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

Susumu Kiyoshima

NTT Research, Sunnyvale, CA, USA
`susumu.kiyoshima@ntt-research.com`

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.)

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 [25]. Obtaining 3-round ZK arguments is notoriously hard, and obtaining 3-round ZK arguments with black-box simulations is known to be impossible [23]. Until recently, 3-round ZK arguments had been obtained only under unfalsifiable knowledge-type assumptions (e.g., [27, 4, 16, 7, 9]) or under weak security definitions (e.g., [38, 8, 6, 30, 36, 1, 12, 11, 20][1]).

Recently, Bitansky, Kalai, and Paneth [10] obtained a 3-round ZK argument by relying on super-polynomial hardness of the learning with errors (LWE) assumption [39] and *keyless multi-collision-resistant hash functions.*[2] Multi-collision resistance [5, 10, 37] 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 $h \in \mathcal{H}$,

⋆ The full version of this paper is available at `https://ia.cr/2022/820`.

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.

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 [10, 12, 13], succinct arguments [10], and non-malleable commitments [12]. (See [10] for more about keyless multi-collision-resistant hash functions.)

Given the result of Bitansky et al. [10], 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 include the classical ZK proofs of Goldreich, Micali, and Wigderson [24] and Blum [14].) 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* [21] and *resettable soundness* [3]. The 3-round ZK argument of Bitansky et al. [10] (as well as the subsequent 3-round statistical ZK argument of Bitansky and Paneth [13]) 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.*

(See Theorem 2 in Section 7 for the formal description.) The assumptions that we use are similar to those that are used by Bitansky et al. [10] 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. [10] 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 [10].

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

2.1 Techniques of Bitansky et al. [10]

The main component of the 3-round ZK argument of Bitansky et al. [10] is a memory delegation scheme [19, 6]. In the setting considered in [6, 10], memory delegation schemes proceed in 3 rounds.

1. The prover sends the verifier a short digest of a long memory string.
2. The verifier chooses a computation to be executed on the memory, and sends the prover the description of the computation 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. [10] 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. [10] 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 [35] (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

[4] In Bitansky et al. [10], 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.

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. [10] 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. [10] 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 delegation scheme by following the idea of an earlier work [6] (roughly speaking, the idea is to reduce the round complexity of the public-coin ZK argument of Barak [2] 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 [32, 29, 17].

Failed attempt #1. First, let us consider using the result of Choudhuri, Jain, and Jin [17] 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. [17] 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. [17] so that it works even when the memory is hashed by a keyless multi-collision-resistant hash function.

[5] Their SNARG works in the common random string model and therefore can be viewed as a public-coin 2-round delegation scheme.

Unfortunately, this approach does not work (at least when naively implemented) since the scheme of Choudhuri et al. [17] 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. [10] for multi-collision-resistant hash functions does not allow such individual opening.) At a high level, the scheme of Choudhuri et al. [17] 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 [32] that obtains a SNARG from the public-coin interactive proof of Goldwasser, Kalai, and Rothblum [26] (the GKR interactive proof in short). The scheme of Jawale et al. [32], 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. [10] 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. [32] 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. [32] requires it.) The problem is that the scheme of Jawale et al. [32] 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. [17] and the scheme of Jawale et al. [32]. We obtain our memory delegation scheme in two steps.

[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.

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. [32] and a keyless multi-collision-resistant hash function as suggested above. The key point is that, as already observed by Goldwasser et al. [26], 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. [32] 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. [17]. 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. [17].
   (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. [32] 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. [32] naively, the encoding needs to be super-polynomially long since the scheme of Jawale et al. [32] uses the GKR interactive scheme with modified parameters.[8] Almost the same problem was already observed in a different context by Bronfman and Rothblum [15], and we avoid our problem as in their work. Namely, instead of directly using the result of Jawale et al. [32], we use the result of Holmgren, Lombardi, and Rothblum [29] that shows, based on Jawale et al. [32], 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 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] $\diamondsuit$

Remark 2 (Our actual approach). Like Bitansky et al. [10], we first focus on *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. (Importantly, we do not need keyless multi-collision-resistant hash functions to construct them.) After obtaining an oracle memory delegation scheme, we upgrade it to a memory delegation scheme by using a keyless multi-collision-resistant hash function. $\diamondsuit$

3 Preliminaries

We denote the security parameter by λ. Due to the space constraint, we only give a minimal set of definitions below. Additional definitions are given in the full version of this paper.

3.1 Notations for (Keyed) Hash Functions

Informally, for a hash function family $\mathcal{H}$, we use $\mathcal{H}_\lambda$ to denote the set of the hash functions that can be used for the security parameter λ. (See the full version of

[8] If the reader is familiar with the GKR interactive proof, we note that the scheme of Jawale et al. [32] 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.

[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.

this paper for the formal meaning.) We usually assume that each $h \in \mathcal{H}_\lambda$ hashes a string of length 2λ to a string of length λ. For a hash function h, we use $\mathsf{TreeHash}_h$ to denote the algorithm that computes tree-hashing using h.

