Locally Verifiable Distributed SNARGs[*]

Eden Aldema Tshuva[†] Elette Boyle[‡] Ran Cohen[§] Tal Moran[¶]
Rotem Oshman[‖]

September 3, 2024

Abstract

The field of *distributed certification* is concerned with certifying properties of distributed networks, where the communication topology of the network is represented as an arbitrary graph; each node of the graph is a separate processor, with its own internal state. To certify that the network satisfies a given property, a prover assigns each node of the network a certificate, and the nodes then communicate with one another and decide whether to accept or reject. We require *soundness* and *completeness*: the property holds if and only if there exists an assignment of certificates to the nodes that causes all nodes to accept. Our goal is to minimize the length of the certificates, as well as the communication between the nodes of the network. Distributed certification has been extensively studied in the distributed computing community, but it has so far only been studied in the information-theoretic setting, where the prover and the network nodes are computationally unbounded.

In this work we introduce and study computationally bounded distributed certification: we define *locally verifiable distributed* SNARG*s* (LVD-SNARGs), which are an analog of SNARGs for distributed networks, and are able to circumvent known hardness results for information-theoretic distributed certification by requiring both the prover and the verifier to be computationally efficient (namely, PPT algorithms).

We give two LVD-SNARG constructions: the first allows us to succinctly certify any network property in P, using a global prover that can see the entire network; the second construction gives an efficient distributed prover, which succinctly certifies the execution of any efficient distributed algorithm. Our constructions rely on non-interactive batch arguments for NP (BARGs) and on RAM SNARGs, which have recently been shown to be constructible from standard cryptographic assumptions.

[*]This is an extended version of the following article: Locally Verifiable Distributed SNARGs, TCC 2023, ©IACR 2023.

[†]Tel-Aviv University. E-mail: `aldematshuva@tau.ac.il`. Research supported in part by AFOSR Award FA9550-23-1-0312, and an Algorand Foundation grant.

[‡]Reichman University and NTT Research. E-mail: `elette.boyle@runi.ac.il`. Research supported in part by AFOSR Award FA9550-21-1-0046 and ERC Project HSS (852952).

[§]Reichman University. E-mail: `cohenran@runi.ac.il`. Research supported in part by NSF grant no. 2055568 and by the Algorand Centres of Excellence programme managed by Algorand Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Algorand Foundation.

[¶]Reichman University. E-mail: `talm@runi.ac.il`. Research supported by ISF grant no. 2337/22.

[‖]Tel-Aviv University. E-mail: `roshman@tau.ac.il`. Research supported by ISF grant no. 2801/20.

Contents

1 Introduction

Distributed algorithms are algorithms that execute on multiple processors, with each processor carrying out part of the computation and often seeing only part of the input. This class of algorithms encompasses a large variety of scenarios and computation models, ranging from a single computer cluster to large-scale distributed networks such as the internet. Distributed algorithms are notoriously difficult to design: in addition to the inherent unpredictability that results from having multiple processors that are usually not tightly coordinated, distributed algorithms are required to be robust and fault-tolerant, coping with an environment that can change over time. Moreover, distributed computation introduces bottlenecks that are not present in centralized computation, including *communication* and *synchronization* costs, which can sometimes outweigh the cost of local computation at each processor. All of these reasons make distributed algorithms hard to design and to reason about.

In this work we study *distributed certification*, a mechanism that is useful for ensuring correctness and fault-tolerance in distributed algorithms: the goal is to efficiently check, on demand, whether the system is in a legal state or not (here, "legal" varies depending on the particular algorithm and its purpose). To that end, we compute in advance auxiliary information in the form of *certificates* stored at the processors, and we design an efficient *verification procedure* that allows the processors to interact with one another and use their certificates to verify that the system is in a legal state. The certificates are computed once, and therefore we are traditionally less interested in how hard they are to compute; however, the verification procedure may be executed many times to check whether the system state is legal, and therefore it must be highly efficient. Since we do not trust that the system is in a legal state, we think of the certificates as given by a *prover*, whose goal is to convince us that the system is in a legal state even when it is not. One can therefore view distributed certification as a distributed analog of NP.

Distributed certification has recently received extensive attention in the context of *distributed network algorithms*, which execute in a network comprising many nodes (processors) that communicate over point-to-point communication links. The communication topology of the network is modeled as an arbitrary undirected network graph, where each node is a vertex; the edges of the graph represent bidirectional communication links. The goal of a network algorithm is to solve some global problem related to the network topology, and so the network graph is in some sense both the input to the computation and also the medium over which the computation is carried out. Typical tasks in this setting include setting up network infrastructure such as low-weight spanning trees or subgraphs, scheduling and routing, and various forms of resource allocation; see the textbook [Pel00] for many examples. We usually assume that the network nodes initially know only their own unique identifier (UID), their immediate neighbors, and possibly a small amount of global information about the network, such as its size or its diameter. An efficient network algorithm will typically have each node learn as little as possible about the network as a whole, as this requires both communication and time. This is sometimes referred to as *locality* [Pel00].

Distributed certification arises naturally in the context of fault tolerance and correctness in network algorithms (even in early work, e.g., [APV91]), but it was first formalized as an object of independent interest in [KKP05]. A certification scheme for a network property $\mathcal{P}$ (for example, "the local states of the network nodes encode a valid spanning tree of the network") consists of a *prover*, which is usually thought of as unbounded, and a *verification procedure*, which is an efficient distributed algorithm that uses the certificates. Here, "efficiency" can take many forms (see the textbook [Pel00] for some), but it is traditionally measured only in *communication* and in *number of synchronized communication rounds*, not in local computation at the nodes. (A *synchronized communication round*, or *round* for short, is a single interaction round during which each network node performs some local computation, sends a possibly-different message on each of its edges, and receives the messages sent by its neighbors.) At the end of the verification procedure, each network node outputs an acceptance bit, and the network as a whole is considered to accept if and only if all nodes accept; it suffices for one node to "raise the alarm" and reject in order to indicate that there is a problem. Our goal is to minimize the length of the certificates while providing soundness and completeness, that is — there should exist a certificate assignment that convinces all nodes to accept if and only if the network satisfies the property $\mathcal{P}$.

To our knowledge, all prior work on distributed certification is in the information-theoretic setting: the prover and the network nodes are computationally unbounded, and we are concerned only with space (the length of the certificates) and communication (at verification time). As might be expected, some strong lower bounds are known: while any property of a communication topology on n nodes can be proven using $O(n^2)$-bit certificates by giving every node the entire network graph, it is shown in [GS16] that some properties do in fact require $\Omega(n^2)$-bit certificates in the deterministic setting, and similar results can be shown when the verification procedure can be randomized [FMO+19].

Our goal in this work is to circumvent the hardness of distributed certification in the information-theoretic setting by moving to the *computational setting*: we introduce and study *computationally sound distributed proofs*, which we refer to as *locally verifiable distributed* SNARG*s* (LVD-SNARGs), extending the centralized notion of a succinct non-interactive argument (SNARG).

Distributed SNARGs. In recent years, the fruitful line of work on delegation of computation has culminated in the construction of succinct, non-interactive arguments (SNARGs) for all properties in P [CJJ21b, WW22, KLVW23, CGJ+23]. A SNARG is a computationally sound proof system under which a polynomial-time prover certifies a statement of the form "$x \in \mathcal{L}$," where x is an input and $\mathcal{L}$ is a language, by providing a polynomial-time verifier with a short proof π. The verifier then examines the input x and the proof π, and decides (in polynomial time) whether to accept or reject. It is guaranteed that an honest prover can convince the verifier to accept any true statement with probability 1 (*perfect completeness*), and at the same time, no poly-size cheating prover can convince the verifier to accept with non-negligible probability (*computational soundness*).

In this work, we first ask:

Can we construct locally verifiable distributed SNARGs *(*LVD-SNARG*s), a distributed analog of* SNARG*s which can be verified by an efficient (i.e., local) distributed algorithm?*

In contrast to prior work on distributed verification, here when we say "efficient" we mean in communication and in rounds, but also in computation, combining both distributed and centralized notions of efficiency. (We defer the precise definition of our model to Section 3.1).

We consider two types of provers: first, as a warm-up, we consider a *centralized prover*, which is a polynomial-time algorithm that sees the entire network and computes succinct certificates for the nodes. We show that in this settings, there is an LVD-SNARG for any property in P, using RAM SNARGs [KP16, KLVW23] as our main building block.

The centralized prover can be applied in the distributed context by first collecting information about the entire network at one node, and having that node act as the prover and compute certificates for all the other nodes. However, this is very inefficient: for example, in terms of total communication, it is easy to see that collecting the entire network topology in one location may require $\Omega(n^2)$ bits of communication to flow on some edge. In contrast, "efficient" network algorithms use sublinear and even polylogarithmic communication.[1] This motivates us

[1]As just one example of many, in [KP98] it is shown that one can construct a k-dominating set of the network graph in $\tilde{O}(k)$ communication per edge, and this is used to construct a minimum-weight spanning tree in $\tilde{O}(\sqrt{n})$ communication per edge.

to consider another type of prover — a *distributed* prover — and ask:

If a property can be decided by an efficient distributed algorithm, can it be succinctly certified by an efficient distributed prover?

Of course, the verifier is required to be an efficient distributed algorithm, as in the case of the centralized prover above. We give a positive answer to this question as well: given a distributed algorithm $\mathcal{D}$, we construct a distributed prover that runs alongside $\mathcal{D}$ with low overhead (in communication and rounds), and produces succinct certificates at the network nodes.

More formal statements of our results are given in Section 1.2; before doing so, we provide more context and background on distributed certification and on delegation of computation.

1.1 Background and Related Works

Distributed Certification The classical model for distributed certification was formally introduced by Korman, Kutten and Peleg in [KKP05] under the name *proof labeling schemes (PLS)*, but was already present implicitly in prior work on self-stabilization, such as [APV91]. To certify a property $\mathcal{P}$ of a network graph $G = (V, E)$,[2] we first run a *marker algorithm* (i.e., a prover), a computationally unbounded algorithm that sees the entire network, to compute a proof in the form of a labeling $\ell : V \to \{0, 1\}^*$. We refer to these labels as *certificates*; each node $v \in V$ is given only its own certificate, $\ell(v)$. We refer to this as the *proving stage*.

Next, whenever we wish to verify that the property $\mathcal{P}$ holds, we carry out the *verification stage*: each node $v \in V$ sends its certificate $\ell(v)$ to its immediate neighbors in the graph. Then, each node examines its direct neighborhood, its certificates, and the certificate it received from its neighbors, and deterministically outputs an acceptance bit.

The proof is considered to be accepted if and only if all nodes accept it. During the verification stage, the nodes are honest; however, the prover may not be honest during the proving stage, and in general it can assign arbitrary certificates to any and all nodes in the network. We require *soundness* and *completeness*: the property $\mathcal{P}$ holds if and only if there exists an assignment of certificates to the nodes that causes all nodes to accept.

The focus in the area of distributed certification is on schemes that use short certificates. Even short certificates can be extremely helpful: to illustrate, and to familiarize the reader with the model, we describe a scheme from [KKP05] for certifying the correctness of a spanning tree: each node $v \in V$ is given a parent pointer $p_v \in V \cup \{\perp\}$, and our goal is to certify that the subgraph induced by these pointers, $\{(v, p_v) : v \in V \text{ and } p_v \neq \perp\}$, is a spanning tree of the network graph G. In the scheme from [KKP05], each node $v \in V$ is given a certificate $\ell(v) = (r_v, d_v)$, containing the following information:

- The purported name r_v of the root of the tree, and
- The distance d_v of v from the root r_v.

(Note that even though the tree has a single root, the prover can try to cheat by claiming different roots at different nodes, and hence we use the notation r_v for the root given to node v.) To verify, the nodes send their certificates to their neighbors, and check that:

- Their root r_v is the same as the root r_u given to each neighbor u, and
- If $p_v \neq \perp$, then $d_{p_v} = d_v - 1$, and if $p_v = \perp$, then $d_v = 0$.

This guarantees the correctness of the spanning tree,[3] and requires only $O(\log n)$-bit certificates, where n is the number of nodes in the network; the verification stage incurs communication

[2]In general, the nodes of the network may have *inputs*, on which the property may depend, but for simplicity we ignore inputs for the time being and discuss only properties of the graph topology itself.

[3]Assuming the underlying network is connected, which is a standard assumption in the area; otherwise additional information, such as the size of the network, is required.

