRR delegation scheme can be converted to a memory delegation scheme as follows. The digest is created by hashing a memory with the keyless hash function. The verifier sends a challenge string and input queries of the KRR delegation scheme.[4] The prover responds with the computation output y, a proof of the KRR delegation scheme for the correctness of y, and local opening of the queried locations of the memory. Intuitively, if local opening of the keyless hash function satisfies a sufficiently strong multi-collision-resistant property, the prover can only create accepting proofs for a small number of values of y. Thus, the prover cannot generate an accepting proof for a randomly chosen value of y.

2. The second observation is that any multi-collision-resistant hash functions can be converted to those that have a local opening property. It should be noted that the standard tree-hashing technique cannot be used for this purpose. Indeed, in the case of multi-collision resistance, an adversary might be able to open each location to two values, and thus, it might be able to open λ locations to 2^λ combinations of such values. The conversion given in Bitansky et al. [BKP18] yields a multi-collision resistant hash function for long inputs while avoiding this exponential deterioration of multi-collision resistance. (Details about this conversion, including the formal definition of multi-collision resistance of local opening, are not important to this overview.) A notable limitation is that when multiple locations are opened, they must be opened simultaneously so that the above "mixed-and-match" attack can be prevented. That is, opening each location individually as in the case of the standard tree-hashing is not allowed. Fortunately, this limitation does not cause a problem for the current purpose since the KRR delegation scheme only makes non-adaptive input queries.

The memory delegation scheme of Bitansky et al. [BKP18] is private coin, and this is the reason why their 3-round ZK argument is private coin. Specifically, their 3-round ZK argument is obtained from a memory

[4] In Bitansky et al. [BKP18], the input queries need to be encrypted by an FHE scheme so that the prover cannot learn the input queries. We ignite this detail in this overview.

delegation scheme by following the idea of an earlier work [BBK+16] (roughly speaking, the idea is to reduce the round complexity of the public-coin ZK argument of Barak [Bar01] by using a memory delegation scheme), and if the underlying memory delegation scheme is public coin, this step can be simplified straightforwardly and yields a public-coin 3-round ZK argument.

2.2 Our Techniques

We obtain a public-coin memory delegation scheme (and as a result a public-coin 3-round ZK argument) by using recent results about *succinct non-interactive arguments (SNARGs)* for deterministic computations [JKKZ21, HLR21b, CJJ22].

Failed attempt #1. First, let us consider using the result of Choudhuri, Jain, and Jin [CJJ22] that gives a SNARG for RAM computations. When their scheme is viewed as a public-coin 2-round RAM delegation scheme,[5] a memory DB is tree-hashed to a digest by using a keyed collision-resistant hash function (the key is sampled by the verifier), the verifier chooses a RAM machine R and a challenge string, and the prover responds with the output $y = R^{\mathsf{DB}}$ and a proof of correctness of y. Choudhuri et al. [CJJ22] showed that their scheme works for all polynomial-time RAM computations under the polynomial hardness of the LWE assumption, and their analysis can be trivially extended for $\lambda^{\omega(1)}$-time RAM computations under the $\lambda^{\omega(1)}$-hardness of the LWE assumption. (The verifier still runs in polynomial time.) Since the prover needs to compute a digest non-interactively in memory delegation schemes, a natural approach is to modify the RAM delegation scheme of Choudhuri et al. [CJJ22] so that it works even when the memory is hashed by a keyless multi-collision-resistant hash function. Unfortunately, this approach does not work (at least when naively implemented) since the scheme of Choudhuri et al. [CJJ22] requires each memory location to be locally opened individually on a locating-by-location basis as explained below. (Recall that the local opening method given by Bitansky et al. [BKP18] for multi-collision-resistant hash functions does not allow such individual opening.) At a high level, the scheme of Choudhuri et al. [CJJ22] proves the correctness of a RAM computation by proving the correctness of multiple evaluations of a single small circuit. Intuitively, this small circuit represents a single step of the RAM computation. That is, it takes as input a local state of the RAM machine, a digest of the memory, and local opening of a single location of the memory, and it outputs an updated local state of the RAM machine along with an updated digest of the memory and a corresponding certificate. The circuit size depends on the memory length only polylogarithmically since local opening of a single location is given as input rather than the entire memory. Since this polylogarithmic dependence is essential for the verifier efficiency of the RAM delegation scheme, it is required that each location of the memory can be locally opened individually.

Failed attempt #2. Next, let us consider using the result of Jawale, Kalai, Khurana, and Zhang [JKKZ21] that obtains a SNARG from the public-coin interactive proof of Goldwasser, Kalai, and Rothblum [GKR15] (the GKR interactive proof in short). The scheme of Jawale et al. [JKKZ21], when viewed as a public-coin 2-round delegation scheme,[6] has the same syntax as the KRR delegation scheme (cf. Section 2.1). In addition, it has the same additional property as the KRR delegation scheme, i.e., soundness holds even when the verifier only makes a small number of non-adaptive queries to (an encoding of) the input. Therefore, just like Bitansky et al. [BKP18] obtained a (private-coin) memory delegation scheme from the KRR delegation scheme, we can obtain a (public-coin) memory delegation scheme from the scheme of Jawale et al. [JKKZ21] by combining it with a keyless multi-collision-resistant hash function. (We need to assume the sub-exponential hardness of the LWE assumption since the scheme of Jawale et al. [JKKZ21] requires it.) The problem is that the scheme of Jawale et al. [JKKZ21] is only shown to work for log-uniform[7] bounded-depth computations. As a result, the memory delegation scheme that we can obtain from it has the same limitation. Unfortunately, for the application to 3-round ZK arguments, memory delegation schemes for such limited computations are insufficient.

Our approach. Given the above two failed attempts, we obtain a public-coin memory delegation scheme by using both the scheme of Choudhuri et al. [CJJ22] and the scheme of Jawale et al. [JKKZ21]. We obtain our

[5] Their SNARG works in the common random string model and therefore can be viewed as a public-coin 2-round delegation scheme.
[6] Their SNARG works in the common random string model and therefore can be viewed as a public-coin 2-round delegation scheme.
[7] A computation is log-uniform if it has a circuit that can be generated by a log-space Turing machine.

memory delegation scheme in two steps.

1. First, we obtain a public-coin *tree-hash memory delegation scheme*, i.e., a public-coin memory delegation scheme for proving the correctness of tree-hash computations. We obtain such a scheme by combining the scheme of Jawale et al. [JKKZ21] and a keyless multi-collision-resistant hash function as suggested above. The key point is that, as already observed by Goldwasser et al. [GKR15], the GKR interactive proof works not only for log-uniform computations but also for any computations that have a certain form of succinct descriptions. Tree-hash computations have the required form of succinct descriptions because of their simple tree structure. Thus, the GKR interactive proof can be used to prove the correctness of tree-hash computations. Then, since the scheme of Jawale et al. [JKKZ21] inherits this property, the memory delegation scheme that we obtain from it also inherits this property, i.e., works for tree-hash computations.

2. Next, we use the above tree-hash memory delegation scheme to obtain a public-coin memory delegation scheme for all $\lambda^{\omega(1)}$-time computations on memories of length $\lambda^{\omega(1)}$. A key observation is that the tree-hash memory delegation scheme can be used to verify whether a digest is correctly computed for the RAM delegation scheme of Choudhuri et al. [CJJ22]. More concretely, we consider the following scheme.

   (a) The digest of a memory DB is obtained as in the tree-hash memory delegation scheme using a keyless multi-collision-resistant hash function.

   (b) The verifier sends the prover (i) a (keyed) collision-resistant hash function h together with a challenge string of the tree-hash delegation scheme and (ii) the description R of the computation to be evaluated on the memory (modeled as a RAM machine) together with a challenge string of the RAM delegation scheme of Choudhuri et al. [CJJ22].

   (c) The prover responds with (i) the tree-hash $\mathsf{rt} := \mathsf{TreeHash}_h(\mathsf{DB})$ of the memory DB w.r.t. h together with the proof of the tree-hash delegation for the correctness of rt and (ii) the output $y := R^{\mathsf{DB}}$ of the computation together with the proof of the RAM delegation scheme for the correctness of y, where rt is used as the digest in the RAM delegation scheme.

   In the above scheme, the digest rt of the RAM delegation scheme is chosen adaptively after the prover learns the challenge string. Still, the tree-hash memory delegation scheme guarantees that rt is correctly computed based on the memory DB, and as a result, we can think as if rt is fixed non-adaptively. Thus, we can use the soundness of the RAM delegation scheme to show the correctness of the computation output y. (In a little more detail, we can show that a cheating prover can give accepting proofs for at most a small number of values of rt and therefore can give accepting proofs for at most a small number of values of y.)

Before concluding the technical overview, we give remarks about the actual construction given in the subsequent sections.

Remark 1 (On tree-hash memory delegation). Firstly, obtaining a tree-hash memory delegation scheme from the scheme of Jawale et al. [JKKZ21] is actually not trivial. To explain the difficulty, we first note that for the soundness of the GKR interactive proof to hold, the verifier should be given oracle access to *an encoding of* the input x, and the length of the encoding is determined by various parameters of the GKR interactive proof. Now, the problem is that if we obtain a tree-hash memory delegation scheme from the scheme of Jawale et al. [JKKZ21] naively, the encoding needs to be super-polynomially long since the scheme of Jawale et al. [JKKZ21] uses the GKR interactive scheme with slightly modified parameters.[8] (This super-polynomially long encoding is not problematic in the settings of SNARGs and 2-round delegation since the encoding is never written down entirely, but it is problematic in the setting of memory delegation since the prover needs to hash the encoding.) Almost the same problem was already observed in a different context by Bronfman and Rothblum [BR22], and we avoid our problem as in their work. Namely, instead of directly using the result of Jawale et al. [JKKZ21], we use the result of Holmgren, Lombardi, and Rothblum [HLR21b] that shows, based on Jawale et al. [JKKZ21], that a SNARG can be obtained from the GKR interactive proof without modifying its parameters.

Secondly, we focus on tree-hash memory delegation schemes for tree-hash computations w.r.t. polylogarithmic-depth collision-resistant hash functions. (By doing so, we can work with the GKR interactive proof in a typical setting, i.e., for polylogarithmic-depth computations.) Such tree-hash memory delegation

[8] If the reader is familiar with the GKR interactive proof, we note that the scheme of Jawale et al. [JKKZ21] uses the GKR interactive proof with a super-polynomially large field, and as a result, the low-degree encoding of the input is super-polynomially long.

schemes are sufficient for our purpose since the sub-exponential hardness of the LWE assumption implies the existence of polylogarithmic-depth collision-resistant hash functions.[9] ◊

Outline of the rest of this paper. After giving necessary notations and definitions in Section 3, we prove our main result from Section 4 to Section 7. Like Bitansky et al. [BKP18], in the actual proof we first consider *oracle memory delegation schemes*, which are simpler than memory delegation schemes in that the verifier publishes (an encoding of) the memory in the clear at the beginning. Specifically, in Section 4, we give a public-coin oracle memory delegation scheme for tree-hash computations, and in Section 5, we upgrade it to a one for all $\lambda^{\omega(1)}$-time computations. Then, in Section 6, we obtain a public-coin memory delegation scheme from our oracle memory delegation scheme and a keyless multi-collision-resistant hash function. In Section 7, we use it to obtain a public-coin 3-round ZK argument.

3 Preliminaries

We denote the security parameter by λ. For editorial simplicity, some definitions are deferred to Appendix A.

3.1 Notations

For any $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, \ldots, n\}$. We use poly to denote an unspecified polynomial, negl to denote an unspecified negligible function, and PPT as an abbreviation of "probabilistic polynomial-time." For any NP language $\mathbf{L}$ and an instance $x \in \mathbf{L}$, we use $\mathbf{R}_{\mathbf{L}}$ to denote the witness relation (i.e., $\mathbf{R}_{\mathbf{L}}$ is the set of all instance-witness pairs of $\mathbf{L}$) and use $\mathbf{R}_{\mathbf{L}}(x)$ to denote the set of all witnesses of x. For any pair of probabilistic interactive Turing machines (P, V), we use $\langle P(w), V(z)\rangle(x)$ for any $x, w, z \in \{0,1\}^*$ to denote the random variable representing the output of V in an interaction between $P(x, w)$ and $V(x, z)$.

For a vector $v = (v_1, \ldots, v_\lambda)$ and a set $S \subseteq [\lambda]$, let $v|_S := \{v_i\}_{i \in S}$. Similarly, for a function $f : D \to R$ and a set $S \subseteq D$, let $f|_S := \{f(i)\}_{i \in S}$. For a set S, we denote by $s \leftarrow S$ the process of obtaining an element $s \in S$ by a uniform sampling from S. For any probabilistic algorithm Algo and an input x, we denote by $y \leftarrow \mathsf{Algo}(x)$ the process of obtaining an output y by running $\mathsf{Algo}(x)$ with uniform randomness. For a sufficiently long string $r \in \{0,1\}^*$, we denote by $y := \mathsf{Algo}(x; r)$ the process of obtaining an output y by running $\mathsf{Algo}(x)$ with randomness r.

3.2 (Keyed) Hash Functions

Recall that a (keyed) hash function family can be modeled by two algorithms $(\mathsf{Gen}, \mathsf{Hash})$ and two functions $\ell_1, \ell_2 : \mathbb{N} \to \mathbb{N}$ as follows.

- $h \leftarrow \mathsf{Gen}(1^\lambda)$: Gen is a PPT algorithm that takes as input a security parameter 1^λ, and it outputs a key h.
- $y := \mathsf{Hash}(h, x)$: Hash is a deterministic polynomial-time algorithm that takes as input a key $h \in \mathsf{Gen}(1^\lambda)$ and a string $x \in \{0,1\}^{\ell_1(\lambda)}$, and it outputs a string $y \in \{0,1\}^{\ell_2(\lambda)}$.

A hash function family is called *public coin* [HR04] if $\mathsf{Gen}(1^\lambda)$ outputs a uniformly random string. Unless otherwise stated, we assume $\ell_1(\lambda) = 2\lambda$ and $\ell_2(\lambda) = \lambda$.

In this paper, we use the following simplified notations. For a hash function family $\mathcal{H} = (\mathsf{Gen}, \mathsf{Hash})$, we use $\mathcal{H}_\lambda$ to denote the range of $\mathsf{Gen}(1^\lambda, \cdot)$, use $h \leftarrow \mathcal{H}_\lambda$ as a shorthand of $h \leftarrow \mathsf{Gen}(1^\lambda)$, and use $h(x)$ as a shorthand of $\mathsf{Hash}(h, x)$.

3.2.1 Collision resistance.

Recall that collision resistance of a hash function family can be defined as follows.

[9] For example, polylogarithmic-depth collision-resistant hash functions can be obtained by using a sub-exponentially hard collision-resistant hash function with a polylogarithmic security parameter.

Definition 1. *A hash function family $\mathcal{H}$ is* collision resistant *if for any* PPT *adversary $\mathcal{A}$ and any sequence of polynomial-size advice $\{z_\lambda\}_{\lambda\in\mathbb{N}}$, there exists a negligible function* negl *such that for every $\lambda \in \mathbb{N}$,*

$$\Pr\left[h(x_1) = h(x_2) \wedge x_1 \neq x_2 \;\middle|\; \begin{array}{l} h \leftarrow \mathcal{H}_\lambda \\ (x_1, x_2) \leftarrow \mathcal{A}(h, z_\lambda) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We note that a public-coin collision-resistant hash function family can be obtained under the LWE assumption.[10]

Remark 2 (Polylogarithmic-depth hash function family). We say that a collision-resistant hash function family is *polylogarithmic depth* if every hash function in the family can be evaluated by a polylogarithmic-depth circuit. Such a hash function family can be obtained from any sub-exponentially secure collision-resistant hash function family as follows. Assume that there exist a hash function family $\mathcal{H}$ and a constant $0 < \epsilon < 1$ such that $\mathcal{H}$ with security parameter κ is collision resistant against 2^{κ^ϵ}-time adversaries. Then, consider a hash function family such that on security parameter λ, we use $\mathcal{H}$ with security parameter $\kappa := (\log \lambda)^{2/\epsilon}$. This hash function family is collision resistant against $\lambda^{\log \lambda}$-time adversaries, and it is polylogarithmic depth (in λ) since every hash function in it can be evaluated by a circuit of size $\mathsf{poly}(\kappa) = \mathsf{poly}(\log \lambda)$. The domain of each hash function can be extended to $\{0,1\}^{\mathsf{poly}(\lambda)}$ without increasing the depth by using each hash function in parallel. ◊

3.2.2 Tree hash.

Recall that for any function $h : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$, the (binary) *tree-hash* of a string $x \in \{0,1\}^{2^\ell \lambda}$ ($\ell \in \mathbb{N}$) is obtained as follows.

1. Separate x into 2^ℓ blocks $x_0, \ldots, x_{2^\ell-1}$ such that $|x_0| = \ldots |x_{2^\ell-1}| = \lambda$.
2. For each $\sigma \in \{0,1\}^i$ ($0 \leq i \leq \ell$), define $X_\sigma \in \{0,1\}^\lambda$ recursively as follows.
   (a) For each $\sigma \in \{0,1\}^\ell$, let $X_\sigma := x_\sigma$, where σ is identified with an integer in $\{0, \ldots, 2^\ell - 1\}$ naturally.
   (b) For each $\sigma \in \{0,1\}^i$ ($0 \leq i \leq \ell - 1$), let $X_\sigma := h(X_{\sigma 0} X_{\sigma 1})$.
3. Let the tree-hash of x be X_ε, where ε is the empty string.

We use $\mathsf{TreeHash}_h(x)$ to denote the tree-hash of x.

3.3 Keyless Multi-Collision Resistant Hash Functions

We recall the definition of multi-collision resistant hash functions from [BKP18], focusing on the keyless version.

Definition 2. *For any functions $K : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and $\gamma : \mathbb{N} \to \mathbb{N}$, a keyless hash function* Hash *is said to be* weakly (K, γ)-collision-resistant *if for every probabilistic $\gamma^{O(1)}$-time adversary $\mathcal{A}$ and every sequence of polynomial-size advice $\{z_\lambda\}_{\lambda\in\mathbb{N}}$, there exists a negligible function* negl *such that for every $\lambda \in \mathbb{N}$, the following holds for $K = K(\lambda, |z_\lambda|)$.*

$$\Pr\left[\begin{array}{l} y_1 = \cdots = y_K \\ \wedge\, \forall i \neq j : x_i \neq x_j \end{array} \;\middle|\; \begin{array}{l} (x_1, \ldots, x_K) \leftarrow \mathcal{A}(1^\lambda, z_\lambda) \\ \forall i : y_i := \mathsf{Hash}(1^\lambda, x_i) \end{array}\right] \leq \mathsf{negl}(\gamma(\lambda)).$$

As in [BKP18], we focus on the case that Hash is polynomially compressing. In particular, we assume that $\mathsf{Hash}(1^\lambda, \cdot)$ takes a string of length λ^2 as input and outputs a string of length λ.

3.4 Zero-Knowledge Arguments

We recall the standard definition of zero-knowledge arguments (see, e.g., [Gol01]).

Definition 3 (Interactive arguments)**.** *For any NP language $\mathbf{L}$, a pair of interactive Turing machines (P, V) is called* an interactive argument for $\mathbf{L}$ *if it satisfies the following.*

[10] For example, the (somewhere statistically binding) hash function of [HW15], which we use indirectly through a result of a prior work [CJJ22], is public coin and collision resistant.

- ***Completeness.*** *There exists a negligible function* negl *such that for every* $(x, w) \in \boldsymbol{R_L}$*,*

$$\Pr\left[\langle P(w), V\rangle(x) = 1\right] \geq 1 - \mathsf{negl}(|x|) \ .$$

- ***Soundness.*** *For every* PPT *interactive Turing machine* P^**, there exists a negligible function* negl *such that for every* $x \in \{0,1\}^* \setminus \boldsymbol{L}$ *and* $z \in \{0,1\}^*$*,*

$$\Pr\left[\langle P^*(z), V\rangle(x) = 1\right] \leq \mathsf{negl}(|x|) \ .$$

Definition 4 (Zero-knowledge)**.** *An interactive argument* (P, V) *for an NP language* $\boldsymbol{L}$ *is called* (computational) zero-knowledge *if for any* PPT *interactive Turing machine* V^**, there exists a* PPT *Turing machine* $\mathcal{S}$ *such that for any sequence* $\{w_x\}_{x\in\boldsymbol{L}}$ *such that* $w_x \in \boldsymbol{R_L}(x)$*, the following ensembles are computationally indistinguishable.*

- $\{\mathsf{view}_{V^*}\langle P(w_x), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$
- $\{\mathcal{S}(x, z)\}_{x\in L, z\in\{0,1\}^*}$

The following definition of public-coin interactive argument is taken from [HLR21b].

Definition 5 (Public coin)**.** *An interactive argument* (P, V) *is said to be* public coin *if the following hold.*

- *For some* $\ell(\lambda) \leq \mathsf{poly}(\lambda)$ *and every* $x \in \{0,1\}^*$*, the messages sent by* $V(x)$ *are independently and identically distributed uniformly random* $\ell(|x|)$*-bit strings.*
- *The final output of* $V(x)$ *when interacting with a prover is a fixed polynomial-time computable function of* x *and the transcript* τ *of its interaction with the prover.*

3.5 Weak Memory Delegations

We recall the definition of 2-round weak memory delegation schemes [BKP18] (which are weaker than those in [CKLR11, BBK+16] because of the soundness definition). We focus on the keyless setting, and use the publicly verifiable version of the definition.

Definition 6. *We say that an efficiently samplable distribution ensemble* $\{Y_\lambda\}_{\lambda\in\mathbb{N}}$ *is* entropic *if*

$$H_\infty(Y_\lambda) := -\log \max_{y\in\mathrm{supp}(Y_\lambda)} \Pr\left[Y_\lambda = y\right] = \Omega(\lambda) \ .$$

Definition 7. *A* publicly verifiable 2-round weak memory delegation scheme *consists of four algorithms* $(\mathsf{Mem}, \mathsf{Query}, \mathsf{Prove}, \mathsf{Ver})$ *that have the following syntax and efficiency.*

Syntax.

- $\mathsf{digest} := \mathsf{Mem}(1^\lambda, \mathsf{DB})$*:* Mem *is a deterministic polynomial-time algorithm that takes as input a security parameter* 1^λ *and a memory* DB*, and it outputs a digest* digest *of the memory.*
- $q \leftarrow \mathsf{Query}(1^\lambda)$*:* Query *is a probabilistic polynomial-time algorithm that takes as input a security parameter* 1^λ*, and it outputs a query* q*.*
- $\pi := \mathsf{Prove}(\mathsf{DB}, \langle M, t, y\rangle, q)$*:* Prove *is a deterministic algorithm that takes as input a memory* DB*, a deterministic Turing machine* M *(possibly with some hardwired inputs), a time bound* t*, an output* y*, and a query* q*, and it outputs a proof* π*.*
- $b := \mathsf{Ver}(\mathsf{digest}, \langle M, t, y\rangle, q, \pi)$*:* Ver *is a deterministic algorithm that takes as input a digest* digest*, a deterministic Turing machine* M *(possibly with some hardwired inputs), a time bound* t*, an output* y*, a query* q*, and a proof* π*, and it outputs a bit* b*.*

Efficiency. *For any polynomial p, there exists polynomials poly_P, poly_V such that for every $\lambda \in \mathbb{N}$, $\langle M, t, y\rangle \in \{0,1\}^{p(\lambda)}$, and $\mathsf{DB} \in \{0,1\}^*$ such that $M(\mathsf{DB})$ outputs y within t steps and $|\mathsf{DB}| \leq t \leq \lambda^{\log \lambda}$,*

- $\mathsf{Prove}(\mathsf{DB}, \langle M, t, y\rangle, q)$ *runs in time* $\mathsf{poly}_P(\lambda, t)$*, and*
- $\mathsf{Ver}(\mathsf{digest}, \langle M, t, y\rangle, q, \pi)$ *runs in time* $\mathsf{poly}_V(\lambda)$.

Security. *For any function $\bar{t} : \mathbb{N} \to \mathbb{N}$, a publicly verifiable 2-round weak memory delegation scheme is called* sound for computation-time bound $\bar{t}$ *if it satisfies the following.*

- ***Correctness.*** *For every $\lambda \in \mathbb{N}$, $\langle M, t, y\rangle \in \{0,1\}^{\mathsf{poly}(\lambda)}$, and $\mathsf{DB} \in \{0,1\}^*$ such that $M(\mathsf{DB})$ outputs y within t steps and $|\mathsf{DB}| \leq t \leq \bar{t}(\lambda)$,*[11]

$$\Pr\left[\mathsf{Ver}(\mathsf{digest}, \langle M, t, y\rangle, q, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{digest} := \mathsf{Mem}(1^\lambda, \mathsf{DB}) \\ q \leftarrow \mathsf{Query}(1^\lambda) \\ \pi := \mathsf{Prove}(\mathsf{DB}, \langle M, t, y\rangle, q) \end{array}\right] = 1 \ .$$

- ***Soundness for computation-time bound $\bar{t}$.*** *For every pair of* PPT *adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ and every sequence of polynomial-size advice $\{z_\lambda\}_{\lambda\in\mathbb{N}}$, there exists a negligible function* negl *such that for every samplable entropic distribution ensemble $\{Y_\lambda\}_{\lambda\in\mathbb{N}}$, every $\lambda \in \mathbb{N}$, and every $t \leq \bar{t}^{O(1)}(\lambda)$,*

$$\Pr\left[\mathsf{Ver}(\mathsf{digest}, \langle M, t, y\rangle, q, \pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{digest}, M, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ q \leftarrow \mathsf{Query}(1^\lambda) \\ y \leftarrow Y_\lambda \\ \pi \leftarrow \mathcal{A}_2(q, y, \mathsf{st}) \end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

A publicly verifiable 2-round weak memory delegation scheme is called public coin *if the query algorithm* Query *is public coin, i.e., it just outputs a string that is sampled uniformly randomly.*

3.6 Oracle Memory Delegations

We recall the definition of 2-round oracle memory delegation schemes [BKP18]. We use the publicly verifiable version of the definition, and for technical reasons, use a slightly modified version of the definition (see Remark 3).

Definition 8. *A* publicly verifiable 2-round oracle memory delegation scheme *consists of five algorithms $(\mathsf{Mem}, \mathsf{Query}_1, \mathsf{Prove}, \mathsf{Query}_2, \mathsf{Ver})$ that have the following syntax and efficiency.*

Syntax.

- $\widehat{\mathsf{DB}} := \mathsf{Mem}(1^\lambda, \mathsf{DB})$*:* Mem *is a deterministic polynomial-time algorithm that takes as input a security parameter 1^λ and a memory* DB*, and it outputs an encoding $\widehat{\mathsf{DB}}$ of the memory.*
- $(q, \sigma) \leftarrow \mathsf{Query}_1(1^\lambda)$*:* Query_1 *is a probabilistic polynomial-time algorithm that takes as input a security parameter 1^λ, and it outputs a query q and a random string σ.*
- $\pi := \mathsf{Prove}(\mathsf{DB}, \langle M, t, y\rangle, q)$*:* Prove *is a deterministic algorithm that takes as input a memory* DB*, a deterministic Turing machine M (possibly with some hardwired inputs), a time bound t, an output y, and a query q, and it outputs a proof π.*
- $I := \mathsf{Query}_2(L_{\mathsf{DB}}, \sigma, \pi)$*:* Query_2 *is a deterministic algorithm that takes as input a length parameter L_{DB}, a random string σ, and a proof π, and it outputs a set $I \subseteq \mathbb{N}$ of oracle queries.*
- $b := \mathsf{Ver}^{(\cdot)}(L_{\mathsf{DB}}, \langle M, t, y\rangle, q, \sigma, \pi)$*:* Ver *is a deterministic oracle algorithm that takes as input a length parameter L_{DB}, a deterministic Turing machine M (possibly with some hardwired inputs), a time bound t, an output y, a query q, a random string σ, and a proof π, and it outputs a bit b.*

[11] We consider a slightly weaker notion of correctness where t is at most $\bar{t}(\lambda)$. (In [BKP18], t is at most 2^λ.) In this paper, $\bar{t}$ is a super-polynomial function, and this version of correctness is sufficient for our purpose.