3.2 Keyless Multi-Collision Resistant Hash Functions

We recall the definition of multi-collision resistant hash functions from [10], focusing on the keyless version.

Definition 1. *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 [10], we focus on the case that Hash is polynomially compressing in the sense that $\mathsf{Hash}(1^\lambda, \cdot)$ takes a string of length λ^2 as input and outputs a string of length λ.

3.3 Weak Memory Delegations

We recall the definition of 2-round weak memory delegation schemes [10], focusing on the keyless setting and the publicly verifiable version of the definition.

Definition 2. *We say that an efficiently samplable distribution ensemble $\{Y_\lambda\}_{\lambda\in\mathbb{N}}$ is* entropic *if $H_\infty(Y_\lambda) := -\log \max_{y\in\mathrm{supp}(Y_\lambda)} \Pr[Y_\lambda = y] = \Omega(\lambda)$.*

Definition 3. *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.*

- $\mathsf{dig} := \mathsf{Mem}(1^\lambda, \mathsf{DB})$*:* 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* dig *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.*
- $b := \mathsf{Ver}(\mathsf{dig}, \langle M, t, y\rangle, q, \pi)$*:* Ver *is a deterministic algorithm that takes as input a digest* dig*, 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 $\mathsf{poly}_P, \mathsf{poly}_V$ 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 $|\mathsf{DB}| \le t \le \lambda^{\log \lambda}$, (i) $\mathsf{Prove}(\mathsf{DB}, \langle M, t, y\rangle, q)$ runs in time $\mathsf{poly}_P(\lambda, t)$, and (ii) $\mathsf{Ver}(\mathsf{dig}, \langle M, t, y\rangle, q, \pi)$ 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 $|\mathsf{DB}| \le t \le \bar{t}(\lambda)$,*[10]

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

- ***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 $t \le \bar{t}^{O(1)}(\lambda)$,*

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

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.*

3.4 Oracle Memory Delegations

We recall the definition of 2-round oracle memory delegation schemes [10]. 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 4. *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.*

- $\widehat{\mathsf{DB}} := \mathsf{Mem}(1^\lambda, \mathsf{DB})$*: 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.*

[10] We consider a slightly weaker notion of correctness where t is at most $\bar{t}(\lambda)$. (In [10], t is at most 2^λ.)

- $\pi := \mathsf{Prove}(\mathsf{DB}, \langle M, t, y\rangle, q)$*:* 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*.*

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* $|\mathsf{DB}| \le t \le \lambda^{\log \lambda}$*, (i)* $\mathsf{Prove}(\mathsf{DB}, \langle M, t, y\rangle, q)$ *runs in time* $\mathsf{poly}_P(\lambda, t)$*, and (ii)* $\mathsf{Ver}^{(\cdot)}(|\mathsf{DB}|, \langle M, t, y\rangle, q, \sigma, \pi)$ *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* $|\mathsf{DB}| \le t \le \bar{t}(\lambda)$*,*

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

- 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* $t \le \bar{t}^{O(1)}(\lambda)$*,*

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

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 [10]). 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'). $\Diamond$

3.5 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 $N \le |\mathbb{H}|^m$, 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., [26, Claim 2.3]).

4 Public-Coin Tree-Hash Oracle Memory Delegation

In this section, we construct a public-coin *tree-hash oracle memory delegation scheme*, i.e., a public-coin oracle memory delegation scheme that is focused on proving the correctness of tree-hash computations. Specifically, we consider a scheme that satisfies the following tailored soundness notion.

Definition 5. *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 4) 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.*[11]
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*

[11] 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.

$(\mathcal{A}_1, \mathcal{A}_2)$ 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 $h \in \mathcal{H}_\lambda$,*

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

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 a L_{DB}-bit string (or $\bot$).*

The goal of this section is to show the following lemma.

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

As a preliminary step, we construct a scheme with a weak soundness notion (where the cheating prover is required to give a valid encoding of a memory).

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. (The differences from Definition 5 are highlighted by underlines.)*

– ***Weak γ-soundness.*** *There exists a deterministic polynomial-time algorithm* Decode *<u>and a predicate</u>* <u>Valid</u> *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 $h \in \mathcal{H}_\lambda$,*

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

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 a L_{DB}-bit string (or $\bot$).*

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 [32, 29] about soundly applying the Fiat–Shamir transformation to the public-coin interactive proof of Goldwasser, Kalai, and Rothblum [26] (the GKR interactive proof in short).

Preliminary 1: the GKR interactive proof. We recall the “bare-bones” version of the GKR interactive proof [26, Section 3], focusing on the parts that are relevant to us. (As in [32], we use a slightly modified version of it [33] so that we can use the recent results about the Fiat–Shamir transformation [32, 29].)

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 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.