$O(\log n)$ on every edge, and requires only one round (each node sends one message to each neighbor). In contrast, *generating* a spanning tree from scratch requires $\Omega(D)$ communication rounds, where D is the diameter of the network; *verifying without certificates* that a given (claimed) spanning tree is correct requires $\tilde{\Omega}(\sqrt{n}/B)$ communication rounds, if each node is allowed to send B bits on every edge in every round [SHK+12].

The original model of [KKP05] is highly restricted: it does not allow randomization, and it allows only one round of communication, during which each node sends its certificate to all of its neighbors (this is the only type of message allowed). Subsequent work studied many variations on this basic model, featuring different generalizations and communication constraints during the verification stage (e.g., [GS16, OPR17, PP17, FFH+21, BFO22]), different restrictions on how certificates may depend on the nodes' identifiers (e.g, [FHK12, FGKS13, BDFO18]), restricted classes of properties and network graphs (e.g., [FBP22, FMRT22]), allowing randomization [FPP19, FMO+19] or interaction with the prover (e.g., [KOS18, NPY20, BKO22]), and in the case of [BKO22], also preserving the privacy of the nodes using a distributed notion of zero knowledge. We refer to the survey [Feu21] for an overview of much of the work in this area.

To our knowledge, all work on distributed certification so far has been in the *information-theoretic* setting, which requires soundness against a computationally unbounded prover, and does not take the local computation time of either the prover or the verifier into consideration as a complexity measure (with one exception, [AO24], where the running time of the nodes is considered, but perfect soundness is still required). Information-theoretic certification is bound to run up against barriers arising from communication complexity: it is easy to construct synthetic properties that essentially encode lower bounds from nondeterministic or Merlin-Arthur communication complexity into a graph problem. More interestingly, it is possible to use reductions from communication complexity to prove lower bounds on some natural problems: for example, in [GS16] it was shown that $\Omega(n^2)$-bit certificates are required to prove the existence of a non-trivial automorphism, or non-3-colorability. In addition to this major drawback, in the information-theoretic setting there is no clear connection between whether a property is efficiently checkable in the traditional sense (P, or even NP) and whether it admits a short distributed proof: even computationally easy properties, such as "the network has diameter at most k" (for some constant k), or "the identifiers of the nodes in the network are unique," are known to require $\tilde{\Omega}(n)$-bit certificates [FMO+19]. (These lower bounds are, again, proven by reduction from two-party communication complexity.) In this work we show that introducing computational assumptions allows us to efficiently certify any property in P, overcoming the limitations of the information-theoretic model.

Delegation of Computation. Computationally sound proof systems were introduced in the seminal work of Micali [Mic00], who gave a construction for such proofs in the random-oracle model (ROM), where we assume all parties have access to a reliably random function. Micali's construction was based on an earlier, *interactive* computationally sound proof introduced by Kilian [Kil92], and on the Fiat-Shamir paradigm [FS86], to turn the latter non-interactive. Following Micali's work, extensive effort went into obtaining non-interactive arguments (SNARGs) in models that are closer to the plain model, such as the *Common Reference String* (CRS) model. Earlier work in this line of research, such as [ABOR00, DLN+04, DL08, Gro10, BCCT12], relied on *knowledge assumptions*, which are non-falsifiable. For languages in NP, Gentry and Wichs [GW11] have shown a strong barrier to constructing SNARGs from falsifiable assumptions, as a black-box reduction to such assumption would disprove the assumption itself. This led the research community to focus some attention on delegating efficient deterministic computation, that is, computation in P.

Initial progress on delegating computation in P assumed the weaker model of a *designated verifier*, where the verifier holds some secret that is related to the CRS [KRR13, KRR14, KP16, BKK+18, HR18]. However, a recent line of work has led to the construction of publicly verifiable

SNARGs for deterministic computation, first for space-bounded computation [KPY19, JKKZ21] and then for general polynomial-time computation [CJJ21b, WW22, KLVW23, CGJ+23]. These latter constructions exploit a connection to *non-interactive batch arguments for* NP *(*BARG*s)*, which can be constructed from various standard cryptographic assumptions [BHK17, CJJ21a, WW22, KLVW23, CGJ+23]. We use BARGs as the basis for the distributed prover that we construct in Section 6.

1.2 Our Results

We are now ready to give a more formal overview of our results, although the full formal definitions of LVD-SNARG with a global and a distributed prover are deferred to Sections 4 and 6. For simplicity, in this overview we restrict attention to network properties that concern only the topology of the network — in other words, in the current section, a property $\mathcal{P}$ is a family of undirected graphs. (In the more general case, a property can also involve the internal states of the network nodes, as in the spanning tree example from Section 1.1. This will be discussed in the Technical Overview.)

Defining LVD-SNARGs. Like centralized SNARGs for P, LVD-SNARGs are defined in the common reference string (CRS) model, where the prover and the verifier both have access to a shared unbiased source of randomness.

An LVD-SNARG for a property $\mathcal{P}$ consists of a pair of algorithms (other than the algorithm generating the CRS, with respect to a security parameter λ):

- A *prover algorithm*: given a network graph $G = (V, E)$ of size $|V| = n$ and the common reference string (CRS), the prover algorithm outputs an assignment of $\text{poly}(\lambda, \log n)$-bit certificates to the nodes of the network. The prover may be either a polynomial-time centralized algorithm, or a distributed algorithm that executes in G in a polynomial number of rounds, sends messages of polynomial length on every edge, and involves only polynomial-time computations at each network node.[4]
- A *verifier algorithm*: the verifier algorithm is a one-round distributed algorithm, where each node of the network simultaneously sends a (possibly different) message of length $\text{poly}(\lambda, \log n)$ on each of its edges, receives the messages sent by its neighbors, carries out some local computation, and then outputs an acceptance bit. Both the computation of the messages to send to the neighbors and the computation of the acceptance bit are done by polynomial-time algorithms.

We require that certificates produced by an honest execution of the prover in the network be accepted by all verifiers with probability 1, whereas for any graph failing to satisfy the property $\mathcal{P}$, certificates produced by any poly-size cheating prover (allowing centralized provers in both cases) will be rejected by at least one node with overwhelming probability, as a function of the security parameter λ.[5] We refer the reader to Section 4 for the formal definition.

LVD-SNARGs with a global prover. We begin by considering a global (i.e., centralized) prover, which sees the entire network graph G. In this setting, we give a very simple construction that makes black-box use of the recently developed RAM SNARGs for P [KP16, CJJ21b, KLVW23, CGJ+23] to obtain the following:

[4]In fact, as we mentioned earlier in the introduction, a centralized prover can also be implemented by a distributed algorithm where one node learns the entire network graph and then generates the certificates. This is easy to do in polynomial rounds and message length.

[5]The schemes we construct actually satisfy *adaptive* soundness: there is no poly-size algorithm that can, with non-negligible probability, output a network graph and certificates for all the nodes, such that the property does not hold for the network graph but all of the nodes accept.

Theorem 1.1 (Informal, see Theorem 4.3.). *Assuming the existence of* RAM SNARG*s for* P *and collision-resistant hash families, for any property $\mathcal{P} \in$* P*, there is an* LVD-SNARG *with a global prover.*

LVD-SNARGs with a distributed prover. As explained earlier in the introduction, one of the main motivations for distributed certification is to be able to quickly check that the network is in a legal state. One natural special case is to check whether the results of a previously executed distributed algorithm are still correct, or whether they have been rendered incorrect by changes or faults in the network. To this end, we ask whether we can augment any given computationally efficient distributed algorithm $\mathcal{D}$ with a *distributed prover*, which runs alongside $\mathcal{D}$ and produces an LVD-SNARG certifying the execution of $\mathcal{D}$ in the specific network. The distributed prover may add some additional overhead in communication and in rounds, but we would like the overhead to be small.

We show that indeed this is possible:

Theorem 1.2 (Informal, see Theorem 6.2). *Let $\mathcal{D}$ be a distributed algorithm that runs in $\mathrm{poly}(n)$ rounds in networks of size n, where in each round, every node sends a $\mathrm{poly}(\log n)$-bit message on every edge, receives the messages sent by its neighbors in the current round, and then carries out $\mathrm{poly}(n)$ local computation steps.*

Assuming the existence of BARG*s for* NP *and collision-resistant hash families, there exists an augmented distributed algorithm $\mathcal{D}'$, which carries out the same computation as $\mathcal{D}$, but also produces an* LVD-SNARG *certificate attesting that $\mathcal{D}$'s output is correct.*

- *The overhead of $\mathcal{D}'$ compared to $\mathcal{D}$ is an additional $O(D)$ rounds on networks with diameter D, during which each node sends only $\mathrm{poly}(\lambda, \log n)$-bit messages, for security parameter λ.*
- *The certificates produced are of size $\mathrm{poly}(\lambda, \log n)$.*

Using known constructions of RAM SNARGs for P and of SNARGs for batch-NP [CJJ21b, CJJ21a, WW22, KLVW23, CGJ+23], we obtain both types of LVD-SNARGs (global or distributed prover) for P from either LWE, DLIN, or subexponential DDH.

Distributed Merkle trees (DMTs). To construct our distributed prover, we develop a data structure that we call *distributed Merkle tree* (DMT), which is essentially a global Merkle tree of a distributed collection of $2|E|$ values, with each node u initially holding a value $x_{u \to v}$ for each neighbor v. (At the "other end of the edge", node v also holds a value $x_{v \to u}$ for node v. There is no relation between the value $x_{u \to v}$ and the value $x_{v \to u}$.)

The unique property of the DMT is that it can be constructed by an efficient distributed algorithm, at the end of which each node u holds both the root of the global Merkle tree and a succinct opening to each value $x_{(u,v)}$ that it held initially.

The DMT is used in the construction of the LVD-SNARG of Theorem 1.2 to allow nodes to "refer" to messages sent by their neighbors. We cannot afford to have node v store these messages, or even a hash of the messages v received on each of its edges, as we do not want the certificates to grow linearly with the degree. Instead, we construct a DMT that allows nodes to "access" the messages sent by their neighbors: we let each value $x_{v \to u}$ be a hash of the messages sent by node v to node u, and construct a DMT over these hashes. When node u needs to "access" a message sent by v to construct its proof, node v produces the appropriate opening path from the root of the DMT, and sends it to node u. All of this happens implicitly, inside a BARG proof asserting that u's local computation is correct.

2 Technical Overview

We begin by giving a more formal overview of our network model; this model is standard in the area of distributed network algorithms (see, e.g., the textbook [Pel00]). In Section 2.1, we describe our construction of an LVD-SNARG with a global prover. This construction is quite simple and can be viewed as a warm-up for our second, more complicated construction: in Section 2.2, we describe an LVD-SNARG with a distributed prover, which certifies that the current state of the system reflects a correct execution of a given distributed algorithm in the current network. Finally, in Section 2.3, we describe our distributed Merkle tree construction.

2.1 LVD-SNARGs with a Global Prover

We begin by describing a simple construction for LVD-SNARGs with a global prover for any property in P. (When we refer to P here, we mean from the centralized point of view: a distributed language $\mathcal{L}$ is in P if and only if there is a deterministic poly-time Turing machine that takes as input a configuration (G, x) and accepts if and only if $(G, x) \in \mathcal{L}$.) Throughout this overview, we assume for simplicity that the nodes of the network are named $V = \{1, \dots, n\}$, with each node knowing its own name (but not necessarily the size n of the network).

Commit-and-prove. Fix a language $\mathcal{L} \in$ P and an instance $(G, x) \in \mathcal{L}$. A global prover that sees the entire instance G can use a (centralized) SNARG for the language $\mathcal{L}$ in a black-box manner, to obtain a succinct proof for the statement "$(G, x) \in \mathcal{L}$." However, regular SNARGs (as opposed to RAM SNARGs) assume that the verifier *holds the entire input* whose membership in $\mathcal{L}$ it would like to verify; in our case, no single node knows the entire instance G, so we cannot use the verification procedure of the SNARG as-is.

Our simple work-around to the nodes' limited view of the network is to ask the prover to give the nodes a *commitment with local openings* C to the entire network graph (for instance, a Merkle tree [Mer89]), and to each node, a proof π_{SNARG} that the graph under the commitment is in the language $\mathcal{L}$.

Note that the language for which π_{SNARG} is a SNARG proof is a set of commitments, not of network configurations — it is the language of all commitments to configurations in $\mathcal{L}$. However, this leaves us with the burden of relating the commitment C to the true instance (G, x) in which the verifier executes, to ensure that the prover did not choose some arbitrary C that is unrelated to the instance at hand. To that end, we ask the prover to provide each node v with the following:

- The commitment C and proof π_{SNARG}. The nodes verify that they all received the same values by comparing with their neighbors, and they verify the SNARG proof π_{SNARG}.
- A succinct opening to v's neighborhood. Node v verifies that indeed, C opens to its true neighborhood $N(v)$.