Efficiency. *For any polynomial p, there exists polynomials $\mathsf{poly}_P, \mathsf{poly}_V$ such that for every $\lambda \in \mathbb{N}$, $\langle M, t, y\rangle \in \{0,1\}^{p(\lambda)}$, and $\mathsf{DB} \in \{0,1\}^*$ such that $M(\mathsf{DB})$ outputs y within t steps and $|\mathsf{DB}| \leq t \leq \lambda^{\log \lambda}$,*

- $\mathsf{Prove}(\mathsf{DB}, \langle M, t, y\rangle, q)$ *runs in time* $\mathsf{poly}_P(\lambda, t)$*, and*
- $\mathsf{Ver}^{(\cdot)}(|\mathsf{DB}|, \langle M, t, y\rangle, q, \sigma, \pi)$ *runs in time* $\mathsf{poly}_V(\lambda)$*.*

Security. *For any functions $\gamma, \bar{t} : \mathbb{N} \to \mathbb{N}$, a publicly verifiable 2-round oracle memory delegation scheme is called* γ-sound for computation-time bound $\bar{t}$ *if it satisfies the following.*

- ***Correctness.*** *For every $\lambda \in \mathbb{N}$, $\langle M, t, y\rangle \in \{0,1\}^{\mathsf{poly}(\lambda)}$, and $\mathsf{DB} \in \{0,1\}^*$ such that $M(\mathsf{DB})$ outputs y within t steps and $|\mathsf{DB}| \leq t \leq \bar{t}(\lambda)$,*

$$\Pr\left[\mathsf{Ver}^{\widehat{\mathsf{DB}}|_I}(|\mathsf{DB}|, \langle M, t, y\rangle, q, \sigma, \pi) = 1 \;\middle|\; \begin{array}{l} \widehat{\mathsf{DB}} := \mathsf{Mem}(1^\lambda, \mathsf{DB}) \\ (q, \sigma) \leftarrow \mathsf{Query}_1(1^\lambda) \\ \pi := \mathsf{Prove}(\mathsf{DB}, \langle M, t, y\rangle, q) \\ I := \mathsf{Query}_2(|\mathsf{DB}|, \sigma, \pi) \end{array}\right] = 1 \ .$$

- ***γ-soundness for computation-time bound $\bar{t}$.*** *For every pair of probabilistic $\gamma^{O(1)}$-time adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ and every sequence of polynomial-size advice $\{z_\lambda\}_{\lambda\in\mathbb{N}}$, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$ and $t \leq \bar{t}^{O(1)}(\lambda)$,*

$$\Pr\left[\begin{array}{l} y \neq y' \\ \wedge\ \mathsf{Ver}^{\widehat{\mathsf{DB}}|_I}(L_{\mathsf{DB}}, \langle M, t, y\rangle, q, \sigma, \pi) = 1 \\ \wedge\ \mathsf{Ver}^{\widehat{\mathsf{DB}}|_{I'}}(L_{\mathsf{DB}}, \langle M, t, y'\rangle, q, \sigma, \pi') = 1 \end{array} \;\middle|\; \begin{array}{l} (\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, M, y, y', \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ (q, \sigma) \leftarrow \mathsf{Query}_1(1^\lambda) \\ (\pi, \pi') \leftarrow \mathcal{A}_2(q, \sigma, \mathsf{st}) \\ I := \mathsf{Query}_2(L_{\mathsf{DB}}, \sigma, \pi) \\ I' := \mathsf{Query}_2(L_{\mathsf{DB}}, \sigma, \pi') \end{array}\right] \leq \mathsf{negl}(\gamma(\lambda)) \ .$$

A publicly verifiable 2-round oracle memory delegation scheme is called public coin *if the query algorithm Query_1 is public coin, i.e., it just outputs a string that is sampled uniformly randomly.*

Remark 3 (Differences from the original definition [BKP18]). First, the syntax is slightly more general since we split the query algorithm into two, Query_1 and Query_2, so that input queries can be chosen based on the proof π. (An additional minor syntax difference is that Ver (and Query_2) is given the memory length, i.e., $|\mathsf{DB}|$.) Second, soundness is slightly stronger since we allow the adversary $\mathcal{A}_2$ to learn σ (which allows $\mathcal{A}_2$ to learn the input queries I, I'). This stronger soundness definition is required for our application. ◊

3.7 Low-Degree Extensions

Let $\mathbb{F}$ be a finite field, $\mathbb{H} \subseteq \mathbb{F}$ be a subset of $\mathbb{F}$, and $m \in \mathbb{N}$ be an integer. Any function $f : \mathbb{H}^m \to \{0,1\}$ can be extended into a (unique) function $\hat{f} : \mathbb{F}^m \to \mathbb{F}$ such that (i) $\hat{f}(z) = f(z)$ for every $z \in \mathbb{H}^m$ and (ii) $\hat{f}$ is an m-variate polynomial of degree at most $|\mathbb{H}| - 1$ in each variable. This function $\hat{f}$ (or the truth table of it) is called the *low-degree extension* (LDE) of f.

Low-degree extensions of strings. The LDE of a binary string x of length N can be obtained by choosing $\mathbb{H}$ and m such that $N \leq |\mathbb{H}|^m$, identifying $\{1, \ldots, |\mathbb{H}|^m\}$ with $\mathbb{H}^m$ in the lexicographical order, and viewing x as a function $x : \mathbb{H}^m \to \{0,1\}$ such that $x(i) = x_i$ for $\forall i \in [N]$ and $x(i) = 0$ for $\forall i \in \{N+1, \ldots, |\mathbb{H}|^m\}$. We use $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}(x)$ to denote the LDE of x. We note that for any $z \in \mathbb{F}^m$, the LDE of x can be evaluated on z in time $|\mathbb{H}|^m \cdot \mathsf{poly}(m, |\mathbb{H}|)$, where we assume that we have $|\mathbb{F}| = \mathsf{poly}(|\mathbb{H}|)$ and field operations over $\mathbb{F}$ can be done in time $\mathsf{poly}(\log|\mathbb{F}|) = \mathsf{poly}(\log|\mathbb{H}|)$ (see, e.g., [GKR15, Claim 2.3]).

3.8 Circuits

We consider arithmetic circuits that have addition and multiplication gates with fan-in 2 over a finite field. We focus on "layered circuits," i.e., we assume that the gates in a circuit can be partitioned into layers such that (i) the first layer contains the input gates and the last layer contains the output gates, and (ii) the gates in the i-th layer have children in the $(i-1)$-st layer. By adding dummy gates, we assume that all layers in a layered circuit have the same width. (The dummy gates always take 0 as their values and their output wires are never connected to any gate.)

4 Public-Coin Tree-Hash Oracle Memory Delegation

In this section, we construct a public-coin *tree-hash oracle memory delegation scheme*. Tree-hash oracle memory delegation schemes are oracle memory delegation schemes that are focused on proving the correctness of tree-hash computations. (For convenience, we consider those that satisfy a tailored soundness notion.) The formal definition, as well as the main lemma of this section, is given below.

Definition 9. *For any hash function family $\mathcal{H}$,* publicly verifiable 2-round tree-hash oracle memory delegation schemes *are defined in the same way as publicly verifiable 2-round oracle memory delegation schemes (Definition 8) except for the following differences.*

1. *Correctness is defined for a statement of the form $\langle M_h, t, y\rangle$ and a memory DB of length $2^i\lambda$ for $\lambda \in \mathbb{N}, h \in \mathcal{H}_\lambda, t \in \mathbb{N}, y \in \{0,1\}^\lambda$, and $i \in [\lfloor \log^2 \lambda \rfloor]$, where M_h is a Turing machine that takes as input a string $\mathsf{DB} \in \{0,1\}^*$ and outputs $\mathsf{TreeHash}_h(\mathsf{DB})$ using the hash function h that is hardwired in it.*[12]

2. *The soundness condition is replaced with the following one.*

   - ***γ-soundness.*** *There exists a probabilistic polynomial-time algorithm Decode such that for every pair of probabilistic $\gamma^{O(1)}$-time adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ and every sequence of polynomial-size advice $\{z_\lambda\}_{\lambda\in\mathbb{N}}$, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$ and $h \in \mathcal{H}_\lambda$,*

$$\Pr\left[\begin{array}{l} \mathsf{rt} \neq \mathsf{TreeHash}_h(\widetilde{\mathsf{DB}}) \\ \wedge\ \mathsf{Ver}^{\widehat{\mathsf{DB}}|_I}(L_{\mathsf{DB}}, \langle M_h, t_{L_{\mathsf{DB}}}, \mathsf{rt}\rangle, q, \sigma, \pi) = 1 \end{array} \middle| \begin{array}{l} (\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ (q, \sigma) \leftarrow \mathsf{Query}_1(1^\lambda) \\ (\mathsf{rt}, \pi) \leftarrow \mathcal{A}_2(h, q, \sigma, \mathsf{st}) \\ I := \mathsf{Query}_2(L_{\mathsf{DB}}, \sigma, \pi) \\ \widetilde{\mathsf{DB}} \leftarrow \mathsf{Decode}(\widehat{\mathsf{DB}}, L_{\mathsf{DB}}) \end{array}\right] \leq \mathsf{negl}(\gamma(\lambda)) \ ,$$

   where $t_{L_{\mathsf{DB}}}$ is the running time of M_h for inputs of length L_{DB}, and $\mathsf{Decode}(\cdot, L_{\mathsf{DB}})$ always outputs an L_{DB}-bit string (or $\perp$).

Lemma 1. *Assume the sub-exponential hardness of the LWE assumption. Then, for any polylogarithmic-depth hash function family and any sufficiently small super-polynomial functions γ (e.g., $\gamma(\lambda) = \lambda^{\log\log\lambda}$), there exists a public-coin 2-round tree-hash oracle memory delegation scheme with γ-soundness.*

4.1 Public-Coin Weak Tree-Hash Oracle Memory Delegation

Before proving Lemma 1, we first construct a scheme with a weak soundness guarantee (where soundness is only required to hold when the prover provides a valid encoding of a memory). Specifically, we show the following lemma.

Lemma 2. *Assume the sub-exponential hardness of the LWE assumption. Then, for any polylogarithmic-depth hash function family $\mathcal{H}$ and any sufficiently small super-polynomial functions γ (e.g., $\gamma(\lambda) = \lambda^{\log\log\lambda}$), there exists a public-coin 2-round tree-hash oracle memory delegation scheme with the following weaker soundness guarantee. (The differences from Definition 9 are highlighted by underlines.)*

[12] We assume that $h \in \mathcal{H}_\lambda$ hashes a string of length 2λ to a string of length λ. Therefore, $\mathsf{TreeHash}_h$ hashes a string of length $2^i\lambda$ to a string of length λ by depth-i tree-hashing (cf. Section 3.2.2).

- ***Weak γ-soundness.*** *There exists a deterministic polynomial-time algorithm* Decode *and a predicate* Valid *such that for every pair of probabilistic $\gamma^{O(1)}$-time adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ and every polynomial-size advice $\{z_\lambda\}_{\lambda\in\mathbb{N}}$, there exists a negligible function* negl *such that for every $\lambda \in \mathbb{N}$ and $h \in \mathcal{H}_\lambda$,*

$$\Pr\left[\begin{array}{l} \mathsf{rt} \neq \mathsf{TreeHash}_h(\widetilde{\mathsf{DB}}) \\ \wedge\, \mathsf{Ver}^{\widehat{\mathsf{DB}}|_I}(L_{\mathsf{DB}}, \langle M_h, t_{L_{\mathsf{DB}}}, \mathsf{rt}\rangle, q, \sigma, \pi) = 1 \\ \wedge\, \underline{\mathsf{Valid}(\widehat{\mathsf{DB}}, L_{\mathsf{DB}}) = 1} \end{array} \middle| \begin{array}{l} (\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ (q, \sigma) \leftarrow \mathsf{Query}_1(1^\lambda) \\ (\mathsf{rt}, \pi) \leftarrow \mathcal{A}_2(h, q, \sigma, \mathsf{st}) \\ I := \mathsf{Query}_2(L_{\mathsf{DB}}, \sigma, \pi) \\ \widetilde{\mathsf{DB}} \leftarrow \mathsf{Decode}(\widehat{\mathsf{DB}}, L_{\mathsf{DB}}) \end{array}\right] \leq \mathsf{negl}(\gamma(\lambda)) \ ,$$

where $t_{L_{\mathsf{DB}}}$ is the running time of M_h for inputs of length L_{DB}, and $\mathsf{Decode}(\cdot, L_{\mathsf{DB}})$ always outputs an L_{DB}-bit string (or $\perp$).

Furthermore, (i) $\mathsf{Mem}(1^\lambda, \mathsf{DB})$ outputs $\mathsf{LDE}_{\mathbb{F},\mathbb{H},m}(\mathsf{DB})$ for some $(\mathbb{F}, \mathbb{H}, m)$, where $(\mathbb{F}, \mathbb{H}, m)$ are the parameters that are determined based on $|\mathsf{DB}|$ and satisfy $2m|\mathbb{H}| < |\mathbb{F}| = \mathsf{poly}(\log\lambda)$ and $|\mathsf{DB}| \leq |\mathbb{H}|^m \leq |\mathbb{F}|^m \leq \mathsf{poly}(|\mathsf{DB}|)$, and (ii) $\mathsf{Valid}(\widehat{\mathsf{DB}}, L_{\mathsf{DB}})$ outputs 1 if and only if $\widehat{\mathsf{DB}}$ is (the truth table of) a polynomial $\hat{x} : \mathbb{F}^m \to \mathbb{F}$ of degree at most $m(|\mathbb{H}| - 1)$.

We prove Lemma 2 by relying on recent results [JKKZ21, HLR21b] about soundly applying the Fiat–Shamir transformation to the public-coin interactive proof of Goldwasser, Kalai, and Rothblum [GKR15] (the GKR interactive proof in short). Therefore, we start by recalling the GKR interactive proof and the results about the Fiat–Shamir transformation [JKKZ21, HLR21b].

4.1.1 Preliminary 1: the GKR interactive proof.

We recall the "bare-bones" version of the GKR interactive proof [GKR15, Section 3], focusing on the parts that are relevant to us. (As in [JKKZ21], we use a slightly modified version of it [KPY18] so that we can use the recent results about the Fiat–Shamir transformation [JKKZ21, HLR21b].)

The GKR interactive proof is a public-coin interactive proof for proving the correctness of computations. The statement consists of a circuit C and an input x, and the prover proves $C(x) = 0$. The circuit C is an arithmetic circuit over a finite field $\mathbb{F}$. The circuit C is assumed to be *layered*, i.e., the gates in C can be partitioned into layers such that (i) the starting layer consists of the input gates and the last layer consists of the output gates, and (ii) the gates in the i-th layer take their inputs from the gates in the $(i-1)$-st layer. For a circuit of depth D and W, we denote the gates in the i-th layer by $(g_{i,0}, \ldots, g_{i,W-1})$ for each $i \in \{0, \ldots, D\}$. We note that we start the index of layers from 0, i.e., the starting layer (which contains the input gates) is "the 0-th layer."

The GKR interactive proof is associated with several parameters. When the statement contains a circuit of depth D and width W, important parameters include the finite field $\mathbb{F}$ over which the circuit is defined, as well as a subset $\mathbb{H} \subset \mathbb{F}$ and an integer $m \in \mathbb{N}$. How exactly these parameters are used in the GKR interactive proof is not important to this paper (essentially, the parameters $(\mathbb{F}, \mathbb{H}, m)$ are used to obtain the LDE of the gate values of each layer). These parameters can be set relatively freely as long as they satisfy certain constraints, such as (i) $|\mathbb{F}|$ is sufficiently larger than D, $|\mathbb{H}|$, and m and (ii) $|\mathbb{H}|^m$ is larger than the width W. When $D = \mathsf{poly}(\log W)$, a typical choice is $|\mathbb{H}| = \mathsf{poly}(\log W)$, $m = \lceil \log_{|\mathbb{H}|} W \rceil = O(\log W / \log\log W)$, and $|\mathbb{F}| = \mathsf{poly}(|\mathbb{H}|) = \mathsf{poly}(\log W)$.

Importantly, in the GKR interactive proof, the verifier does not explicitly take a statement as input. Indeed, if the verifier explicitly takes a circuit C as input, the running time of the verifier becomes $\Omega(|C|)$, and the GKR interactive proof cannot have significant efficiency benefits. In the bare-bones version of the GKR interactive proof, the verifier learns about C by making queries to certain polynomials that are guaranteed to satisfy the following conditions. Let D and W be the depth and width of C. For each $i \in [D]$, let $\mathsf{add}_i : \{0, \ldots, W-1\}^3 \to \{0, 1\}$ be the function such that on input (u, v, w), it outputs 1 if and only if $g_{i,u} = g_{i-1,v} + g_{i-1,w}$, i.e., the u-th gate in the i-th layer is an addition gate such that its inputs come from the v-th and w-th gates in the $(i-1)$-st layer. Let $\{\mathsf{mult}_i\}_{i\in[D]}$ be defined similarly about multiplication gates. The functions $\{\mathsf{add}_i, \mathsf{mult}_i\}_{i\in[D]}$ are called *the functions that specify* C. Then, the verifier of the GKR interactive proof is given oracle access to *extensions* $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$ of $\{\mathsf{add}_i, \mathsf{mult}_i\}_{i\in[D]}$, where each $\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i : \mathbb{F}^{3m} \to \mathbb{F}$ are guaranteed to

satisfy the following for each $(z_u, z_v, z_w) \in \mathbb{H}^{3m}$. Let $\alpha : \mathbb{H}^m \to \{0, \ldots, |\mathbb{H}|^m - 1\}$ be the mapping that returns the lexicographic order of the input.

$$\widetilde{\mathsf{add}}_i(z_u, z_v, z_w) = \begin{cases} \mathsf{add}_i(\alpha(z_u), \alpha(z_v), \alpha(z_w)) & \text{if } \alpha(z_u), \alpha(z_v), \alpha(z_w) \leq W - 1 \\ 0 & \text{otherwise} \end{cases}.$$

$$\widetilde{\mathsf{mult}}_i(z_u, z_v, z_w) = \begin{cases} \mathsf{mult}_i(\alpha(z_u), \alpha(z_v), \alpha(z_w)) & \text{if } \alpha(z_u), \alpha(z_v), \alpha(z_w) \leq W - 1 \\ 0 & \text{otherwise} \end{cases}.$$

Extensions $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$ do not need to be the LDEs of $\{\mathsf{add}_i, \mathsf{mult}_i\}_{i \in [D]}$. Still, for the soundness to hold, they need to be low-degree polynomials, which roughly means that the individual degree δ of each $\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i$ is much smaller than the field size $|\mathbb{F}|$.

In [GKR15], the bare-bones version of the GKR interactive proof is used as a stepping-stone toward their main results. For example, the bare-bones version is used to obtain an interactive proof for log-space uniform bounded-depth circuit computations. (The verifier can evaluate extensions $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$ efficiently for such computations.[13]) It is also used to obtain an interactive proof for any (not necessarily log-space uniform) bounded-depth circuit computations by considering a model where the verifier evaluates $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$ in an offline pre-processing phase. Jumping ahead, we use the bare-bones version to obtain a protocol for a circuit that is not necessarily log-space uniform. The key point is that for the circuit that we consider, the verifier can evaluate $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$ efficiently because of the simple structure of the circuit.

Below, we summarize the properties of the GKR interactive proof that we use. (It is based on [GKR15, Theorem 3.1] and its analysis with slight adaptation. The differences are explained in footnotes.)

Lemma 3 (Soundness and efficiency of the GKR interactive proof)**.** *There exists a constant $c_{GKR} \in \mathbb{N}$ such that the GKR interactive proof is sound (with constant soundness error) when it is used with a finite field $\mathbb{F}$, an arithmetic circuit C over $\mathbb{F}$, and parameters $\mathbb{H} \subset \mathbb{F}$, $m \in \mathbb{N}$ that satisfy the following condition.*

- ***GKR compatibility:*** *Let W and D be the width and the depth of C. Then, there exists $\delta \in \mathbb{N}$ $(m(|\mathbb{H}|-1) \leq \delta < |\mathbb{F}|)$ for which the following hold.*[14]
  1. *The field $\mathbb{F}$ is sufficiently large. Concretely, it satisfies $c_{GKR} D m \delta \leq |\mathbb{F}| \leq \mathsf{poly}(|\mathbb{H}|)$.*
  2. *The parameters $\mathbb{H}$ and m satisfy $\max(D, \log W) \leq |\mathbb{H}| \leq \mathsf{poly}(D, \log W)$ and $W \leq |\mathbb{H}|^m \leq \mathsf{poly}(W)$.*
  3. *There exist polynomials $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$ such that (i) each $\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i$ are of individual degree at most δ and (ii) $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$ are extensions of the functions $\{\mathsf{add}_i, \mathsf{mult}_i\}_{i \in [D]}$ that specify C.*

Furthermore, soundness holds even in a model where the verifier is not given the statement (C, x) and instead given (i) oracle access to $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$ and (ii) oracle access to a polynomial $\widehat{x} : \mathbb{F}^m \to \mathbb{F}$ that is of (total) degree at most $m(\mathbb{H} - 1)$ and has x as a prefix of $\widehat{x}|_{\mathbb{H}^m}$.[15] *In such a model, the verifier runs in time $\mathsf{poly}(D, \log W)$ while the prover runs in time $\mathsf{poly}(D, W)$. The verifier queries the encoding $\widehat{x}$ at two points*[16] *(where the queries are determined by the (public) randomness of the verifier) and makes $O(D)$ queries to $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$.*

4.1.2 Preliminary 2: the Fiat–Shamir transformation for the GKR interactive proof.

We next recall a recent result by Holmgren, Lombardi, and Rothblum [HLR21b], where it is shown (based on an observation by Jawale, Kalai, Khurana, and Zhang [JKKZ21]) that we can obtain a public-coin non-interactive

[13] More precisely, in [GKR15], it is observed that the verifier can delegate the evaluation of $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$ to the prover, and in a subsequent work [Gol17b], it is observed that the verifier can evaluate $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$ efficiently.

[14] For convenience, we use a slightly stronger lower bound for δ. (In [GKR15], the requirement is $|\mathbb{H}|-1 \leq \delta < |\mathbb{F}|$.) See Footnote 15.

[15] In [GKR15, Theorem 3.1], the encoding $\widehat{x}$ is required to be the LDE of x. However, the only requirement that is used in the analysis of [GKR15, Theorem 3.1] is that the individual degree of $\widehat{x}$ is upper bounded by the degree parameter δ. Since we guarantee $\delta \geq m(|\mathbb{H}| - 1)$, it suffices to require that the total degree of $\widehat{x}$ is at most $m(\mathbb{H} - 1)$ (which implies that the individual degree is at most δ).

[16] Unlike the original version of the GKR interactive proof [GKR15], the version given in [KPY18] (which is the version that we use) requires the verifier to read $\widehat{x}$ at two points.

argument by applying the Fiat–Shamir transformation to the parallel repetition of the GKR interactive proof. (As stated in Lemma 3, the GKR interactive proof without repetitions has constant soundness.) The result that we use is summarized in the following lemma, which is a rephrasing of a result given in [HLR21a, Section 6.2.2] (see also [JKKZ20, Corollary 6.6, Theorem 4.5]).

Lemma 4. *Let $W, D, \delta : \mathbb{N} \to \mathbb{N}$ be functions such that $W(\lambda) \leq \mathsf{poly}(\lambda^{\log \lambda})$,[17] $D(\lambda) \leq \mathsf{poly}(\log \lambda)$, and $\delta(\lambda) \leq \mathsf{poly}(D(\lambda), \log W(\lambda))$. Then, under the sub-exponential hardness of the LWE assumption, there exists a public-coin hash family $\mathcal{H}_{\mathsf{FS}}$ that satisfies the following for $t(\lambda) = \mathsf{poly}(\lambda, D(\lambda), \log W(\lambda))$ and a sub-exponential function T.*

Let $\Pi = (\mathsf{Setup}, P, V)$ be the public-coin non-interactive argument that is obtained by applying the Fiat-Shamir transformation to the t-repetition of the GKR interactive proof w.r.t. the hash family $\mathcal{H}_{\mathsf{FS}}$ (cf. Section A.2).

1. ***Correctness.*** *For every $\lambda \in \mathbb{N}$, fix any $(C, \mathbb{F}, \mathbb{H}, m)$ such that (i) $\mathbb{F}$ is a finite field such that $|\mathbb{F}| = \mathsf{poly}(D(\lambda), \log W(\lambda))$ is sufficiently large, (ii) C is a layered arithmetic circuit over $\mathbb{F}$ with output length λ, where the width and the depth of C are at most $W(\lambda)$ and $D(\lambda)$ respectively, and (iii) $(C, \mathbb{F}, \mathbb{H}, m)$ is GKR compatible for $\delta \leq \delta(\lambda)$ (cf. Lemma 3). Then, when Π is used with λ and $(C, \mathbb{F}, \mathbb{H}, m)$, the following hold for every input x to C and $y := C(x)$.*

$$\Pr\left[V^{\mathcal{F}}(\mathsf{crs}, x, y, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ \pi \leftarrow P(\mathsf{crs}, C, x, y) \end{array}\right] = 1 \ ,$$

   where $\mathcal{F} := \{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$ are the polynomials that are guaranteed to exist by the GKR compatibility of $(C, \mathbb{F}, \mathbb{H}, m)$.

2. ***T-soundness.*** *For every probabilistic $\mathsf{poly}(T)$-time prover P^* and every sequence of polynomial-size advice $\{z_\lambda\}_{\lambda \in \mathbb{N}}$, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$, every $(C, \mathbb{F}, \mathbb{H}, m)$ as above, and every input x to C,*

$$\Pr\left[\begin{array}{l} y \neq C(x) \\ \wedge\, V^{\mathcal{F}}(\mathsf{crs}, x, y, \pi) = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (y, \pi) \leftarrow P^*(\mathsf{crs}, C, x, z_\lambda) \end{array}\right] \leq \mathsf{negl}(T(\lambda)) \ ,$$

   where $\mathcal{F}$ is defined as above.

3. ***Efficiency.*** *For every $\lambda \in \mathbb{N}$ and every $(C, \mathbb{F}, \mathbb{H}, m)$ as above, the prover P runs in time $D(\lambda) \cdot \mathsf{poly}(\lambda, D(\lambda), \log W(\lambda)) + T_{\mathrm{GKR},P}$ and the verifier V runs in time $D(\lambda) \cdot \mathsf{poly}(\lambda, D(\lambda), \log W(\lambda)) + T_{\mathrm{GKR},V}$, where $T_{\mathrm{GKR},P}$ and $T_{\mathrm{GKR},V}$ are the running time of the prover and the verifier in the t-repetition of the GKR interactive proof.*

Remark 4 (On how we obtain Lemma 4 from [HLR21b]). In [HLR21b], it is observed that once we fix several parameters of the GKR interactive proof, we can set the parameters of the hash function of [HLR21b] accordingly so that we can obtain a hush function family $\mathcal{H}_{\mathsf{FS}}$ for soundly applying the Fiat–Shamir transformation to the GKR interactive proof (see [HLR21a, Section 6.2.2]). In particular, once we fix upper bounds W, D, δ for the circuit depth, the circuit width, and the degree parameter, we can fix upper bounds for other relevant parameters of the GKR interactive proof (namely $|\mathbb{F}|$, $|\mathbb{H}|$, and m) using the conditions in Lemma 3, and then based on these bounds, we can obtain $\mathcal{H}_{\mathsf{FS}}$. Although the analyses in [JKKZ21, HLR21b] only consider the case of log-uniform bounded-depth computations, the same analyses can be used for the case of any computation with GKR compatible $(C, \mathbb{F}, \mathbb{H}, m)$.[18] ◊

Remark 5 (On adaptive choice of y in the definition of soundness). Lemma 4 differs from what is shown in [JKKZ21, HLR21b] in that (i) the output length of the circuit C is λ and (ii) the cheating prover in the definition of soundness is allowed to choose the output y adaptively. (In [JKKZ21, HLR21b], the output length of C is 1, and as in the GKR interactive proof described above, the output y is fixed to be 0). Still, it is easy to see that

[17] This is a super-polynomial upper bound that is sufficient for our purpose.

[18] We note that $\mathcal{H}_{\mathsf{FS}}$ does not depend on the evaluation time of $\mathcal{F} = \{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$. (In particular, the function BAD, which is used to define the size of hash functions of $\mathcal{H}_{\mathsf{FS}}$ in [JKKZ20, Section 6.2], depends on $\mathcal{F}$ only via polynomial-size non-uniform advice.)

the results in [JKKZ21, HLR21b] can be used to obtain Lemma 4. Consider, for simplicity, that C outputs a binary output. (This is the case that we are interested in.) First, if the output length of C is 1 and the cheating prover adaptively chooses the output $y \in \{0,1\}$, it suffices to consider a protocol where the verifier initiates the protocol of [JKKZ21, HLR21b] twice in parallel, one for the statement $C(x) = 0$ and the other for the statement $C(x) = 1$, and the prover chooses one of them according to the actual output. Next, if the output length of C is λ, the prover and the verifier run this single-bit protocol λ times in parallel, one for each output bit. In total, the protocol of [JKKZ21, HLR21b] is executed 2λ times in parallel, and it is easy to see that if there exists a cheating prover that breaks the multi-bit version with probability ϵ, there exists a cheating prover that breaks the original version with probability at least ϵ/λ. ◊

Remark 6 (On soundness when the verifier is not given (C, x) explicitly). The Fiat–Shamir transformation preserves the furthermore part of Lemma 3, i.e., soundness (and completeness) holds even when the verifier only has (i) oracle access to $\mathcal{F} = \{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$ and (ii) oracle access to a low-degree polynomial $\widehat{x}$ that encodes x. Also, the number of queries to $\mathcal{F}$ and $\widehat{x}$ is not increased by the Fiat–Shamir transformation.[19] Furthermore, since the queries to $\widehat{x}$ are determined by the verifier randomness in the GKR interactive proof, they are determined by the proof π after the Fiat–Shamir transformation. Thus, there is a deterministic polynomial-time algorithm $\mathsf{InpQuery}$ such that the correctness and soundness of Π hold even when we replace $V^{\mathcal{F}}(\mathsf{crs}, x, y, \pi)$ with $V^{\widehat{x}|_I, \mathcal{F}}(\mathsf{crs}, y, \pi)$, where $I := \mathsf{InpQuery}(\pi)$. Since Π is obtained from t-repetition of the GKR interactive proof for $t(\lambda) = \mathsf{poly}(\lambda, D(\lambda), \log W(\lambda))$, we have $|I| = \mathsf{poly}(\lambda, D(\lambda), \log W(\lambda))$. ◊

4.1.3 Proof of Lemma 2.

We are ready to give our weak tree-hash memory delegation scheme. Our approach is to use Lemma 4 for tree-hash computations. That is, we consider the GKR interactive proof for tree-hash computations and then obtain the desired protocol by applying the Fiat–Shamir transformation. By Lemma 3, we need a circuit that computes tree-hashing in a "GKR friendly" way, i.e., we need a tree-hashing circuit that satisfies properties such as having efficiently computable low-degree extensions $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$. Motivated by this observation, we prove the following lemma in Appendix C.