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

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

Extensions $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$ do not need to be the LDEs of $\{\mathsf{add}_i, \mathsf{mult}_i\}_{i\in[D]}$, but 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 [26], 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.[12]) 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 [26, 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 $c_{\mathrm{GKR}} \in \mathbb{N}$ 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 $\delta \in \mathbb{N}$ ($m(|\mathbb{H}| - 1) \le \delta < |\mathbb{F}|$) for which the following hold.*[13]
  1. *The field $\mathbb{F}$ is large; concretely, $c_{\mathrm{GKR}} D m \delta \le |\mathbb{F}| \le \mathsf{poly}(|\mathbb{H}|)$.*
  2. *The parameters $\mathbb{H}$ and m satisfy $\max(D, \log W) \le |\mathbb{H}| \le \mathsf{poly}(D, \log W)$ and $W \le |\mathbb{H}|^m \le \mathsf{poly}(W)$.*

[12] More precisely, in [26], 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 [22], it is observed that the verifier can evaluate $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$ efficiently.

[13] For convenience, we use a slightly stronger lower bound for δ. (In [26], the requirement is $|\mathbb{H}| - 1 \le \delta < |\mathbb{F}|$.) See Footnote 14.

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* $\widehat{x}|_{\mathbb{H}^m}$*.*[14] *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*[15] *(where the queries are determined by the (public) randomness of the verifier) and makes* $O(D)$ *queries to* $\{\widetilde{\mathsf{add}}_i, \widetilde{\mathsf{mult}}_i\}_{i\in[D]}$*.*

Preliminary 2: the Fiat–Shamir transformation for the GKR interactive proof. We next recall a result by Holmgren et al. [29], which shows (based on an observation by Jawale et al. [32]) that we can obtain a public-coin non-interactive argument by applying the Fiat–Shamir transformation to the parallel repetition of the GKR interactive proof. The result that we use is summarized in the following lemma, which is a rephrasing of a result given in [28, Section 6.2.2] (see also [31, Corollary 6.6, Theorem 4.5]).

Lemma 4. *Let* $W, D, \delta : \mathbb{N} \to \mathbb{N}$ *be functions such that* $W(\lambda) \le \mathsf{poly}(\lambda^{\log \lambda})$,[16] $D(\lambda) \le \mathsf{poly}(\log \lambda)$*, and* $\delta(\lambda) \le \mathsf{poly}(D(\lambda), \log W(\lambda))$*. 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* $\mathcal{H}_{\mathsf{FS}}$*.*

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* $\delta \le \delta(\lambda)$ *(cf. Lemma 3). Then, when* Π *is used with* λ *and* $(C, \mathbb{F}, \mathbb{H}, m)$*, the following hold for every input* x *and* $y := C(x)$*.*

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

[14] In [26, 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 [26, Theorem 3.1] is that the individual degree of $\widehat{x}$ is upper bounded by the degree parameter δ. Since we guarantee $\delta \ge m(|\mathbb{H}|-1)$, 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).

[15] Unlike the original version [26], the version given in [33] (which is the version that we use) requires the verifier to read $\widehat{x}$ at two points.

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

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,*

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

3. ***Efficiency.*** *For every $\lambda \in \mathbb{N}$ and every $(C, \mathbb{F}, \mathbb{H}, m)$ as above, P runs in time $D(\lambda) \cdot \mathsf{poly}(\lambda, D(\lambda), \log W(\lambda)) + T_{\mathrm{GKR},P}$ and 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.*

(See the full version of this paper about how we obtain Lemma 4 from [29].)

Remark 4 (On adaptive choice of y in the definition of soundness). Lemma 4 differs from what is shown in [32, 29] 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 [32, 29], 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 the results in [32, 29] 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 [32, 29] 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 [32, 29] 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. $\diamondsuit$

Remark 5 (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.[17] 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.

[17] Note that the Fiat–Shamir transformation only requires hashing the transcript (excluding x) as shown in [31, Figure 1].

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 $|I| = \mathsf{poly}(\lambda, D(\lambda), \log W(\lambda))$. $\Diamond$

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 show the following lemma.

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 $\lambda, \ell \in \mathbb{N}$ ($\ell \leq \log^2 \lambda$) and any finite field $\mathbb{F}$ of sufficiently large size $|\mathbb{F}| \leq \mathsf{poly}(\log \lambda)$, 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 $(C, \mathbb{F}, \mathbb{H}, m)$ 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, $(\mathbb{H}, m)$ 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 the full version of this paper.

Now, we proceeds to the proof of Lemma 2.

Proof (of Lemma 2). We first note that from Lemma 3, Lemma 4, Remark 5, and Lemma 5, we have the following corollary. (See the full version of this paper for the proof.)

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* $y := \mathsf{TreeHash}_h(x)$*,*
$$\Pr\left[V^{\widehat{x}|_I}(\mathsf{crs}, \ell, h, y, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ \pi \leftarrow P(\mathsf{crs}, h, x, y) \\ I := \mathsf{InpQuery}(\pi) \end{array} \right] = 1 \enspace .$$
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* $m(\mathbb{H} - 1)$*,*
$$\Pr\left[\begin{array}{l} y \neq \mathsf{TreeHash}_h(x) \\ \wedge\; V^{\widehat{x}|_I}(\mathsf{crs}, \ell, h, y, \pi) = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (y, \pi) \leftarrow P^*(\mathsf{crs}, h, x, z_\lambda) \\ I := \mathsf{InpQuery}(\pi) \end{array} \right] \leq \mathsf{negl}(T(\lambda)) \enspace ,$$
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.*** P *runs in time* $\mathsf{poly}(\lambda, L)$ *and* 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 Overview of 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. 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 $\widehat{\mathsf{DB}}$. For a formal proof, see the full version of this paper.