Intuitively, by verifying that the commitment is consistent with the view of all the nodes, and by verifying the SNARG that the graph "under the commitment" is in the language $\mathcal{L}$, we verify that the true instance (G, x) is in fact in $\mathcal{L}$.

Although the language $\mathcal{L}$ is in P, if we proceed carelessly, we might find ourselves asking the prover to prove an NP-statement, such as "there exists a graph configuration (G, x) whose commitment is C, such that $(G, x) \in \mathcal{L}$." Moreover, proving the soundness of such a scheme requires *extracting* the configuration (G, x) from the proof π_{SNARG}, so that a cheating adversary who produces a convincing proof of a false statement can be used to break either the SNARG or the commitment scheme. Essentially, we would require a SNARK (a succinct non-interactive argument *of knowledge*) for NP, but significant barriers are known on constructing SNARKs from standard assumptions [GW11]. To avoid this barrier, we use RAM SNARG*s*.

RAM SNARGs for P. A RAM SNARG ([KP16, BHK17]) is a SNARG that proves that a given RAM machine M performs some computation correctly;[6] however, instead of holding the input x to the computation, the verifier is given only a *digest* of x — a hash value, typically obtained from a hash family with local openings (for instance, the root of a Merkle tree of x). In our case, we ask the prover to use a polynomial-time machine $M_{\mathcal{L}}$ that decides $\mathcal{L}$ as the RAM machine for the SNARG, and the commitment C as the digest; the prover computes a RAM SNARG proof for the statement "$M_{\mathcal{L}}(G, x) = 1$."

Defining the soundness of RAM SNARGs is delicate: because the verifier is not given the full instance but only a digest of it, there is no well-defined notion of a "false statement" — a given digest d could be the digest of multiple instances, some of which satisfy the claim and some of which do not. However, the digest is collision resistant, so intuitively, it is hard for the adversary to *find* two instances that have the same digest. We adopt the original RAM SNARG soundness definition from [KP16, BHK17, KLVW23], which requires that it be computationally hard for an adversary to prove "contradictory statements"; given the common reference string, it must be hard for an adversary to find:

- A digest d, and
- Two different proofs π_0 and π_1, *which are both accepted with input digest* d, such that π_0 proves that the output of the computation is 0, and π_1 proves that the output of the computation is 1.

In our construction, the prover is asked to provide the nodes with a digest C, which is a commitment to the configuration (G, x), and a RAM SNARG proof π_{SNARG} for the statement "$(G, x) \in \mathcal{L}$," which the prover constructs using a RAM machine $M_{\mathcal{L}}$ that decides membership in $\mathcal{L}$ in polynomial time.

Tying the digest to the real network graph. By themselves, the digest C and the RAM SNARG proof π_{SNARG} do not say much about the actual instance (G, x). As explained above, the digest can be related to the real network by having every node verify that it opens correctly to its local view (neighborhood). However, this is not enough: the prover can commit to (i.e., provide a digest of) a graph $G' \in \mathcal{L}$ that is *larger* than the true network graph G, such that G' agrees with G on the neighborhoods of all the "real nodes" (the nodes of G).[7] We prevent the prover from doing this by:

- Asking the prover to provide the nodes with the size n of the network, and a certificate proving that the size is indeed n. There is a simple and elegant scheme for doing this [KKP05], based on building and certifying a rooted spanning tree of the network; it has perfect soundness and completeness, and requires $O(\log n)$-bit certificates.
- The Turing machine $M_{\mathcal{L}}$ for verifying membership in $\mathcal{L}$ is assumed to take its input in the form of an adjacency list $L_{G,x} = \big((v_1, x(v_1), N(v_1)), \ldots, (v_n, x(v_n), N(v_n)), \perp\big)$, where $\perp$ is a special symbol marking the end of the list, and each triplet $(v_i, x(v_i), N(v_i))$ specifies a node v_i, its input $x(v_i)$, and its neighborhood $N(v_i)$. Since $\perp$ marks the end of the list, the machine $M_{\mathcal{L}}$ is assumed (without loss of generality) to ignore anything following the symbol $\perp$ in its input.
- Recall that we assumed for simplicity that $V = \{1, \ldots, n\}$. The prover computes a digest C of $L_{G,x}$, and gives each node i the opening to the i^{th} entry. Each node verifies that its entry opens correctly to its local view (name, input, and neighborhood).
- The last node, node n, is also given the opening to the $(n+1)^{\text{th}}$ entry, and verifies that

[6] A RAM machine M is given query access to an input x and an unbounded random-access memory array, and returns some output y. Each query to the input x or the memory is considered a unit-cost operation.

[7] This requires that G' not be connected, but that is not necessarily a problem for the prover, depending on the property $\mathcal{L}$.

it opens to $\perp$. Node n knows that it is the last node, because the prover gave all nodes the size n of the network (and certified it).

To prove the soundness of the resulting scheme, we show that if all nodes accept, then C is a commitment to some adjacency list L' which has $L_{G,x}$ as a prefix — in the format outlined above, including the end-of-list symbol $\perp$. Since the machine $M_{\mathcal{L}}$ interprets $\perp$ as the end of its input, it ignores anything past this point, and thus, the prover's SNARG proof is essentially a proof for the statement "$M_{\mathcal{L}}$ accepts (G, x)." If we assume for the sake of contradiction that $(G, x) \notin \mathcal{L}$ then we can generate an honest SNARG proof π_0 for the statement "$M_{\mathcal{L}}$ rejects (G, x)," using the same digest C,[8] and this breaks the soundness of the SNARG.

2.2 LVD-SNARGs with a Distributed Prover

One of the main motivations for distributed certification is to help build fault-tolerant distributed algorithms. In this setting, there is no omniscient global prover that can provide certificates to all the nodes. Instead, the labels must themselves be produced by a distributed algorithm, and comprise a proof that a previous execution phase completed successfully and that its outputs are still valid (in particular, they are still relevant given the current state of the communication graph and the network nodes). Formally, given a distributed algorithm $\mathcal{D}$, we want to construct a distributed prover $\mathcal{D}'$ that certifies the language

$$\mathcal{L}_{\mathcal{D}} = \left\{ (G, x, y) : \begin{array}{c} \text{when } \mathcal{D} \text{ executes in the network } G \\ \text{with inputs } x : V \to \{0,1\}^*, \\ \text{it produces the outputs } y : V \to \{0,1\}^* \end{array} \right\}.$$

Furthermore, $\mathcal{D}'$ should not have much overhead compared to $\mathcal{D}$ in terms of communication and rounds.

Certifying the execution of the distributed algorithm $\mathcal{D}$ essentially amounts to proving a collection of "local" statements, each asserting that at a specific node $v \in V(G)$, the algorithm $\mathcal{D}$ indeed produces the claimed output $y(v)$ when it executes in G. The prover at node v can record the local computation at node v as $\mathcal{D}$ executes, including the messages that node v sends and receives. As a first step towards certifying that $\mathcal{D}$ executes correctly, we could store at each node v a (centralized) SNARG proving that in every round, v produced the correct messages according to $\mathcal{D}$, handled incoming messages correctly, and performed its local computation correctly, eventually outputting $y(v)$. However, this does not suffice to guarantee that the *global* computation is correct, because we must verify consistency across the nodes: how can we be sure that incoming messages recorded at node v were indeed sent by v's neighbors when $\mathcal{D}$ ran, and vice-versa?

A naïve solution would be for node v to record, for each neighbor $u \in N(v)$, a hash $H_{(v,u)}$ of all the messages that v sends and receives on the edge $\{v, u\}$; at the other end of the edge, node u would do the same, producing a hash $H_{(u,v)}$. At verification time, nodes u and v could compare their hashes, and reject if $H_{(v,u)} \neq H_{(u,v)}$. Unfortunately, this solution would require too much space, as node v can have up to $n - 1$ neighbors; we cannot afford to store a separate hash for each edge as part of the certificate. Our solution is instead to hash all the messages sent in the entire network together, but in a way that allows each node to "access" the messages sent by itself and its neighbors. To do this we use an object we call a *distributed Merkle tree* (DMT), which we introduce next.

Distributed Merkle trees. A DMT is a single Merkle tree that represents a commitment to an unordered collection of values $\{x_{u\to v}\}_{\{u,v\}\in E}$, one value for every directed edge $u \to v$

[8]This step is a little delicate, and relies on the fact that in recent RAM SNARG constructions (e.g., [CJJ21b, KLVW23]), completeness holds for any digest d that opens to the input instance at every location the RAM machine reads from. See Section 3.3.1 for more detail.

such that $\{u, v\} \in E$. (The total number of values is $2|E|$.) It is constructed by a distributed algorithm called $\mathsf{DistMake}$, at the end of which each node v obtains the following information:

- val: the global root of the DMT.
- rt_v: the "local root" of node v, which is the root of a Merkle tree over the local values $\{x_{v\to u}\}_{u\in N(v)}$.
- I_v and ρ_v: the index of rt_v inside the global DMT, and the corresponding opening path ρ_v for rt_v from the global root val.
- $\beta_v = \{(I_{v\to u}, \rho_{v\to u})\}_{u\in N(v)}$: for each neighbor $u \in N(v)$, the index $I_{v\to u}$ is a relative index for the position of $x_{v\to u}$ under the local root rt_v, and the opening path $\rho_{v\to u}$ is the corresponding relative opening path from rt_v. For every pair of neighbors v and u, the index $I_{v\to u}$ also equals the number of the port of u in v's neighborhood.

The DMT is built such that for any value $x_{v\to u}$, the index of the value in the DMT is given by $I_v||I_{v\to u}$, and the corresponding opening path is $\rho_v||\rho_{v\to u}$. Thus, node v holds enough information to produce an opening and to verify any value that it holds.[9] (Here and throughout, $||$ denotes concatenation; we treat indices as binary strings representing paths from the root down, with "0" representing a left turn, and "1" a right.)

The novelty of the DMT is that it can be constructed by an efficient distributed algorithm, which runs in $O(D)$ synchronized rounds (where D is the diameter of the graph), and sends $\mathrm{poly}(\lambda, \log n)$-bit messages on every each in each round. We remark that it would be trivial to construct a DMT in a *centralized* manner, but the key to the efficiency of our distributed prover is to provide an efficient *distributed* construction; in particular, we cannot afford to, e.g., collect all the values $\{x_{u\to v}\}_{\{u,v\}\in E}$ in one place, as this would require far too much communication. We avoid this by giving a distributed construction where each node does some of the work of constructing the DMT, and eventually obtains only the information it needs.

We give an overview of the construction of the DMT in Section 2.3, but first we explain how we use it in the distributed prover.

Using the DMT. While running the original distributed algorithm $\mathcal{D}$, the distributed prover stores the internal computation steps, the messages sent and the messages received at every node.[10] For each node v and neighbor u, node v computes two hashes:

- A hash $h_{v\to u}$ of the messages v sent to u, and
- a hash $h_{u\to v}$ of the messages v received from u.

A message sent from v to u in round r is hashed at index r in $h_{v\to u}$. Note that both endpoints of the edge $\{u, v\}$ compute the same hashes $h_{u\to v}$ and $h_{v\to u}$, but they "interpret" them differently: node v views $h_{u\to v}$ as a hash of the messages it received from u, while node u views it as a hash of the messages it sent to v, and vice-versa for $h_{v\to u}$.

The messages hashes are used to construct the proof, but they are discarded at the end of the proving stage, so as not to exceed our storage requirements. We use a hash family with local openings, so that node v is able to produce a succinct opening from $h_{v\to u}$ or $h_{u\to v}$ to any specific message that was sent or received in a given round.

Next we construct a DMT over the values $\{h_{u\to v}\}_{\{u,v\}\in E}$. Let $\mathsf{val}^{\mathsf{msg}}$ be the root of the DMT. For each neighbor $u \in N(v)$, node v obtains from the DMT the index and opening for

[9] For simplicity we assume that nodes can query the communication infrastructure for a consistent order of their neighbors (e.g., by "port number"); thus the relative ordering $I_{v\to u}$ does not count against v's storage. This is a standard assumption in the area. In the general case, the port numbers themselves, which may stand for MAC addresses or similar, do not necessarily need to be consecutive numbers from $1, \ldots, \deg(v)$, but we can order v's neighbors in order of increasing port number.