Lemma 5. *Let $\mathcal{H}$ be any polylogarithmic-depth hash function family. Then, there exist polynomials $\mathsf{poly}_W, \mathsf{poly}_D, \mathsf{poly}_\delta$ such that for any $\lambda, \ell \in \mathbb{N}$ ($\ell \le \log^2 \lambda$) and any finite field $\mathbb{F}$ of sufficiently large size $|\mathbb{F}| \le \mathsf{poly}(\log \lambda)$, there exist a subset $\mathbb{H} \subset \mathbb{F}$ and an integer $m \in \mathbb{N}$ such that for any $h : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda \in \mathcal{H}_\lambda$, there exists a layered arithmetic circuit $C : \mathbb{F}^L \to \mathbb{F}^\lambda$ that satisfies the following, where L is defined as $L := 2^\ell \lambda$.*

1. *The circuit C computes $\mathsf{TreeHash}_h$ for every input $x \in \{0,1\}^L$, and it outputs values in $\mathbb{F}^\lambda \setminus \{0,1\}^\lambda$ for inputs in $x \in \mathbb{F}^L \setminus \{0,1\}^L$. The circuit C is of width $W := \mathsf{poly}_W(\lambda, L)$ and of depth $D := \mathsf{poly}_D(\log \lambda, \ell)$.*
2. *There exist $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$ such that the tuple $(C, \mathbb{F}, \mathbb{H}, m)$ is GKR compatible for $\delta := \mathsf{poly}_\delta(D, \log W)$ and $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$. Furthermore, $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$ can be evaluated in time $\mathsf{poly}(\lambda)$ given the description of h.*

Furthermore, the parameters $(\mathbb{H}, m)$ can be obtained in polynomial time given λ and ℓ.

The proof of this lemma is straightforward. The circuit C is defined by connecting many copies of the polylogarithmic-depth circuit C_h of h in the tree structure. The key point is that, because of the tree structure of C, there exist extensions $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$ that can be evaluated almost as efficiently as the LDEs of the functions that specify C_h, which in turn can be evaluated in time $\mathsf{poly}(\lambda)$ since C_h is of size $\mathsf{poly}(\lambda)$. For a formal proof, see Appendix C.

Now, we proceed to the proof of Lemma 2.

Proof of Lemma 2. We first note that from Lemma 3, Lemma 4, Remark 6, and Lemma 5, we have the following corollary. (See Appendix B for the proof.)

[19] Note that the Fiat–Shamir transformation only requires hashing the transcript (excluding x) as shown in [JKKZ20, Figure 1] (cf. Section A.2).

Corollary 1. *Let $\mathcal{H}$ be any polylogarithmic-depth hash function family. Then, under the sub-exponential hardness of the LWE assumption, there exists a public-coin non-interactive argument $\Pi = (\mathsf{Setup}, P, V)$ and a deterministic polynomial-time algorithm $\mathsf{InpQuery}$ such that the following hold for a sub-exponential function T.*

1. ***Parameters.*** *For each $\lambda, \ell \in \mathbb{N}$ such that $\ell \leq \log^2 \lambda$, the non-interactive argument Π has $(\mathbb{F}, \mathbb{H}, m)$ as parameters, where $\mathbb{F}$ is a finite field, $\mathbb{H} \subset \mathbb{F}$ is a subset, and $m \in \mathbb{N}$ is an integer such that $2m|\mathbb{H}| < |\mathbb{F}| = \mathsf{poly}(\log \lambda)$ and $L \leq |\mathbb{H}|^m \leq |\mathbb{F}|^m \leq \mathsf{poly}(L)$, where $L = 2^\ell \lambda$.*

2. ***Completeness.*** *For every $\lambda, \ell \in \mathbb{N}$ ($\ell \leq \log^2 \lambda$), $h \in \mathcal{H}_\lambda$, $x \in \{0,1\}^L$, $\widehat{x} := \mathsf{LDE}_{\mathbb{F},\mathbb{H},m}(x)$, and $y := \mathsf{TreeHash}_h(x)$,*

$$\Pr\left[V^{\widehat{x}|_I}(\mathsf{crs}, \ell, h, y, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ \pi \leftarrow P(\mathsf{crs}, h, x, y) \\ I := \mathsf{InpQuery}(\pi) \end{array} \right] = 1 \ .$$

3. ***T-soundness.*** *For every probabilistic $\mathsf{poly}(T)$-time prover P^* and every sequence of polynomial-size advice $\{z_\lambda\}_{\lambda \in \mathbb{N}}$, there exists a negligible function negl such that for every $\lambda, \ell \in \mathbb{N}$ ($\ell \leq \log^2 \lambda$), every $h \in \mathcal{H}_\lambda$, and every polynomial $\widehat{x} : \mathbb{F}^m \to \mathbb{F}$ that is of degree at most $m(\mathbb{H} - 1)$,*

$$\Pr\left[\begin{array}{l} y \neq \mathsf{TreeHash}_h(x) \\ \wedge\, V^{\widehat{x}|_I}(\mathsf{crs}, \ell, h, y, \pi) = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (y, \pi) \leftarrow P^*(\mathsf{crs}, h, x, z_\lambda) \\ I := \mathsf{InpQuery}(\pi) \end{array} \right] \leq \mathsf{negl}(T(\lambda)) \ ,$$

   where x is the length-L prefix of $\widehat{x}|_{\mathbb{H}^m}$ if it is in $\{0,1\}^L$ and $x := \bot$ otherwise, where $L := 2^\ell \lambda$.

4. ***Efficiency.*** *The prover P runs in time $\mathsf{poly}(\lambda, L)$ and the verifier V runs in time $\mathsf{poly}(\lambda)$.*

Given Corollary 1, the proof of Lemma 2 is trivial. Consider the delegation scheme given in Algorithm 1. The efficiency and security conditions can be verified by inspection. (We note that Mem runs in polynomial time since we have $|\mathbb{F}|^m \leq \mathsf{poly}(|\mathsf{DB}|)$.) This completes the proof of Lemma 2. □

4.2 Proof of Lemma 1

We are ready to explain how we obtain our tree-hash oracle memory delegation scheme. The idea is to upgrade the soundness of our weak tree-hash oracle memory delegation scheme (Lemma 2) by considering a verifier that additionally checks the validity of the encoded memory. Fortunately, such a check can be implemented easily by relying on well-known techniques about low-degree polynomials, namely low-degree tests and self-correction (cf. Section A.3). As stated in Lemma 2, in our weak tree-hash oracle memory delegation scheme, an encoding of a memory is valid if it is a polynomial of degree at most $m(|\mathbb{H}| - 1)$. Thus, the verifier can use low-degree tests to check whether it is given an encoding $\widehat{\mathsf{DB}}$ that is close to a valid encoding $\widehat{\mathsf{DB}}'$, and then it can use self-correction to make queries to $\widehat{\mathsf{DB}}'$ through $\widehat{\mathsf{DB}}$. Details are given below.

First, we list the building blocks. Let $(\mathsf{LDTest.Q}, \mathsf{LDTest.D})$ and $(\mathsf{SelfCorr}'.\mathsf{Q}, \mathsf{SelfCorr}'.\mathsf{Rec})$ be the algorithms for low-degree tests and public-coin local self-correction as described in Section A.3. Let $\mathsf{wTHDel} = (\mathsf{wTHDel.Mem}, \mathsf{wTHDel.Query}_1, \mathsf{wTHDel.Prove}, \mathsf{wTHDel.Query}_2, \mathsf{wTHDel.Ver})$ be the public-coin 2-round weak tree-hash oracle memory delegation scheme that is given in Lemma 2, and $\mathsf{wTHDel.Decode}$ and $\mathsf{wTHDel.Valid}$ be the corresponding algorithm and predicate that are guaranteed to exist by the definition of weak soundness.

Our tree-hash oracle memory delegation scheme THDel is described in Algorithm 2 (including the decoding algorithm Decode that is required to exist by the definition of soundness). Since wTHDel is public coin, THDel is also public coin. Completeness can be verified by inspection. The efficiency condition of THDel follows from the efficiency condition of wTHDel. (We note that $|\mathbb{F}|$, $|\mathbb{H}|$, and m are $\mathsf{poly}(\log \lambda)$ even when $|\mathsf{DB}| = \lambda^{\log \lambda}$ (cf. the furthermore part of Lemma 2), and thus, both Query_1 and Query_2 output queries of total length $\mathsf{poly}(\lambda)$.) In the rest of this proof, we focus on soundness.

Algorithm 1 Public-coin weak oracle memory tree-hash delegation wTHDel.

Let $\Pi = (\mathsf{Setup}, P, V)$ and $\mathsf{InpQuery}$ be the public-coin non-interactive argument and the deterministic algorithm given by Corollary 1.

- $\widehat{\mathsf{DB}} := \mathsf{Mem}(1^\lambda, \mathsf{DB})$:
  1. Let ℓ^* be the integer such that $|\mathsf{DB}| = 2^{\ell^*}\lambda$, and let $(\mathbb{F}, \mathbb{H}, m)$ be the parameters that Π uses for λ and ℓ^*. Then, output $\widehat{\mathsf{DB}} := \mathsf{LDE}_{\mathbb{F},\mathbb{H},m}(\mathsf{DB})$.
- $(q, \sigma) \leftarrow \mathsf{Query}_1(1^\lambda)$:
  1. Run $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$.
  2. Output $q := \mathsf{crs}$ and $\sigma := \bot$.
- $\pi := \mathsf{Prove}(\mathsf{DB}, \langle M_h, t_{|\mathsf{DB}|}, \mathsf{rt}\rangle, q)$:
  1. Parse q as crs.
  2. Output $\pi \leftarrow P(\mathsf{crs}, h, \mathsf{DB}, \mathsf{rt})$, where h is the hash function that is hardwired in M_h.
- $I := \mathsf{Query}_2(|\mathsf{DB}|, \sigma, \pi)$:
  1. Output $I := \mathsf{InpQuery}(\pi)$.
- $b := \mathsf{Ver}^{\widehat{\mathsf{DB}}|_I}(|\mathsf{DB}|, \langle M_h, t_{|\mathsf{DB}|}, \mathsf{rt}\rangle, q, \sigma, \pi)$:
  1. Obtain ℓ^* as in Mem, and obtain crs from q as in Prove. (If there does not exist ℓ^* such that $|\mathsf{DB}| = 2^{\ell^*}\lambda$, output 0.)
  2. Output $b := V^{\widehat{\mathsf{DB}}|_I}(\mathsf{crs}, \ell^*, h, \mathsf{rt}, \pi)$.
- $\widetilde{\mathsf{DB}} \leftarrow \mathsf{Decode}(\widehat{\mathsf{DB}}, L_{\mathsf{DB}})$:
  1. Let $\widetilde{\mathsf{DB}}$ be the length-L_{DB} prefix of $\widehat{\mathsf{DB}}|_{\mathbb{H}^m}$.
  2. Output $\widetilde{\mathsf{DB}}$ if $\widetilde{\mathsf{DB}} \in \{0,1\}^{L_{\mathsf{DB}}}$, and output $\bot$ otherwise.

Algorithm 2 Public-coin tree-hash oracle memory delegation THDel.

- $\widehat{\mathsf{DB}} \coloneqq \mathsf{Mem}(1^\lambda, \mathsf{DB})$:
  1. Output $\widehat{\mathsf{DB}} \coloneqq \mathsf{wTHDel.Mem}(1^\lambda, \mathsf{DB})$. As stated in Lemma 2, it holds $\widehat{\mathsf{DB}} = \mathsf{LDE}_{\mathbb{F},\mathbb{H},m}(\mathsf{DB})$ for some $(\mathbb{F}, \mathbb{H}, m)$ that is determined based on $|\mathsf{DB}|$.
- $(q, \sigma) \leftarrow \mathsf{Query}_1(1^\lambda)$:
  1. Run $(q_{\mathsf{wTHDel}}, \sigma_{\mathsf{wTHDel}}) \leftarrow \mathsf{wTHDel.Query}_1(1^\lambda)$.
  2. Sample a random string r_1 that is long enough to run the λ^2-repetition of $\mathsf{LDTest.Q}$ for $\widehat{\mathsf{DB}}$. Since Query_1 does not know $(\mathbb{F}, \mathbb{H}, m)$, the length of r_1 is determined assuming that DB has the longest possible length, i.e., $|\mathsf{DB}| = \lambda^{\log \lambda}$.
  3. Sample a random string r_2 that is long enough to run $\mathsf{SelfCorr}'.\mathsf{Q}$ for $\widehat{\mathsf{DB}}$, where the length of r_2 is determined as above.
  4. Output $q \coloneqq q_{\mathsf{wTHDel}}$ and $\sigma \coloneqq (\sigma_{\mathsf{wTHDel}}, r_1, r_2)$.
- $\pi \coloneqq \mathsf{Prove}(\mathsf{DB}, \langle M_h, t_{|\mathsf{DB}|}, \mathsf{rt}\rangle, q)$:
  1. Output $\pi \coloneqq \mathsf{wTHDel.Prove}(\mathsf{DB}, \langle M_h, t_{|\mathsf{DB}|}, \mathsf{rt}\rangle, q)$.
- $I \coloneqq \mathsf{Query}_2(|\mathsf{DB}|, \sigma, \pi)$:
  1. Parse σ as $(\sigma_{\mathsf{wTHDel}}, r_1, r_2)$.
  2. Run the λ^2-repetition of $\mathsf{LDTest.Q}$ using r_1 as randomness, where the parameters $\mathbb{F}$, m, and $d \coloneqq m(|\mathbb{H}| - 1)$ are determined based on $|\mathsf{DB}|$. Let $((\mathsf{st}_1, Q_1), \ldots, (\mathsf{st}_{\lambda^2}, Q_{\lambda^2}))$ denote the output.
  3. Run $I_{\mathsf{wTHDel}} \coloneqq \mathsf{wTHDel.Query}_2(|\mathsf{DB}|, \sigma_{\mathsf{wTHDel}}, \pi)$, and for each $v \in I_{\mathsf{wTHDel}}$, run $(\mathsf{st}_v, Q_v) \coloneqq \mathsf{SelfCorr}'.\mathsf{Q}(v; r_2)$ with the parameters $(\mathbb{F}, m, d)$ that are determined as above.
  4. Output $I \coloneqq (\bigcup_{i\in[\lambda^2]} Q_i) \cup (\bigcup_{v\in I_{\mathsf{wTHDel}}} Q_v)$.
- $b \coloneqq \mathsf{Ver}^{\widehat{\mathsf{DB}}|_I}(|\mathsf{DB}|, \langle M_h, t_{|\mathsf{DB}|}, \mathsf{rt}\rangle, q, \sigma, \pi)$:
  1. Parse σ as $(\sigma_{\mathsf{wTHDel}}, r_1, r_2)$, and obtain $\{(\mathsf{st}_i, Q_i)\}_{i\in[\lambda^2]}$ and $\{(\mathsf{st}_v, Q_v)\}_{v\in I_{\mathsf{wTHDel}}}$ as in Query_2.
  2. Output 1 if both of the following hold.
     - $\mathsf{LDTest.D}(\mathsf{st}_i, \widehat{\mathsf{DB}}|_{Q_i}) = 1$ for $\forall i \in [\lambda^2]$.
     - $\mathsf{wTHDel.Ver}^{\widehat{\mathsf{DB}}'}(|\mathsf{DB}|, \langle M_h, t_{|\mathsf{DB}|}, \mathsf{rt}\rangle, q, \sigma_{\mathsf{wTHDel}}, \pi) = 1$, where $\widehat{\mathsf{DB}}'$ is defined by $\widehat{\mathsf{DB}}'(v) \coloneqq \mathsf{SelfCorr}'.\mathsf{Rec}(\mathsf{st}_v, \widehat{\mathsf{DB}}|_{Q_v})$ for $\forall v \in I_{\mathsf{wTHDel}}$.

     Otherwise, output 0.
- $\widetilde{\mathsf{DB}} \leftarrow \mathsf{Decode}(\widehat{\mathsf{DB}}, L_{\mathsf{DB}})$:
  1. Apply the global self-correction of low-degree polynomials to $\widehat{\mathsf{DB}}$ (cf. Section A.3.3). Let $\widehat{\mathsf{DB}}'$ be the result.
  2. Output $\mathsf{wTHDel.Decode}(\widehat{\mathsf{DB}}', L_{\mathsf{DB}})$.

Assume for contradiction that soundness does not hold w.r.t. Decode, i.e., there exists a pair of probabilistic $\gamma^{O(1)}$-time adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, a polynomial-size advice $\{z_\lambda\}_{\lambda\in\mathbb{N}}$, and a polynomial p such that for infinitely many $\lambda \in \mathbb{N}$, there exists $h \in \mathcal{H}_\lambda$ such that

$$\Pr\left[\begin{array}{l|l} \mathsf{rt} \neq \mathsf{TreeHash}_h(\widetilde{\mathsf{DB}}) & (\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ \wedge\ \mathsf{Ver}^{\widehat{\mathsf{DB}}|_I}(L_{\mathsf{DB}}, \langle M_h, t_{L_{\mathsf{DB}}}, \mathsf{rt}\rangle, q, \sigma, \pi) = 1 & (q, \sigma) \leftarrow \mathsf{Query}_1(1^\lambda) \\ & (\mathsf{rt}, \pi) \leftarrow \mathcal{A}_2(h, q, \sigma, \mathsf{st}) \\ & I := \mathsf{Query}_2(L_{\mathsf{DB}}, \sigma, \pi) \\ & \widetilde{\mathsf{DB}} \leftarrow \mathsf{Decode}(\widehat{\mathsf{DB}}, L_{\mathsf{DB}}) \end{array}\right] \geq \frac{1}{p(\gamma(\lambda))} \ . \quad (1)$$

We use $(\mathcal{A}_1, \mathcal{A}_2)$ to obtain a pair of $\gamma^{O(1)}$-time adversaries $(\mathcal{B}_1, \mathcal{B}_2)$ that breaks the weak soundness of wTHDel. For any $\lambda \in \mathbb{N}$, the pair $(\mathcal{B}_1, \mathcal{B}_2)$ is described in Algorithm 3. Note that $(\mathcal{B}_1, \mathcal{B}_2)$ run in time $\gamma^{O(1)}(\lambda)$ since $(\mathcal{A}_1, \mathcal{A}_2)$ run in time $\gamma^{O(1)}(\lambda)$.

Algorithm 3 Pair of adversaries $(\mathcal{B}_1, \mathcal{B}_2)$ against the weak soundness of wTHDel.

- $(\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, \mathsf{st}) \leftarrow \mathcal{B}_1(1^\lambda, z_\lambda)$:
  1. Run $(\widehat{\mathsf{DB}}_{\mathsf{THDel}}, L_{\mathsf{DB}}, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda)$.
  2. Apply the global self-correction of low-degree polynomials to $\widehat{\mathsf{DB}}_{\mathsf{THDel}}$ as in Decode. Let $\widehat{\mathsf{DB}}$ be the result.
  3. Output $(\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, \mathsf{st})$.
- $(\mathsf{rt}, \pi) \leftarrow \mathcal{B}_2(h, q, \sigma, \mathsf{st})$:
  1. Sample sufficiently long random strings r_1, r_2 as in Query_1.
  2. Output $(\mathsf{rt}, \pi) \leftarrow \mathcal{A}_2(h, q, \sigma_{\mathsf{THDel}}, \mathsf{st})$, where $\sigma_{\mathsf{THDel}} := (\sigma, r_1, r_2)$.

Let us see that the pair $(\mathcal{B}_1, \mathcal{B}_2)$ indeed breaks the soundness of wTHDel. Fix any $\lambda \in \mathbb{N}$ and $h \in \mathcal{H}_\lambda$ for which we have (1). Unless otherwise stated, in the following each probability is taken over the randomness of the following experiment, which is obtained from the experiment considered in (1) by inlining Query_1, Query_2, and Decode.

$$\begin{array}{l} (\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ (q, \sigma) \leftarrow \mathsf{wTHDel.Query}_1(1^\lambda) \\ r_1, r_2 \leftarrow \{0,1\}^* \\ (\mathsf{rt}, \pi) \leftarrow \mathcal{A}_2(h, q, (\sigma, r_1, r_2), \mathsf{st}) \\ \{(\mathsf{st}_i, Q_i)\}_{i\in[\lambda^2]} \leftarrow \mathsf{LDTest.Q}^{\otimes\lambda^2}(r_1) \\ I := \mathsf{wTHDel.Query}_2(L_{\mathsf{DB}}, \sigma, \pi) \\ \forall v \in I, (\mathsf{st}_v, Q_v) := \mathsf{SelfCorr}'.\mathsf{Q}(v; r_2) \\ \text{Obtain } \widehat{\mathsf{DB}}' \text{ by global self correction to } \widehat{\mathsf{DB}} \\ \widetilde{\mathsf{DB}} \leftarrow \mathsf{wTHDel.Decode}(\widehat{\mathsf{DB}}', L_{\mathsf{DB}}) \end{array}$$

(In the above experiment, $\mathsf{LDTest.Q}^{\otimes\lambda^2}(r_1)$ is the λ^2-repetition of $\mathsf{LDTest.Q}$.) Since the above experiment is identical with the experiment considered in (1), we obtain the following from (1) and our construction of Ver.

$$\Pr\left[\begin{array}{l} \mathsf{rt} \neq \mathsf{TreeHash}_h(\widetilde{\mathsf{DB}}) \\ \wedge\ \forall i \in [\lambda^2], \mathsf{LDTest.D}(\mathsf{st}_i, \widehat{\mathsf{DB}}|_{Q_i}) = 1 \\ \wedge\ \mathsf{wTHDel.Ver}^{\widehat{\mathsf{DB}}''}(L_{\mathsf{DB}}, \langle M_h, t_{L_{\mathsf{DB}}}, \mathsf{rt}\rangle, q, \sigma, \pi) = 1 \end{array}\right] \geq \frac{1}{p(\gamma(\lambda))} \ ,$$

where $\widehat{\mathsf{DB}}''$ is defined by $\widehat{\mathsf{DB}}''(v) := \mathsf{SelfCorr}'.\mathsf{Rec}(\mathsf{st}_v, \widehat{\mathsf{DB}}|_{Q_v})$ for $\forall v \in I$. Then, since $\widehat{\mathsf{DB}}$ must be $1/\lambda$-close

to a degree-$m(|\mathbb{H}|-1)$ polynomial when the λ^2-repetition of LDTest passes (Corollary 2), we have

$$\Pr\left[\begin{array}{l}\mathsf{rt} \neq \mathsf{TreeHash}_h(\widetilde{\mathsf{DB}})\\ \wedge\ \widehat{\mathsf{DB}} \text{ is } 1/\lambda\text{-close to a degree-}m(|\mathbb{H}|-1) \text{ polynomial}\\ \wedge\ \mathsf{wTHDel.Ver}^{\widehat{\mathsf{DB}}''}(L_{\mathsf{DB}}, \langle M_h, t_{L_{\mathsf{DB}}}, \mathsf{rt}\rangle, q, \sigma, \pi) = 1\end{array}\right] \geq \frac{1}{p(\gamma(\lambda))} - \mathsf{negl}(\gamma(\lambda)) \ .$$

Then, since $\widehat{\mathsf{DB}}'$ is obtained by applying global self correction to $\widehat{\mathsf{DB}}$, when $\widehat{\mathsf{DB}}$ is $1/\lambda$-close to a degree-$m(|\mathbb{H}|-1)$ polynomial, $\widehat{\mathsf{DB}}'$ must be such a polynomial (Corollary 2), and it holds $\mathsf{wTHDel.Valid}(\widehat{\mathsf{DB}}', L_{\mathsf{DB}}) = 1$ from the definition of wTHDel.Valid. Thus, we have

$$\Pr\left[\begin{array}{l}\mathsf{rt} \neq \mathsf{TreeHash}_h(\widetilde{\mathsf{DB}})\\ \wedge\ \mathsf{wTHDel.Valid}(\widehat{\mathsf{DB}}', L_{\mathsf{DB}}) = 1\\ \wedge\ \widehat{\mathsf{DB}} \text{ is } 1/\lambda\text{-close to } \widehat{\mathsf{DB}}'\\ \wedge\ \mathsf{wTHDel.Ver}^{\widehat{\mathsf{DB}}''}(L_{\mathsf{DB}}, \langle M_h, t_{L_{\mathsf{DB}}}, \mathsf{rt}\rangle, q, \sigma, \pi) = 1\end{array}\right] \geq \frac{1}{p(\gamma(\lambda))} - \mathsf{negl}(\gamma(\lambda)) \ .$$

Then, since $\widehat{\mathsf{DB}}''$ (which is given to wTHDel.Ver in the above expression) is defined by $\widehat{\mathsf{DB}}''(v) := \mathsf{SelfCorr}'.\mathsf{Rec}(\mathsf{st}_v, \widehat{\mathsf{DB}}|_{Q_v})$ for $\forall v \in I$, when $\widehat{\mathsf{DB}}$ is $1/\lambda$-close to $\widehat{\mathsf{DB}}'$, we have $\widehat{\mathsf{DB}}'' = \widehat{\mathsf{DB}}'|_I$ from the security of $\mathsf{SelfCorr}'$ (Corollary 2). Thus, we have

$$\Pr\left[\begin{array}{l}\mathsf{rt} \neq \mathsf{TreeHash}_h(\widetilde{\mathsf{DB}})\\ \wedge\ \mathsf{wTHDel.Valid}(\widehat{\mathsf{DB}}', L_{\mathsf{DB}}) = 1\\ \wedge\ \mathsf{wTHDel.Ver}^{\widehat{\mathsf{DB}}'|_I}(L_{\mathsf{DB}}, \langle M_h, t_{L_{\mathsf{DB}}}, \mathsf{rt}\rangle, q, \sigma, \pi) = 1\end{array}\right] \geq \frac{1}{p(\gamma(\lambda))} - \mathsf{negl}(\gamma(\lambda)) \ . \tag{2}$$

We can rewrite (2) as follows by rewriting the above-specified experiment by using $(\mathcal{B}_1, \mathcal{B}_2)$ and removing unnecessary parts.

$$\Pr\left[\begin{array}{l}\mathsf{rt} \neq \mathsf{TreeHash}_h(\widetilde{\mathsf{DB}})\\ \wedge\ \mathsf{wTHDel.Valid}(\widehat{\mathsf{DB}}', L_{\mathsf{DB}}) = 1\\ \wedge\ \mathsf{wTHDel.Ver}^{\widehat{\mathsf{DB}}'|_I}(L_{\mathsf{DB}}, \langle M_h, t_{L_{\mathsf{DB}}}, \mathsf{rt}\rangle, q, \sigma, \pi) = 1\end{array} \middle| \begin{array}{l}(\widehat{\mathsf{DB}}', L_{\mathsf{DB}}, \mathsf{st}) \leftarrow \mathcal{B}_1(1^\lambda, z_\lambda)\\ (q, \sigma) \leftarrow \mathsf{wTHDel.Query}_1(1^\lambda)\\ (\mathsf{rt}, \pi) \leftarrow \mathcal{B}_2(h, q, \sigma, \mathsf{st})\\ I := \mathsf{wTHDel.Query}_2(L_{\mathsf{DB}}, \sigma, \pi)\\ \widetilde{\mathsf{DB}} \leftarrow \mathsf{wTHDel.Decode}(\widehat{\mathsf{DB}}', L_{\mathsf{DB}})\end{array}\right] \geq \frac{1}{p(\gamma(\lambda))} - \mathsf{negl}(\gamma(\lambda)) \ .$$

Thus, the pair $(\mathcal{B}_1, \mathcal{B}_2)$ indeed breaks the soundness of wTHDel. This completes the proof of Lemma 1.

5 Public-Coin Oracle Memory Delegation

In this section, we construct a public-coin oracle memory delegation scheme.

Lemma 6. *Assume the sub-exponential hardness of the LWE assumption. Then, there exists a public-coin 2-round oracle memory delegation scheme with γ-soundness for computation-time bound γ for any (sufficiently small) super-polynomial function γ (e.g., $\gamma(\lambda) = O(\lambda^{\log\log\lambda})$).*

As explained in Section 2, our strategy is to combine our tree-hash memory delegation scheme (Lemma 1) with the RAM delegation scheme of Choudhuri, Jain, and Jin [CJJ22]. Therefore, we start by recalling the definition of RAM delegation schemes and the scheme of Choudhuri et al. [CJJ22].