5 Public-Coin Oracle Memory Delegation

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

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 $\ell^*$ be the integer such that $|\mathsf{DB}| = 2^{\ell^*}\lambda$, and let $(\mathbb{F}, \mathbb{H}, m)$ be the parameters
       that $\Pi$ uses for $\lambda$ and $\ell^*$. 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 $\sigma := \perp$.
– $\pi := \mathsf{Prove}(\mathsf{DB}, \langle M_h, t_{|\mathsf{DB}|}, \mathsf{rt}\rangle, q)$:
    1. Parse $q$ as $\mathsf{crs}$.
    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 $\ell^*$ as in $\mathsf{Mem}$, and obtain $\mathsf{crs}$ from $q$ as in $\mathsf{Prove}$. (If there does not
       exist $\ell^*$ 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-$L_{\mathsf{DB}}$ 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 $\perp$ otherwise.
```

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., $\gamma(\lambda) = O(\lambda^{\log\log\lambda})$).*

We prove Lemma 1 by combining our tree-hash memory delegation scheme (Lemma 1) with the RAM delegation scheme of Choudhuri et al. [17].

5.1 Preliminary: RAM delegation

We recall the definition of publicly verifiable non-interactive RAM delegation schemes from [34, 17]. (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 [34, 17] 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 [34, 17], 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*.[18] 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 6. *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.*

– $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{Setup}(1^\lambda, T)$*:* 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*.*
– $\mathsf{dig} := \mathsf{Mem}(\mathsf{dk}, \mathsf{ms})$*:* Mem *is a deterministic algorithm that takes as input a digest key* dk *and a memory-state pair* ms*, and it outputs a digest* dig *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.*
– $b := \mathsf{Ver}(\mathsf{vk}, x, \mathsf{dig}, \mathsf{dig}', \pi)$*:* Ver *is a deterministic algorithm that takes as input a verifier key* vk*, an input x to R, source and destination digests* $\mathsf{dig}, \mathsf{dig}'$*, 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 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 $L_\pi(\lambda, T, |x|)$.*
– $\mathsf{Ver}(\mathsf{vk}, x, \mathsf{dig}, \mathsf{dig}', \pi)$ *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 $(\lambda, x, \mathsf{ms}, \mathsf{ms}', T) \in U_R$,*

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

[18] Unlike [34, 17], we refrain from using the term "configuration" to refer to the memory and state since we allow RAM machines to additionally have inputs.

– 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 $T \leq \gamma(\lambda)$,*

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

– 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 $T \leq \gamma(\lambda)$,*

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

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 [17] 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 $\mathsf{dig} = (\mathsf{st},\mathsf{rt},|\mathsf{DB}|)$ 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 [17] in that (i) RAM machines are defined in a 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 [17], 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 [17]. In particular, the analysis given in [17] 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 dig 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),[19] 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 [17]. 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.[20] The properties of Setup and Mem can be easily verified by inspecting the scheme description given in [18, Figure 5].[21]

5.2 Proof of Lemma 6

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

- R is given as input a description of a Turing machine M and given as memory a string DB. Then, R internally executes $M(\mathsf{DB})$.[22] 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. We

[19] For those who are familiar with the RAM delegation of [17], we note that we allow the statements of the batch-NP argument to contain the input of the RAM machine.

[20] Technically, the public-coin property can be verified by observing that under the LWE assumption, all the components of the scheme of [17] can be made public coin by using, e.g., an FHE scheme with pseudorandom public keys and ciphertexts.

[21] Actually, Mem in [18, Figure 5] outputs a pair $\mathsf{dig} = (\mathsf{st}, \mathsf{rt})$, but as noted above, we consider an extended version that additionally includes $|\mathsf{DB}|$ in dig.

[22] 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.)

can assume that this hash function family is polylogarithmic depth (and secure against $\lambda^{\log \lambda}$-time adversaries) under the sub-exponential hardness of the LWE assumption (cf. Footnote 9). 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 6 (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 6). This is because the output length of $\mathsf{RDel.Setup}(1^\lambda, T)$ is bounded by $\mathsf{poly}_{\mathsf{Setup}}(\lambda)$ for any $T \leq 2^\lambda$ 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.[23] 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 2. (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 $\mathsf{RDel.Mem}$).[24] The efficiency condition of ODel follows from the efficiency conditions of THDel and RDel. 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 $t \leq \gamma^{O(1)}(\lambda)$ such that

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

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 6, breaking the soundness of RDel requires generating an input x, source and destination memory-state pairs $(\mathsf{ms}, \mathsf{ms}')$, source and destination digests $(\mathsf{dig}, \mathsf{dig}')$,

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

[24] 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 22).

Algorithm 2 Public-coin oracle memory delegation scheme ODel.