[10] We believe that this additional temporary storage requirement can be avoided using incrementally verifiable computation ([Val08],[PP17],[DGKV22]), but we have not gone through the details.

the message hash $h_{v\to u}$, and it sends them to the corresponding neighbor u.

For a given node v and a neighbor of it, u, let $I^{\mathsf{msg}}_{v,u,r}$ be the index in the DMT of the message sent by node v to node u in round r, which is given by $I_v||I_{v\to u}||r$ (recall that r is the index of the r-round message inside $h_{v\to u}$). Node v is able to compute both $I^{\mathsf{msg}}_{v,u,r}$ and $I^{\mathsf{msg}}_{u,v,r}$ and the corresponding opening paths, since it holds both hashes $h_{v\to u}$ and $h_{u\to v}$, learns I_v and $\beta_v = \{I_{v\to u}\}_{u\in N(v)}$ during the construction of the DMT, and receives $I_u||I_{u\to v}$ from node v.

With these values in hand, the nodes can jointly use $\mathsf{val}^{\mathsf{msg}}$ as a hash of all the messages sent or received during the execution of $\mathcal{D}$. Each node v holds indices and openings for all the messages that it sent or received during the execution. Note that this is the *only* information that v obtains; although $\mathsf{val}^{\mathsf{msg}}$ is a hash of *all* the messages sent in the network, each node can only access the messages that it "handled" (sent or received) during its own execution. This is all that is required to certify the execution of $\mathcal{D}$, because a message that was neither sent nor received by a node does not influence its immediate execution.

Modeling the distributed algorithm in detail. Before proceeding with the construction we must give a formal model for the internal computation at each network node, as our goal will be to certify that each step of this computation was carried out correctly. It is convenient to think of each round of a distributed algorithm as comprising three phases:

1. A *compute* phase, where each node computes the messages it will send in the current round and writes them on a special output tape. In this phase nodes may also change their internal state.

2. A *send* phase, where nodes send the messages that they produced in the compute phase. The internal states of the nodes do not change.

3. A *receive* phase, where nodes receive the messages sent by their neighbors and write them on a special tape. The internal states of the nodes do not change.

The compute phase at each node is modeled by a RAM machine $M_{\mathcal{D}}$ that uses the following memory sections:

- Env: a read-only memory section describing the node's environment — its neighbors and port numbers, and any additional prior information it might have about the network before the computation begins.
- In: a read-only memory section that contains the input to the node.
- Read: a read-only input memory section that contains the messages that the node received in the previous round.
- Mem: a read-write working memory section, which contains the node's internal state.
- Write: a write-only memory section where the machine writes the messages that the node sends to its neighbors in the current round. In the final round of the distributed algorithm, this memory section contains the final output of the node.

The state of the RAM machine, which we denote by st, includes the following information:

- Whether the machine will read or write in the current step,
- The memory location that will be accessed,
- If the next step is a write, the value to be written and the next state to which the RAM machine will transition after writing,
- If the next step is a read, the states to which the RAM machine will transition upon reading 0 or 1 (respectively).

(We assume for simplicity that the memory is Boolean, that is, each cell contains a single bit.)

The send and receive phases can be thought of as follows:

- The send phase is a sequence of $2|E|$ *send steps*, each indexed by a directed edge $v \to u$, ordered lexicographically, first by sender v and then by receiver u. In send step $v \to u$ the message created by v for u in the current round is sent on the edge between them.
- The receive phase is similarly a sequence of $2|E|$ *receive steps*, indexed by the directed edges of the graph, and ordered lexicographically, again first by the sending node and then the receiving node. In receive step $v \to u$ the message created by v for u in the current round is received at node u.

Intuitively, using the same ordering for both the send and the receive phase means that messages are received in the exact same order in which they are sent.

Certifying the computation of one node. After constructing the DMT, each node has access to hashes of the messages it received during the execution of the algorithm. It would be tempting think of these hashes as input digests, since in some sense incoming messages do serve as inputs, and to use a RAM SNARG in a black-box manner to certify that the node carried out its computation correctly. The problem with this approach is the notion of soundness we require, which is similar to that of a plain SNARG, but differs from the soundness of a RAM SNARG: in our model, the nodes have access to their neighborhoods and their individual inputs at verification time, so in some sense they jointly have the entire input to the computation. We require that the prover should not be able to *prove a false statement*, that is, find a configuration (G, x) and a convincing proof that $\mathcal{D}(G, x)$ outputs a value y which is not the true output of $\mathcal{D}$ on (G, x). In contrast, the RAM SNARG verifier has only a digest of the input — although it may also have a short explicit input, the bulk of the input is implicit and is "specified" only by the digest, i.e., it is not uniquely specified. The soundness of RAM SNARGs, in turn, is weaker: they only require that the prover not be able to find a single digest and two convincing proofs for contradictory statements about the same digest. Because of this difference, we cannot use RAM SNARGs as a black box, and instead we directly build the LVD-SNARG from the same primary building block used in recent RAM SNARG constructions [CJJ21b, KLVW23, CGJ+23]: a *non-interactive batch argument for* NP (BARG).

A (non-interactive) BARG is an argument that proves a set (a batch) of NP statements $x_1, \ldots, x_k \in \mathcal{L}$, for an NP language $\mathcal{L}$, such that the size of the proof increases very slowly (typically, polylogarithmically) with the number of statements k. (This is not a SNARG for NP, since the proof size does grow polynomially with the length of *one* witness.) Several recent works [CJJ21a, KLVW23, CGJ+23] have constructed from standard assumptions BARGs with proof size $\mathrm{poly}(\lambda, s, \log k)$, where s is the size of the circuit that verifies the NP-language. These BARGs were then used in [CJJ21b, KLVW23] to construct RAM SNARGs for P. Following their approach, we use BARGs to construct our desired LVD-SNARG. Roughly, our method is as follows.

At each node v, we use a hash family with local openings to commit to the sequence of RAM machine configurations that v goes through: for example, if the history of the memory section Read at node v is given by $\mathsf{Read}_v^0, \mathsf{Read}_v^1, \ldots$ (with Read_v^0 being the initial contents of the memory section, Read_v^1 being the contents following the first step of the algorithm, and so on), then we first compute individual hashes of $\mathsf{Read}_v^0, \mathsf{Read}_v^1, \ldots$, and then hash together all these hashes to obtain a hash $\mathsf{val}_v^{\mathsf{Read}}$ representing the sequence of contents on this memory section at node v. Similarly, let $\mathsf{val}_v^{\mathsf{Mem}}, \mathsf{val}_v^{\mathsf{Write}}$ be commitments to the memory section contents of Mem and Write at v, and let $\mathsf{val}_v^{\mathsf{st}}$ be a hash of the sequence of internal RAM machine states that node v went through during the execution of $\mathcal{D}$ (in all rounds).

We now construct a BARG to prove the following statement (roughly speaking): for each round r and each internal step i of that round, there exist openings of $\mathsf{val}_v^{\mathsf{Read}}$, $\mathsf{val}_v^{\mathsf{Mem}}$, $\mathsf{val}_v^{\mathsf{Write}}$ and

$\mathsf{val}^{\mathsf{st}}_v$ in indices (r,i) and $(r,i+1)$ to values $\mathsf{st}_{r,i}$, $\mathsf{st}_{r,i}$, $\mathsf{hRead}_{r,i}$, $\mathsf{hRead}_{r,i+1}$, $\mathsf{hMem}_{r,i}$, $\mathsf{hMem}_{r,i+1}$, $\mathsf{hWrite}_{r,i}$, $\mathsf{hWrite}_{r,i+1}$, such that the following holds:

- If i is a step of the compute phase, and $\mathsf{st}_{r,i}$ indicates that the machine reads from location ℓ in memory section $\mathsf{TP} \in \{\mathsf{Read}, \mathsf{Mem}, \mathsf{Write}\}$, then there exists an opening of $\mathsf{hTP}_{r,i}$ in location ℓ to a bit b such that upon reading b, $M_{\mathcal{D}}$ transitions to $\mathsf{st}_{r,i+1}$. Moreover, the hash values of the memory sections $\mathsf{hRead}, \mathsf{hMem}, \mathsf{hWrite}$ do not change in step (r,i): we have $\mathsf{hRead}_{r,i} = \mathsf{hRead}_{r,i+1}$, $\mathsf{hMem}_{r,i} = \mathsf{hMem}_{r,i+1}$, and $\mathsf{hWrite}_{r,i} = \mathsf{hWrite}_{r,i+1}$.
- If i is a step of the compute phase, and $\mathsf{st}_{r,i}$ indicates that the machine writes the value b to location ℓ in memory section $\mathsf{TP} \in \{\mathsf{Mem}, \mathsf{Write}\}$, then there exists an opening of $\mathsf{hTP}_{r,i+1}$ in location ℓ to the bit b. Moreover, the hash values of the other memory sections $\{\mathsf{hRead}, \mathsf{hMem}, \mathsf{hWrite}\} \setminus \mathsf{TP}$ do not change in step (r,i).
- If i is a step of the send phase indexed by $v \to u$ (i.e., a step where v sends a message to u), then there exists a message $m$such that $\mathsf{val}^{\mathsf{msg}}$ opens to m in index $I^{\mathsf{msg}}_{v,u,r}$ and hWrite opens to m in index d.
- If i is a step of the receive phase indexed by $u \to v$ (i.e., a step where v receives a message from u), and u is the d^{th} neighbor of v, then there exists a message m such that $\mathsf{val}^{\mathsf{msg}}$ opens to m in index $I^{\mathsf{msg}}_{u,v,r}$ and hRead opens to m in index d.

In addition to the requirements above, we must ensure that whenever the contents of a memory section change, they change only in the location to which the machine writes, and the hash value for the memory section changes accordingly; for example, if in step i of the compute phase of round r the machine writes value b to location ℓ of memory section TP, then we must ensure not only that $\mathsf{TP}_{r,i+1}$ opens to b in location ℓ, but also that $\mathsf{hTP}_{r,i}$ and $\mathsf{hTP}_{r,i+1}$ are hash values of arrays that differ *only* in location ℓ. To do so, we use a hash family that also supports write operations (in addition to local openings), as in the definition of a *hash tree* in [KPY19]. For example, a Merkle tree [Mer89] satisfies all of the requirements for a hash tree.

We use the hash write operations to include the following additional requirements as part of our BARG statement:

- For each step i of the compute phase of each round r, if $\mathsf{st}_{r,i}$ indicates that the machine writes value b to location ℓ in memory section $\mathsf{TP} \in \{\mathsf{Mem}, \mathsf{Write}\}$, then there exists an opening showing that $\mathsf{hTP}_{r,i}$ and $\mathsf{hTP}_{r,i+1}$ differ only in location ℓ.
- For each step of the receive phase of each round r, if the message received in this step is written to location ℓ of Read, then there exists an opening showing that $\mathsf{hRead}_{r,i}$, $\mathsf{hRead}_{r,i+1}$ differ only location ℓ.

There is one main obstacle remaining: in all known BARG constructions, the BARG is only as succinct as the circuit that verifies the statements it claims. In our case, the statements involve the indices $I^{\mathsf{msg}}_{v,u,r}$, as well as port numbers of the various neighbors of v, and the corresponding opening paths. These must be "hard-wired" into the circuit, because they are obtained from the DMT, i.e., they are external to the BARG itself. Each node v may need to use up to $n-1$ indices and openings, one for every neighbor, so we cannot afford to use a circuit that explicitly encodes them.

Indirect indexing. To avoid hard-wiring the indices and openings into the BARG, each node v computes a commitment to the indices, in the form of a locally openable hash of the following arrays:

- $\mathsf{Ind}^{\mathsf{in}}(v)$, an array containing at each index $I_{v \to u}$ the value $I_v || I_{v \to u}$.
- $\mathsf{Ind}^{\mathsf{out}}(v)$, an array containing at each index $I_{v \to u}$ the value $I_u || I_{u \to v}$.
- $\mathsf{Port}(v)$, an array containing at each index k the value $\perp$ if $v_k \notin N(v)$, or the value d if v_k

is the d$^{\text{th}}$ neighbor of v.

Denote these hash values by $\mathsf{val}^{\mathsf{in}}(v)$, $\mathsf{val}^{\mathsf{out}}(v)$, and $\mathsf{val}^{\mathsf{Port}}(v)$, respectively.