5.1 Preliminary: RAM delegation

We recall the definition of publicly verifiable non-interactive RAM delegation schemes from [KPY19, CJJ22]. (We straightforwardly generalize the definition to consider security against slightly super-polynomial-time adversaries.)

A RAM machine R with word size ℓ is modeled as a deterministic machine with random access to a memory of at most 2^ℓ bits and a local state of $O(\ell)$ bits. At every step, the machine reads or writes a single memory bit and updates its state. For simplicity, we use the security parameter λ as the word size. Also, for convenience, we consider a slightly more general model than [KPY19, CJJ22] and think of a RAM machine that has access to a memory of at most 2^ℓ bits and additionally takes a short input. (In [KPY19, CJJ22], a RAM machine has access to a memory of (exactly) 2^ℓ bits and takes no input other than the memory and the initial local state.) In this paper, the memory and state of a RAM machine at a given time-step are referred to as its *memory-state pair*.[20] For any RAM machine R, let U_R denote the language such that $(\ell, x, \mathsf{ms}, \mathsf{ms}', T) \in U_R$ if and only if R with word size ℓ and on input x transitions from memory-state pair ms to memory-state pair ms' in T steps.

Definition 10. *For any RAM machine R, a* publicly verifiable non-interactive RAM delegation scheme *for R consists of four algorithms* $(\mathsf{Setup}, \mathsf{Mem}, \mathsf{Prove}, \mathsf{Ver})$ *that have the following syntax.*

Syntax.

- $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\lambda, T)$*:* Setup *is a probabilistic algorithm that takes as input a security parameter 1^λ and a time bound T, and it outputs a triple of public keys: a prover key* pk*, a verifier key* vk*, and a digest key* dk*.*
- $\mathsf{digest} := \mathsf{Mem}(\mathsf{dk}, \mathsf{ms})$*:* Mem *is a deterministic algorithm that takes as input a digest key* dk *and a memory-state pair* ms*, and it outputs a digest* digest *of the memory-state pair.*
- $\pi := \mathsf{Prove}(\mathsf{pk}, x, \mathsf{ms}, \mathsf{ms}')$*:* Prove *is a deterministic algorithm that takes as input a prover key* pk*, an input x to R, source and destination memory-state pairs* $\mathsf{ms}, \mathsf{ms}'$*, and it outputs a proof π.*
- $b := \mathsf{Ver}(\mathsf{vk}, x, \mathsf{digest}, \mathsf{digest}', \pi)$*:* Ver *is a deterministic algorithm that takes as input a verifier key* vk*, an input x to R, source and destination digests* $\mathsf{digest}, \mathsf{digest}'$*, and a proof π, and it outputs a bit b.*

Efficiency. *For any functions $T_{\mathsf{Setup}} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and $L_\pi : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, a publicly verifiable non-interactive RAM delegation scheme is said to have* setup time T_{Setup} and proof length L_π *if it satisfies the following: For every $\lambda, T \in \mathbb{N}$ such that $T \leq 2^\lambda$ and for every $x, \mathsf{ms}, \mathsf{ms}' \in \{0,1\}^*$ such that $(\lambda, x, \mathsf{ms}, \mathsf{ms}', T) \in U_R$:*

- $\mathsf{Setup}(1^\lambda, T)$ *runs in time $T_{\mathsf{Setup}}(\lambda, T)$.*
- $\mathsf{Mem}(\mathsf{dk}, \mathsf{ms})$ *runs in time $|\mathsf{ms}| \cdot \mathsf{poly}(\lambda)$ and outputs a digest of length λ.*
- $\mathsf{Prove}(\mathsf{pk}, x, \mathsf{ms}, \mathsf{ms}')$ *runs in time $\mathsf{poly}(\lambda, T, |x|, |\mathsf{ms}|)$ and outputs a proof of length $L_\pi(\lambda, T, |x|)$.*
- $\mathsf{Ver}(\mathsf{vk}, x, \mathsf{digest}, \mathsf{digest}', \pi)$ *runs in time $O(L_\pi(\lambda, T, |x|)) + \mathsf{poly}(\lambda, |x|)$.*

Security. *For any function $\gamma : \mathbb{N} \to \mathbb{N}$, a publicly verifiable non-interactive RAM delegation scheme is called* γ-sound *if it satisfies the following.*

- ***Correctness.*** *For every $\lambda, T \in \mathbb{N}$ such that $T \leq 2^\lambda$ and for every $x, \mathsf{ms}, \mathsf{ms}' \in \{0,1\}^*$ such that $(\lambda, x, \mathsf{ms}, \mathsf{ms}', T) \in U_R$,*

$$\Pr\left[\mathsf{Ver}(\mathsf{vk}, x, \mathsf{digest}, \mathsf{digest}', \pi) = 1 \;\middle|\; \begin{array}{l}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \mathsf{digest} := \mathsf{Mem}(\mathsf{dk}, \mathsf{ms}) \\ \mathsf{digest}' := \mathsf{Mem}(\mathsf{dk}, \mathsf{ms}') \\ \pi := \mathsf{Prove}(\mathsf{pk}, x, \mathsf{ms}, \mathsf{ms}')\end{array}\right] = 1 \ .$$

- ***γ-collision resistance.*** *For every probabilistic $\gamma^{O(1)}$-time adversary $\mathcal{A}$ and every sequence of polynomial-size non-uniform advise $\{z_\lambda\}_{\lambda \in \mathbb{N}}$, there exists a negligible function* negl *such that for every $\lambda \in \mathbb{N}$ and $T \leq \gamma(\lambda)$,*

$$\Pr\left[\begin{array}{l}\mathsf{ms} \neq \mathsf{ms}' \\ \wedge\, \mathsf{Mem}(\mathsf{dk}, \mathsf{ms}) = \mathsf{Mem}(\mathsf{dk}, \mathsf{ms}')\end{array} \;\middle|\; \begin{array}{l}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\lambda, T) \\ (\mathsf{ms}, \mathsf{ms}') \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}, z_\lambda)\end{array}\right] \leq \mathsf{negl}(\gamma(\lambda)) \ .$$

[20] Unlike [KPY19, CJJ22], we refrain from using the term "configuration" to refer to the memory and state since we allow RAM machines to additionally have inputs.

- ***γ-soundness.*** *For every probabilistic $\gamma^{O(1)}$-time adversary $\mathcal{A}$ and every sequence of polynomial-size non-uniform advise $\{z_\lambda\}_{\lambda\in\mathbb{N}}$, there exists a negligible function* negl *such that for every $\lambda \in \mathbb{N}$ and $T \leq \gamma(\lambda)$,*

$$\Pr\left[\begin{array}{l|l} \mathsf{Ver}(\mathsf{vk}, x, \mathsf{digest}, \mathsf{digest}', \pi) = 1 & \\ \wedge\ (\lambda, x, \mathsf{ms}, \mathsf{ms}', T) \in U_R & (\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \wedge\ \mathsf{digest} = \mathsf{Mem}(\mathsf{dk}, \mathsf{ms}) & (x, \mathsf{ms}, \mathsf{ms}', \mathsf{digest}, \mathsf{digest}', \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}, z_\lambda) \\ \wedge\ \mathsf{digest}' \neq \mathsf{Mem}(\mathsf{dk}, \mathsf{ms}') & \end{array}\right] \leq \mathsf{negl}(\gamma(\lambda)) \ .$$

A publicly verifiable non-interactive RAM delegation scheme is called public coin *if the setup algorithm* Setup *is public coin, i.e., it just outputs a triple of strings that are sampled uniformly randomly.*

We use the following prior result [CJJ22] with straightforward adaptation.

Theorem 1. *Let γ be any (sufficiently small) super-polynomial function (e.g., $\gamma(\lambda) = \lambda^{\log\log\lambda}$), and assume the γ-hardness of the LWE assumption. Then, for any RAM machine R, there exists a publicly verifiable non-interactive RAM delegation scheme with γ-soundness, where the setup time is $T_{\mathsf{Setup}}(\lambda, T) = \mathsf{poly}(\lambda, \log T)$ and proof length is $L_\pi(\lambda, T, |x|) = \mathsf{poly}(\lambda, \log T, |x|)$. Furthermore, this scheme is public coin, and (i) the setup algorithm* Setup *outputs a hash function as a digest key, where the hash function is sampled from a collision-resistant hash function family that is independent of the computation-time bound T, and (ii) the digest algorithm* Mem*, on input a digest key* dk *and a memory-state pair* $\mathsf{ms} = (\mathsf{DB}, \mathsf{st})$*, outputs a triple* $\mathsf{digest} = (\mathsf{st}, \mathsf{rt}, |\mathsf{DB}|)$ *that consists of the local state* st*, the tree-hash* $\mathsf{rt} := \mathsf{TreeHash}_{\mathsf{dk}}(\mathsf{DB})$ *of the memory* DB*, and the memory length* $|\mathsf{DB}|$*, where the tree-hash is computed by using the digest key* dk *as a hash function.*

Two remarks about Theorem 1 are given below.

1. The first part of Theorem 1 differs from what is shown in [CJJ22] in that (i) RAM machines are defined in a slightly more general model where a RAM machine has access to a memory of at most 2^ℓ bits (rather than exactly 2^ℓ bits) and takes a short input in addition to a local state and a memory, and (ii) soundness is required to hold for $\lambda^{\omega(1)}$-time adversaries and $\lambda^{\omega(1)}$-time RAM computations. (In [CJJ22], soundness is shown for polynomial-time adversaries and polynomial-time RAM computations under the polynomial hardness of the LWE assumption.) Still, the first part of Theorem 1 can be easily obtained from [CJJ22]. In particular, the analysis given in [CJJ22] can be easily extended (i) for memories of at most 2^ℓ bits by appending the length $|\mathsf{DB}|$ of the memory to the digest digest so that the verification algorithm Verify can learn $|\mathsf{DB}|$, (ii) for RAM machines that take additional short inputs by allowing the proof length to be polynomial in the input length (but still polylogarithmic in the computation-time bound T),[21] and (iii) for $\lambda^{\omega(1)}$-time adversaries and $\lambda^{\omega(1)}$-time RAM computations by assuming the $\lambda^{\omega(1)}$-hardness of the LWE assumption.

2. Regarding the furthermore part of Theorem 1, the public-coin property is implicitly mentioned in [CJJ22]. In particular, it is mentioned that the RAM delegation scheme (or more precisely its main component) works in the common random string model rather than the common reference string model, implying that its setup algorithm Setup outputs uniformly random strings.[22] The properties of Setup and Mem can be easily verified by inspecting the scheme description given in [CJJ21, Figure 5].[23]

5.2 Proof of Lemma 6

Fix any sufficiently small super-polynomial function γ. Let R be the following RAM machine.

[21] For those who are familiar with the RAM delegation of [CJJ22], we note that we allow the statements of the batch-NP argument to contain the input of the RAM machine.

[22] Technically, the public-coin property can be verified by observing that under the LWE assumption, all the components of the scheme of [CJJ22] can be made public coin by using, e.g., an FHE scheme with pseudorandom public keys and ciphertexts.

[23] Actually, Mem in [CJJ21, Figure 5] outputs a pair $\mathsf{digest} = (\mathsf{st}, \mathsf{rt})$, but as noted above, we consider an extended version that additionally includes $|\mathsf{DB}|$ in digest.

- R is given as input a description of a Turing machine M and given as memory a string DB. Then, R internally executes $M(\mathsf{DB})$.[24] When M terminates, R writes (y, t) at the beginning of the memory and terminates, where y is the output of M and t is the running time of M.

Without loss of generality, we assume that there exists a (non-decreasing) polynomial poly_R such that when the running time of $M(\mathsf{DB})$ is t, the running time of $R^{\mathsf{DB}}(M)$ is $\mathsf{poly}_R(t)$, and $R^{\mathsf{DB}}(M)$ only reads and writes the first $\mathsf{poly}_R(t)$ bits of DB (hence, we assume that DB is of length $\mathsf{poly}_R(t)$). Let $\mathsf{RDel} = (\mathsf{RDel.Setup}, \mathsf{RDel.Mem}, \mathsf{RDel.Prove}, \mathsf{RDel.Ver})$ be the public-coin non-interactive RAM delegation scheme given by Theorem 1 for the RAM machine R with γ-soundness. Recall that $\mathsf{RDel.Setup}$ outputs as a digest key a hash function that is sampled from a collision-resistant hash family (see the furthermore part of Theorem 1). We can assume that this hash function family is polylogarithmic depth (and secure against $\lambda^{\log \lambda}$-time adversaries) since we assume the sub-exponential hardness of the LWE assumption, which implies the existence of sub-exponentially secure collision-resistant hash function families (see Remark 2 in Section 3.2.1). For this hash function family, let $\mathsf{THDel} = (\mathsf{THDel.Mem}, \mathsf{THDel.Query}_1, \mathsf{THDel.Query}_2, \mathsf{THDel.Prove}, \mathsf{THDel.Ver})$ be any public-coin 2-round tree-hash oracle memory delegation scheme with γ-soundness (e.g., the one given in Lemma 1).

Remark 7 (Simplified syntax of $\mathsf{RDel.Setup}$). Without loss of generality, we can think as if $\mathsf{RDel.Setup}$ only takes 1^λ as input (rather than $(1^\lambda, T)$ as defined in Definition 10). This is because the output length of $\mathsf{RDel.Setup}(1^\lambda, T)$ is bounded by $\mathsf{poly}_{\mathsf{Setup}}(\lambda)$ for any $T \le 2^\lambda$ for a fixed polynomial $\mathsf{poly}_{\mathsf{Setup}}$. (Recall that the setup time is $T_{\mathsf{Setup}}(\lambda, T) = \mathsf{poly}(\lambda, \log T)$.) Indeed, in this case, we can assume without loss of generality that $\mathsf{RDel.Setup}$ outputs a triple of sufficiently long random strings $(\bar{\mathsf{pk}}, \bar{\mathsf{vk}}, \bar{\mathsf{dk}})$ (whose length is longer than $\mathsf{poly}_{\mathsf{Setup}}(\lambda)$), and $\mathsf{RDel.Mem}$, $\mathsf{RDel.Prove}$, and $\mathsf{RDel.Ver}$ use prefixes of $\bar{\mathsf{pk}}$, $\bar{\mathsf{vk}}$, and $\bar{\mathsf{dk}}$ as the actual keys.[25] Thus, in the following, we use this simplified syntax of $\mathsf{RDel.Setup}$. Also, since $\mathsf{RDel.Mem}$, $\mathsf{RDel.Prove}$, and $\mathsf{RDel.Ver}$ need to know T to determine the lengths of the actual keys, we write them as $\mathsf{RDel.Mem}_T$, $\mathsf{RDel.Prove}_T$, and $\mathsf{RDel.Ver}_T$ to make it explicit what value of T they depend on. ◊

Our oracle memory delegation scheme $\mathsf{ODel} = (\mathsf{Mem}, \mathsf{Query}_1, \mathsf{Prove}, \mathsf{Query}_2, \mathsf{Ver})$ is given in Algorithm 4. (A high-level idea is explained in Section 2.) Since THDel and RDel are public coin, ODel is also public coin. Completeness holds due to the furthermore part of Theorem 1 (in particular, the part about the digest algorithm $\mathsf{RDel.Mem}$).[26] The efficiency condition of ODel follows from the efficiency conditions of THDel and RDel. (In particular, Prove runs in time $\mathsf{poly}(\lambda, t, T) = \mathsf{poly}(\lambda, t)$, and Ver runs in time $\mathsf{poly}(\lambda, \log t, \log T) = \mathsf{poly}(\lambda)$.) In the following, we focus on soundness.

Assume for contradiction that soundness does not hold, i.e., there exist a pair of probabilistic $\gamma^{O(1)}$-time adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, a sequence of polynomial-size advice $\{z_\lambda\}_{\lambda\in\mathbb{N}}$, and a polynomial p such that for infinitely many $\lambda \in \mathbb{N}$, there exists $t \le \gamma^{O(1)}(\lambda)$ such that

$$\Pr\left[\begin{array}{l|l} y_0 \neq y_1 & (\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, M, y_0, y_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ \wedge\, \mathsf{Ver}^{\widehat{\mathsf{DB}}|_{I_0}}(L_{\mathsf{DB}}, \langle M, t, y_0\rangle, q, \sigma, \pi_0) = 1 & (q, \sigma) \leftarrow \mathsf{Query}_1(1^\lambda) \\ \wedge\, \mathsf{Ver}^{\widehat{\mathsf{DB}}|_{I_1}}(L_{\mathsf{DB}}, \langle M, t, y_1\rangle, q, \sigma, \pi_1) = 1 & (\pi_0, \pi_1) \leftarrow \mathcal{A}_2(q, \sigma, \mathsf{st}) \\ & I_0 := \mathsf{Query}_2(L_{\mathsf{DB}}, \sigma, \pi_0) \\ & I_1 := \mathsf{Query}_2(L_{\mathsf{DB}}, \sigma, \pi_1) \end{array}\right] \ge \frac{1}{p(\gamma(\lambda))} \ . \tag{3}$$

We use $(\mathcal{A}_1, \mathcal{A}_2)$ to obtain a $\gamma^{O(1)}$-time adversary $\mathcal{B}$ that breaks the soundness of RDel. A high-level strategy is as follows. As defined in Definition 10, breaking the soundness of RDel requires generating an input x, source and destination memory-state pairs $(\mathsf{ms}, \mathsf{ms}')$, source and destination digests $(\mathsf{digest}, \mathsf{digest}')$, and a proof π such that (i) π is accepting w.r.t. $(\mathsf{digest}, \mathsf{digest}')$, (ii) ms' is the correct destination memory-state pair that can be obtained by running R starting from input x and memory-state pair ms, (iii) digest is the correct digest of ms, but (iv) digest' is not the correct digest of ms'. Now, suppose the adversary pair $(\mathcal{A}_1, \mathcal{A}_2)$ generates two proofs of ODel that are accepting w.r.t. a single encoded memory $\widehat{\mathsf{DB}}$ and two different outputs as shown in (3).

[24] R emulates the working tape of M by writing it to the memory DB. (It is assumed that DB contains a padding string as a suffix so that it is long enough for the emulation of the working tape. It is also assumed that M is designed to ignore this padding part of DB.)

[25] Recall that $\mathsf{RDel.Setup}$ is public coin.

[26] Formally, completeness holds under a slightly modified definition where for each $\langle M, t, y\rangle \in \{0,1\}^{\mathsf{poly}(\lambda)}$, we only consider a memory DB that contains a padding string as a suffix so that it is of length $T := \mathsf{poly}_R(t)$ (cf. Footnote 24).

Algorithm 4 Public-coin oracle memory delegation scheme ODel.

- $\widehat{\mathsf{DB}} \coloneqq \mathsf{Mem}(1^\lambda, \mathsf{DB})$:
  1. Output $\widehat{\mathsf{DB}} \coloneqq \mathsf{THDel.Mem}(1^\lambda, \mathsf{DB})$.
- $(q, \sigma) \leftarrow \mathsf{Query}_1(1^\lambda)$:
  1. Run $(q_{\mathsf{THDel}}, \sigma_{\mathsf{THDel}}) \leftarrow \mathsf{THDel.Query}_1(1^\lambda)$.
  2. Run $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.Setup}(1^\lambda)$.
  3. Output $q \coloneqq (q_{\mathsf{THDel}}, \mathsf{pk}, \mathsf{vk}, \mathsf{dk})$ and $\sigma \coloneqq \sigma_{\mathsf{THDel}}$.
- $\pi \coloneqq \mathsf{Prove}(\mathsf{DB}, \langle M, t, y\rangle, q)$:
  1. Parse q as $(q_{\mathsf{THDel}}, \mathsf{pk}, \mathsf{vk}, \mathsf{dk})$, and let $T \coloneqq \mathsf{poly}_R(t)$.
  2. Run $R^{\mathsf{DB}}(M)$. If $R^{\mathsf{DB}}(M)$ does not terminate in T steps, abort. Otherwise, let DB' denote the content of the memory at the termination of $R^{\mathsf{DB}}(M)$.
  3. Run $\pi_{\mathsf{RDel}} \coloneqq \mathsf{RDel.Prove}_T(\mathsf{pk}, M, \mathsf{ms}, \mathsf{ms}')$ for $\mathsf{ms} \coloneqq (\mathsf{DB}, \mathsf{st}_{\text{START}})$ and $\mathsf{ms}' \coloneqq (\mathsf{DB}', \mathsf{st}_{\text{END}})$, where $\mathsf{st}_{\text{START}}$ and st_{END} are the initial and the terminating states of R.
  4. Run $\pi_{\mathsf{THDel}} \coloneqq \mathsf{THDel.Prove}(\mathsf{DB}, \langle M_h, t_{|\mathsf{DB}|}, \mathsf{rt}\rangle, q_{\mathsf{THDel}})$, where M_h and $t_{|\mathsf{DB}|}$ are defined as in Definition 9 for the hash function $h \coloneqq \mathsf{dk}$ (more precisely, h is a prefix of dk; cf. Remark 7) and rt is the tree-hash that is obtained by $\mathsf{rt} \coloneqq \mathsf{TreeHash}_{\mathsf{dk}}(\mathsf{DB})$.
  5. Let (y', t') be the prefix of DB' that R wrote before the termination, rt' be the tree-hash that is obtained by $\mathsf{rt}' \coloneqq \mathsf{TreeHash}_{\mathsf{dk}}(\mathsf{DB}')$, and π_{TreeHash} be the local opening for (y', t') w.r.t. rt'.
  6. Output $\pi \coloneqq (\mathsf{rt}, \mathsf{rt}', (y', t'), \pi_{\mathsf{RDel}}, \pi_{\mathsf{THDel}}, \pi_{\mathsf{TreeHash}})$.
- $I \coloneqq \mathsf{Query}_2(|\mathsf{DB}|, \sigma, \pi)$:
  1. Parse π as $(\mathsf{rt}, \mathsf{rt}', (y', t'), \pi_{\mathsf{RDel}}, \pi_{\mathsf{THDel}}, \pi_{\mathsf{TreeHash}})$.
  2. Output $I \coloneqq \mathsf{THDel.Query}_2(|\mathsf{DB}|, \sigma, \pi_{\mathsf{THDel}})$.
- $b \coloneqq \mathsf{Ver}^{\widehat{\mathsf{DB}}|_I}(|\mathsf{DB}|, \langle M, t, y\rangle, q, \sigma, \pi)$:
  1. Parse q as $(q_{\mathsf{THDel}}, \mathsf{pk}, \mathsf{vk}, \mathsf{dk})$ and π as $(\mathsf{rt}, \mathsf{rt}', (y', t'), \pi_{\mathsf{RDel}}, \pi_{\mathsf{THDel}}, \pi_{\mathsf{TreeHash}})$. Also, obtain T as in Prove, and abort if $|\mathsf{DB}| \neq T$. Let $\mathsf{digest} \coloneqq (\mathsf{st}_{\text{START}}, \mathsf{rt}, T)$ and $\mathsf{digest}' \coloneqq (\mathsf{st}_{\text{END}}, \mathsf{rt}', T)$.
  2. Output 1 if all of the following hold.
     (a) $y = y'$ and $t' \leq t$.
     (b) $\mathsf{RDel.Ver}_T(\mathsf{vk}, M, \mathsf{digest}, \mathsf{digest}', \pi_{\mathsf{RDel}}) = 1$.
     (c) $\mathsf{THDel.Ver}^{\widehat{\mathsf{DB}}|_I}(|\mathsf{DB}|, \langle M_h, t_{|\mathsf{DB}|}, \mathsf{rt}\rangle, q_{\mathsf{THDel}}, \sigma, \pi_{\mathsf{THDel}}) = 1$, where $h \coloneqq \mathsf{dk}$.
     (d) π_{TreeHash} is a valid local opening for (y', t') w.r.t. rt'.

     If any of the above does not hold, output 0.

Then, at least one of the proofs must be accepting w.r.t. an incorrect output (i.e., an output that differs from the correct output that is obtained based on $\widehat{\mathsf{DB}}$). In that case, one of the proofs must contain a proof of RDel that is accepting w.r.t. an incorrect destination digest (i.e., a digest that differs from the correct destination digest that is obtained based on $\widehat{\mathsf{DB}}$). We consider an adversary that internally runs $(\mathcal{A}_1, \mathcal{A}_2)$ and outputs such a proof.

Formally, we obtain the adversary $\mathcal{B}$ as follows. Let $\mathsf{Decode}_{\mathsf{THDel}}$ be the algorithm that is guaranteed to exist by the soundness of THDel (cf. Definition 9). Then, for any $\lambda \in \mathbb{N}$ and $t \leq \gamma^{O(1)}(\lambda)$, the adversary $\mathcal{B}$ is described in Algorithm 5. Note that $\mathcal{B}$ runs in time $\gamma^{O(1)}(\lambda)$ since $\mathcal{A}_1$ and $\mathcal{A}_2$ are $\gamma^{O(1)}$-time adversaries and $\mathcal{B}$ runs $R^{\mathsf{DB}}(M)$ at most $T = \mathsf{poly}_R(t) \leq \mathsf{poly}(\gamma(\lambda))$ steps.

Algorithm 5 Adversary $\mathcal{B}$ against the soundness of RDel.

On input $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk})$, do the following. Let $T \coloneqq \mathsf{poly}_R(t)$.

1. Run $(\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, M, y_0, y_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda)$.
2. Run $(q_{\mathsf{THDel}}, \sigma_{\mathsf{THDel}}) \leftarrow \mathsf{THDel.Query}_1(1^\lambda)$.
3. Run $(\pi_0, \pi_1) \leftarrow \mathcal{A}_2(q, \sigma, \mathsf{st})$, where $q \coloneqq (q_{\mathsf{THDel}}, \mathsf{pk}, \mathsf{vk}, \mathsf{dk})$ and $\sigma \coloneqq \sigma_{\mathsf{THDel}}$.
4. For each $b \in \{0,1\}$, parse π_b as $(\mathsf{rt}_b, \mathsf{rt}'_b, (y'_b, t'_b), \pi_{\mathsf{RDel},b}, \pi_{\mathsf{THDel},b}, \pi_{\mathsf{TreeHash},b})$, and let $\mathsf{digest}_b \coloneqq (\mathsf{st}_{\mathrm{START}}, \mathsf{rt}_b, T)$ and $\mathsf{digest}'_b \coloneqq (\mathsf{st}_{\mathrm{END}}, \mathsf{rt}'_b, T)$.
5. Find $b^* \in \{0,1\}$ that satisfies all of the following.
   (a) $\mathsf{RDel.Ver}_T(\mathsf{vk}, M, \mathsf{digest}_{b^*}, \mathsf{digest}'_{b^*}, \pi_{\mathsf{RDel},b^*}) = 1$.
   (b) $\mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}) = \mathsf{digest}_{b^*}$, where $\mathsf{ms} \coloneqq (\widetilde{\mathsf{DB}}, \mathsf{st}_{\mathrm{START}})$ for $\widetilde{\mathsf{DB}} \leftarrow \mathsf{Decode}_{\mathsf{THDel}}(\widehat{\mathsf{DB}}, L_{\mathsf{DB}})$.
   (c) $\mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}') \neq \mathsf{digest}'_{b^*}$, where ms' is the memory-state pair of R after T steps starting from input M and memory-state pair ms.

   If such b^* exists, output $(M, \mathsf{ms}, \mathsf{ms}', \mathsf{digest}_{b^*}, \mathsf{digest}'_{b^*}, \pi_{\mathsf{RDel},b^*})$. Otherwise, abort.

Let us see that $\mathcal{B}$ indeed breaks the soundness of RDel. Fix any $\lambda \in \mathbb{N}$ and $t \leq \gamma^{O(1)}(\lambda)$ for which we have (3). Let $T \coloneqq \mathsf{poly}_R(t)$. We start by giving a sequence of claims about various values that $\mathcal{B}$ computes. The first claim says that in $\mathcal{B}$, the internally emulated $(\mathcal{A}_1, \mathcal{A}_2)$ succeed with non-negligible probability as shown in (3).

Claim 1. *In an execution of* $\mathcal{B}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk})$ *for* $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.Setup}(1^\lambda)$,

$$\Pr\left[\begin{array}{l} y'_0 \neq y'_1 \\ \wedge\ L_{\mathsf{DB}} = T \\ \wedge\ \forall b \in \{0,1\} : \mathsf{RDel.Ver}_T(\mathsf{vk}, M, \mathsf{digest}_b, \mathsf{digest}'_b, \pi_{\mathsf{RDel},b}) = 1 \\ \wedge\ \forall b \in \{0,1\} : \mathsf{THDel.Ver}^{\widehat{\mathsf{DB}}|_{I_b}}(L_{\mathsf{DB}}, \langle M_{\mathsf{dk}}, t_{L_{\mathsf{DB}}}, \mathsf{rt}_b\rangle, q_{\mathsf{THDel}}, \sigma, \pi_{\mathsf{THDel},b}) = 1 \\ \wedge\ \forall b \in \{0,1\} : \pi_{\mathsf{TreeHash},b} \textit{ is a valid opening for } (y'_b, t'_b) \textit{ w.r.t. } \mathsf{rt}'_b \end{array}\right] \geq \frac{1}{p(\gamma(\lambda))} \ ,$$

where $I_b \coloneqq \mathsf{THDel.Query}_2(L_{\mathsf{DB}}, \sigma, \pi_{\mathsf{THDel},b})$.