```
– $\widehat{\mathsf{DB}} := \mathsf{Mem}(1^\lambda, \mathsf{DB})$:
    1. Output $\widehat{\mathsf{DB}} := \mathsf{THDel.Mem}(1^\lambda, \mathsf{DB})$.
– $(q, \sigma) \leftarrow \mathsf{Query}_1(1^\lambda)$:
    1. Run $(q_{\mathsf{THDel}}, \sigma_{\mathsf{THDel}}) \leftarrow \mathsf{THDel.Query}_1(1^\lambda)$ and $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.Setup}(1^\lambda)$.
    2. Output $q := (q_{\mathsf{THDel}}, \mathsf{pk}, \mathsf{vk}, \mathsf{dk})$ and $\sigma := \sigma_{\mathsf{THDel}}$.
– $\pi := \mathsf{Prove}(\mathsf{DB}, \langle M, t, y \rangle, q)$:
    1. Parse $q$ as $(q_{\mathsf{THDel}}, \mathsf{pk}, \mathsf{vk}, \mathsf{dk})$, and let $T := \mathsf{poly}_R(t)$.
    2. Run $R^{\mathsf{DB}}(M)$. If $R^{\mathsf{DB}}(M)$ does not terminate in $T$ steps, abort. Otherwise, let
       $\mathsf{DB}'$ denote the content of the memory at the termination of $R^{\mathsf{DB}}(M)$.
    3. Run $\pi_{\mathsf{RDel}} := \mathsf{RDel.Prove}_T(\mathsf{pk}, M, \mathsf{ms}, \mathsf{ms}')$ for $\mathsf{ms} := (\mathsf{DB}, \mathsf{st}_{\mathrm{START}})$ and $\mathsf{ms}' :=$
       $(\mathsf{DB}', \mathsf{st}_{\mathrm{END}})$, where $\mathsf{st}_{\mathrm{START}}$ and $\mathsf{st}_{\mathrm{END}}$ are the initial and the terminating states
       of $R$.
    4. Run $\pi_{\mathsf{THDel}} := \mathsf{THDel.Prove}(\mathsf{DB}, \langle M_h, t_{|\mathsf{DB}|}, \mathsf{rt} \rangle, q_{\mathsf{THDel}})$, where $M_h$ and $t_{|\mathsf{DB}|}$ are
       defined as in Definition 5 for the hash function $h := \mathsf{dk}$, and $\mathsf{rt}$ is the tree-hash
       that is obtained by $\mathsf{rt} := \mathsf{TreeHash}_{\mathsf{dk}}(\mathsf{DB})$.
    5. Let $(y', t')$ be the prefix of $\mathsf{DB}'$ that $R$ wrote before the termination, $\mathsf{rt}'$ be the
       tree-hash that is obtained by $\mathsf{rt}' := \mathsf{TreeHash}_{\mathsf{dk}}(\mathsf{DB}')$, and $\pi_{\mathsf{TreeHash}}$ be the local
       opening for $(y', t')$ w.r.t. $\mathsf{rt}'$.
    6. Output $\pi := (\mathsf{rt}, \mathsf{rt}', (y', t'), \pi_{\mathsf{RDel}}, \pi_{\mathsf{THDel}}, \pi_{\mathsf{TreeHash}})$.
– $I := \mathsf{Query}_2(|\mathsf{DB}|, \sigma, \pi)$:
    1. Parse $\pi$ as $(\mathsf{rt}, \mathsf{rt}', (y', t'), \pi_{\mathsf{RDel}}, \pi_{\mathsf{THDel}}, \pi_{\mathsf{TreeHash}})$.
    2. Output $I := \mathsf{THDel.Query}_2(|\mathsf{DB}|, \sigma, \pi_{\mathsf{THDel}})$.
– $b := \mathsf{Ver}^{\widehat{\mathsf{DB}}|_I}(|\mathsf{DB}|, \langle M, t, y \rangle, q, \sigma, \pi)$:
    1. Parse $q$ as $(q_{\mathsf{THDel}}, \mathsf{pk}, \mathsf{vk}, \mathsf{dk})$ and $\pi$ as $(\mathsf{rt}, \mathsf{rt}', (y', t'), \pi_{\mathsf{RDel}}, \pi_{\mathsf{THDel}}, \pi_{\mathsf{TreeHash}})$.
       Also, obtain $T$ as in $\mathsf{Prove}$, and abort if $|\mathsf{DB}| \neq T$. Let $\mathsf{dig} := (\mathsf{st}_{\mathrm{START}}, \mathsf{rt}, T)$
       and $\mathsf{dig}' := (\mathsf{st}_{\mathrm{END}}, \mathsf{rt}', T)$.
    2. Output 1 if all of the following hold.
       (a) $y = y'$ and $t' \leq t$.
       (b) $\mathsf{RDel.Ver}_T(\mathsf{vk}, M, \mathsf{dig}, \mathsf{dig}', \pi_{\mathsf{RDel}}) = 1$.
       (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 $h := \mathsf{dk}$.
       (d) $\pi_{\mathsf{TreeHash}}$ is a valid local opening for $(y', t')$ w.r.t. $\mathsf{rt}'$.
       If any of the above does not hold, output 0.
```

and a proof π such that (i) π is accepting w.r.t. $(\mathsf{dig}, \mathsf{dig}')$, (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) dig is the correct digest of ms, but (iv) dig' 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 (1). 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 ob-

Algorithm 3 Adversary $\mathcal{B}$ against the soundness of RDel.

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

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 $q := (q_{\mathsf{THDel}}, \mathsf{pk}, \mathsf{vk}, \mathsf{dk})$ and $\sigma := \sigma_{\mathsf{THDel}}$.
4. For each $b \in \{0,1\}$, parse $\pi_b$ 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
   $\mathsf{dig}_b := (\mathsf{st}_{\text{START}}, \mathsf{rt}_b, T)$ and $\mathsf{dig}'_b := (\mathsf{st}_{\text{END}}, \mathsf{rt}'_b, T)$.
5. Find $b^* \in \{0,1\}$ that satisfies all of the following.
   (a) $\mathsf{RDel.Ver}_T(\mathsf{vk}, M, \mathsf{dig}_{b^*}, \mathsf{dig}'_{b^*}, \pi_{\mathsf{RDel},b^*}) = 1$.
   (b) $\mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}) = \mathsf{dig}_{b^*}$, where $\mathsf{ms} := (\widetilde{\mathsf{DB}}, \mathsf{st}_{\text{START}})$ for $\widetilde{\mathsf{DB}} \leftarrow$
       $\mathsf{Decode}_{\mathsf{THDel}}(\widehat{\mathsf{DB}}, L_{\mathsf{DB}})$.
   (c) $\mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}') \neq \mathsf{dig}'_{b^*}$, where $\mathsf{ms}'$ is the memory-state pair of $R$ after $T$
       steps starting from input $M$ and memory-state pair $\mathsf{ms}$.
   If such $b^*$ exists, output $(M, \mathsf{ms}, \mathsf{ms}', \mathsf{dig}_{b^*}, \mathsf{dig}'_{b^*}, \pi_{\mathsf{RDel},b^*})$. Otherwise, abort.
```