Now we can augment the BARG, and have it prove the following: at every round r and step i of the send phase, there exists a port number d, an index I, a message m, and appropriate openings to the hash values $\mathsf{val}^{\mathsf{Port}}, \mathsf{val}^{\mathsf{out}}, \mathsf{hWrite}_{r,i}, \mathsf{val}^{\mathsf{msg}}$ such that

- $\mathsf{val}^{\mathsf{Port}}$ opens to d in location ℓ such that v_ℓ is the node that v sends a message to in step i of every send phase,
- $\mathsf{val}^{\mathsf{out}}$ opens to I in location d,
- $\mathsf{hWrite}_{r,i}$ opens to m in location d, and
- $\mathsf{val}^{\mathsf{msg}}$ opens to m in location $I \parallel r$.

Similarly, at every round r and step i of the receive phase, there exist a port number d, an index I, a message m, and appropriate openings to the hash values $\mathsf{val}^{\mathsf{Port}}, \mathsf{val}^{\in}, \mathsf{hRead}_{r,i+1}, \mathsf{val}^{\mathsf{msg}}$ such that

- $\mathsf{val}^{\mathsf{Port}}$ opens to d in location k such that v_k is the node that v receives a message to in step i of every send phase,
- $\mathsf{val}^{\mathsf{in}}$ opens to I in location d,
- $\mathsf{hRead}_{r,i+1}$ opens to m in location d,[11] and
- $\mathsf{val}^{\mathsf{msg}}$ opens to m in location $I \parallel r$.

The circuit verifying this BARG's statement requires only the following values to be hard-wired: $\mathsf{val}^{\mathsf{st}}$, $\mathsf{val}^{\mathsf{msg}}$, $\mathsf{val}^{\mathsf{in}}$, $\mathsf{val}^{\mathsf{out}}$, $\mathsf{val}^{\mathsf{Port}}$, $\mathsf{val}^{\mathsf{Read}}$, $\mathsf{val}^{\mathsf{Mem}}$, $\mathsf{val}^{\mathsf{Write}}$. During verification, however, node v must verify that indeed, the hashes $\mathsf{val}^{\mathsf{in}}(v)$, $\mathsf{val}^{\mathsf{out}}(v), \mathsf{val}^{\mathsf{Port}}(v)$ are correct: node v can do this by re-computing the hashes, using the index I_v which is stored as part of its certificate, the port numbers $\{I_{v \to u}\}_{u \in N(v)}$ that it accesses during verification, and also indices $\{I_u\}_{u \in N(v)}$ and port numbers $\{I_{u \to v}\}_{u \in N(v)}$ that v's neighbors can provide in verification time.

The soundness of our construction. Following recent works, instead of using regular BARGs, we use *somewhere extractable* BARGs (seBARGs): an seBARG is a BARG with the following somewhere argument of knowledge property: for some index i, using the appropriate trapdoor, the seBARG proof completely reveals an NP-witness for the i^{th} statement. Importantly, the trapdoor is generated alongside the crs and the crs *hides* the binding index i: the (computationally bounded) prover cannot tell from the crs alone the binding index i. Conveniently, BARGs can be easily transformed into seBARGs [CJJ21b, KLVW23], without adding more assumptions.

The overall idea of our soundness proof is similar to the one in [CJJ21b, KLVW23], although there are some complications (e.g., the need to switch between different nodes of the network as we argue correctness). Assume for the sake of contradiction that a cheating prover is able to convince the network to accept a false statement with non-negligible probability. We proceed by induction over the rounds and internal steps (inside each compute, send and receive phase) of the distributed algorithm: in the induction we track the true state of the distributed algorithm, and compare witnesses extracted from the seBARG to this state. Informally speaking, we prove that from a proof that is accepted, using the appropriate trapdoor and crs, we can extract at each step a witness that must be compatible with the true execution of the distributed algorithm, otherwise we break the seBARG. In the last round, this means that the output encoded in the witness is the correct output of the distributed algorithm. But this contradicts our assumption that the adversary convinces the network of a *false* statement.

[11] As explained above, we actually require that this opening show that $\mathsf{hRead}_{r,i}$ and $\mathsf{hRead}_{r,i+1}$ only differ in the location d and $\mathsf{hRead}_{r,i+1}$ opens to m in that location.

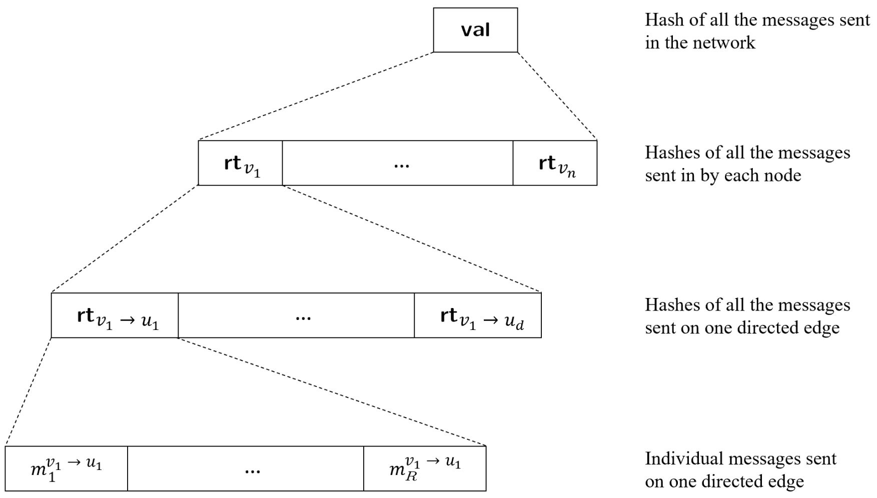
val
Hash of all the messages sent in the network
rt_{v_1}
...
rt_{v_n}
Hashes of all the messages sent in by each node
$\mathsf{rt}_{v_1 \to u_1}$
...
$\mathsf{rt}_{v_1 \to u_d}$
Hashes of all the messages sent on one directed edge
$m_1^{v_1 \to u_1}$
...
$m_R^{v_1 \to u_1}$
Individual messages sent on one directed edge

Figure 2.1: The structure of the DMT constructed over the messages.

2.3 Distributed Merkle Trees

Finally, we briefly sketch the construction of the distributed Merkle tree used in the previous section.

The structure of the DMT. Recall that our goal with the distributed Merkle tree (DMT) is to hash together all the messages sent during the execution of the distributed algorithm, in such a way that a node can produce openings for its own sent messages. Accordingly, we construct the DMT in several layers (see Figure 2.1):

- At the lowest level, for each node v and neighbor $u \in N(v)$, node v hashes together the messages $(m_1^{v\to u}, m_2^{v\to u}, \ldots)$ that it sent to node u, obtaining a hash $\mathsf{rt}_{v\to u}$.
- At the second level, each node v hashes together the hashes of its different edges, $\{\mathsf{rt}_{v\to u}\}_{u\in N(v)}$, ordered by the port numbers $I_{v\to u}$, obtaining a hash rt_v which we refer to as *v's local root.*
- Finally, the nodes collaborate to hash their local roots $\{\mathsf{rt}_v\}_{v\in V}$ together to obtain a global root val. The nodes are initially not ordered, but during the creation of the DMT, the local roots $\{\mathsf{rt}_v\}_{v\in V}$ are ordered; and each node v obtains an index I_v for its local root, and the corresponding opening path from val to rt_v.

Constructing the DMT. After each node computes the hash values $\mathsf{rt}_{v\to u}$ for each of its neighbors $u \in N(v)$, we continue by having the network nodes compute a spanning tree ST of the network, with each node v learning its parent $p_v \in N(v) \cup \{\perp\}$, and its children $C_v \subseteq N(v)$. The root v_0 of the spanning tree is the only node that has a null parent, i.e., $p_{v_0} = \perp$.

We note that using standard techniques, a rooted spanning tree can be constructed in $O(D)$ rounds in networks of diameter D, using $O(\log n)$-bit messages in every round; this can be done even if the nodes do not initially know the diameter D or the size n of the network, and it does not require the root to be chosen or known in advance [Lyn96].

After constructing the spanning tree, we compute the DMT in three stages: in the first stage nodes compute a Merkle tree of their own values, in the second we go “up the spanning tree” to compute the global Merkle tree, and the third stage goes “down the tree” to obtain the indices and the openings.

Stage 1: Local hash trees. Let $\vec{x}_v$ be a vector containing the values $\{\mathsf{rt}_{v \to u}\}_{u \in N(v)}$ held by node v, ordered by the port number of the neighbor $u \in N(v)$ at node v (padded up to a power of 2, if necessary). For each node v and neighbor $u \in N(v)$, let $I_{v \to u}$ be a binary representation of the port number of u at v (again, possibly padded).

Each node v computes its local root rt_v by building a Merkle tree over the vector $\vec{x}_v$, as well as an opening $\rho_{v \to u}$ for the index $I_{v \to u}$, for each neighbor $u \in N(v)$. We let $\beta_v = \{(I_{v \to u}, \rho_{v \to u})\}_{u \in N(v)}$.

Stage 2: Spanning tree computation. The nodes jointly compute a spanning tree ST of the network, storing at every node v the parent $p_v \in N(v)$ of v and the children $C_v \subseteq N(v)$ of v. In the sequel, we denote by v_0 the root of the spanning tree.

Stage 3: Convergecast of hash-tree forests. In this stage, we compute the global hash tree up the spanning tree ST, with each node v merging some or all of the hash-trees received from its children and sending the result upwards in the form of a set of HT-roots annotated with height information.

Each node v receives from each child $c \in C_v$ a set S_c of pairs (rt, h), where rt is a Merkle-tree root, and $h \in \mathbb{N}$ is the cumulative height of the Merkle tree. Node v now creates a forest F_v, as follows:

1. Initially, F_v contains the roots sent up by v's children, and a new leaf representing v's local hash tree: $F_v = \{(\mathsf{rt}_v, 0)\} \cup \bigcup_{c \in C_v} S_c$.

2. While there remain two trees in F_v whose roots rt_0 and rt_1 have the same cumulative height h (note — we do not care about the actual height of the trees in the forest F_v, but rather about their cumulative height, represented by the value h in the node (rt, h)): node v chooses two such trees and merges them, creating a new root rt of cumulative height $h+1$ and placing (rt_0, h) and (rt_1, h) as the left and right children of $(\mathsf{rt}, h+1)$, respectively.

3. When there no longer remain two trees in F_v whose roots have the same cumulative height:

   - If $v \neq v_0$ (that is, v is not the root of the spanning tree), node v sends its parent, p_v, the set S_v of tree-roots in F_v. The size of this set is at most $O(\log n)$, since it contains at most one root of any given cumulative height (if there were two roots of the same cumulative height, node v would merge them).
   - At the root v_0, we do not want to halt until F_v is a single tree. If F_v is not yet a single tree, node v_0 must pad the forest by adding "dummy trees" so that it can continue to merge. To do so, node v_0 finds the tree-root (rt, h) that has the smallest cumulative height h in F_v. It then creates a "dummy" Merkle-tree of height h, with root $(\perp, h)$, and adds it to F_{v_0}. Following this addition, there exist two tree-roots of cumulative-height h (the original tree-root (rt, h) and the "dummy" tree-root $(\perp, h)$), which v_0 now merges. It continues on with this process, at each step choosing a tree with the smallest remaining height, and either merging it with another same-height tree if there is one, or creating a dummy tree and merging the shortest tree with it.

When the last stage completes, the forest F_{v_0} computed by node v_0 (the root of the spanning tree) is in fact a single tree, whose root, dented by val, is the root of the global Merkle tree.

Stage 4: Computing hash-tree indices and openings. In this stage we proceed down the spanning tree, forwarding the global root val downwards. In addition, as we move down the tree, each node v annotates its forest F_v with indices and opening paths: first, it receives from its parent p_v an index and opening for every tree-root $(\mathsf{rt}, h) \in F_v$ that it sent upwards to p_v. Then, it extends this information "downwards" inside F_v, annotating each inner node and leaf in F_v with their index and opening path from the global root val: for example, if (rt_0, h) and (rt_1, h) are the left and right children of $(\mathsf{rt}, h+1)$ in F_v, and the index and opening path for $(\mathsf{rt}, h+1)$ are already known to be I and ρ (resp.), then the index and opening path for (rt_0, h) are $I||0$ and $\rho||\mathsf{rt}_1$ (resp.), and the index and opening path for (rt_1, h) are $I||1$ and $\rho||\mathsf{rt}_0$ (resp.).

Outputs. The final output at node v is $(\mathsf{val}, \mathsf{rt}_v, I_v, \rho_v, \beta_v)$. (For the LVD-SNARG, at the end of the proving stage, β_v is discarded, as it is too long to store. However, $\mathsf{val}, \mathsf{rt}_v, I_v$ and ρ_v are part of node v's certificate.)