Proof. This claim follows immediately from (3) (specifically, it suffices to rewrite (3) by inlining Query_1, Query_2, and Ver). We note that when π_0 and π_1 are accepted and $y_0 \neq y_1$, we have $y'_0 \neq y'_1$ and $L_{\mathsf{DB}} = T$ since Ver checks $y_b \stackrel{?}{=} y'_b$ and $L_{\mathsf{DB}} \stackrel{?}{=} T$. □

The second claim says that in $\mathcal{B}$, if the internally emulated $(\mathcal{A}_1, \mathcal{A}_2)$ output an accepting proof $\pi_{\mathsf{THDel},b}$ of THDel, the corresponding tree-hash rt_b is correctly computed.

Claim 2. *In an execution of* $\mathcal{B}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk})$ *for* $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.Setup}(1^\lambda)$, *for each* $b \in \{0,1\}$,

$$\Pr\left[\begin{array}{l} \mathsf{rt} \neq \mathsf{TreeHash}_{\mathsf{dk}}(\widetilde{\mathsf{DB}}) \\ \wedge\ \mathsf{THDel.Ver}^{\widehat{\mathsf{DB}}|_{I_b}}(L_{\mathsf{DB}}, \langle M_{\mathsf{dk}}, t_{L_{\mathsf{DB}}}, \mathsf{rt}_b\rangle, q_{\mathsf{THDel}}, \sigma, \pi_{\mathsf{THDel},b}) = 1 \end{array}\right] \leq \mathsf{negl}(\gamma(\lambda)),$$

where $I_b \coloneqq \mathsf{THDel.Query}_2(L_{\mathsf{DB}}, \sigma, \pi_{\mathsf{THDel},b})$.

Proof. This claim follows immediately from the γ-soundness of THDel. □

The third claim says that in $\mathcal{B}$, if the internally emulated $(\mathcal{A}_1, \mathcal{A}_2)$ output two distinct outputs y'_0, y'_1 and the corresponding openings $\pi_{\mathsf{TreeHash},0}, \pi_{\mathsf{TreeHash},1}$ are valid, the corresponding destination digests $\mathsf{digest}'_0, \mathsf{digest}'_1$ must be distinct.

Claim 3. *In an execution of* $\mathcal{B}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk})$ *for* $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.Setup}(1^\lambda)$,

$$\Pr\left[\begin{array}{l} \mathsf{digest}'_0 = \mathsf{digest}'_1 \\ \wedge\, y'_0 \neq y'_1 \\ \wedge\, \forall b \in \{0,1\} : \pi_{\mathsf{TreeHash},b} \textit{ is a valid opening for } (y'_b, t'_b) \textit{ w.r.t. } \mathsf{rt}'_b \end{array}\right] \leq \mathsf{negl}(\gamma(\lambda)).$$

Proof. Since $\mathsf{digest}'_0 = \mathsf{digest}'_1$ implies $\mathsf{rt}'_0 = \mathsf{rt}'_1$ (recall $\mathsf{digest}'_b := (\mathsf{st}_{\mathrm{END}}, \mathsf{rt}'_b, T)$), this claim follows immediately from the binding property of tree-hashing. □

Now, we analyze $\mathcal{B}$. Combined with Claim 2 and Claim 3, Claim 1 implies the following when executing $\mathcal{B}(\mathsf{pk}, \mathsf{vk}, \mathsf{dk})$ for $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.Setup}(1^\lambda)$.

$$\Pr\left[\begin{array}{l} \mathsf{digest}'_0 \neq \mathsf{digest}'_1 \\ \wedge\, L_{\mathsf{DB}} = T \\ \wedge\, \forall b \in \{0,1\} : \mathsf{RDel.Ver}_T(\mathsf{vk}, M, \mathsf{digest}_b, \mathsf{digest}'_b, \pi_{\mathsf{RDel},b}) = 1 \\ \wedge\, \forall b \in \{0,1\} : \mathsf{rt} = \mathsf{TreeHash}_{\mathsf{dk}}(\widetilde{\mathsf{DB}}) \end{array}\right] \geq \frac{1}{p(\gamma(\lambda))} - \mathsf{negl}(\gamma(\lambda)) \ .$$

Note that $L_{\mathsf{DB}} = T \wedge \mathsf{rt} = \mathsf{TreeHash}_{\mathsf{dk}}(\widetilde{\mathsf{DB}})$ implies $\mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}) = \mathsf{digest}_b$ since $\mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}) = (\mathsf{st}_{\mathrm{START}}, \mathsf{rt}, |\widetilde{\mathsf{DB}}|) = (\mathsf{st}_{\mathrm{START}}, \mathsf{rt}, L_{\mathsf{DB}}) = (\mathsf{st}_{\mathrm{START}}, \mathsf{rt}, T) = \mathsf{digest}_b$ (the first equality holds due to the furthermore part of Theorem 1 and the second equality holds since $\widetilde{\mathsf{DB}}$ is obtained by $\mathsf{Decode}_{\mathsf{THDel}}(\widetilde{\mathsf{DB}}, L_{\mathsf{DB}})$, which outputs an L_{DB}-bit string as stated in Definition 9). Thus, we obtain

$$\Pr\left[\begin{array}{l} \mathsf{digest}'_0 \neq \mathsf{digest}'_1 \\ \wedge\, \forall b \in \{0,1\} : \mathsf{RDel.Ver}_T(\mathsf{vk}, M, \mathsf{digest}_b, \mathsf{digest}'_b, \pi_{\mathsf{RDel},b}) = 1 \\ \wedge\, \forall b \in \{0,1\} : \mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}) = \mathsf{digest}_b \end{array}\right] \geq \frac{1}{p(\gamma(\lambda))} - \mathsf{negl}(\gamma(\lambda)) \ .$$

Then, since $\mathsf{digest}'_0 \neq \mathsf{digest}'_1$ implies $\exists b^* \in \{0,1\}$ s.t. $\mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}') \neq \mathsf{digest}'_{b^*}$, we obtain

$$\Pr\left[\begin{array}{l} \exists b^* \in \{0,1\} : \\ \mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}') \neq \mathsf{digest}'_{b^*} \\ \wedge\, \mathsf{RDel.Ver}_T(\mathsf{vk}, M, \mathsf{digest}_{b^*}, \mathsf{digest}'_{b^*}, \pi_{\mathsf{RDel},b^*}) = 1 \\ \wedge\, \mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}) = \mathsf{digest}_{b^*} \end{array}\right] \geq \frac{1}{p(\gamma(\lambda))} - \mathsf{negl}(\gamma(\lambda)) \ .$$

Thus, $\mathcal{B}$ does not abort with probability at least $1/p(\gamma(\lambda)) - \mathsf{negl}(\gamma(\lambda))$. Then, since the definition of ms' guarantees $(\lambda, M, \mathsf{ms}, \mathsf{ms}', T) \in U_R$ in $\mathcal{B}$ when it does not abort, we have the following about the output $(M, \mathsf{ms}, \mathsf{ms}', \mathsf{digest}_{b^*}, \mathsf{digest}'_{b^*}, \pi_{\mathsf{RDel},b^*})$ of $\mathcal{B}$.

$$\Pr\left[\begin{array}{l} \mathsf{RDel.Ver}_T(\mathsf{vk}, M, \mathsf{digest}_{b^*}, \mathsf{digest}'_{b^*}, \pi_{\mathsf{RDel},b^*}) = 1 \\ \wedge\, (\lambda, M, \mathsf{ms}, \mathsf{ms}', T) \in U_R \\ \wedge\, \mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}) = \mathsf{digest}_{b^*} \\ \wedge\, \mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}') \neq \mathsf{digest}'_{b^*} \end{array}\right] \geq \frac{1}{p(\gamma(\lambda))} - \mathsf{negl}(\gamma(\lambda)) \ .$$

Thus, $\mathcal{B}$ breaks the γ-soundness of RDel. This completes the proof of Lemma 6.

6 Public-Coin Weak Memory Delegation

In this section, we construct a public-coin memory delegation scheme.

Lemma 7. *Assume the sub-exponential hardness of the LWE assumption, and assume the existence of a keyless weakly* (K, γ)*-collision-resistant hash function for* $K(\lambda, \zeta) = \mathsf{poly}(\lambda, \zeta)$ *and* $\gamma(\lambda) = \lambda^{\tau(\lambda)}$ *for a super-constant function* $\tau(\lambda) = \omega(1)$*. Then, there exists* $\bar{t}(\lambda) = \lambda^{\omega(1)}$ *such that there exists a two-round memory delegation scheme with weak soundness for computation-time bound* $\bar{t}$*.*

As suggested in Section 2, our strategy is to combine our oracle memory delegation scheme (Lemma 6) with the keyless multi-collision-resistant hash function with local opening of Bitansky, Kalai, and Paneth [BKP18]. Therefore, we start by recalling the definition of multi-collision-resistant hash functions with local opening and the result of Bitansky et al. [BKP18].

6.1 Preliminary: Multi-Collision-Resistant Hash with Local Opening

We recall the definition of (the keyless version of) multi-collision-resistant hash functions with local opening [BKP18]. Roughly speaking, they are hash functions that hash a long string to a short digest in such a way that any locations of the hashed string can be opened without opening the entire string. Their multi-collision resistance is formalized as an extractability property, which roughly guarantees that if an adversary makes local opening about a length-L string, we can extract a set of length-L strings such that any local opening by the adversary must agree with one of the strings in the set. (When the adversary locally opens multiple locations simultaneously, the opened values are guaranteed to agree with a single length-L string in the set.) The formal definition is given below.

Definition 11. *A* keyless multi-collision-resistant hash functions with local opening *consists of four algorithms* $(\mathsf{Hash}, \mathsf{Chal}, \mathsf{Auth}, \mathsf{Ver})$ *that have the following syntax and security.*

Syntax.

- $\mathsf{rt} \coloneqq \mathsf{Hash}(1^\lambda, x)$*:* Hash *is a deterministic polynomial-time algorithm that takes as input a security parameter* 1^λ *and an input* $x \in \{0,1\}^L$ *of length* $L \le 2^\lambda$*, and it outputs a digest* $\mathsf{rt} \in \{0,1\}^\lambda$*.*
- $\mathsf{ch} \leftarrow \mathsf{Chal}(1^\lambda, 1^\rho)$*:* Chal *is a probabilistic polynomial-time algorithm that takes as input a security parameter* 1^λ *and an opening-size parameter* 1^ρ*, and it outputs a challenge* ch*.*
- $\pi \coloneqq \mathsf{Auth}(1^\lambda, x, I, \mathsf{ch})$*:* Auth *is a deterministic polynomial-time algorithm that takes as input a security parameter* 1^λ*, an input* $x \in \{0,1\}^L$*, an index set* $I \subseteq [L]$*, and a challenge* ch*, and it outputs a proof* π*.*
- $b \coloneqq \mathsf{Ver}(1^\lambda, L, \mathsf{rt}, I, A, \mathsf{ch}, \pi)$*:* Ver *is a deterministic polynomial-time algorithm that takes as input a security parameter* 1^λ*, an input length* $L \in \mathbb{N}$*, a digest* $\mathsf{rt} \in \{0,1\}^\lambda$*, an index set* I*, an assignment* $A : I \to \{0,1\}$*, a challenge* ch*, and a proof* π*, and it outputs a bit* b*.*

Security.

- ***Correctness.*** *For every* $\lambda, \rho, L \in \mathbb{N}$ *such that* $L \le 2^\lambda$*, every* $x \in \{0,1\}^L$*, and every* $I \subseteq [L]$*,*

$$\Pr\left[\mathsf{Ver}(1^\lambda, L, \mathsf{rt}, I, A, \mathsf{ch}, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{rt} \leftarrow \mathsf{Hash}(1^\lambda, x) \\ \mathsf{ch} \leftarrow \mathsf{Chal}(1^\lambda, 1^\rho) \\ \pi \leftarrow \mathsf{Auth}(1^\lambda, x, I, \mathsf{ch}) \end{array}\right] = 1 \ .$$

- ***Succinctness.*** *For every* $\lambda, \rho, L \in \mathbb{N}$ *such that* $L \le 2^\lambda$*, every* $x \in \{0,1\}^L$*, every* $I \subseteq [L]$*, and every* $\mathsf{ch} \in \{0,1\}^*$ *that has the same length as* $\mathsf{ch}' \leftarrow \mathsf{Chal}(1^\lambda, 1^\rho)$*, the authentication algorithm* $\mathsf{Auth}(1^\lambda, x, I, \mathsf{ch})$ *outputs a proof* π *such that* $|\pi| = \mathsf{poly}(\lambda, \rho, |I|)$*.*
- K***-collision resistance for length bound*** $\bar{L}$*.* *There exists a* PPT *extractor* Ext *such that for every* PPT *adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, every sequence of polynomial-size advice* $\{z_\lambda\}_{\lambda \in \mathbb{N}}$*, every noticeable function* ε*, every function* L *such that* $L(\lambda) \le \bar{L}^{O(1)}(\lambda)$*, every sufficiently large security parameter* $\lambda \in \mathbb{N}$*, and every opening-size parameter* $\rho \le L(\lambda)$*, the following holds for* $L \coloneqq L(\lambda)$ *and* $K \coloneqq K(\lambda, |z_\lambda|, L)$*.*

$$\Pr\left[\begin{array}{l} |I| \le \rho \\ \wedge\, \mathsf{Ver}(1^\lambda, L, \mathsf{rt}, I, A, \mathsf{ch}, \pi) = 1 \\ \wedge\, A \notin \{x|_I \mid x \in S\} \end{array} \;\middle|\; \begin{array}{l} (\mathsf{rt}, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ \mathsf{ch} \leftarrow \mathsf{Chal}(1^\lambda, 1^\rho) \\ (I, A, \pi) \leftarrow \mathcal{A}_2(\mathsf{ch}, \mathsf{st}) \\ S \leftarrow \mathsf{Ext}^{\mathcal{A}_2(\cdot, \mathsf{st})}(1^\lambda, 1^\rho, 1^L, 1^K, 1^{1/\varepsilon(\lambda)}) \end{array}\right] \le \varepsilon(\lambda) \ .$$

  Furthermore, in the above experiment, Ext *always outputs a set* S *of size at most* K*.*

A multi-collision-resistant hash function with local opening is called public coin *if the query algorithm* Chal *is public coin, i.e., it just outputs a string that is sampled uniformly randomly.*

We use the following theorem, which is shown in Bitansky et al. [BKP18].

Theorem 2. *For any (arbitrarily small) $\tau(\lambda) = \omega(1)$, there exists $\bar{L}(\lambda) = \lambda^{\omega(1)}$ such that if there exists a keyless weakly (K, γ)-collision-resistant hash function for $K(\lambda, \zeta) = \mathsf{poly}(\lambda, \zeta)$ and $\gamma(\lambda) = \lambda^{\tau(\lambda)}$, there exists a public-coin[27] keyless K^{τ}-collision-resistant hash function with local opening for input-length bound $\bar{L}$.*

6.2 Proof of Lemma 7

Let us first describe the building blocks of our memory delegation scheme. Fix any $\tau(\lambda) = \omega(1)$ such that we have a weakly (K_{H}, γ)-collision-resistant hash function for $K_{\mathsf{H}}(\lambda, \zeta) = \mathsf{poly}(\lambda, \zeta)$ and $\gamma(\lambda) = \lambda^{\tau(\lambda)}$.

- $\mathsf{HLO} = (\mathsf{HLO.Hash}, \mathsf{HLO.Chal}, \mathsf{HLO.Auth}, \mathsf{HLO.Ver})$: a public-coin keyless K_{HLO}-collision-resistant hash function with local opening for input-length bound $\bar{L}$, where $\bar{L}$ is an arbitrarily small super-polynomial function and $K_{\mathsf{HLO}}(\lambda, \zeta, L) := K_{\mathsf{H}}(\lambda, \zeta)^{\tau(\lambda)}$. (Such a hash function is guaranteed to exist by Theorem 2.) Without loss of generality, we assume $\bar{L}(\lambda) \leq \lambda^{\tau(\lambda)}$.
- $\mathsf{ODel} = (\mathsf{ODel.Mem}, \mathsf{ODel.Query}_1, \mathsf{ODel.Query}_2, \mathsf{ODel.Prove}, \mathsf{ODel.Ver})$: a public-coin 2-round oracle memory delegation scheme with γ-soundness for computation-time bound $\bar{t}$, where γ is defined as above and $\bar{t}$ is an arbitrarily small super-polynomial function. (Such an oracle memory delegation scheme is guaranteed to exist by Lemma 6.)

  Let $\ell_{\widehat{\mathsf{DB}}}$ be an upper bound such that for a memory of length $L_{\mathsf{DB}} \leq \bar{t}(\lambda)$, the encoding algorithm $\mathsf{ODel.Mem}$ outputs an encoding of length at most $\ell_{\widehat{\mathsf{DB}}}(L_{\mathsf{DB}})$. Without loss of generality, we assume $\ell_{\widehat{\mathsf{DB}}}(L_{\mathsf{DB}}) \leq \bar{L}(\lambda)$ for every $L_{\mathsf{DB}} \leq \bar{t}(\lambda)$ by assuming that $\bar{t}$ is sufficiently smaller than $\bar{L}$.

  Let ℓ_I be an upper bound such that for any $L_{\mathsf{DB}} \leq \bar{t}(\lambda)$, the input query algorithm $\mathsf{ODel.Query}_2(L_{\mathsf{DB}}, \cdot, \cdot)$ outputs an input query of length at most $\ell_I(L_{\mathsf{DB}})$. Since $\mathsf{ODel.Query}_2(L_{\mathsf{DB}}, \cdot, \cdot)$ queries to an encoding of length $\ell_{\widehat{\mathsf{DB}}}(L_{\mathsf{DB}})$, we have $\ell_I(L_{\mathsf{DB}}) \leq \ell_{\widehat{\mathsf{DB}}}(L_{\mathsf{DB}})$. Also, from the efficiency condition of $\mathsf{ODel.Ver}$, we have $\ell_I(L_{\mathsf{DB}}) = \mathsf{poly}(\lambda)$ for any $L_{\mathsf{DB}} \leq \bar{t}(\lambda)$.

Using these building blocks, we obtain a two-round memory delegation scheme with weak soundness for computation-time bound $\bar{t}$.

Remark 8 (Simplified syntax of $\mathsf{HLO.Chal}$). In this paper, without loss of generality we can think as if $\mathsf{HLO.Chal}$ only takes 1^λ as input (rather than $(1^\lambda, 1^\rho)$ as defined in Definition 11). This is because we only use $\mathsf{HLO.Chal}$ with $\rho := \ell_I(L_{\mathsf{DB}})$ for $L_{\mathsf{DB}} \leq \bar{t}(\lambda)$. (Recall that ρ is the upper bound of the number of locations that can be locally opened.) Indeed, in this case, we have $\rho \leq \mathsf{poly}_\rho(\lambda)$ for a fixed polynomial poly_ρ as observed above, and thus, we can assume without loss of generality that $\mathsf{HLO.Chal}$ just outputs a sufficiently long random string $\bar{\mathsf{ch}}$ (which is longer than the output of $\mathsf{HLO.Chal}(1^\lambda, 1^\rho)$ for any $\rho \leq \mathsf{poly}_\rho(\lambda)$), and $\mathsf{HLO.Auth}$ and $\mathsf{HLO.Ver}$ just use a prefix of $\bar{\mathsf{ch}}$ as the actual challenge.[28] Thus, in the following, we use this simplified syntax of $\mathsf{HLO.Chal}$. Also, since $\mathsf{HLO.Auth}$ and $\mathsf{HLO.Ver}$ need to know $\rho = \ell_I(L_{\mathsf{DB}})$ to determine the length of the actual challenge, we write them as $\mathsf{HLO.Auth}_{L_{\mathsf{DB}}}$ and $\mathsf{HLO.Ver}_{L_{\mathsf{DB}}}$ to make it explicit what value of ρ they depend on. $\diamondsuit$

Our memory delegation scheme $\mathsf{Del} = (\mathsf{Mem}, \mathsf{Query}, \mathsf{Prove}, \mathsf{Ver})$ is described in Algorithm 6. Since HLO and ODel are public coin, Del is also public coin. Completeness can be verified by inspection. The efficiency condition of Del follows from the efficiency condition of ODel and the succinctness of HLO. In the rest of this proof, we focus on soundness.

Assume for contradiction that there exists a pair of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, a sequence of polynomial-size advice $\{z_\lambda\}_{\lambda \in \mathbb{N}}$, a samplable entropic distribution ensemble $\{Y_\lambda\}_{\lambda \in \mathbb{N}}$, and a polynomial p such that for infinitely many $\lambda \in \mathbb{N}$, there exists $t \leq \bar{t}^{O(1)}(\lambda)$ such that

$$\Pr\left[\mathsf{Ver}(\mathsf{digest}, \langle M, t, y\rangle, q, \pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{digest}, M, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ q \leftarrow \mathsf{Query}(1^\lambda) \\ y \leftarrow Y_\lambda \\ \pi \leftarrow \mathcal{A}_2(q, y, \mathsf{st}) \end{array}\right] \geq \frac{1}{p(\lambda)} \ . \tag{4}$$

[27] The public-coin property of the construction of [BKP18] is mentioned in [BKP18] and can be verified by inspection.

[28] Recall that $\mathsf{HLO.Chal}$ is public coin.

Algorithm 6 Public-coin weak memory delegation scheme Del.

- $\mathsf{digest} \coloneqq \mathsf{Mem}(1^\lambda, \mathsf{DB})$:
  1. Run $\widehat{\mathsf{DB}} \coloneqq \mathsf{ODel.Mem}(1^\lambda, \mathsf{DB})$.
  2. Run $\mathsf{digest}_{\mathsf{HLO}} \leftarrow \mathsf{HLO.Hash}(1^\lambda, \widehat{\mathsf{DB}})$.
  3. Output $\mathsf{digest} \coloneqq (\mathsf{digest}_{\mathsf{HLO}}, |\mathsf{DB}|)$.
- $q \leftarrow \mathsf{Query}(1^\lambda)$:
  1. Run $(q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}) \leftarrow \mathsf{ODel.Query}_1(1^\lambda)$.
  2. Run $\mathsf{ch} \leftarrow \mathsf{HLO.Chal}(1^\lambda)$.
  3. Output $q \coloneqq (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \mathsf{ch})$.
- $\pi \coloneqq \mathsf{Prove}(\mathsf{DB}, \langle M, t, y\rangle, q)$:
  1. Parse q as $(q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \mathsf{ch})$.
  2. Run $\pi_{\mathsf{ODel}} \coloneqq \mathsf{ODel.Prove}(\mathsf{DB}, \langle M, t, y\rangle, q_{\mathsf{ODel}})$.
  3. Run $I \coloneqq \mathsf{ODel.Query}_2(|\mathsf{DB}|, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}})$.
  4. Run $\pi_{\mathsf{HLO}} \leftarrow \mathsf{HLO.Auth}_{L_{\mathsf{DB}}}(1^\lambda, \widehat{\mathsf{DB}}, I, \mathsf{ch})$, where $L_{\mathsf{DB}} \coloneqq |\mathsf{DB}|$.
  5. Output $\pi \coloneqq (\pi_{\mathsf{ODel}}, I, \widehat{\mathsf{DB}}|_I, \pi_{\mathsf{HLO}})$.
- $b \coloneqq \mathsf{Ver}(\mathsf{digest}, \langle M, t, y\rangle, q, \pi)$:
  1. Parse digest as $(\mathsf{digest}_{\mathsf{HLO}}, L_{\mathsf{DB}})$, parse q as $(q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \mathsf{ch})$, parse π as $(\pi_{\mathsf{ODel}}, I, \widehat{\mathsf{DB}}|_I, \pi_{\mathsf{HLO}})$, and let $L_{\widehat{\mathsf{DB}}} \coloneqq \ell_{\widehat{\mathsf{DB}}}(L_{\mathsf{DB}})$.
  2. Output 1 if all of the following hold.
     (a) $L_{\widehat{\mathsf{DB}}} \leq \bar{L}(\lambda)$.
     (b) $\mathsf{ODel.Ver}^{\widehat{\mathsf{DB}}|_I}(L_{\mathsf{DB}}, \langle M, t, y\rangle, q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) = 1$.
     (c) $\mathsf{ODel.Query}_2(L_{\mathsf{DB}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) = I$.
     (d) $\mathsf{HLO.Ver}_{L_{\mathsf{DB}}}(1^\lambda, L_{\widehat{\mathsf{DB}}}, \mathsf{digest}_{\mathsf{HLO}}, I, \widehat{\mathsf{DB}}|_I, \mathsf{ch}, \pi_{\mathsf{HLO}}) = 1$.

We use $(\mathcal{A}_1, \mathcal{A}_2)$ to obtain a pair of $\gamma^{O(1)}$-time adversaries $(\mathcal{B}_1, \mathcal{B}_2)$ that breaks the soundness of ODel. Roughly speaking, the pair $(\mathcal{B}_1, \mathcal{B}_2)$ works as follows. Since $\mathcal{A}_1$ outputs a digest while $\mathcal{B}_1$ is required to output an encoded memory in the clear, $\mathcal{B}_1$ needs to extract a memory from $(\mathcal{A}_1, \mathcal{A}_2)$. Naturally, $\mathcal{B}_1$ uses the multi-collision resistance of HLO for this purpose. Specifically, $\mathcal{B}_1$ uses the extractor of HLO to extract a set of memories such that local opening from $\mathcal{A}_2$ is guaranteed to be consistent with one of the memories in the set. Then, $\mathcal{B}_1$ simply outputs a memory that is randomly chosen from the set, and $\mathcal{B}_2$ uses $\mathcal{A}_2$ to obtain a proof while hoping that the obtained proof is accepting w.r.t. the memory that was chosen by $\mathcal{B}_1$. The key point is that $\mathcal{B}_2$ obtains such a proof with non-negligible probability since the size of the extracted set is guaranteed to be not too large. The formal description of $(\mathcal{B}_1, \mathcal{B}_2)$ is given below.

- $(\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, M, y, y', \mathsf{st}) \leftarrow \mathcal{B}_1(1^\lambda, z_\lambda)$:
  1. Run $(\mathsf{digest}, M, \mathsf{st}_{\mathsf{Del}}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda)$, and parse digest as $(\mathsf{digest}_{\mathsf{HLO}}, L_{\mathsf{DB}})$. Define $L_{\widehat{\mathsf{DB}}} := \ell_{\widehat{\mathsf{DB}}}(L_{\mathsf{DB}})$. If $L_{\widehat{\mathsf{DB}}} > \bar{L}(\lambda)$, abort.
  2. Use the extractor $\mathsf{Ext}_{\mathsf{HLO}}$ of HLO to extract a set S of memories that $\mathcal{A}_2$ can locally open. Concretely, let $\mathcal{C}_2(\cdot, \mathsf{st}_{\mathsf{HLO}})$ for $\mathsf{st}_{\mathsf{HLO}} := (\lambda, \mathsf{st}_{\mathsf{Del}})$ be the adversary that converts $\mathcal{A}_2$ to an adversary against HLO as follows: On input ch, (i) run $(q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}) \leftarrow \mathsf{ODel.Query}_1(1^\lambda)$, (ii) sample $y \leftarrow Y_\lambda$, (iii) run $\pi_{\mathsf{Del}} \leftarrow \mathcal{A}_2(q_{\mathsf{Del}}, y, \mathsf{st}_{\mathsf{Del}})$ for $q_{\mathsf{Del}} := (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \mathsf{ch})$, (iv) parse π_{Del} as $(\pi_{\mathsf{ODel}}, I, A, \pi_{\mathsf{HLO}})$, and (v) output $(I, A, \pi_{\mathsf{HLO}})$. Then, the extractor $\mathsf{Ext}_{\mathsf{HLO}}$ is used against $\mathcal{C}_2$ with parameters $(\rho, L, K, \varepsilon) := (\ell_I(L_{\mathsf{DB}}), L_{\widehat{\mathsf{DB}}}, K_{\mathsf{HLO}}(\lambda, |z_\lambda|, L_{\widehat{\mathsf{DB}}}), 1/2p(\lambda))$.
  3. Sample $\widehat{\mathsf{DB}} \leftarrow S$ and $y, y' \leftarrow Y_\lambda$, and output $(\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, M, y, y', \mathsf{st})$, where $\mathsf{st} := (y, y', \mathsf{st}_{\mathsf{Del}})$.
- $(\pi_{\mathsf{ODel}}, \pi'_{\mathsf{ODel}}) \leftarrow \mathcal{B}_2(q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \mathsf{st})$:
  1. Parse st as $(y, y', \mathsf{st}_{\mathsf{Del}})$.
  2. $\mathsf{ch} \leftarrow \mathsf{HLO.Chal}(1^\lambda)$.
  3. Run $\pi_{\mathsf{Del}} \leftarrow \mathcal{A}_2(q_{\mathsf{Del}}, y, \mathsf{st}_{\mathsf{Del}})$ and $\pi'_{\mathsf{Del}} \leftarrow \mathcal{A}_2(q_{\mathsf{Del}}, y', \mathsf{st}_{\mathsf{Del}})$ for $q_{\mathsf{Del}} := (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \mathsf{ch})$.
  4. Parse π_{Del} as $(\pi_{\mathsf{ODel}}, I, A, \pi_{\mathsf{HLO}})$, and parse π'_{Del} as $(\pi'_{\mathsf{ODel}}, I', A', \pi'_{\mathsf{HLO}})$.
  5. Output $(\pi_{\mathsf{ODel}}, \pi'_{\mathsf{ODel}})$.