tained 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 5). Then, for any $\lambda \in \mathbb{N}$ and $t \leq \gamma^{O(1)}(\lambda)$, the adversary $\mathcal{B}$ is described in Algorithm 3. Note that $\mathcal{B}$ runs in time $\gamma^{O(1)}(\lambda)$.

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 (1). Let $T := \mathsf{poly}_R(t)$. 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 (1).

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

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

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

Proof. This claim follows from (1) (it suffices to rewrite (1) 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 $y_b \overset{?}{=} y'_b$ and $L_{\mathsf{DB}} \overset{?}{=} T$. $\square$

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* $b \in \{0,1\}$*,*

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

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

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 $\mathsf{dig}'_0, \mathsf{dig}'_1$ must be distinct.

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

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

Proof. Since $\mathsf{dig}'_0 = \mathsf{dig}'_1$ implies $\mathsf{rt}'_0 = \mathsf{rt}'_1$ (recall $\mathsf{dig}'_b := (\mathsf{st}_{\mathrm{END}}, \mathsf{rt}'_b, T)$), 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 $(\mathsf{pk}, \mathsf{vk}, \mathsf{dk}) \leftarrow \mathsf{RDel.Setup}(1^\lambda)$.

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

Note that $L_{\mathsf{DB}} = T \wedge \mathsf{rt} = \mathsf{TreeHash}_{\mathsf{dk}}(\widetilde{\mathsf{DB}})$ implies $\mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}) = \mathsf{dig}_b$ since $\mathsf{RDel.Mem}_T(\mathsf{dk}, \mathsf{ms}) = (\mathsf{st}_{\mathrm{START}}, \mathsf{rt}, |\widetilde{\mathsf{DB}}|) = (\mathsf{st}_{\mathrm{START}}, \mathsf{rt}, L_{\mathsf{DB}}) = (\mathsf{st}_{\mathrm{START}}, \mathsf{rt}, T) = \mathsf{dig}_b$ (the first equality holds due to the furthermore part of Theorem 1 and the second equality holds since $\widetilde{\mathsf{DB}}$ is obtained by $\mathsf{Decode}_{\mathsf{THDel}}(\widehat{\mathsf{DB}}, L_{\mathsf{DB}})$, which outputs an L_{DB}-bit string as stated in Definition 5). Thus, we obtain

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

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

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

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 $(M, \mathsf{ms}, \mathsf{ms}', \mathsf{dig}_{b^*}, \mathsf{dig}'_{b^*}, \pi_{\mathsf{RDel}, b^*})$ of $\mathcal{B}$.