We remark that for our purposes, it is not necessary for the nodes to *certify* that they computed the DMT correctly: after obtaining the global root and the relevant openings, the nodes simply use the DMT as they would use a centralized hash with local openings. The completeness proof of our LVD-SNARG relies on the fact that a correctly-computed DMT will open to the correct information everywhere, but the soundness proof does not rely the details of the construction, only on the fact that the value obtained by opening various locations of the DMT matches the true execution of the algorithm.

3 Preliminaries

3.1 Modeling Distributed Networks

In this section, we give a more formal overview of our network model; this model is standard in the area of distributed network algorithms (see, e.g., the textbook [Pel00]).

A distributed network is modeled as an undirected, connected[12] graph $G = (V, E)$, where the nodes V of the network are the processors participating in the computation, and the edges E represent bidirectional communication links between them.

For a node $v \in V$, we denote by $N_G(v)$ (or by $N(v)$, if G is clear from context) the neighborhood of v in the graph G. The communication links (i.e., edges) of node v are indexed by *port numbers*, with $I_{u \to v} \in [n]$ denoting the port number of the channel from v to its neighbor u. The port numbers of a given node need not be contiguous, nor do they need to be symmetric (that is, it might be that $I_{v \to u} \neq I_{u \to v}$). We assume that the neighborhood $N(v)$ and the port numbering at node v are known to node v during the verification stage; the node does not necessarily need to have them stored in memory at the beginning of the verification stage, but it should be able to generate them at verification time (e.g., by probing its neighborhood, opening communication sessions with its neighbors one after the other; or, in the case of a wireless network, by running a neighbor-discovery protocol).

In addition to knowing their neighborhood, we assume that each node $v \in V$ has a unique identifier; for convenience we conflate the unique identifier of a node v with the vertex v representing v in the network graph. We assume that the UID is represented by a logarithmic number of bits in the size of the graph. No other information is available; in particular, we do not assume that the nodes know the size of the network, its diameter, or any other global properties.

A *(synchronous) distributed network algorithm* proceeds in synchronized rounds, where in each round, each node $v \in V$ performs some internal computation, then sends a (possibly different) message on each edge $\{v, u\} \in E$. The nodes then receive the messages sent to them, and the next round begins. Eventually, each node halts and produces some output.

[12]We consider only connected networks, since in disconnected networks one can never hope to carry out any computation involving more than one connected component. Also, it is fairly standard to assume an undirected graph topology, i.e., bidirectional communication links, although directed networks are also considered sometimes (for instance, in [BFO22]).

Distributed decision tasks. In the literature on distributed decision and certification, network properties are referred to as *distributed languages*. A distributed language is a family of *configurations* (G, x), where G is a network graph and $x : V \to \{0,1\}^*$ assigns a string $x(v)$ to each node $v \in V$. The assignment x may represent, for example, the input to a distributed computation, or the internal states of the network nodes. We assume that $|x(v)|$ is polynomial of the size of the graph. We usually refer to x as an *input assignment*, since for our purposes it represents an input to the decision task.

A distributed decision algorithm is a distributed algorithm at the end of which each node of the network outputs an acceptance bit. The standard notion of *acceptance* in distributed decision [Pel00] is that the network accepts if and only if all nodes accept; if any node rejects, then the network is considered to have rejected.

Notation. When describing the syntax (interface) of a distributed algorithm, we describe the input to the algorithm as a triplet $(\alpha; G; \beta)$, where

- α is a value that is given to all the nodes in the network. Typically this will be the common reference string.
- $G = (V, E)$ is the network topology on which the algorithm will run.
- $\beta : V \to \{0,1\}^*$ is a mapping assigning a local input to every network node. Each node $v \in V$ receives only $\beta(v)$ at the beginning of the algorithm, and does not initially know the local values $\beta(u)$ of other nodes $u \neq v$.

We frequently abuse notation by writing a sequence of values or mappings instead of a single one for α or β (respectively); e.g., when we write that the input to a distributed algorithm is $(a, b; G; x, y)$, we mean that every node $v \in V(G)$ is initially given $a, b, x(v), y(v)$, and the algorithm executes in the network described by the graph G.

The unique identifier (UID) of each node is drawn from a set $\mathcal{U}$, and we assume that it can be described in $O(\log n)$ bits when the network size is n. The UID is given to the node as part of its local input, but we omit it from the notation, that is, by input $(\alpha; G; \beta)$, we mean each node $v \in V(G)$ inputs α, $\beta(v)$, and v, where v is it's UID.

The *output* of a distributed algorithm in a network $G = (V, E)$ is described by a mapping $o : V \to \{0,1\}^*$ which specifies the output $o(v)$ of each node $v \in V$. As we explained above, in the case of *decision algorithms*, the output is a mapping $o : V \to \{0,1\}$, and we say that the algorithm *accepts* if and only if all nodes output 1 (i.e., $\bigwedge_{v \in V} o(v) = 1$). We denote this event by "$\mathcal{D}(\alpha; G; \beta) = 1$," where $\mathcal{D}$ is the distributed algorithm, and $(\alpha; G; \beta)$ is its input (as explained above).

In general, when describing objects that depend on a specific graph G, we include G as a subscript: e.g., the neighborhood of node v in G is denoted $N_G(v)$. However, when G is clear from the context, we omit the subscript and write, e.g., $N(v)$.

3.2 Recursive Hash Families with Local Openings

A hash family with local openings allows a sender to produce a short hash value of a long input (which we will refer to as an input vector), and then locally open specific locations in the input while providing a short certificate for them. In [KPY19], this definition was extended to a *"hash tree,"* where the sender can also perform write operations on its long input, change the hash value accordingly, and provide a short certificate for the update. Merkle [Mer89] constructed such a family from collision-resistant hash functions (CRH). The original construction by Merkle has some useful properties that we can properly define and use once we open the black-box and externalize the underlying CRH. So, we first give here the definition of CRH for completeness and then proceed to define recursive hash families with local openings (MT, which stands for Merkle tree). We remark that we use the algorithm name conventions of a hash family with local openings and not those of a hash tree, to avoid conflating names later on, but we do extend the syntax and properties to match a hash tree, as in [KPY19].

Definition 3.1 (CRH). *An ensemble of hash function families* $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda \in \mathbb{N}}$*, where for every* $\lambda \in \mathbb{N}$*,* $\mathcal{H}_\lambda = \left\{h : \{0,1\}^{2\lambda} \to \{0,1\}^\lambda\right\}$ *is a family of functions, is collision-resistant if there exists a negligible function* $\operatorname{negl}(\cdot)$ *such that for any poly-time adversary* $\mathcal{A}$ *and every* $\lambda \in \mathbb{N}$*, the following holds:*

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

HT: Syntax. A hash family (HT) with succinct local openings consists of the following algorithms:

$\mathsf{Gen}(1^\lambda) \to \mathsf{hk}$. A randomized algorithm that takes the security parameter λ in unary representation, and outputs a hash key.

$\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{val}$. A polynomial-time algorithm that takes a bit vector x and a hash key hk, and outputs a hash value val.

$\mathsf{Open}(\mathsf{hk}, x, i) \to (b, \rho_i)$. A polynomial-time algorithm that takes a hash key hk, a bit vector x and an index i, and outputs a bit b and an opening ρ_i.

$\mathsf{WOpen}(\mathsf{hk}, x, i, b) \to (\mathsf{val}', \rho)$. A polynomial-time algorithm that takes a hash key hk, a vector x, an index i and a bit b, and outputs a hash value val' and an opening ρ.

$\mathsf{Verify}(\mathsf{hk}, \mathsf{val}, i, b, \rho) \in \{0,1\}$. A polynomial-time verification algorithm that takes a hash key hk, a value val, an index i, a bit b, and an opening ρ, and outputs an acceptance bit.

$\mathsf{WVerify}(\mathsf{hk}, \mathsf{val}, \mathsf{val}', i, b, \rho) \in \{0,1\}$. A polynomial-time verification algorithm that takes a hash key hk, two hash values val and val', an index i, a bit b, and an opening ρ, and outputs an acceptance bit.

Definition 3.2 (Properties of HT). *We say* $\mathsf{HT} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{WOpen}, \mathsf{Verify}, \mathsf{WVerify})$ *is a hash family with local openings if it satisfies the following properties:*

Opening completeness. *For any* $\lambda \in \mathbb{N}$*, any* $N \leq 2^\lambda$*, any* $x \in \{0,1\}^N$*, and any index* $i \in [N]$

$$\Pr\left[\begin{array}{l} b = x_i \\ \wedge\ \mathsf{Verify}(\mathsf{hk}, \mathsf{val}, i, b, \rho) = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{val} \leftarrow \mathsf{Hash}(\mathsf{hk}, x) \\ (b, \rho) = \mathsf{Open}(\mathsf{hk}, x, i) \end{array} \right] = 1.$$

Writing completeness. *For any* $\lambda \in \mathbb{N}$*, any* $N \leq 2^\lambda$*, any* $x \in \{0,1\}^N$ *and any index* $i \in [N]$*, let* x' *be the vector that equals to* b *in location* i *and equals to* x_j *in every location* $j \neq i$*. We have that:*

$$\Pr\left[\begin{array}{l} \mathsf{val}' = \mathsf{Hash}(\mathsf{hk}, x') \\ \wedge\ \mathsf{WVerify}(\mathsf{hk}, \mathsf{val}, \mathsf{val}', i, b, \rho) = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{val} \leftarrow \mathsf{Hash}(\mathsf{hk}, x) \\ (\mathsf{val}', \rho) = \mathsf{WOpen}(\mathsf{hk}, x, i, b) \end{array} \right] = 1$$

Succinctness. *In the completeness experiment above, the size of* val *and of* ρ_i *for every* $i \in [|x|]$ *is* $\mathrm{poly}(\lambda)$.

Collision resistance with respect to opening. *For any poly-size adversary* $\mathcal{A}$ *there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$ *and every* $N \leq 2^\lambda$,

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{hk}, \mathsf{val}, i, 0, \rho_0) = 1 & \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda) \\ \wedge\ \mathsf{Verify}(\mathsf{hk}, \mathsf{val}, i, 1, \rho_1) = 1 & (\mathsf{val}, i, \rho_0, \rho_1) \leftarrow \mathcal{A}(\mathsf{hk}) \end{array}\right] \leq \mathrm{negl}(\lambda).$$

Collision resistance with respect to writing. *For any poly-size adversary* $\mathcal{A}$ *there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$ *and every* $N \leq 2^\lambda$,

$$\Pr\left[\begin{array}{l|l} \mathsf{val}'_0 \neq \mathsf{val}'_1 & \\ \wedge\ \mathsf{WVerify}(\mathsf{hk}, \mathsf{val}, \mathsf{val}'_0, i, b, \rho_0) = 1 & \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda) \\ \wedge\ \mathsf{WVerify}(\mathsf{hk}, \mathsf{val}, \mathsf{val}'_1, i, b, \rho_1) = 1 & (\mathsf{val}, \mathsf{val}'_0, \mathsf{val}'_1, i, b, \rho_0, \rho_1) \leftarrow \mathcal{A}(\mathsf{hk}) \end{array}\right] \leq \mathrm{negl}(\lambda).$$

Remark 3.3. Hash families with local openings are typically defined for bit vectors. This could be generalized to accommodate string arrays by simply applying Open and Verify on a set of consecutive indices (but still applying Hash to the string array). This is equivalent to simply open in multiple locations, and verify accordingly. In this work, we sometimes abuse notation and refer to Open and Verify for arrays of strings.

Definition 3.4 (MT: recursively cinstructable HT)**.** *A* recursively constructable *hash family with local openings* $\mathsf{MT} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{WOpen}, \mathsf{Verify}, \mathsf{WVerify})$ *satisfies all of the properties of Definition 3.2, with the additional following property:*

Recursive constructability. *There exists a collision-resistant ensemble of hash families* $\mathcal{H}$*, such that the following holds. For every* $\lambda \in \mathbb{N}$*, every* $N \leq 2^\lambda$ *and* hk *such that* $\Pr[\mathsf{hk} = \mathsf{Gen}(1^\lambda)] > 0$*, there exists a function* $h \in \mathcal{H}_\lambda$*, such that for every two vectors* $x_1, x_2 \in \{0,1\}^N$*, we have:*

- $\mathsf{Hash}(\mathsf{hk}, x_1||x_2) = h(\mathsf{hk}, \mathsf{Hash}(\mathsf{hk}, x_1)||\mathsf{Hash}(\mathsf{hk}, x_2))$.
- *For* $i \in [N]$*, let:* $(b_1, \rho_1) = \mathsf{Open}(\mathsf{hk}, x_1, i)$ *and* $(b_2, \rho_2) = \mathsf{Open}(\mathsf{hk}, x_2, i)$*. Then,*
  - $\mathsf{Open}(\mathsf{hk}, x_1||x_2, i) = (b_1, \rho_1||\mathsf{Hash}(\mathsf{hk}, x_2))$*, and*
  - $\mathsf{Open}(\mathsf{hk}, x_1||x_2, N + i) = (b_2, \rho_2||\mathsf{Hash}(\mathsf{hk}, x_1))$.