We note that $\mathcal{B}_1$ and $\mathcal{B}_2$ indeed run in time $\mathsf{poly}(\gamma(\lambda))$ since their running time is dominated by the running time of $\mathsf{Ext}_{\mathsf{HLO}}$, which is bounded by $\mathsf{poly}(L_{\widehat{\mathsf{DB}}}, K_{\mathsf{HLO}}(\lambda, |z_\lambda|, L_{\widehat{\mathsf{DB}}})) = \mathsf{poly}(\lambda^{\tau(\lambda)})$.

Now, we show that the pair $(\mathcal{B}_1, \mathcal{B}_2)$ indeed breaks the soundness of ODel. First, we rewrite (4) by inlining Query and Ver to obtain

$$\Pr\left[\begin{array}{l} L_{\widehat{\mathsf{DB}}} \le \bar{L}(\lambda) \\ \mathsf{ODel.Ver}^A(L_{\mathsf{DB}}, \langle M, t, y\rangle, q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) = 1 \\ \mathsf{ODel.Query}_2(L_{\mathsf{DB}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) = I \\ \mathsf{HLO.Ver}_{L_{\mathsf{DB}}}(1^\lambda, L_{\widehat{\mathsf{DB}}}, \mathsf{digest}_{\mathsf{HLO}}, I, A, \mathsf{ch}, \pi_{\mathsf{HLO}}) = 1 \end{array} \middle| \begin{array}{l} ((\mathsf{digest}_{\mathsf{HLO}}, L_{\mathsf{DB}}), M, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}) \leftarrow \mathsf{ODel.Query}_1(1^\lambda) \\ \mathsf{ch} \leftarrow \mathsf{HLO.Chal}(1^\lambda) \\ q := (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \mathsf{ch}) \\ y \leftarrow Y_\lambda \\ (\pi_{\mathsf{ODel}}, I, A, \pi_{\mathsf{HLO}}) \leftarrow \mathcal{A}_2(q, y, \mathsf{st}) \end{array}\right] \ge \frac{1}{p(\lambda)} \ ,$$

where $L_{\widehat{\mathsf{DB}}} := \ell_{\widehat{\mathsf{DB}}}(L_{\mathsf{DB}})$. Then, since the multi-collision resistance of HLO and the construction of the adversary $\mathcal{C}_2$ guarantee that we can extract a set of memories with which the adversary's local opening must agree, we have

$$\Pr\left[\begin{array}{l} L_{\widehat{\mathsf{DB}}} \le \bar{L}(\lambda) \\ \mathsf{ODel.Ver}^A(L_{\mathsf{DB}}, \langle M, t, y\rangle, q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) = 1 \\ \mathsf{ODel.Query}_2(L_{\mathsf{DB}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) = I \\ A \in \{\widehat{\mathsf{DB}}|_I \mid \widehat{\mathsf{DB}} \in S\} \end{array} \middle| \begin{array}{l} ((\mathsf{digest}_{\mathsf{HLO}}, L_{\mathsf{DB}}), M, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ S \leftarrow \mathsf{Ext}_{\mathsf{HLO}}^{\mathcal{C}_2(\cdot, \mathsf{st}_{\mathsf{HLO}})}(1^\lambda, 1^\rho, 1^L, 1^K, 1^{1/\varepsilon(\lambda)}) \\ (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}) \leftarrow \mathsf{ODel.Query}_1(1^\lambda) \\ \mathsf{ch} \leftarrow \mathsf{HLO.Chal}(1^\lambda) \\ q := (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \mathsf{ch}) \\ y \leftarrow Y_\lambda \\ (\pi_{\mathsf{ODel}}, I, A, \pi_{\mathsf{HLO}}) \leftarrow \mathcal{A}_2(q, y, \mathsf{st}) \end{array}\right]$$
$$\ge \frac{1}{p(\lambda)} - \frac{1}{2p(\lambda)} = \frac{1}{2p(\lambda)} \ ,$$

where $\mathsf{st}_{\mathsf{HLO}} := (\lambda, \mathsf{st})$ and $(\rho, L, K, \varepsilon) := (\ell_I(L_{\mathsf{DB}}), L_{\widehat{\mathsf{DB}}}, K_{\mathsf{HLO}}(\lambda, |z_\lambda|, L_{\widehat{\mathsf{DB}}}), 1/2p(\lambda))$ when the extractor $\mathsf{Ext}_{\mathsf{HLO}}$ is used. Then, since $\mathsf{Ext}_{\mathsf{HLO}}$ outputs a set of size at most $K = K_{\mathsf{HLO}}(\lambda, |z_\lambda|, L_{\widehat{\mathsf{DB}}})$, under the condition that the extraction succeeds, a memory that is randomly chosen from the extracted set is consistent with the adversary's local opening with probability $1/K$. Therefore, by using a randomly chosen extracted memory in the verification, we obtain

$$\Pr\left[\begin{array}{l} \\ L_{\widehat{\mathsf{DB}}} \le \bar{L}(\lambda) \\ \mathsf{ODel.Ver}^{\widehat{\mathsf{DB}}|_I}(L_{\mathsf{DB}}, \langle M, t, y\rangle, q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) = 1 \\ \mathsf{ODel.Query}_2(L_{\mathsf{DB}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) = I \\ \\ \end{array} \middle| \begin{array}{l} ((\mathsf{digest}_{\mathsf{HLO}}, L_{\mathsf{DB}}), M, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ S \leftarrow \mathsf{Ext}_{\mathsf{HLO}}^{\mathcal{C}_2(\cdot, \mathsf{st}_{\mathsf{HLO}})}(1^\lambda, 1^\rho, 1^L, 1^K, 1^{1/\varepsilon(\lambda)}) \\ \widehat{\mathsf{DB}} \leftarrow S \\ (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}) \leftarrow \mathsf{ODel.Query}_1(1^\lambda) \\ \mathsf{ch} \leftarrow \mathsf{HLO.Chal}(1^\lambda) \\ q := (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \mathsf{ch}) \\ y \leftarrow Y_\lambda \\ (\pi_{\mathsf{ODel}}, I, A, \pi_{\mathsf{HLO}}) \leftarrow \mathcal{A}_2(q, y, \mathsf{st}) \end{array}\right] \ge \frac{1}{2Kp(\lambda)} .$$

From an average argument, when we consider sampling two random outputs y, y' from Y_λ and obtaining a proof from $\mathcal{A}_2$ for each of them, we obtain accepting proofs for both of them with non-negligible probability, i.e., we have

$$\Pr\left[\begin{array}{l} \\ L_{\widehat{\mathsf{DB}}} \le \bar{L}(\lambda) \\ \mathsf{ODel.Ver}^{\widehat{\mathsf{DB}}|_I}(L_{\mathsf{DB}}, \langle M, t, y\rangle, q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) = 1 \\ \mathsf{ODel.Query}_2(L_{\mathsf{DB}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) = I \\ \mathsf{ODel.Ver}^{\widehat{\mathsf{DB}}|_{I'}}(L_{\mathsf{DB}}, \langle M, t, y'\rangle, q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \pi'_{\mathsf{ODel}}) = 1 \\ \mathsf{ODel.Query}_2(L_{\mathsf{DB}}, \sigma_{\mathsf{ODel}}, \pi'_{\mathsf{ODel}}) = I' \\ \\ \end{array} \middle| \begin{array}{l} ((\mathsf{digest}_{\mathsf{HLO}}, L_{\mathsf{DB}}), M, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda) \\ S \leftarrow \mathsf{Ext}_{\mathsf{HLO}}^{\mathcal{C}_2(\cdot, \mathsf{st}_{\mathsf{HLO}})}(1^\lambda, 1^\rho, 1^L, 1^K, 1^{1/\varepsilon(\lambda)}) \\ \widehat{\mathsf{DB}} \leftarrow S \\ (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}) \leftarrow \mathsf{ODel.Query}_1(1^\lambda) \\ \mathsf{ch} \leftarrow \mathsf{HLO.Chal}(1^\lambda) \\ q := (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \mathsf{ch}) \\ y, y' \leftarrow Y_\lambda \\ (\pi_{\mathsf{ODel}}, I, A, \pi_{\mathsf{HLO}}) \leftarrow \mathcal{A}_2(q, y, \mathsf{st}) \\ (\pi'_{\mathsf{ODel}}, I', A', \pi'_{\mathsf{HLO}}) \leftarrow \mathcal{A}_2(q, y', \mathsf{st}) \end{array}\right] \ge \left(\frac{1}{4Kp(\lambda)}\right)^3 . \quad (5)$$

Note that $\Pr[y \neq y' \mid y, y' \leftarrow Y_\lambda] \ge 1 - 2^{-\Omega(\lambda)}$ since Y_λ is a samplable entropic distribution. Therefore, from (5) and the constructions of $(\mathcal{B}_1, \mathcal{B}_2)$, we have

$$\Pr\left[\begin{array}{l} y \neq y' \\ \mathsf{ODel.Ver}^{\widehat{\mathsf{DB}}|_I}(L_{\mathsf{DB}}, \langle M, t, y\rangle, q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) = 1 \\ \mathsf{ODel.Ver}^{\widehat{\mathsf{DB}}|_{I'}}(L_{\mathsf{DB}}, \langle M, t, y'\rangle, q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \pi'_{\mathsf{ODel}}) = 1 \\ \\ \end{array} \middle| \begin{array}{l} (\widehat{\mathsf{DB}}, L_{\mathsf{DB}}, M, y, y', \mathsf{st}) \leftarrow \mathcal{B}_1(1^\lambda, z_\lambda) \\ (q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}) \leftarrow \mathsf{ODel.Query}_1(1^\lambda) \\ (\pi_{\mathsf{ODel}}, \pi'_{\mathsf{ODel}}) \leftarrow \mathcal{B}_2(q_{\mathsf{ODel}}, \sigma_{\mathsf{ODel}}, \mathsf{st}) \\ I := \mathsf{ODel.Query}_2(L_{\mathsf{DB}}, \sigma_{\mathsf{ODel}}, \pi_{\mathsf{ODel}}) \\ I' := \mathsf{ODel.Query}_2(L_{\mathsf{DB}}, \sigma_{\mathsf{ODel}}, \pi'_{\mathsf{ODel}}) \end{array}\right] \ge \left(\frac{1}{4Kp(\lambda)}\right)^3 - \frac{1}{2^{\Omega(\lambda)}} .$$

Since $K = K_{\mathsf{HLO}}(\lambda, |z_\lambda|, L_{\widehat{\mathsf{DB}}}) = \mathsf{poly}(\lambda^{\tau(\lambda)})$, the above implies that the pair $(\mathcal{B}_1, \mathcal{B}_2)$ indeed breaks the γ-soundness of ODel. This concludes the proof of Lemma 7.

7 Public-coin 3-round Zero-Knowledge Argument

In this section, we construct a public-coin 3-round zero-knowledge argument.

Theorem 3. *Assume the sub-exponential hardness of the LWE assumption, and assume the existence of a keyless weakly (K, γ)-collision-resistant hash function for $K(\lambda, \zeta) = \mathsf{poly}(\lambda, \zeta)$ and $\gamma(\lambda) = \lambda^{\tau(\lambda)}$ for a super-constant function $\tau(\lambda) = \omega(1)$. Then, there exists a public-coin 3-round zero-knowledge argument for NP.*

Following prior works [BBK+16, BKP18], we prove Theorem 3 by using our weak memory delegation scheme (Lemma 7) to reduce the round complexity of Barak's public-coin zero-knowledge argument [Bar01].

In particular, as in the prior works [BBK+16, BKP18], we combine our weak memory delegation scheme with a primitive called *a witness-indistinguishable argument of knowledge with first-message-dependent instances* [BBK+16, BKP18]. Thus, we start by recalling this primitive.

7.1 Preliminary: Witness Indistinguishability with First-Message-Dependent Instances

We recall the definition of witness-indistinguishable arguments of knowledge with first-message-dependent instances from [BBK+16, BKP18], where we make straightforward modifications to focus on public-coin ones.

Definition 12 (WIAOK with first-message-dependent instances)**.** *For any NP language $\boldsymbol{L}$, a public-coin 3-round argument (P, V) for $\boldsymbol{L}$ is said to be* a public-coin witness-indistinguishable argument of knowledge (WIAOK) with first-message-dependent instances *if it satisfies the following.*

- ***Completeness with first-message-dependent instances:*** *For any instance choosing function X and any $\lambda, \ell \in \mathbb{N}$,*

$$\Pr\left[V(x, \mathsf{wi}_1, \mathsf{wi}_2, \mathsf{wi}_3) = 1 \;\middle|\; \begin{array}{l} (\mathsf{wi}_1, \mathsf{st}_P) \leftarrow P(1^\lambda, \ell) \\ (x, w) \leftarrow X(\mathsf{wi}_1), \text{ where } x \in \boldsymbol{L} \cap \{0,1\}^\ell \text{ and } w \in \boldsymbol{R_L}(x) \\ \mathsf{wi}_2 \leftarrow V(1^\lambda, \ell) \\ \mathsf{wi}_3 \leftarrow P(x, w, \mathsf{wi}_1, \mathsf{wi}_2, \mathsf{st}_P) \end{array}\right] = 1 \ .$$

  Furthermore, the honest prover's first message wi_1 is of length λ, independent of the length ℓ of the statement x.

- ***Adaptive witness indistinguishability:*** *For every polynomial ℓ, every* PPT *interactive Turing machine V^*, and every sequence of polynomial-size advice $\{z_\lambda\}_{\lambda\in\mathbb{N}}$, there exists a negligible function* negl *such that for every $\lambda \in \mathbb{N}$,*

$$\Pr\left[V^*(x, \mathsf{wi}_1, \mathsf{wi}_2, \mathsf{wi}_3, \mathsf{st}_V) = b \;\middle|\; \begin{array}{l} (\mathsf{wi}_1, \mathsf{st}_P) \leftarrow P(1^\lambda, \ell(\lambda)) \\ (x, w_0, w_1, \mathsf{wi}_2, \mathsf{st}_V) \leftarrow V^*(\mathsf{wi}_1, z_\lambda), \\ \quad \text{where } x \in \boldsymbol{L} \cap \{0,1\}^{\ell(\lambda)} \text{ and } w_0, w_1 \in \boldsymbol{R_L}(x) \\ b \leftarrow \{0,1\} \\ \mathsf{wi}_3 \leftarrow P(x, w_b, \mathsf{wi}_1, \mathsf{wi}_2, \mathsf{st}_P) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda) \ .$$

- ***Adaptive argument of knowledge:*** *There exists a uniform* PPT *oracle Turing machine* Ext *such that for every polynomial ℓ and every deterministic polynomial-time interactive Turing machine P^* (possibly with some hardwired inputs), there exists a negligible function* negl *such that for every sufficiently large $\lambda \in \mathbb{N}$ and every $\varepsilon > 0$, if*

$$\Pr\left[V(x, \mathsf{wi}_1, \mathsf{wi}_2, \mathsf{wi}_3) = 1 \;\middle|\; \begin{array}{l} \mathsf{wi}_1 \leftarrow P^* \\ \mathsf{wi}_2 \leftarrow V(1^\lambda, \ell(\lambda)) \\ (x, \mathsf{wi}_3) \leftarrow P^*(\mathsf{wi}_1, \mathsf{wi}_2) \end{array}\right] \geq \varepsilon \ ,$$

  then

$$\Pr\left[\begin{array}{l} V(x, \mathsf{wi}_1, \mathsf{wi}_2, \mathsf{wi}_3) = 1 \\ \wedge\, w \notin \boldsymbol{R_L}(x) \end{array} \;\middle|\; \begin{array}{l} \mathsf{wi}_1 \leftarrow P^* \\ \mathsf{wi}_2 \leftarrow V(1^\lambda, \ell(\lambda)) \\ (x, \mathsf{wi}_3) \leftarrow P^*(\mathsf{wi}_1, \mathsf{wi}_2) \\ w \leftarrow \mathsf{Ext}^{P^*}(1^{1/\varepsilon}, x, \mathsf{wi}_1, \mathsf{wi}_2, \mathsf{wi}_3) \end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

It is observed in [BKP18] that a public-coin 3-round WIAOK with first-message-dependent instances can be obtained from a keyless multi-collision-resistant hash function.

Theorem 4 ([BKP18])**.** *Assume the existence of a weakly K-collision-resistant hash function for $K(\lambda, \zeta) = \mathsf{poly}(\lambda, \zeta)$. Then, for any language in NP, there exists a public-coin 3-round WIAOK with first-message-dependent instances.*

7.2 Proof of Theorem 3

Let us first lists the building blocks of our 3-round zero-knowledge argument.

- Com: a non-interactive perfectly binding commitment scheme (which can be constructed under the LWE assumption [GHKW17, LS19]). Without loss of generality, we assume that the committer algorithm uses λ-bit randomness.
- $(P_{\mathsf{WIAOK}}, V_{\mathsf{WIAOK}})$: a public-coin 3-round WIAOK with first-message-dependent instances (e.g., the one that we can obtain by Theorem 4).
- $\mathsf{Del} = (\mathsf{Del.Mem}, \mathsf{Del.Query}, \mathsf{Del.Prove}, \mathsf{Del.Ver})$: a public-coin 2-round memory delegation scheme with weak soundness for slightly super-polynomial computation-time bound $\bar{t}$ (e.g., the one that we can obtain by Lemma 7). We assume for simplicity that $\mathsf{Del.Mem}$ outputs a digest of length λ, and we use ℓ_π to denote the proof length. (Because of the efficiency condition of memory delegation schemes, we have $\ell_\pi(\lambda) = \mathsf{poly}(\lambda)$.)

For an NP language $\mathbf{L}$, our 3-round zero-knowledge argument (P, V) is described in Algorithm 7. Clearly, the prover and the verifier run in polynomial time. (Importantly, because of the efficiency condition of weak memory delegation schemes, the language $\mathbf{L}'$ is NP, and thus, $(P_{\mathsf{WIAOK}}, V_{\mathsf{WIAOK}})$ for the statement Ψ can be executed in polynomial time.) Also, since $(P_{\mathsf{WIAOK}}, V_{\mathsf{WIAOK}})$ and Del are public coin, (P, V) is public coin. Now, since the completeness of (P, V) can be verified by inspection, in the following, we show the soundness and zero-knowledge of (P, V).

7.2.1 Proof of soundness.

Assume for contradiction that there exists a PPT cheating prover P^* and a polynomial p such that for infinitely many $x \notin \mathbf{L}$, there exists $z \in \{0,1\}^*$ such that

$$\Pr\left[\langle P^*(z), V\rangle(x) = 1\right] \geq \frac{1}{p(\lambda)} \quad . \tag{6}$$

Without loss of generality, we assume that P^* is deterministic. Let us call a pair (x, z) be *bad* if we have (6) for (x, z). Let $\Lambda \subseteq \mathbb{N}$ be the set such that we have $\lambda \in \Lambda$ if and only if there exists a bad (x, z) such that $|x| = \lambda$.

We use P^* to obtain a pair of adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ that breaks the weak soundness of Del. Since P^* is deterministic, for any (x, z), the first-round message $(\mathsf{wi}_1, \mathsf{cmt})$ that $P^*(x, z)$ outputs is uniquely determined; thus, due to the perfect binding property of Com, the committed value of cmt, denoted as $\mathsf{digest}^*|t^*$, is also uniquely determined. For each $\lambda \in \Lambda$, pick any bad (x, z) such that $|x| = \lambda$, and let $z_\lambda := (x, z, \mathsf{digest}^*, t^*)$. Then, we consider the following adversaries $(\mathcal{A}_1, \mathcal{A}_2)$.

- $(\mathsf{digest}, M, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, z_\lambda)$**:**
  1. Parse z_λ as $(x, z, \mathsf{digest}^*, t^*)$.
  2. Run $(\mathsf{wi}_1, \mathsf{cmt}, \mathsf{st}_{P^*}) := P^*(x, z)$.
  3. Output $(\mathsf{digest}^*, M_{\mathsf{wi}_1, \mathsf{cmt}}, \mathsf{st})$, where $\mathsf{st} = (x, z, \mathsf{digest}^*, t^*, \mathsf{wi}_1, \mathsf{cmt}, \mathsf{st}_{P^*})$.
- $\pi \leftarrow \mathcal{A}_2(q, y, \mathsf{st})$**:**
  1. Parse st as $(x, z, \mathsf{digest}^*, t^*, \mathsf{wi}_1, \mathsf{cmt}, \mathsf{st}_{P^*})$.
  2. Run $\mathsf{wi}_3 := P^*(y, \mathsf{wi}_2, q, \mathsf{st}_{P^*})$, where wi_2 is obtained by $\mathsf{wi}_2 \leftarrow V_{\mathsf{WIAOK}}(1^\lambda, \ell_\Psi(\lambda))$.
  3. Use the extractor $\mathsf{Ext}_{\mathsf{WIAOK}}$ of $(P_{\mathsf{WIAOK}}, V_{\mathsf{WIAOK}})$ to extract a witness w from P^*. Concretely, let P^*_{WIAOK} be the cheating prover that converts P^* to a prover against the adaptive argument-of-knowledge property of $(P_{\mathsf{WIAOK}}, V_{\mathsf{WIAOK}})$ as follows. P^*_{WIAOK} has (x, z, q, y) as a hardwired input. In Round 1, P^*_{WIAOK} runs $(\mathsf{wi}_1, \mathsf{cmt}, \mathsf{st}_{P^*}) := P^*(x, z)$ and outputs wi_1. After receiving wi_2, P^*_{WIAOK} runs $\mathsf{wi}_3 := P^*(y, \mathsf{wi}_2, q, \mathsf{st}_{P^*})$ and outputs $\Psi = (x, \mathsf{wi}_1, \mathsf{cmt}, y, q)$ and wi_3 in Round 3. Then, the extractor $\mathsf{Ext}_{\mathsf{WIAOK}}$ is used against P^*_{WIAOK} with parameter $\varepsilon = 1/2p(\lambda)$ to obtain a witness w. If w is not a valid witness for $(\mathsf{wi}_1, \mathsf{cmt}, y, q) \in \mathbf{L}'$, abort.

Algorithm 7 Public-coin 3-round zero-knowledge argument (P, V).

- **Input.** Let $x \in \mathbf{L}$ be the common input to the prover P and the verifier V, and $w \in \mathbf{R}_{\mathbf{L}}(x)$ be the private input to P. Let $\lambda := |x|$ be the security parameter.
- **Round 1.** The prover P does the following.
  1. Run $(\mathsf{wi}_1, \mathsf{st}_P) \leftarrow P_{\mathsf{WIAOK}}(1^\lambda, \ell_\Psi(\lambda))$, where ℓ_Ψ is the length of the statement Ψ that is defined in Round 3 below.
  2. Run $\mathsf{cmt} \leftarrow \mathsf{Com}(0^\lambda | 0^{\log \bar{t}(\lambda)})$, where $0^\lambda | 0^{\log \bar{t}(\lambda)}$ is the concatenation of 0^λ and $0^{\log \bar{t}(\lambda)}$. (Recall that $\bar{t}$ is the computation-time bound that is defined in the soundness of Del.)
  3. Send $(\mathsf{wi}_1, \mathsf{cmt})$ to V.
- **Round 2.** On receiving $(\mathsf{wi}_1, \mathsf{cmt})$, the verifier V does the following.
  1. Sample a uniformly random string $y \in \{0,1\}^\lambda$.
  2. Run $\mathsf{wi}_2 \leftarrow V_{\mathsf{WIAOK}}(1^\lambda, \ell_\Psi(\lambda))$.
  3. Run $q \leftarrow \mathsf{Del.Query}(1^\lambda)$.
  4. Send (y, wi_2, q) to P.
- **Round 3.** On receiving (y, wi_2, q), the prover P does the following.
  1. Let $\mathbf{L}'$ be the language defined by

$$\mathbf{L}' := \left\{ (\mathsf{wi}_1, \mathsf{cmt}, y, q) \;\middle|\; \begin{array}{l} \exists \mathsf{digest}, r_{\mathsf{cmt}} \in \{0,1\}^\lambda, \pi \in \{0,1\}^{\ell_\pi(\lambda)}, t \le \bar{t}(\lambda) \\ \text{s.t. } \mathsf{Com}(\mathsf{digest}|t; r_{\mathsf{cmt}}) = \mathsf{cmt} \\ \quad \mathsf{Del.Ver}(\mathsf{digest}, \langle M_{\mathsf{wi}_1, \mathsf{cmt}}, t, y \rangle, q, \pi) = 1 \end{array} \right\},$$

     where $M_{\mathsf{wi}_1, \mathsf{cmt}}$ is a Turing machine that has $(\mathsf{wi}_1, \mathsf{cmt})$ as a hardwired input, and on input $\mathsf{DB} \in \{0,1\}^*$, it does the following: (i) view DB as a Turing machine M (possibly with some hardwired inputs), (ii) run $M(\mathsf{wi}_1, \mathsf{cmt})$, (iii) parse the output of M as (y, wi_2, q), and (iv) output y.
  2. Let $\Psi = (x, \mathsf{wi}_1, \mathsf{cmt}, y, q)$ be the following statement: $x \in \mathbf{L} \vee (\mathsf{wi}_1, \mathsf{cmt}, y, q) \in \mathbf{L}'$. (Note that the length of Ψ is determined by $\lambda = |x|$ alone.)
  3. Send $\mathsf{wi}_3 \leftarrow P_{\mathsf{WIAOK}}(\Psi, w, \mathsf{wi}_1, \mathsf{wi}_2, \mathsf{st}_P)$ to V.
- **Verification.** On receiving wi_3, the verifier does the following.
  1. Output $b := V_{\mathsf{WIAOK}}(\Psi, \mathsf{wi}_1, \mathsf{wi}_2, \mathsf{wi}_3)$.

4. Parse w as $(\mathsf{digest}, r_{\mathsf{cmt}}, \pi, t)$, and output π.

We show that the pair $(\mathcal{A}_1, \mathcal{A}_2)$ indeed breaks the weak soundness of Del. For any $\lambda \in \Lambda$, let $z_\lambda = (x, z, \mathsf{digest}^*, t^*)$ be defined as above and Y_λ be the samplable entropic distribution that outputs a uniformly random string $y \in \{0,1\}^\lambda$. Consider the experiment for the weak soundness of Del (Definition 7) w.r.t. $(\mathcal{A}_1, \mathcal{A}_2)$, $\{z_\lambda\}_{\lambda\in\Lambda}$, $\{Y_\lambda\}_{\lambda\in\Lambda}$, $\lambda \in \Lambda$, and t^*. First, since the pair $(\mathcal{A}_1, \mathcal{A}_2)$ perfectly emulates an honest verifier of (P, V) for P^*, the transcript $(\mathsf{wi}_1, \mathsf{wi}_2, \mathsf{wi}_3)$ that is obtained in $\mathcal{A}_2$ is accepting with probability at least $1/p(\lambda)$. Then, from an average argument, with probability at least $1/2p(\lambda)$ over the choice of the input (q, y) to $\mathcal{A}_2$, the input (q, y) to $\mathcal{A}_2$ is "good" in the sense that when the input to $\mathcal{A}_2$ is (q, y), the transcript $(\mathsf{wi}_1, \mathsf{wi}_2, \mathsf{wi}_3)$ that is obtained in $\mathcal{A}_2$ is accepting with probability at least $1/2p(\lambda)$ over the choice of wi_2. Then, since $x \notin \mathbf{L}$, the adaptive argument-of-knowledge property of $(P_{\mathsf{WIAOK}}, V_{\mathsf{WIAOK}})$ guarantees that under the condition that the input (q, y) to $\mathcal{A}_2$ is good, $\mathcal{A}_2$ obtains a valid witness w for $(\mathsf{wi}_1, \mathsf{cmt}, y, q) \in \mathbf{L}'$ with probability at least $1/2p(\lambda) - \mathsf{negl}(\lambda)$. Now, the definition of $\mathbf{L}'$ and the binding property of Com guarantee that when $w = (\mathsf{digest}, r_{\mathsf{cmt}}, \pi, t)$ is a valid witness for $(\mathsf{wi}_1, \mathsf{cmt}, y, q) \in \mathbf{L}'$, we have $\mathsf{Del.Ver}(\mathsf{digest}^*, \langle M_{\mathsf{wi}_1,\mathsf{cmt}}, t^*, y\rangle, q, \pi) = 1$ and $t^* \le \bar{t}(\lambda)$. Thus, $\mathcal{A}_2$ outputs an accepting proof π with probability at least $(1/2p(\lambda) - \mathsf{negl}(\lambda)) \cdot 1/2p(\lambda)$, and therefore, the pair $(\mathcal{A}_1, \mathcal{A}_2)$ breaks the weak soundness of Del.