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

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 $\bar{t}$.*

We prove this lemma by following the same approach as Bitansky et al. [10] (where a private-coin memory delegation scheme is obtained from a private-coin oracle memory delegation scheme). Specifically, we obtain a public-coin memory delegation scheme by augmenting our public-coin oracle memory delegation scheme (Lemma 6) with a keyless multi-collision-resistant hash function. Roughly speaking, we modify our oracle memory delegation scheme as follows: (i) the verifier is no longer given oracle access to an encoded memory, and instead given a digest that is obtained by hashing the encoded memory with a multi-collision-resistant hash function with a local opening property (such a hash function can be obtained generically from any multi-collision-resistant hash function [10]), and (ii) the prover provides local opening of the encoded memory w.r.t. the locations that are necessary for the verification.[25] After these modifications, soundness can be shown as in Bitansky et al. [10] by relying on the multi-collision resistance of the hash function and the soundness of our oracle memory delegation scheme. (Our proof is simpler since we consider the public-coin setting.) For a formal proof, see the full version of this paper.

7 Public-coin 3-round Zero-Knowledge Argument

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

Theorem 2. *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.*

[25] These locations can be determined based on the proof and the verifier query.

Following prior works [6, 10], we prove Theorem 2 by using our weak memory delegation scheme (Lemma 7) to reduce the round complexity of Barak's public-coin zero-knowledge argument [2]. The high-level strategy is quite simple. Very roughly speaking, Barak's public-coin zero-knowledge argument uses a 4-round interactive argument to prove a statement about a digest of a long string, and the verification of this interactive argument is required to run in polynomial time even for a statement about a slightly super-polynomial computation. We reduce the round complexity of Barak's zero-knowledge argument by using our public-coin weak memory delegation scheme instead of this 4-round interactive argument. Soundness and zero-knowledge can be shown as in [6, 10], and our proof is simpler since we consider the public-coin setting. For a formal proof, see the full version of this paper.

References

1. Badrinarayanan, S., Goyal, V., Jain, A., Kalai, Y.T., Khurana, D., Sahai, A.: Promise zero knowledge and its applications to round optimal MPC. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 459–487. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_16
2. Barak, B.: How to go beyond the black-box simulation barrier. In: 42nd FOCS. pp. 106–115. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959885
3. Barak, B., Goldreich, O., Goldwasser, S., Lindell, Y.: Resettably-sound zero-knowledge and its applications. In: 42nd FOCS. pp. 116–125. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959886
4. Bellare, M., Palacio, A.: The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 273–289. Springer, Heidelberg (Aug 2004). https://doi.org/10.1007/978-3-540-28628-8_17
5. Berman, I., Degwekar, A., Rothblum, R.D., Vasudevan, P.N.: Multi-collision resistant hash functions and their applications. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 133–161. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78375-8_5
6. Bitansky, N., Brakerski, Z., Kalai, Y.T., Paneth, O., Vaikuntanathan, V.: 3-message zero knowledge against human ignorance. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part I. LNCS, vol. 9985, pp. 57–83. Springer, Heidelberg (Oct / Nov 2016). https://doi.org/10.1007/978-3-662-53641-4_3
7. Bitansky, N., Canetti, R., Chiesa, A., Goldwasser, S., Lin, H., Rubinstein, A., Tromer, E.: The hunting of the SNARK. Journal of Cryptology **30**(4), 989–1066 (Oct 2017). https://doi.org/10.1007/s00145-016-9241-9
8. Bitansky, N., Canetti, R., Paneth, O., Rosen, A.: On the existence of extractable one-way functions. In: Shmoys, D.B. (ed.) 46th ACM STOC. pp. 505–514. ACM Press (May / Jun 2014). https://doi.org/10.1145/2591796.2591859
9. Bitansky, N., Eizenstadt, N., Paneth, O.: Weakly extractable one-way functions. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part I. LNCS, vol. 12550, pp. 596–626. Springer, Heidelberg (Nov 2020). https://doi.org/10.1007/978-3-030-64375-1_21