Remark 3.5. Note that Merkle trees do satisfy these properties, if we define the verification algorithm to take an opening that is written from the leaf to the root, and not the other way around.

Remark 3.6. For some of our use cases, and in particular in Section 4.1, we only require a hash family with local openings and do not use the recursive constructability property. In these cases, we denote the hash family HT instead of MT.

Theorem 3.7 ([Mer89, Ajt96, GGH11, Dam87])**.** *Recursive hash families with local openings exist assuming either (1) Discrete log or (2) LWE.*

3.3 RAM SNARGs

A $\mathsf{RAM\ SNARG}$ allows a prover to prove to a verifier that $M(x) = 1$ for some machine M, where the verifier does not have access to x itself but to a *digest* of it, which is much shorter. We use the notion of *flexible* $\mathsf{RAM\ SNARGs}$ that are defined with respect to a hash family with local openings $\mathsf{HT} = (\mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Open}, \mathsf{HT.Verify})$ [KLVW23].[13]

[13] A flexible $\mathsf{RAM\ SNARG}$ is defined with respect to a HT family, and its soundness is dependent on the collision resistance with respect to opening of the HT.

Syntax. A $\mathsf{RAM\ SNARG}$s delegation of a machine M consists of the following algorithms:

$\mathsf{Gen}(1^\lambda, T) \to \mathsf{crs}$. A randomized setup algorithm that takes as input a security parameter 1^λ and a time bound T, and outputs a common reference string crs.

$\mathcal{P}(\mathsf{crs}, x) \to (o, \pi)$. A polynomial-time algorithm that takes the crs and an instance x, and outputs a bit o and a proof π.

$\mathcal{V}(\mathsf{crs}, d, o, \pi) \to b$. A polynomial-time verification algorithm that takes the crs, a digest d, an output bit o and a proof π, and returns an acceptance bit b.

Remark 3.8 (Digest algorithm)**.** In the definition in [KLVW23], the common reference string crs includes a hash key hk, and the $\mathsf{RAM\ SNARG}$ includes also a digest algorithm $\mathsf{Digest}(\mathsf{crs}, x)$, that is defined to be $\mathsf{HT.Hash}(\mathsf{hk}, x)$. This definition is equivalent.

Definition 3.9 ($\mathsf{RAM\ SNARG}$)**.** *A* $\mathsf{RAM\ SNARG}$ *for a machine M which has a local state of size $S = S(n)$ and runs in time $T = T(n)$ on inputs of size n, satisfies the following properties.*

Completeness. *For any $\lambda, N \in \mathbb{N}$ such that $N \leq T(N) \leq 2^\lambda$ and any $x \in \{0,1\}^N$ such that $M(x)$ halts within T time steps, we have that*

$$\Pr\left[\begin{array}{l} o = M(x) \\ \wedge\ \mathcal{V}(\mathsf{crs}, \mathsf{hk}, d, o, \pi) = 1 \end{array} \middle| \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, T) \\ \mathsf{hk} \leftarrow \mathsf{HT.Gen}(1^\lambda) \\ d \leftarrow \mathsf{HT.Hash}(\mathsf{hk}, x) \\ (o, \pi) \leftarrow \mathcal{P}(\mathsf{crs}, x) \end{array}\right] = 1.$$

Succinctness. *In the completeness experiment above the size of the reference string* crs *and the proof π is at most* $\mathrm{poly}(\lambda, S, \log T)$*.*

(Adaptive) Soundness. *For any poly-size adversary $\mathcal{A}$ and polynomial $T = T(\lambda)$, there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for every $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, d, 0, \pi_0) = 1 \\ \wedge\ \mathcal{V}(\mathsf{crs}, d, 1, \pi_1) = 1 \end{array} \middle| \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, 1^T) \\ \mathsf{hk} \leftarrow \mathsf{HT.Gen}(1^\lambda) \\ (d, \pi_0, \pi_1) \leftarrow \mathcal{P}^*(\mathsf{crs}, \mathsf{hk}) \end{array}\right] \leq \mathrm{negl}(\lambda).$$

Remark 3.10 (Size of the local state)**.** The size S of the local state of the machine M, could usually be thought of as polylogarithmic in n since we could think about the entire work tape of the machine as part of the RAM, and the part of the memory that changes in each computation step must be of constant size. However, this requires us to consider machines that both read and write, where mostly, for the rest of the use cases, it is more convenient to think of read-only machines. Nevertheless, the known constructions either already support read-and-write operations, or easily transform from read-only to read-and-write without loss of generality.

3.3.1 $\mathsf{RAM\ SNARG}$s with Extended Completeness

The above definition for completeness of a $\mathsf{RAM\ SNARG}$ is standard. However, in the latest constructions of $\mathsf{RAM\ SNARG}$s [CJJ21b, KLVW23, CGJ+23], the actual completeness property achieved is slightly stronger: the input digest d is computed using a fixed, pre-chosen HT-family, and it does not matter whether the prover is the one who computed the digest d and the corresponding openings $\{\rho_i\}_{i \in [|x|]}$ to the input bits, or whether the prover is *given* the digest and the openings and then asked to compute the $\mathsf{RAM\ SNARG}$ proof. In fact, the prover can be given *any* digest d' that opens to the input at all the locations that the Turing machine accesses during its computation, even if d' is actually the digest of some other input x', as long as the real input x and the other input x' agree on all locations accessed by the Turing machine. For example, if x is a prefix of x', and the machine never reads past the end of x, then the prover is able to work with a digest of x' instead of x (still if x' is much longer then x, the efficiency and succinctness will be damaged, of course). This seemingly minor observation turns out to be useful for our construction in Section 4.1, and we refer to the stronger property as *extended completeness*.

To formalize extended completeness, we add to the syntax a new prover algorithm $\mathcal{P}'$:

$\mathcal{P}'(\mathsf{crs}, \mathsf{hk}, x, d, \{\rho_i\}_{i\in[|x|]}) \to (o, \pi)$. A polynomial time algorithm that takes crs, a hash key hk, an instance x, a digest d, and a set of openings $\{\rho_i\}_{i\in[|x|]}$, and outputs a bit o and a proof π.

Using the same definitions of Gen and $\mathcal{V}$ and $\mathcal{P}$ as above, we now require the following:

Definition 3.11. *(*RAM SNARG *with extended completeness) A* RAM SNARG $= (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ *with extended completeness for a machine M that has a local state of size $S = S(n)$ and runs in time $T = T(n)$ on inputs of size n, satisfies the succinctness, efficiency, and soundness properties from Definition 3.9. In addition, there exists another prover algorithm $\mathcal{P}'$, that satisfies the following property.*

Extended Completeness. *For any $\lambda, N \in \mathbb{N}$ such that $N \le T(N) \le 2^\lambda$, any $x \in \{0,1\}^N$, such that $M(x)$ halts within T time steps, for any d and any $\{\rho_i\}_{i\in[|x|]}$, we have that*

$$\Pr\left[\begin{array}{l} \left(\begin{array}{l} \quad o = M(x) \\ \quad \wedge\ \mathcal{V}(\mathsf{crs}, \mathsf{hk}, d, o, \pi) = 1 \end{array}\right) \\ \vee\ \exists i \in [|x|]: \\ \quad \mathsf{HT.Verify}(\mathsf{hk}, d, i, x[i], \rho_i) = 0 \end{array} \middle| \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, T) \\ \mathsf{hk} \leftarrow \mathsf{HT.Gen}(1^\lambda) \\ (o, \pi) \leftarrow \mathcal{P}'(\mathsf{crs}, \mathsf{hk}, x, d, \{\rho_i\}_{i\in[|x|]}) \end{array}\right] = 1.$$

In other words, *if* the prover is given an input x, and a digest d together with openings that open to x correctly, then it proves the statement with the digest d. If the prover is given a digest that does not match the input, it can (and should, by soundness and collision-resistance of the HT family) fail.

Theorem 3.12 ([CJJ21b, WW22, KLVW23, CGJ+23]). RAM SNARG *for* P *(with extended completeness) exist assuming either: (1)* LWE*, (2)* DLIN*, or (3) subexponential* DDH.

3.4 Somewhere Extractable Batch Arguments (seBARGs)

A (non-interactive) batch argument for NP [CJJ21a] allows us to prove k NP-statements with a certificate whose length dependence on k is sublinear. Specifically, we are interested in BARGs for index languages [CJJ21a]. An *index language* is a language of the form:

$$\{(C, i) \mid \exists w : \ C(i, w) = 1\}$$

where C is a Boolean circuit. Since the instances for index language are simply indices, when the batch argument is for the statements $(C, 1), \dots, (C, k)$, we can omit i from the inputs to the prover and the verifier, which allows a significant efficiency boost to the verifier, as it no longer has to read k inputs. Moreover, we are interested in a *somewhere-extractable* version of index-language BARGs (seBARGs), where we can program the crs to contain a trapdoor that allows to extract one witness from the BARG.

Syntax. A seBARG for index language consists of the following algorithms:

$\mathsf{Gen}(1^\lambda, k, 1^s, i) \to (\mathsf{crs}, \mathsf{td})$. A randomized setup procedure that takes a security parameter λ, the number of statements k, the size of the circuit 1^s, and an optional index i, and generates a common reference string crs and if provided an index i, a trapdoor td.

$\mathcal{P}(\mathsf{crs}, C, w_1, \ldots, w_k) \to (b, \pi)$. A polynomial-time prover algorithm that takes the crs, a circuit C and a list of witnesses $w_1, \ldots, w_k$, and outputs a bit b and a proof π.

$\mathcal{V}(\mathsf{crs}, C, \pi) \to b$. A polynomial-time verification algorithm that takes the crs, a circuit C, and a proof π and outputs an acceptance bit.

$\mathcal{E}(\mathsf{td}, C, \pi) \to w_i$. A polynomial-time extraction algorithm that takes a trapdoor td, a circuit C, and a proof π, and outputs a witness w_i.

Definition 3.13 (seBARG). *A* seBARG *satisfies the following requirements.*

Succinctness. *The length of the* crs *and of the proof* π *is at most* $\mathrm{poly}(s, \lambda, \log k)$.

Verifier Efficiency. *The verifier runs in time* $\mathrm{poly}(s, \lambda, \log k)$.

Completeness. *For any* $\lambda \in \mathbb{N}$ *and* $s = s(\lambda)$ *of size at most* 2^λ, *for any circuit* $C : [k] \times \{0,1\}^m \to \{0,1\}$ *of size at most* s, *any witnesses* $w_1, \ldots, w_k \in \{0,1\}^m$ *and any index* $i^* \in [k]$

$$\Pr\left[\mathcal{V}(\mathsf{crs}, C, \pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i^*) \\ \pi \leftarrow \mathcal{P}(\mathsf{crs}, C, w_1, \ldots, w_k) \end{array} \right] = 1.$$

Index hiding. *For any poly-size adversary* $\mathcal{A}$ *and polynomials* $k = k(\lambda)$ *and* $s = s(\lambda)$, *there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$

$$\Pr\left[\begin{array}{l} i_0, i_1 \in [k] \\ \mathcal{A}(\mathsf{crs}) = b \end{array} \;\middle|\; \begin{array}{l} (i_0, i_1) \leftarrow \mathcal{A}(1^\lambda) \\ b \leftarrow \{0,1\} \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_b) \end{array} \right] \leq \frac{1}{2} + \mathrm{negl}(\lambda).$$

Somewhere argument of knowledge. *For any poly-size adversary* $\mathcal{A}$, *polynomials* $k = k(\lambda)$ *and* $s = s(\lambda)$, *and index* $i^* = i^*(\lambda) \in [k(\lambda)]$, *there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, C, \pi) = 1 \\ \wedge\ C(i^*, w) = 0 \end{array} \;\middle|\; \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i^*) \\ (C, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ w \leftarrow \mathcal{E}(\mathsf{td}, C, \pi) \end{array} \right] \leq \mathrm{negl}(\lambda).$$

Theorem 3.14 ([CJJ21a, WW22, KLVW23, CGJ+23]). seBARG*s for* NP, *and in particular, for the index languages, exist assuming either: (1)* LWE, *(2)* DLIN, *or (3) subexponential* DDH.