7.2.2 Proof of zero-knowledge.

For any polynomial-time cheating verifier V^* (which is assumed to be deterministic without loss of generality), our simulator $\mathcal{S}$ is described in Algorithm 8. Because of the efficiency condition of memory delegation schemes, $\mathcal{S}$ runs in time $\mathsf{poly}(\lambda, t^*) = \mathsf{poly}(\lambda)$. Also, by the definition of $\mathbf{L}'$ and the trivial fact that $V^*_{x,z}(\mathsf{wi}_1, \mathsf{cmt}) = (y, \mathsf{wi}_2, q)$, the witness w in Round 3 is a valid witness for $(\mathsf{wi}_1, \mathsf{cmt}, y, q) \in \mathbf{L}'$.[29] Thus, the indistinguishability between honest executions and simulation can be shown by using a standard hybrid argument relying on the hiding property of Com and the adaptive witness indistinguishability of $(P_{\mathsf{WIAOK}}, V_{\mathsf{WIAOK}})$.

Algorithm 8 Simulator $\mathcal{S}(x, z)$.

$\mathcal{S}$ internally invokes $V^*(x, z)$ and emulates an interaction of (P, V) for V^* as follows. In the following, we use $V^*_{x,z}$ to denote the description of V^* in which (x, z) is hardwired as the input. Let $\lambda := |x|$.

- **Round 1.**
  1. Run $(\mathsf{wi}_1, \mathsf{st}_P) \leftarrow P_{\mathsf{WIAOK}}(1^\lambda, \ell_\Psi(\lambda))$
  2. Run $\mathsf{digest} := \mathsf{Del.Mem}(1^\lambda, V^*_{x,z})$.
  3. Run $\mathsf{cmt} := \mathsf{Com}(\mathsf{digest}|t^*; r_{\mathsf{cmt}})$ for a uniformly random string $r_{\mathsf{cmt}} \in \{0,1\}^\lambda$, where t^* is an upper bound of the running time of $M_{\mathsf{wi}_1,\mathsf{cmt}}(V^*_{x,z})$. (It can be assumed without loss of generality that t^* depends on $|\mathsf{wi}_1|$ and $|\mathsf{cmt}|$ but otherwise does not depend on $\mathsf{wi}_1, \mathsf{cmt}$. Therefore, there is no circularity, i.e., t^* can be defined before obtaining cmt.) Note that since V^* runs in polynomial time, $t^* = \mathsf{poly}(\lambda) \le \bar{t}(\lambda)$.
  4. Send $(\mathsf{wi}_1, \mathsf{cmt})$ to V^*.
- **Round 3.** On receiving (y, wi_2, q) from V^*, the simulator $\mathcal{S}$ does the following.
  1. Run $\pi := \mathsf{Del.Prove}(V^*_{x,z}, \langle M_{\mathsf{wi}_1,\mathsf{cmt}}, t^*, y\rangle, q)$.
  2. Send $\mathsf{wi}_3 \leftarrow P_{\mathsf{WIAOK}}(\Psi, w, \mathsf{wi}_1, \mathsf{wi}_2, \mathsf{st}_P)$ to V^*, where $w := (\mathsf{digest}, r_{\mathsf{cmt}}, \pi, t^*)$.

Then, $\mathcal{S}$ outputs the view of V^* in the above interaction.

This completes the proof of Theorem 3.

[29] Since Del is public coin, its correctness guarantees that the proof π in Round 3 is accepting for any q.

References

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd FOCS*, pages 106–115. IEEE Computer Society Press, October 2001.

[BBK+16] Nir Bitansky, Zvika Brakerski, Yael Tauman Kalai, Omer Paneth, and Vinod Vaikuntanathan. 3-message zero knowledge against human ignorance. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 57–83. Springer, Heidelberg, October / November 2016.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, October 2017.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th ACM STOC*, pages 505–514. ACM Press, May / June 2014.

[BDRV18] Itay Berman, Akshay Degwekar, Ron D. Rothblum, and Prashant Nalini Vasudevan. Multi-collision resistant hash functions and their applications. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 133–161. Springer, Heidelberg, April / May 2018.

[BEP20] Nir Bitansky, Noa Eizenstadt, and Omer Paneth. Weakly extractable one-way functions. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 596–626. Springer, Heidelberg, November 2020.

[BGGL01] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. In *42nd FOCS*, pages 116–125. IEEE Computer Society Press, October 2001.

[BGJ+18] Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal MPC. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 459–487. Springer, Heidelberg, August 2018.

[BKP18] Nir Bitansky, Yael Tauman Kalai, and Omer Paneth. Multi-collision resistance: a paradigm for keyless hash functions. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 671–684. ACM Press, June 2018.

[BKP19] Nir Bitansky, Dakshita Khurana, and Omer Paneth. Weak zero-knowledge beyond the black-box barrier. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1091–1102. ACM Press, June 2019.

[BL18] Nir Bitansky and Huijia Lin. One-message zero knowledge and non-malleable commitments. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 209–234. Springer, Heidelberg, November 2018.

[Blu86] Manuel Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, volume 2, pages 1444–1451, 1986.

[BP04] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 273–289. Springer, Heidelberg, August 2004.

[BP19] Nir Bitansky and Omer Paneth. On round optimal statistical zero knowledge arguments. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 128–156. Springer, Heidelberg, August 2019.

[BR22] Liron Bronfman and Ron D. Rothblum. PCPs and Instance Compression from a Cryptographic Lens. In Mark Braverman, editor, *ITCS 2022*, volume 215, pages 30:1–30:19. LIPIcs, January 2022.

[CD08] Ran Canetti and Ronny Ramzi Dakdouk. Extractable perfectly one-way functions. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 449–460. Springer, Heidelberg, July 2008.

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. Cryptology ePrint Archive, Report 2021/808, Version 20211108:181325, 2021. https://eprint.iacr.org/2021/808. An extended version of [CJJ22].

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

[CKLR11] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 151–168. Springer, Heidelberg, August 2011.

[Den20] Yi Deng. Individual simulations. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 805–836. Springer, Heidelberg, December 2020.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[GHKW17] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. A generic approach to constructing and proving verifiable random functions. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 537–566. Springer, Heidelberg, November 2017.

[GJS11] Sanjam Garg, Abhishek Jain, and Amit Sahai. Leakage-resilient zero knowledge. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 297–315. Springer, Heidelberg, August 2011.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *Journal of the ACM*, 62(4):27:1–27:64, 2015.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

[Gol17a] Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, Cambridge, UK, 2017. A draft is available at http://www.wisdom.weizmann.ac.il/~oded/PDF/pt-v3.pdf.

[Gol17b] Oded Goldreich. On the doubly-efficient interactive proof systems of GKR. Electronic Colloquium on Computational Complexity, 2017.

[HLR21a] Justin Holmgren, Alex Lombardi, and Ron D. Rothblum. Fiat-Shamir via list-recoverable codes (or: Parallel repetition of GMW is not zero-knowledge). Cryptology ePrint Archive, Report 2021/286, Version: 20210307:022349, 2021. https://eprint.iacr.org/2021/286. An extended version of [HLR21b].

[HLR21b] Justin Holmgren, Alex Lombardi, and Ron D. Rothblum. Fiat–Shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge). In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, page 750–760. ACM Press, June 2021.

[HR04] Chun-Yuan Hsiao and Leonid Reyzin. Finding collisions on a public road, or do secure hash functions need secret coins? In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 92–105. Springer, Heidelberg, August 2004.

[HT98] Satoshi Hada and Toshiaki Tanaka. On the existence of 3-round zero-knowledge protocols. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 408–423. Springer, Heidelberg, August 1998.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015.

[JKKR17] Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Ron Rothblum. Distinguisher-dependent simulation in two rounds and its applications. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 158–189. Springer, Heidelberg, August 2017.

[JKKZ20] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. Cryptology ePrint Archive, Report 2020/980, Version 20200819:035531, 2020. https://eprint.iacr.org/2020/980. An extended version of [JKKZ21].

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

[KNY18] Ilan Komargodski, Moni Naor, and Eylon Yogev. Collision resistant hashing for paranoids: Dealing with multiple collisions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 162–194. Springer, Heidelberg, April / May 2018.

[KPY18] Yael Kalai, Omer Paneth, and Lisa Yang. On publicly verifiable delegation from standard assumptions. Cryptology ePrint Archive, Report 2018/776, 2018. https://eprint.iacr.org/2018/776.

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

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In David B. Shmoys, editor, *46th ACM STOC*, pages 485–494. ACM Press, May / June 2014.

[KS17] Dakshita Khurana and Amit Sahai. How to achieve non-malleability in one or two rounds. In Chris Umans, editor, *58th FOCS*, pages 564–575. IEEE Computer Society Press, October 2017.

[LS19] Alex Lombardi and Luke Schaeffer. A note on key agreement and non-interactive commitments. Cryptology ePrint Archive, Report 2019/279, 2019. https://eprint.iacr.org/2019/279.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 160–176. Springer, Heidelberg, May 2003.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6), 2009.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

A Additional Preliminaries

A.1 The Learning with Errors (LWE) Assumption

The formal definition of the learning with errors assumption [Reg09] is not needed to understand this paper since we only use it indirectly to use several results of prior works [CJJ22, HLR21b, JKKZ21]. The sub-exponential hardness of the LWE assumption requires that there exists a constant $0 < \epsilon < 1$ such that any non-uniform 2^{λ^ϵ}-time adversary cannot break the LWE assumption with advantage more than 2^{λ^ϵ}.

A.2 The Fiat–Shamir Transformation

In this paper, the Fiat–Shamir transformation [FS87] is used to transform a multi-round public-coin interactive proof into a non-interactive argument. Let (P, V) be a public-coin interactive proof for a language L. By adding dummy rounds if necessary, we assume that there exists $\ell \in \mathbb{N}$ such that P and V interact in 2ℓ rounds and the prover sends the first message. Then, for a hash function family $\mathcal{H}$ (with some appropriate domain and range), the Fiat–Shamir transformation transforms (P, V) into a non-interactive argument $(P_{\mathsf{FS}}, V_{\mathsf{FS}})$ in the CRS model as follows. (The following description is based on [JKKZ20, Figure 1].)

- The common reference string is $\mathsf{crs} := (h_1, \ldots, h_\ell)$, where $h_i \leftarrow \mathcal{H}_\lambda$ for $\forall i \in [\lambda]$.
- The prover P_{FS} takes as input (crs, x) and does the following.
  1. Set $i := 1$ and $\tau_0 := \emptyset$.
  2. Compute $\alpha_i \leftarrow P(x, \tau_{i-1})$ and $\beta_i := h_i(\tau_{i-1}|\alpha_i)$, where $\tau_{i-1}|\alpha_i$ is the concatenation of τ_{i-1} and α_i.
  3. Set $\tau_i := \tau_{i-1}|\alpha_i|\beta_i$.
  4. If $i = \ell$, output τ_i. Otherwise, set $i := i + 1$ and go to item 2.
- The verifier V_{FS} takes as input (crs, x, τ) and does the following.
  1. Parse crs as $(h_1, \ldots, h_\ell)$, and parse τ as $\alpha_1|\beta_1|\cdots|\alpha_\ell|\beta_\ell$.
  2. Output 1 if and only if $V(x, \tau) = 1$ and $\beta_i = h_i(\tau_{i-1}|\alpha_i)$ for every $i \in [\ell]$, where $\tau_{i-1} := \alpha_1|\beta_1|\cdots|\alpha_{i-1}|\beta_{i-1}$.

Note that when $\mathcal{H}$ is public coin, the common reference string crs is uniformly random.

A.3 Algorithms for Low-Degree Polynomials

We recall some basic notations and definitions about low-degree polynomials. These are used only in Section 4 to work with the interactive proof of Goldwasser, Kalai, and Rothblum [GKR15].

Notations. For any finite field $\mathbb{F}$, integers $m, d \in \mathbb{N}$, and a constant $0 < \epsilon < 1$, we say that a function $f : \mathbb{F}^m \to \mathbb{F}$ is *ϵ-close* to degree-d polynomials if there exists a polynomial $\widehat{f} : \mathbb{F}^m \to \mathbb{F}$ of (total) degree at most d such that the relative Hamming distance between f and $\widehat{f}$ is at most ϵ, i.e., $\Pr[f(x) \neq \widehat{f}(x) \mid x \leftarrow \mathbb{F}^m] \leq \epsilon$.

A.3.1 Low-degree tests.

We first recall a basic result about low-degree tests, which check whether a function f is close to a low-degree polynomial by making a small number of queries.

Lemma 8 ([Gol17a]; see also [RS96])**.** *Let $\mathbb{F}$ be a finite field of prime order, $m, d \in \mathbb{N}$ be integers such that $d < |\mathbb{F}|/2$, and $0 < \epsilon < 1$ be a constant. Then, the algorithms $(\mathsf{LDTest.Q}, \mathsf{LDTest.D})$ in Algorithm 9 satisfy the following.*

1. *For any polynomial $f : \mathbb{F}^m \to \mathbb{F}$ of degree at most d,*

$$\Pr\left[\mathsf{LDTest.D}(\mathsf{st}, f|_Q) = 1 \mid (\mathsf{st}, Q) \leftarrow \mathsf{LDTest.Q}\right] = 1 \ .$$

2. *For any function $f : \mathbb{F}^m \to \mathbb{F}$ that is not ϵ-close to degree-d polynomials,*

$$\Pr\left[\mathsf{LDTest.D}(\mathsf{st}, f|_Q) = 0 \mid (\mathsf{st}, Q) \leftarrow \mathsf{LDTest.Q}\right] \geq \min\left(\frac{\epsilon}{2}, \Omega(d^{-2})\right) \ .$$

Algorithm 9 Low-degree test $\mathsf{LDTest} = (\mathsf{LDTest.Q}, \mathsf{LDTest.D})$.

- $\mathsf{LDTest.Q}$:
  1. Choose uniformly random points $x, h \in \mathbb{F}^m$.
  2. Output $\mathsf{st} := (x, h)$ and $Q := \{x + ih\}_{i \in \{0,\ldots,d+1\}}$.
- $\mathsf{LDTest.D}(\mathsf{st}, A)$:
  1. Reconstruct a (unique) degree-d polynomial $g : \mathbb{F} \to \mathbb{F}$ that satisfies $g(i) = A(x + ih)$ for every $i \in [d + 1]$ (where i is viewed as an element of $\mathbb{F}$ when it is given to g).
  2. Output 1 if $g(0) = A(x)$ and output 0 otherwise.

A.3.2 Local self-correction.

We next recall a basic result about (local) self-correction. When given a function f that is close to a low-degree polynomial $\tilde{f}$, self-correction allows us to evaluate $\tilde{f}$ by making a small number of queries to f.

Lemma 9. *Let $\mathbb{F}$ be a finite field of prime order, $m, d \in \mathbb{N}$ be integers such that $d < |\mathbb{F}|$, and $0 < \epsilon_1, \epsilon_2 < 1$ be constants such that $\epsilon_1 < \min((1 - d/|\mathbb{F}|)/2, \epsilon_2/(d + 1))$. Then, for any $v \in \mathbb{F}^m$ and any function $f : \mathbb{F}^m \to \mathbb{F}$ that is ϵ_1-close to a degree-d polynomial $\tilde{f}$,[30] the algorithms $(\mathsf{SelfCorr.Q}, \mathsf{SelfCorr.Rec})$ in Algorithm 10 satisfy the following.*

$$\Pr\left[\mathsf{SelfCorr.Rec}(\mathsf{st}, f|_Q) = \tilde{f}(v) \;\middle|\; (\mathsf{st}, Q) \leftarrow \mathsf{SelfCorr.Q}(v)\right] \geq 1 - \epsilon_2 \ . \tag{7}$$

Furthermore, if f itself is a degree-d polynomial,

$$\Pr\left[\mathsf{SelfCorr.Rec}(\mathsf{st}, f|_Q) = f(v) \mid (\mathsf{st}, Q) \leftarrow \mathsf{SelfCorr.Q}(v)\right] = 1 \ .$$

Proof. We have (7) since from a union bound, we have

$$\Pr\left[\forall i \in [d + 1], f(v + ih) = \tilde{f}(v + ih) \;\middle|\; h \leftarrow \mathbb{F}^m\right] \geq 1 - (d + 1)\epsilon_1 > 1 - \epsilon_2 \ .$$

The furthermore part can be verified by inspection. □

[30] $\tilde{f}$ is the unique degree-d polynomial that is ϵ_1-close to f since $\epsilon_1 < (1 - d/|\mathbb{F}|)/2$.

Algorithm 10 Self-correction $\mathsf{SelfCorr} = (\mathsf{SelfCorr.Q}, \mathsf{SelfCorr.Rec})$

- $\mathsf{SelfCorr.Q}(v)$:
  1. Choose a uniformly random point $h \in \mathbb{F}^m$.
  2. Output $\mathsf{st} := (v, h)$ and $Q := \{v + ih\}_{i \in [d+1]}$.
- $\mathsf{SelfCorr.Rec}(\mathsf{st}, A)$:
  1. Reconstruct a (unique) degree-d polynomial $f : \mathbb{F} \to \mathbb{F}$ that satisfies $f(i) = A(v + ih)$ for every $i \in [d+1]$ (where i is viewed as an element of $\mathbb{F}$ when it is given to f).
  2. Output $f(0)$.

We next give a "public-coin" version of self-correction, which is public coin in the sense that even when the randomness of the query algorithm is published, no one can find a point for which self-correction fails (as such a point does not exist with high probability).

Lemma 10. *Let $\mathbb{F}$, m, d, ϵ_1, ϵ_2 be as in Lemma 9, and additionally assume $\epsilon_2 \le 1/12$. Then, for any $\lambda \in \mathbb{N}$ and any function $f : \mathbb{F}^m \to \mathbb{F}$ that is ϵ_1-close to a degree-d polynomial $\tilde{f}$, the algorithms $(\mathsf{SelfCorr}'.\mathsf{Q}, \mathsf{SelfCorr}'.\mathsf{Rec})$ in Algorithm 11 satisfies the following.*

$$\Pr\left[\forall v \in \mathbb{F}^m, a_v = \tilde{f}(v) \;\middle|\; \begin{array}{l} \bar{h} = (h_1, \ldots, h_\lambda) \leftarrow (\mathbb{F}^m)^\lambda \\ \forall v \in \mathbb{F}^m, (\mathsf{st}_v, Q_v) := \mathsf{SelfCorr}'.\mathsf{Q}(v; \bar{h}) \\ \forall v \in \mathbb{F}^m, a_v := \mathsf{SelfCorr}'.\mathsf{Rec}(\mathsf{st}_v, f|_{Q_v}) \end{array}\right] \ge 1 - \frac{|\mathbb{F}|^m}{2^{\lambda/2}} \ . \tag{8}$$

Furthermore, if f itself is a degree-d polynomial,

$$\Pr\left[\forall v \in \mathbb{F}^m, a_v = f(v) \;\middle|\; \begin{array}{l} \bar{h} = (h_1, \ldots, h_\lambda) \leftarrow (\mathbb{F}^m)^\lambda \\ \forall v \in \mathbb{F}^m, (\mathsf{st}_v, Q_v) := \mathsf{SelfCorr}'.\mathsf{Q}(v; \bar{h}) \\ \forall v \in \mathbb{F}^m, a_v := \mathsf{SelfCorr}'.\mathsf{Rec}(\mathsf{st}_v, f|_{Q_v}) \end{array}\right] = 1 \ .$$

Algorithm 11 Public-coin self-correction $\mathsf{SelfCorr}' = (\mathsf{SelfCorr}'.\mathsf{Q}, \mathsf{SelfCorr}'.\mathsf{Rec})$

- $\mathsf{SelfCorr}'.\mathsf{Q}(v)$:
  1. For every $i \in [\lambda]$, sample a uniformly random point $h_i \in \mathbb{F}^m$ and run $(\mathsf{st}_i, Q_i) := \mathsf{SelfCorr.Q}(v; h_i)$.
  2. Output $\mathsf{st} := (\mathsf{st}_1, \ldots, \mathsf{st}_\lambda)$ and $Q := \bigcup_{i \in [\lambda]} Q_i$.
- $\mathsf{SelfCorr}'.\mathsf{Rec}(\mathsf{st}, A)$:
  1. For every $i \in [\lambda]$, run $\tilde{a}_i := \mathsf{SelfCorr.Rec}(\mathsf{st}_i, A|_{Q_i})$.
  2. Output the most frequent value $\tilde{a}$ among $\{\tilde{a}_1, \ldots, \tilde{a}_\lambda\}$.

Proof. For every $v \in \mathbb{F}^m$, we have $a_v \ne \tilde{f}(v)$ only when $\mathsf{SelfCorr}$ with randomness h_i fails to output $\tilde{f}(v)$ for at least $\lambda/2$ values of $i \in [\lambda]$. That is, we have

$$\begin{aligned} &\Pr\left[a_v \ne \tilde{f}(v) \;\middle|\; \begin{array}{l} \bar{h} = (h_1, \ldots, h_\lambda) \leftarrow (\mathbb{F}^m)^\lambda \\ (\mathsf{st}_v, Q_v) := \mathsf{SelfCorr}'.\mathsf{Q}(v; \bar{h}) \\ a_v := \mathsf{SelfCorr}'.\mathsf{Rec}(\mathsf{st}_v, f|_{Q_v}) \end{array}\right] \\ &\le \Pr\left[\left|\{i \in [\lambda] \mid \mathsf{SelfCorr.Rec}(\mathsf{st}_i, f|_{Q_i}) \ne \tilde{f}(v)\}\right| \ge \frac{\lambda}{2} \;\middle|\; \begin{array}{l} \forall i \in [\lambda], \\ (\mathsf{st}_i, Q_i) \leftarrow \mathsf{SelfCorr.Q}(v) \end{array}\right] \end{aligned} \tag{9}$$

Since the expected number of such i is at most $\epsilon_2 \lambda \le \lambda/12$ due to Lemma 9, we can use a suitable version of the Chernoff bound (e.g., [MU05, Theorem 4.4]) to upper bound the probability in (9) by $2^{-\lambda/2}$. Thus, we obtain (8) from a union bound. The furthermore part can be verified by inspection. □

A.3.3 Global self-correction.

In a setting where Lemma 10 holds, we can make global self-correction of low-degree polynomials. That is, for $\mathbb{F}$, m, d, ϵ_1, and ϵ_2 for which Lemma 10 holds, when we are given a function $f : \mathbb{F}^m \to \mathbb{F}$ that is ϵ_1-close to a degree-d polynomial $\tilde{f} : \mathbb{F}^m \to \mathbb{F}$, we can obtain (the truth table of) $\tilde{f}$ from f efficiently by just using $\mathsf{SelfCorr}'$ for every point in $\mathbb{F}^m$. The error probability is at most $|\mathbb{F}|^m/2^{\lambda/2}$.

A.3.4 The setting that we consider.

We use the above lemmas in the setting where $|\mathbb{F}|$, m, and d are $\mathsf{poly}(\log \lambda)$ and satisfy $|\mathbb{F}| \gg d$. In this setting, we have the following.

Corollary 2. *Assume $|\mathbb{F}|$, m, and d are $\mathsf{poly}(\log \lambda)$ and satisfies $|\mathbb{F}| \gg d$.*[31]

- LDTest*: Lemma 8 with $\epsilon = 1/\lambda$ guarantees that we can reject any function f that is not $1/\lambda$-close to degree-d polynomials with error probability $1/2^{\lambda/2}$ by using the λ^2-repetition of* LDTest*. The total length of the queries is $\mathsf{poly}(\lambda)$, and the total length of the randomness is $\mathsf{poly}(\lambda)$.*
- $\mathsf{SelfCorr}$*: Lemma 9 with $\epsilon_1 = 1/\lambda$ and $\epsilon_2 = (d+2)/\lambda$ guarantees that when we are given oracle access to a function f that is $1/\lambda$-close to a degree-d polynomial $\tilde{f}$, we can evaluate $\tilde{f}$ using f with error probability $\mathsf{poly}(\log \lambda)/\lambda$. The total length of the queries is $\mathsf{poly}(\log \lambda)$ and the total length of the randomness is $\mathsf{poly}(\log \lambda)$.*
- $\mathsf{SelfCorr}'$*: Lemma 10 with $\epsilon_1 = 1/\lambda$ and $\epsilon_2 = (d+2)/\lambda$ guarantees that for sufficiently large λ (in particular, if $(d+2)/\lambda < 1/12$), when we are given oracle access to a function f that is $1/\lambda$-close to a degree-d polynomial $\tilde{f}$, we can evaluate $\tilde{f}$ using f in the public-coin manner with error probability $|\mathbb{F}|^m/2^{\lambda/2}$. The total length of the queries is $\lambda \cdot \mathsf{poly}(\log \lambda)$ and the total length of the randomness is $\lambda \cdot \mathsf{poly}(\log \lambda)$.*

B Proof of Corollary 1

Let $\mathsf{poly}_W, \mathsf{poly}_D, \mathsf{poly}_\delta$ be the polynomials that are guaranteed to exist by Lemma 5, and for $\ell_{\max}(\lambda) = \lfloor \log^2 \lambda \rfloor$ and $L_{\max}(\lambda) = 2^{\ell_{\max}(\lambda)}\lambda$, let $W(\lambda) := \mathsf{poly}_W(\lambda, L_{\max}(\lambda)) = \mathsf{poly}(\lambda^{\log \lambda})$, $D(\lambda) := \mathsf{poly}_D(\log \lambda, \ell_{\max}(\lambda)) = \mathsf{poly}(\log \lambda)$, and $\delta(\lambda) := \mathsf{poly}_\delta(D(\lambda), \log W(\lambda))$. Then, the desired non-interactive argument Π is obtained by using Lemma 4 for W, D, and δ with straightforward adaptation. For completeness, we give more details below.

- For each $\lambda, \ell \in \mathbb{N}$ such that $\ell \leq \log^2 \lambda$, the parameters $(\mathbb{F}, \mathbb{H}, m)$ is determined by obtaining $(\mathbb{H}, m)$ from Lemma 5 with sufficiently large field $\mathbb{F}$. The conditions about the parameters are satisfied because of GKR compatibility (Lemma 3).
- Setup is identical with the one obtained from Lemma 4.
- The prover P is identical with the one obtained from Lemma 4 except for the following modifications: instead of taking a circuit as input as in Lemma 4, the prover takes a hash function h as input, and it obtains a circuit C of $\mathsf{TreeHash}_h$ on its own by using Lemma 5, where the value ℓ for Lemma 5 is determined based on $|x|$.
- The verifier V is identical with the one obtained from Lemma 4 except for the following modifications: (i) instead of being given oracle access to $\mathcal{F}$ as in Lemma 4, the verifier takes a length parameter ℓ and a hash function h as input, and evaluates $\mathcal{F} = \{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i \in [D]}$ on its own by relying on Lemma 5; (ii) instead of being given x as input as in Lemma 4, the verifier is given oracle access to $\widehat{x}$ as in Remark 6; (iii) the verifier outputs 0 whenever $y \notin \{0,1\}^\lambda$.
- The input query algorithm $\mathsf{InpQuery}$ is obtained as in Remark 6.

[31] In particular, it is assumed that it holds $1/(d+1) < (1 - d/|\mathbb{F}|)/2$.

C Proof of Lemma 5

Fix any $\lambda, \ell \in \mathbb{N}$ ($\ell \le \log^2 \lambda$) and $h \in \mathcal{H}_\lambda$, where $h : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda$. Let $\mathbb{F}$ be a finite field of sufficiently large size $|\mathbb{F}| \le \mathsf{poly}(\log \lambda)$. (Concrete requirements about $|\mathbb{F}|$ are discussed at the end of the proof.) Let $L := 2^\ell \lambda$.

Let $C_h : \mathbb{F}^{2\lambda} \to \mathbb{F}$ be a layered arithmetic circuit that computes h for every input $x \in \{0,1\}^{2\lambda}$. Since $\mathcal{H}$ is polylogarithmic depth, we assume that the death D_h of C_h is bounded by $\mathsf{poly}(\log \lambda)$. Also, since h can be computed in polynomial time, we assume that the width W_h of C_h is bounded by $\mathsf{poly}(\lambda)$. We assume without loss of generality that W_h is a power of 2 and all layers of C_h have width W_h, where the first 2λ gates in the input layer are the input gates and the first λ gates in the output layer are the output gates. (The other gates in the input and output layers are dummy gates.) Also, for convenience, we assume that the last gate of each layer is guaranteed to be a dummy gate. Let $\{\mathsf{add}_{h,i}, \mathsf{mult}_{h,i}\}_{i\in[D_h]}$ denote the functions that specify C_h. Without loss of generality, we assume that C_h outputs a value in $\mathbb{F}^\lambda \setminus \{0,1\}^\lambda$ when the input is in $\mathbb{F}^{2\lambda} \setminus \{0,1\}^{2\lambda}$.

We note that layers of a depth-D circuit are indexed by $\{0, \dots, D\}$, where the input layer is "the 0-th layer" and the output layer is "the D-th layer."