10. Bitansky, N., Kalai, Y.T., Paneth, O.: Multi-collision resistance: a paradigm for keyless hash functions. In: Diakonikolas, I., Kempe, D., Henzinger, M. (eds.) 50th ACM STOC. pp. 671–684. ACM Press (Jun 2018). https://doi.org/10.1145/3188745.3188870
11. Bitansky, N., Khurana, D., Paneth, O.: Weak zero-knowledge beyond the black-box barrier. In: Charikar, M., Cohen, E. (eds.) 51st ACM STOC. pp. 1091–1102. ACM Press (Jun 2019). https://doi.org/10.1145/3313276.3316382
12. Bitansky, N., Lin, H.: One-message zero knowledge and non-malleable commitments. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018, Part I. LNCS, vol. 11239, pp. 209–234. Springer, Heidelberg (Nov 2018). https://doi.org/10.1007/978-3-030-03807-6_8
13. Bitansky, N., Paneth, O.: On round optimal statistical zero knowledge arguments. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 128–156. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8_5
14. Blum, M.: How to prove a theorem so no one else can claim it. In: Proceedings of the International Congress of Mathematicians. vol. 2, pp. 1444–1451 (1986)
15. Bronfman, L., Rothblum, R.D.: PCPs and Instance Compression from a Cryptographic Lens. In: Braverman, M. (ed.) ITCS 2022. vol. 215, pp. 30:1–30:19. LIPIcs (Jan 2022). https://doi.org/10.4230/LIPIcs.ITCS.2022.30
16. Canetti, R., Dakdouk, R.R.: Extractable perfectly one-way functions. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 449–460. Springer, Heidelberg (Jul 2008). https://doi.org/10.1007/978-3-540-70583-3_37
17. Choudhuri, A.R., Jain, A., Jin, Z.: SNARGs for $\mathcal{P}$ from LWE. In: 62nd FOCS. pp. 68–79. IEEE Computer Society Press (Feb 2022). https://doi.org/10.1109/FOCS52979.2021.00016
18. Choudhuri, A.R., Jain, A., Jin, Z.: 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 [17]
19. Chung, K.M., Kalai, Y.T., Liu, F.H., Raz, R.: Memory delegation. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 151–168. Springer, Heidelberg (Aug 2011). https://doi.org/10.1007/978-3-642-22792-9_9
20. Deng, Y.: Individual simulations. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part III. LNCS, vol. 12493, pp. 805–836. Springer, Heidelberg (Dec 2020). https://doi.org/10.1007/978-3-030-64840-4_27
21. Garg, S., Jain, A., Sahai, A.: Leakage-resilient zero knowledge. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 297–315. Springer, Heidelberg (Aug 2011). https://doi.org/10.1007/978-3-642-22792-9_17
22. Goldreich, O.: On the doubly-efficient interactive proof systems of GKR. Electronic Colloquium on Computational Complexity (2017), https://eccc.weizmann.ac.il/report/2017/101
23. Goldreich, O., Krawczyk, H.: On the composition of zero-knowledge proof systems. SIAM Journal on Computing **25**(1), 169–192 (1996)
24. Goldreich, O., Micali, S., Wigderson, A.: 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)
25. Goldreich, O., Oren, Y.: Definitions and properties of zero-knowledge proof systems. Journal of Cryptology **7**(1), 1–32 (Dec 1994). https://doi.org/10.1007/BF00195207

26. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: Interactive proofs for muggles. Journal of the ACM **62**(4), 27:1–27:64 (2015)
27. Hada, S., Tanaka, T.: On the existence of 3-round zero-knowledge protocols. In: Krawczyk, H. (ed.) CRYPTO'98. LNCS, vol. 1462, pp. 408–423. Springer, Heidelberg (Aug 1998). https://doi.org/10.1007/BFb0055744
28. Holmgren, J., Lombardi, A., Rothblum, R.D.: 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 [29]
29. Holmgren, J., Lombardi, A., Rothblum, R.D.: Fiat–Shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge). In: Khuller, S., Williams, V.V. (eds.) 53rd ACM STOC. p. 750–760. ACM Press (Jun 2021). https://doi.org/10.1145/3406325.3451116
30. Jain, A., Kalai, Y.T., Khurana, D., Rothblum, R.: Distinguisher-dependent simulation in two rounds and its applications. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 158–189. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63715-0_6
31. Jawale, R., Kalai, Y.T., Khurana, D., Zhang, R.: 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 [32]
32. Jawale, R., Kalai, Y.T., Khurana, D., Zhang, R.: SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. In: Khuller, S., Williams, V.V. (eds.) 53rd ACM STOC. p. 708–721. ACM Press (Jun 2021). https://doi.org/10.1145/3406325.3451055
33. Kalai, Y., Paneth, O., Yang, L.: On publicly verifiable delegation from standard assumptions. Cryptology ePrint Archive, Report 2018/776 (2018), https://eprint.iacr.org/2018/776
34. Kalai, Y.T., Paneth, O., Yang, L.: How to delegate computations publicly. In: Charikar, M., Cohen, E. (eds.) 51st ACM STOC. pp. 1115–1124. ACM Press (Jun 2019). https://doi.org/10.1145/3313276.3316411
35. Kalai, Y.T., Raz, R., Rothblum, R.D.: How to delegate computations: the power of no-signaling proofs. In: Shmoys, D.B. (ed.) 46th ACM STOC. pp. 485–494. ACM Press (May / Jun 2014). https://doi.org/10.1145/2591796.2591809
36. Khurana, D., Sahai, A.: How to achieve non-malleability in one or two rounds. In: Umans, C. (ed.) 58th FOCS. pp. 564–575. IEEE Computer Society Press (Oct 2017). https://doi.org/10.1109/FOCS.2017.58
37. Komargodski, I., Naor, M., Yogev, E.: Collision resistant hashing for paranoids: Dealing with multiple collisions. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 162–194. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78375-8_6
38. Pass, R.: Simulation in quasi-polynomial time, and its application to protocol composition. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 160–176. Springer, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_10
39. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM **56**(6) (2009)