C.1 Circuit C

We obtain the circuit C straightforwardly by using many copies of C_h in a tree structure as described below. First, we have $L/2\lambda$ copies of C_h, which we call the *level-0 circuits*, and their input gates serve as the input gates of C. Next, just above the level-0 circuits, we have $L/2^2\lambda$ copies of C_h, which we call the *level-1 circuits*, and their input layer are replaced with the output layer of the level-0 gates appropriately. (That is, for every $k \in \{0, \dots, L/2^2\lambda - 1\}$, the first λ input gates of the k-th level-1 circuit are replaced with the λ output gates of the $2k$-th level-0 circuit, and the other λ input gates of the k-th level-1 circuit are replaced with the λ output gates of the $(2k+1)$-st level-0 circuit.[32]) We repeat this process until we have a single copy of C_h as the level-$(\ell-1)$ circuit, and the output gates of this copy serve as the output gates of C. Finally, we add dummy gates to each layer so that all layers in C have the same width.

Clearly, C computes $\mathsf{TreeHash}_h$ for every input $x \in \{0,1\}^L$, and it outputs a value in $\mathbb{F}^\lambda \setminus \{0,1\}^\lambda$ when the input is in $\mathbb{F}^L \setminus \{0,1\}^L$ because of our assumption about C_h. The width of C is $W := W_h \cdot L/2\lambda$, and the depth of C is $D := D_h \cdot \ell$. We assume without loss of generality that C_h has the same width and depth for every $h \in \mathcal{H}_\lambda$. Therefore, we have $W = \mathsf{poly}_W(\lambda, L)$ and $D = \mathsf{poly}_D(\log \lambda, \ell)$ for polynomials $\mathsf{poly}_W, \mathsf{poly}_D$ that depend only on $\mathcal{H}$.

C.2 Parameters $(\mathbb{H}, m)$

Fix a subset $\mathbb{H} \subset \mathbb{F}$ such that $|\mathbb{H}|$ is the smallest power-of-2 that satisfies $|\mathbb{H}| \ge \max(D, \log W) = \mathsf{poly}(\log \lambda)$, and let $m := \lceil \log_{|\mathbb{H}|} W \rceil = \mathsf{poly}(\log \lambda)$ so that $W \le |\mathbb{H}|^m \le \mathsf{poly}(W)$. Note that $\mathbb{H}$ and m can be determined once poly_W, poly_D, λ, ℓ, and $\mathbb{F}$ are fixed.

C.3 Functions $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$

For any $k \in \mathbb{N}$, let $\mathsf{num}_k : \mathbb{H}^k \to \{0, \dots, |\mathbb{H}|^k - 1\}$ be the function that outputs the lexicographical order of the input. Recall that $\{\mathsf{add}_{h,i}, \mathsf{mult}_{h,i}\}_{i\in[D_h]}$ are the functions that specify C_h. Let $\{\widetilde{\mathsf{add}}_{h,i}, \widetilde{\mathsf{mult}}_{h,i} : \mathbb{F}^{3m_h} \to \mathbb{F}\}_{i\in[D_h]}$ denote the LDE of $\{\mathsf{add}_{h,i}, \mathsf{mult}_{h,i}\}_{i\in[D_h]}$ w.r.t. $\mathbb{F}$, $\mathbb{H}$, and m_h, where $m_h := \lceil \log_{|\mathbb{H}|} W_h \rceil$ so that $|\mathbb{H}|^{m_h - 1} < W_h \le |\mathbb{H}|^{m_h}$. Note that $\{\widetilde{\mathsf{add}}_{h,i}, \widetilde{\mathsf{mult}}_{h,i}\}_{i\in[D_h]}$ have individual degree at most $|\mathbb{H}|$, and since we have $W_h = \mathsf{poly}(\lambda)$, they can be evaluated in time $|\mathbb{H}|^{3m_h} \cdot \mathsf{poly}(m_h, |\mathbb{H}|) \le \mathsf{poly}(\lambda)$ given C_h (cf. Section 3.7).

We now describe the polynomials $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$. Recall that $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$ need to be extensions of the functions $\{\mathsf{add}_i, \mathsf{mult}_i\}_{i\in[D]}$ that specify C. Below, we define $\widetilde{\mathsf{add}}_i$ by considering 3 cases depending on the value of $i \in [D]$. ($\widetilde{\mathsf{mult}}_i$ can be defined similarly.) In the following, we say that layer i of C *corresponds* to layer i_h of C_h if the i-th layer of C is the i_h-th layer of the level-κ circuits for some κ.

[32] The dummy gates in the input layer of each level-1 circuit (which we have added to C_h so that the input layer of C_h has width W_h) are simply removed so that we have layer 1 of the level-1 circuits just above the output layer of the level-0 circuits.

Case 1. Layer i of C corresponds to layer i_h of C_h for some $i_h \geq 2$. Let $\kappa \in \mathbb{N}$ be the integer such that both the i-th and the $(i-1)$-st layers of C belong to the level-κ circuits. (Such κ is guaranteed to exist because of the condition of this case.) We note that by the construction of C, the level-κ circuits consist of $L/2^{\kappa+1}\lambda = 2^{\ell-\kappa-1}$ copies of C_h.

As a preliminary step, we first give a pseudocode that computes $\mathsf{add}_i : \{0,\dots,W-1\}^3 \to \{0,1\}$. Note that for any $j_1, j_2, j_3 \in \{0,\dots,W-1\}$, we can compute $\mathsf{add}_i(j_1,j_2,j_3)$ by using add_{h,i_h} trivially if gates $g_{i,j_1}, g_{i-1,j_2}, g_{i-1,j_3}$ of C belong to the same copy of C_h, and we have $\mathsf{add}_i(j_1,j_2,j_3) = 0$ in any other cases. Thus, observing that gate g_{i,j_1} corresponds to the $(j_1 \bmod W_h)$-th gate in the i_h-th layer of the $\lfloor j_1/W_h \rfloor$-th level-κ circuit (assuming that the $\lfloor j_1/W_h \rfloor$-th level-κ circuit exist, i.e., $\lfloor j_1/W_h \rfloor \leq 2^{\ell-\kappa-1}-1$) and observing the same for g_{i-1,j_2} and g_{i-1,j_3}, we consider computing $\mathsf{add}_i(j_1,j_2,j_3)$ as follows.

if $\lfloor j_1/W_h \rfloor = \lfloor j_2/W_h \rfloor = \lfloor j_3/W_h \rfloor$ and $\lfloor j_1/W_h \rfloor \leq 2^{\ell-\kappa-1}-1$ **then**
  return $\mathsf{add}_{h,i_h}(j_1 \bmod W_h, j_2 \bmod W_h, j_3 \bmod W_h)$
else
  return 0
end if

Now, we define $\widetilde{\mathsf{add}}_i$ based on the above pseudocode.

1. First, we obtain a polynomial $\widetilde{\mathsf{sameC}_\mathsf{h}} : \mathbb{F}^{2m} \to \mathbb{F}$ that checks whether two gates belong to the same copy of C_h. Concretely, when given $(v,v') \in \mathbb{H}^{2m}$ as input, $\widetilde{\mathsf{sameC}_\mathsf{h}}$ outputs 1 if $\lfloor \mathsf{num}_m(v)/W_h \rfloor = \lfloor \mathsf{num}_m(v')/W_h \rfloor$ and outputs 0 otherwise. (Recall that $\mathsf{num}_m : \mathbb{H}^m \to \{0,\dots,|\mathbb{H}|^m-1\}$ is the function that outputs the lexicographical order of the input.) Let $\mathsf{EQ} : \mathbb{H}^2 \to \{0,1\}$ be the function that outputs 1 if and only if the input $(v,v') \in \mathbb{H}^2$ satisfies $v = v'$, and let $\widetilde{\mathsf{EQ}} : \mathbb{F}^2 \to \mathbb{F}$ be its LDE. Let $\mathsf{EQ}' : \mathbb{H}^{2m_h} \to \{0,1\}$ be the function that outputs 1 if and only if the input $(v,v') \in \mathbb{H}^{2m_h}$ satisfies $\lfloor \mathsf{num}_{m_h}(v)/W_h \rfloor = \lfloor \mathsf{num}_{m_h}(v')/W_h \rfloor$, and let $\widetilde{\mathsf{EQ}}' : \mathbb{F}^{2m_h} \to \mathbb{F}$ be its LDE. Then, for any $z = (z_m,\dots,z_1), z' = (z'_m,\dots,z'_1) \in \mathbb{F}^m$, let

$$\widetilde{\mathsf{sameC}_\mathsf{h}}(z,z') \coloneqq \widetilde{\mathsf{EQ}'}(z_{\leq m_h}, z'_{\leq m_h}) \prod_{k=m_h+1}^{m} \widetilde{\mathsf{EQ}}(z_k, z'_k) \ ,$$

where $z_{\leq m_h} \coloneqq (z_{m_h},\dots,z_1)$ and $z'_{\leq m_h} \coloneqq (z'_{m_h},\dots,z'_1)$.

It is easy to see that given $(v,v') \in \mathbb{H}^{2m}$ as input, $\widetilde{\mathsf{sameC}_\mathsf{h}}$ indeed outputs 1 if $\lfloor \mathsf{num}_m(v)/W_h \rfloor = \lfloor \mathsf{num}_m(v')/W_h \rfloor$ and outputs 0 otherwise.[33] Note that $\widetilde{\mathsf{EQ}}$ has individual degree at most $|\mathbb{H}|$ and can be evaluated in time $|\mathbb{H}|^2 \cdot \mathsf{poly}(|\mathbb{H}|)$, and $\widetilde{\mathsf{EQ}}'$ has individual degree at most $|\mathbb{H}|$ and can be evaluated in time $|\mathbb{H}|^{2m_h} \cdot \mathsf{poly}(m_h, |\mathbb{H}|)$. Thus, $\widetilde{\mathsf{sameC}_\mathsf{h}}$ has individual degree at most $|\mathbb{H}|$ and can be evaluated in time $\mathsf{poly}(\lambda)$.

2. Next, we obtain a polynomial $\widetilde{\mathsf{valid}} : \mathbb{F}^m \to \mathbb{F}$ that checks whether a gate belongs to a copy of C_h. Concretely, when given $v \in \mathbb{H}^m$ as input, $\widetilde{\mathsf{valid}}$ outputs 1 if $\lfloor \mathsf{num}_m(v)/W_h \rfloor \leq 2^{\ell-\kappa-1}-1$ and outputs 0 otherwise. Let $m' \coloneqq \lceil \log_{|\mathbb{H}|} 2^{\ell-\kappa+\log W_h} \rceil$ so that we have $(m'-1)\log|\mathbb{H}| < \ell - \kappa + \log W_h \leq m' \log|\mathbb{H}|$. Let $\widetilde{\mathsf{EQ}}$ be defined as above. Let $\mathsf{EQ}'' : \mathbb{H}^2 \to \{0,1\}$ be the function that outputs 1 if and only if the input $(v,v') \in \mathbb{H}^2$ satisfies $\lfloor \mathsf{num}_1(v)/2^{\ell-\kappa-1+\log W_h-(m'-1)\log|\mathbb{H}|} \rfloor = \lfloor \mathsf{num}_1(v')/2^{\ell-\kappa-1+\log W_h-(m'-1)\log|\mathbb{H}|} \rfloor$, and let $\widetilde{\mathsf{EQ}}''$: $\mathbb{F}^2 \to \mathbb{F}$ be its LDE. Then, for any $z = (z_m,\dots,z_1) \in \mathbb{F}^m$, let

$$\widetilde{\mathsf{valid}}(z) \coloneqq \widetilde{\mathsf{EQ}}''(z_{m'}, 0) \prod_{k=m'+1}^{m} \widetilde{\mathsf{EQ}}(z_k, 0) \ .$$

[33] To see this, observe the following. Since we assume that W_h is a power of 2, we have $\lfloor \mathsf{num}_m(v)/W_h \rfloor = \lfloor \mathsf{num}_m(v')/W_h \rfloor$ if and only if the binary representations of $\mathsf{num}_m(v)$ and $\mathsf{num}_m(v')$ are equal except for the least significant $\log W_h$ bits. Also, since we assume that $|\mathbb{H}|$ is a power of 2, the binary representation of $\mathsf{num}_m(v)$ (resp., $\mathsf{num}_m(v')$) is the concatenation of the binary representations of $\mathsf{num}_1(v_m),\dots,\mathsf{num}_1(v_1)$ (resp., $\mathsf{num}_1(v'_m),\dots,\mathsf{num}_1(v'_1)$). Therefore, we have $\lfloor \mathsf{num}_m(v)/W_h \rfloor = \lfloor \mathsf{num}_m(v')/W_h \rfloor$ if and only if (i) the binary representations of $\mathsf{num}_{m_h}(v_{\leq m_h})$ and $\mathsf{num}_{m_h}(v'_{\leq m_h})$ are equal except for the least significant $\log W_h$ bits (i.e., $\lfloor \mathsf{num}_{m_h}(v_{\leq m_h})/W_h \rfloor = \lfloor \mathsf{num}_{m_h}(v'_{\leq m_h})/W_h \rfloor$) and (ii) $\mathsf{num}_1(v_k) = \mathsf{num}_1(v_k)$ for every $k \in \{m_h+1,\dots,m\}$.

It is easy to see that given $v \in \mathbb{H}^m$ as input, $\widetilde{\mathsf{valid}}$ indeed outputs 1 if $\lfloor \mathsf{num}_m(v)/W_h \rfloor \leq 2^{\ell-\kappa-1}-1$ and outputs 0 otherwise.[34] Note that $\widetilde{\mathsf{EQ}}''$ has individual degree at most $|\mathbb{H}|$ and can be evaluated in time $|\mathbb{H}|^2 \cdot \mathsf{poly}(|\mathbb{H}|)$. Thus, $\widetilde{\mathsf{valid}}$ has individual degree at most $|\mathbb{H}|$ and can be evaluated in time $\mathsf{poly}(\log \lambda)$.

3. Next, we obtain a set of polynomials $\widetilde{\mathsf{relative\text{-}index}} = (\widetilde{\mathsf{relative\text{-}index}}_{m_h}, \ldots, \widetilde{\mathsf{relative\text{-}index}}_1)$ that take an index of a gate in C and convert it to the index that the gate has as a gate in C_h. Concretely, it holds $\widetilde{\mathsf{relative\text{-}index}}_k : \mathbb{F}^{m_h} \to \mathbb{F}$ for each $k \in [m_h]$, and for any input $v = (v_m, \ldots, v_1) \in \mathbb{H}^m$, the output $v' := \widetilde{\mathsf{relative\text{-}index}}(v) := (\widetilde{\mathsf{relative\text{-}index}}_{m_h}(v_{\leq m_h}), \ldots, \widetilde{\mathsf{relative\text{-}index}}_1(v_{\leq m_h})) \in \mathbb{H}^{m_h}$ satisfies $\mathsf{num}_{m_h}(v') = \mathsf{num}_m(v) \bmod W_h$. Let $\mathsf{relative\text{-}index} = (\mathsf{relative\text{-}index}_1, \ldots, \mathsf{relative\text{-}index}_{m_h})$ be the set of the functions such that $\mathsf{relative\text{-}index}_k : \mathbb{H}^{m_h} \to \mathbb{H}$ for each $k \in [m_h]$, and for any input $v = (v_{m_h}, \ldots, v_1) \in \mathbb{H}^{m_h}$, the output $v' := \mathsf{relative\text{-}index}(v) := (\mathsf{relative\text{-}index}_{m_h}(v), \ldots, \mathsf{relative\text{-}index}_1(v)) \in \mathbb{H}^{m_h}$ satisfies $\mathsf{num}_{m_h}(v') = \mathsf{num}_{m_h}(v) \bmod W_h$. Then, each $\widetilde{\mathsf{relative\text{-}index}}_k$ is the LDE of $\mathsf{relative\text{-}index}_k$.

   It is easy to see that given $v \in \mathbb{H}^m$ as input, $\widetilde{\mathsf{relative\text{-}index}}$ indeed outputs $v' \in \mathbb{H}^{m_h}$ such that $\mathsf{num}_{m_h}(v') = \mathsf{num}_m(v) \bmod W_h$. Note that each $\widetilde{\mathsf{relative\text{-}index}}_k$ has individual degree at most $|\mathbb{H}|$ and can be evaluated in time $|\mathbb{H}|^{m_h} \cdot \mathsf{poly}(m_h, |\mathbb{H}|) = \mathsf{poly}(\lambda)$.

4. Finally, we define $\widetilde{\mathsf{add}}_i$ as follows. For every $z_1, z_2, z_3 \in \mathbb{F}^m$,

$$\widetilde{\mathsf{add}}_i(z_1, z_2, z_3) := \widetilde{\mathsf{same}\mathsf{C}_\mathsf{h}}(z_1, z_2) \cdot \widetilde{\mathsf{same}\mathsf{C}_\mathsf{h}}(z_2, z_3) \cdot \widetilde{\mathsf{valid}}(z_1) \times \widetilde{\mathsf{add}}_{h,i_h}(\widetilde{\mathsf{relative\text{-}index}}(z_1), \widetilde{\mathsf{relative\text{-}index}}(z_2), \widetilde{\mathsf{relative\text{-}index}}(z_3)) \ .$$

It is easy to see that $\widetilde{\mathsf{add}}_i$ is indeed an extension of add_i (cf. the above pseudocode). It has individual degree at most $3m_h|\mathbb{H}|^2$ and can be evaluated in time $\mathsf{poly}(\lambda)$.

Case 2. Layer i of C corresponds to layer 1 of the level-0 circuits. In this case, the i-th and the $(i-1)$-st layers belong to the level-0 circuits. Therefore, it suffices to define $\widetilde{\mathsf{add}}_i$ as in Case 1.

Case 3. Layer i of C corresponds to layer 1 of the level-κ circuits for some $\kappa > 0$. As a preliminary step, we first give a pseudocode that computes $\mathsf{add}_i : \{0, \ldots, W-1\}^3 \to \{0,1\}$. Note that the $(i-1)$-st layer of C corresponds to the output layer of the level-$(\kappa-1)$ circuits. Thus, for any $j_1, j_2, j_3 \in \{0, \ldots, W-1\}$, we can compute $\mathsf{add}_i(j_1, j_2, j_3)$ by using $\mathsf{add}_{h,1}$ if the copy of C_h that the gate g_{i,j_1} belongs to is the parent of those that the gates g_{i-1,j_2}, g_{i-1,j_3} belong to (i.e., if $\lfloor j_1/W_h \rfloor = \lfloor \lfloor j_2/W_h \rfloor /2 \rfloor = \lfloor \lfloor j_3/W_h \rfloor /2 \rfloor$). Also, by construction, each copy of C_h in the level-κ circuits uses the λ output gates of its left-child C_h as the first λ gates of its input layer, and uses the λ output gates of its right-child as the next λ gates of its input layer. Thus, we consider computing $\mathsf{add}_i(j_1, j_2, j_3)$ as follows.[35]

if $\lfloor j_1/W_h \rfloor = \lfloor \lfloor j_2/W_h \rfloor /2 \rfloor = \lfloor \lfloor j_3/W_h \rfloor /2 \rfloor$ and $\lfloor j_1/W_h \rfloor \leq 2^{\ell-\kappa-1}-1$ **then**
  $j_1' := j_1 \bmod W_h$
  if $j_2 \bmod W_h < \lambda$ **then**
    $j_2' := j_2 \bmod W_h + (\lfloor j_2/W_h \rfloor \bmod 2) \cdot \lambda$
  else
    $j_2' := W_h - 1$
  end if
  if $j_3 \bmod W_h < \lambda$ **then**
    $j_3' := j_3 \bmod W_h + (\lfloor j_3/W_h \rfloor \bmod 2) \cdot \lambda$
  else
    $j_3' := W_h - 1$

[34] To see this, observe that we have $\lfloor \mathsf{num}_m(v)/W_h \rfloor \leq 2^{\ell-\kappa-1}-1$ if and only if each bit in the binary representation of $\mathsf{num}_m(v)$ is 0 except for the least significant $\ell - \kappa - 1 + \log W_h$ bits.

[35] This pseudocode relies on our assumption that the last gate of each layer of C_h (i.e., the (W_h-1)-st gate) is guaranteed to be a dummy gate. In particular, it is used to force $\mathsf{add}_{h,1}$ to output 0 when $j_2 \bmod W_h \geq \lambda$ and $j_3 \bmod W_h \geq \lambda$.

12: **end if**
13: **return** $\mathsf{add}_{h,1}(j'_1, j'_2, j'_3)$
14: **else**
15: **return** 0
16: **end if**

Now, we define $\widetilde{\mathsf{add}}_i$ following the above pseudocode.

1. First, we obtain a polynomial $\widetilde{\mathsf{child}} : \mathbb{F}^{2m} \to \mathbb{F}$ that takes two gates as input and checks whether the copy of C_h that the second gate belongs to is a child of the one that the first one belongs to. Concretely, when given $(v, v') \in \mathbb{H}^{2m}$ as input, $\widetilde{\mathsf{child}}$ outputs 1 if $\lfloor \mathsf{num}_m(v)/W_h \rfloor = \lfloor \lfloor \mathsf{num}_m(v')/W_h \rfloor /2 \rfloor$ and outputs 0 otherwise. Let $\mathsf{right\text{-}shift} : \mathbb{H}^2 \to \mathbb{H}$ be the function such that when given input $(v, v') \in \mathbb{H}^2$, it outputs $\bar{v}' \in \mathbb{H}$ such that $\mathsf{num}_1(\bar{v}') = \lfloor \mathsf{num}_1(v')/2 \rfloor + (\mathsf{num}_1(v) \bmod 2) \cdot |\mathbb{H}|/2$, and $\widetilde{\mathsf{right\text{-}shift}}$ be its LDE. Let $\widetilde{\mathsf{sameC_h}}$ be the polynomial that is defined in Case 1. Then, for any $z = (z_m, \ldots, z_1), z' = (z'_m, \ldots, z'_1) \in \mathbb{F}^m$, let

$$\widetilde{\mathsf{child}}(z, z') \coloneqq \widetilde{\mathsf{sameC_h}}(z, \bar{z}'),$$
$$\text{where } \bar{z}' \coloneqq (\widetilde{\mathsf{right\text{-}shift}}(0, z'_m), \widetilde{\mathsf{right\text{-}shift}}(z'_m, z'_{m-1}), \ldots, \widetilde{\mathsf{right\text{-}shift}}(z'_2, z'_1)) \ .$$

   It is easy to see that given $(v, v') \in \mathbb{H}^{2m}$ as input, $\widetilde{\mathsf{child}}$ indeed outputs 1 if $\lfloor \mathsf{num}_m(v)/W_h \rfloor = \lfloor \lfloor \mathsf{num}_m(v')/W_h \rfloor /2 \rfloor$ and outputs 0 otherwise.[36] Note that $\widetilde{\mathsf{right\text{-}shift}}$ has individual degree at most $|\mathbb{H}|$ and can be evaluated in time $|\mathbb{H}|^2 \cdot \mathsf{poly}(|\mathbb{H}|)$, and $\widetilde{\mathsf{sameC_h}}$ has individual degree at most $|\mathbb{H}|$ and can be evaluated in time $\mathsf{poly}(\lambda)$. Thus, $\widetilde{\mathsf{child}}$ has individual degree at most $2|\mathbb{H}|^2$ and can be evaluated in time $\mathsf{poly}(\lambda)$.

2. Next, we obtain a set of polynomials $\widetilde{\mathsf{relative\text{-}index}}' = (\widetilde{\mathsf{relative\text{-}index}}'_{m_h}, \ldots, \widetilde{\mathsf{relative\text{-}index}}'_1)$ such that it holds $\widetilde{\mathsf{relative\text{-}index}}'_k : \mathbb{F}^{m_h+1} \to \mathbb{F}$ for each $k \in [m_h]$, and for any input $v = (v_m, \ldots, v_1) \in \mathbb{H}^m$, the output $v' \coloneqq \widetilde{\mathsf{relative\text{-}index}}'(v) \coloneqq (\widetilde{\mathsf{relative\text{-}index}}'_{m_h}(v_{\leq m_h+1}), \ldots, \widetilde{\mathsf{relative\text{-}index}}'_1(v_{\leq m_h+1})) \in \mathbb{H}^{m_h}$ satisfies

$$\mathsf{num}_{m_h}(v') = \begin{cases} \mathsf{num}_m(v) \bmod W_h + (\lfloor \mathsf{num}_m(v)/W_h \rfloor \bmod 2) \cdot \lambda & (\text{if } \mathsf{num}_m(v) \bmod W_h < \lambda) \\ W_h - 1 & (\text{otherwise}) \end{cases}$$

   Let $\mathsf{relative\text{-}index}' = (\mathsf{relative\text{-}index}'_{m_h}, \ldots, \mathsf{relative\text{-}index}'_1)$ be the set of the functions such that it holds $\mathsf{relative\text{-}index}'_k : \mathbb{H}^{m_h+1} \to \mathbb{H}$ for each $k \in [m_h]$, and for any input $v = (v_{m_h+1}, \ldots, v_1) \in \mathbb{H}^{m_h+1}$, the output $v' \coloneqq \mathsf{relative\text{-}index}'(v) \coloneqq (\mathsf{relative\text{-}index}'_{m_h}(v), \ldots, \mathsf{relative\text{-}index}'_1(v)) \in \mathbb{H}^{m_h}$ satisfies

$$\mathsf{num}_{m_h}(v') = \begin{cases} \mathsf{num}_{m_h+1}(v) \bmod W_h + (\lfloor \mathsf{num}_{m_h+1}(v)/W_h \rfloor \bmod 2) \cdot \lambda & (\text{if } \mathsf{num}_{m_h+1}(v) \bmod W_h < \lambda) \\ W_h - 1 & (\text{otherwise}) \end{cases}$$

   Then, each $\widetilde{\mathsf{relative\text{-}index}}'_k$ is the LDE of $\mathsf{relative\text{-}index}'_k$.

   It is easy to see that given $v \in \mathbb{H}^m$ as input, $\widetilde{\mathsf{relative\text{-}index}}'$ outputs the desired point in $\mathbb{H}^{m_h}$. Note that each $\widetilde{\mathsf{relative\text{-}index}}'_k$ has individual degree at most $|\mathbb{H}|$ and can be obtained and evaluated in time $|\mathbb{H}|^{m_h+1} \cdot \mathsf{poly}(m_h, |\mathbb{H}|) = \mathsf{poly}(\lambda)$.

3. Finally, we define $\widetilde{\mathsf{add}}_i$ as follows. Let $\widetilde{\mathsf{valid}}$ and $\widetilde{\mathsf{relative\text{-}index}}$ be defined as in Case 1. Then, for every $z_1, z_2, z_3 \in \mathbb{F}^m$,

$$\widetilde{\mathsf{add}}_i(z_1, z_2, z_3) \coloneqq \widetilde{\mathsf{child}}(z_1, z_2) \cdot \widetilde{\mathsf{child}}(z_1, z_3) \cdot \widetilde{\mathsf{valid}}(z_1)$$
$$\times \widetilde{\mathsf{add}}_{h,1}(\widetilde{\mathsf{relative\text{-}index}}(z_1), \widetilde{\mathsf{relative\text{-}index}}'(z_2), \widetilde{\mathsf{relative\text{-}index}}'(z_3)) \ .$$

[36] To see this, observe that we have $\lfloor \mathsf{num}_m(v)/W_h \rfloor = \lfloor \lfloor \mathsf{num}_m(v')/W_h \rfloor /2 \rfloor$ if and only if the most significant $(m \log|\mathbb{H}| - \log W_h)$ bits of the binary representation of $\mathsf{num}_m(v)$ are equal to the most significant $(m \log|\mathbb{H}| - \log W_h - 1)$ bits of the binary representation of $\mathsf{num}_m(v)$ with preceding 0.

It is easy to see that $\widetilde{\mathsf{add}}_i$ is indeed an extension of add_i (cf. the above pseudocode). It has individual degree at most $6m_h|\mathbb{H}|^2$ and can be evaluated in time $\mathsf{poly}(\lambda)$.

C.4 GKR Compatibility

Let poly_δ be a polynomial such that $6m_h|\mathbb{H}|^2 \leq \mathsf{poly}_\delta(D, \log W) \leq \mathsf{poly}(\log \lambda)$. Note that from the above analysis, the extensions $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$ have individual degree at most $\delta := \mathsf{poly}_\delta(D, \log W)$. Also, we have $\delta > m(|\mathbb{H}| - 1)$. Finally, since $|\mathbb{H}|$ is determined by D and $\log W$, the polynomial poly_δ can be determined by $\mathcal{H}$.

For the field $\mathbb{F}$, we require that $c_{\text{GKR}} Dm\delta \leq |\mathbb{F}| \leq \mathsf{poly}(|\mathbb{H}|)$, where c_{GKR} is the constant that is used in the definition of GKR compatibility (Lemma 3). Since m, $|\mathbb{H}|$, and $\delta := \mathsf{poly}_\delta(D, \log W)$ are determined by $D = \mathsf{poly}_D(\log \lambda, \ell)$ and $W = \mathsf{poly}_W(\lambda, L)$, the field $\mathbb{F}$ can be determined once λ and ℓ are fixed.

Now, the GKR compatibility of $(C, \mathbb{F}, \mathbb{H}, m)$ can be verified by inspection. This completes the proof of Lemma 5.