4 Locally Verifiable Distributed SNARGs

In this section we give the formal definition of locally-verifiable distributed SNARGs (LVD-SNARGs), in the case of a global (centralized) prover (see Section 6 for the definition for a distributed prover). Next, in Section 4.1 we present our construction of LVD-SNARG for P with a global prover.

Syntax. A locally verifiable distributed SNARG consists of the following algorithms.

$\mathsf{Gen}(1^\lambda, n) \to \mathsf{crs}$. A randomized algorithm that takes as input a security parameter 1^λ and a graph size n, and outputs a common reference string crs.

$\mathcal{P}(\mathsf{crs}, G, x) \to \pi$. A prover algorithm that takes a crs and a configuration (G, x), and outputs an assignment of outputs to the nodes $y : V(G) \to \{0,1\}^*$ and an assignment of certificates to the nodes $\pi : V(G) \to \{0,1\}^*$.

$\mathcal{V}(\mathsf{crs}; G; x, \pi) \to b$. A *distributed decision algorithm* that takes as a common input to the entire network a common reference string crs, executes in the network G, where each node $v \in V(G)$ is assigned with an input $x(v)$ and a proof $\pi(v)$, and outputs acceptance bits $b : V \to \{0,1\}^*$.

Definition 4.1 (LVD-SNARG)**.** *Let $\mathcal{L}$ be a distributed language. An* LVD-SNARG $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ *for $\mathcal{L}$ must satisfy the following properties:*

Completeness. *For any $(G, x) \in \mathcal{L}$,*

$$\Pr\left[\ \mathcal{V}(\mathsf{crs}; G; x, \pi) = 1 \ \middle|\ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \pi \leftarrow \mathcal{P}(\mathsf{crs}; G; x) \end{array} \right] = 1.$$

Soundness. *For any poly-size algorithm $\mathcal{P}^*$ and polynomial $n = n(\lambda)$, there exists a negligible function* $\mathrm{negl}(\cdot)$ *such that*

$$\Pr\left[\begin{array}{l} (G, x) \notin \mathcal{L} \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, \pi) = 1 \end{array} \middle|\ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ (G, x, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \end{array} \right] \leq \mathrm{negl}(\lambda).$$

Succinctness. *The* crs *and the proof $\pi(v)$ at each node v are of length at most* $\mathrm{poly}(\lambda, \log n)$*.*

Verifier efficiency. *$\mathcal{V}$ runs in a single synchronized communication round, during which each node sends a (possibly different) message of length* $\mathrm{poly}(\lambda, \log n)$ *to each neighbor. At each node v, the local computation executed by $\mathcal{V}$ runs in time* $\mathrm{poly}(\lambda, |\pi(v)|, |x(v)|, \deg(v)) = \mathrm{poly}(\lambda, n)$*.*

Prover efficiency. *The prover runs in time* $\mathrm{poly}(\lambda, n)$*.*

4.1 LVD-SNARGs for P with a Global Prover

We proceed to show an LVD-SNARG for any property in P. Our construction uses a hash family with local openings (HT, see Section 3.2), a RAM SNARG for P (see Section 3.3), and a (perfectly secure and perfectly sound) distributed certification scheme for the size of the network. To certify the size of the network, we use the proof labeling scheme from [KKP05]:

Theorem 4.2 ([KKP05])**.** *There exists a distributed certification scheme for the size of the network, with certificate size $O(\log n)$, and one round of verification with message length $O(\log n)$.*

For a graph $G = (V, E)$, let $\{\sigma_G(v)\}_{v \in V}$ denote the (deterministic) labels assigned by the scheme of [KKP05] to each node $v \in V$.

4.1.1 Construction from $\mathsf{RAM\ SNARG}$s

Theorem 4.3. *Assume the existence of a* $\mathsf{RAM\ SNARG}$ *for* P *and a hash family with local openings. Then, for every graph language* $\mathcal{L} \in \mathsf{P}$ *there exists an* $\mathsf{LVD\text{-}SNARG}$ *with a global prover.*

Theorem 4.3, alongside with Theorems 3.7 and 3.12 imply the following corollary:

Corollary 4.4. *Let* $\mathcal{L} \in \mathsf{P}$*. Then, there exists an* $\mathsf{LVD\text{-}SNARG}$ *with global prover for* $\mathcal{L}$ *assuming a collision-resistant hash functions and either* LWE*, or* DLIN*, or subexponential* DDH*.*

Proof of Theorem 4.3. Given a configuration (G, x), let $V = \{v_1, \dots, v_n\}$ be the nodes of G in lexicographic order. (The order does not matter, but for concreteness, since the prover will use it, we fix a specific ordering.) Here, we do not assume anything about the UIDs of the nodes, other than the standard assumption that they can be represented in $O(\log n)$ bits). Let $L_{G,x}$ be an adjacency-list representation of (G, x): the list has length $n + 1$, where for each $i \in \{1, \dots, n\}$,

$$L_{G,x}[i] = (v_i, x(v_i), N(v_i)),$$

and the final entry is $L_{G,x}[n + 1] = \perp$. Let $m = m(n)$ be the size of $|L_{G,x}[i]|$. Note that $m = \mathrm{poly}(n)$.

Let $\mathcal{L} \in \mathsf{P}$ be a distributed language over configurations represented as adjacency lists (as explained above), and let $M_{\mathcal{L}}$ be a RAM machine deciding membership in $\mathcal{L}$. We assume that $M_{\mathcal{L}}$ never reads past the symbol $\perp$ in its input.

Let HT be a hash family with local openings:[14]

$$\mathsf{HT} = (\mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Open}, \mathsf{HT}.\mathcal{V}),$$

and let SNARG be a $\mathsf{RAM\ SNARG}$ for $\mathcal{L}$ defined with respect to HT:

$$\mathsf{SNARG} = (\mathsf{SNARG.Gen}, \mathsf{SNARG.Hash}, \mathsf{SNARG}.\mathcal{P}, \mathsf{SNARG}.\mathcal{V}).$$

The $\mathsf{LVD\text{-}SNARG}$ for $\mathcal{L}$, denoted $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$, is specified as follows, given an input configuration (G, x) represented as an adjacency list $L_{G,x}$, with $n = |V(G)|$.

$\underline{\mathsf{Gen}(1^\lambda, 1^n)}$**.**

1. Compute $\mathsf{hk} = \mathsf{HT.Gen}(1^\lambda)$.
2. Compute $\mathsf{SNARG.crs} = \mathsf{SNARG.Gen}(1^\lambda, 1^n)$.
3. Output $\mathsf{crs} = (\mathsf{hk}, \mathsf{SNARG.crs})$.

$\underline{\mathcal{P}(\mathsf{crs}, G, x)}$**.**

1. Parse $\mathsf{crs} = (\mathsf{hk}, \mathsf{SNARG.crs})$.
2. Compute the size certificates $\{\sigma_G(v_i)\}_{i=1}^n$.
3. For each $i \in [n + 1]$, compute $\mathsf{h}_i = \mathsf{HT.Hash}(\mathsf{hk}, L_{G,x}[i])$. Let $\mathsf{h} = \{\mathsf{h}_i\}_{i \in [n]}$.
4. Compute $C = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{h})$.
5. For every $i \in [n + 1]$, compute ρ_i by $(\mathsf{h}_i, \rho_i) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{h}, i)$.
6. Compute $\mathsf{SNARG}.\pi = \mathsf{SNARG}.\mathcal{P}(\mathsf{crs}, L_{G,x})$.
7. For every $v_i \in V(G)$, set $\pi(v_i) = (n, C, i, \rho_i, \rho_{n+1}, \mathsf{SNARG}.\pi)$.[15]
8. Output $\pi(v_i)$ at each node $v_i \in V(G)$.

[14] HT is not required to be recursive for this section, but a Merkle tree [Mer89] is still a convenient example.
[15] For convenience of notation, we give ρ_{n+1} to all nodes, but only the node v_n will use it.

$\mathcal{V}(\mathsf{crs}; G; x, \pi)$ **at node** $v \in V(G)$**.**

1. Parse $\mathsf{crs} = (\mathsf{hk}, \mathsf{SNARG.crs})$.
2. Parse $\pi(v) = (n, C, i, \rho_i, \rho_{n+1}, \sigma_v, \mathsf{SNARG}.\pi)$.
3. Verify that $1 \leq i \leq n$.
4. Send $\pi(v)$ to every neighbor $u \in N(v)$, and receive $\{\pi(u)\}_{u \in N(v)}$.
5. Parse all received messages to obtain $\{(n_u, C_u, i_u, \rho_{i_u}, \rho_{n+1}, \sigma_u, \mathsf{SNARG}.\pi_u)\}_{u \in N(v)}$, and verify that for each $u \in N(v)$ we have $n_u = n$, $C_u = C$, and $\mathsf{SNARG}.\pi_u = \mathsf{SNARG}.\pi$.
6. Verify the following:
   (a) *Graph size.* Apply the verification procedure from [KKP05] to the network size n and the certificates σ_v and $\{\sigma_u\}_{u \in N(v)}$.
   (b) *Local view under commitment.* Compute $\mathsf{h}_i = \mathsf{HT.Hash}(\mathsf{hk}, (v, x(v), N(v)))$. Check that $\mathsf{HT.Verify}(\mathsf{hk}, C, i, \mathsf{h}_i, \rho_i) = 1$.
   (c) *Committed list size.* If $i = n$, check that $\mathsf{HT.Verify}(\mathsf{hk}, C, n+1, \perp, \rho_{n+1}) = 1$.
   (d) SNARG. Verify that $\mathsf{SNARG}.\mathcal{V}(\mathsf{SNARG.crs}, C, 1, \mathsf{SNARG}.\pi) = 1$.

We proceed to show that the construction above $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ satisfies the properties of LVD-SNARG per Definition 4.1.

Completeness. Follows immediately from the opening completeness of the HT family, the completeness of the underlying RAM SNARG scheme, and the completeness of the size certification scheme from [KKP05].

Succinctness. The components of the proof π are UIDs and numbers of representation length $O(\log n)$, output of the HT family on input of size $\text{poly}(n)$, and a RAM SNARG proof π. Therefore succinctness follows from the corresponding property of the individual components in our scheme.

Verifier efficiency. The round complexity of the verifier is 1. The local computation time is $\text{poly}(\lambda, |\pi(v)|, |x(v)|, \deg(v))$, as follows from the respective properties of the HT family and the RAM SNARG.

Prover efficiency. The running time of the prover is the combined running time of the prover from [KKP05], of $M_{\mathcal{L}}$, of $\mathsf{HT.Hash}$, of $\mathsf{SNARG}.\mathcal{P}$, and $n \cdot m$ times the running time of $\mathsf{HT.Open}$, which are all $\text{poly}(\lambda, n)$.

Soundness. The soundness proof is more subtle.

Claim 4.5. *The construction* $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ *is sound.*

Proof. Assume towards contradiction that there exists a poly-size prover algorithm $\mathcal{P}^*$, a polynomial $n = n(\lambda)$ and non-negligible function ϵ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} (G, x) \notin \mathcal{L} \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, \pi) = 1 \end{array} \middle| \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ (G, x, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \end{array}\right] \geq \epsilon(\lambda).$$

For each node $v \in V(G)$, parse: $\pi(v) = (n_v, C_v, i_v, \rho_v, \rho_{n+1_v}, \sigma_v, \mathsf{SNARG}.\pi_v)$. The event that all nodes accept, i.e., $\mathcal{V}(\mathsf{crs}; G; x, \pi) = 1$, implies that:

- For every $v \in V(G)$ and neighbor $u \in N(v)$ we have $n_v = n_u$, $C_v = C_u$, and $\mathsf{SNARG}.\pi_v = \mathsf{SNARG}.\pi_u$. Since the graph is connected, this implies that all nodes agree on these values; let $\hat{n}$, $\hat{C}$, and $\hat{\pi}$ denote these common values in the sequel, and let us use the shortened notation $[\hat{n}, \hat{C}, \hat{\pi}]$ to denote the event that all nodes receive these values.
- By the soundness of the scheme from [KKP05], we have $\hat{n} = n = |V(G)|$.
- For every $v \in V(G)$ we have $\mathsf{HT.Verify}(\mathsf{hk}, C, i_v, \mathsf{h}_{i_v}, \rho_v) = 1$.

We therefore have for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} (G,x) \notin \mathcal{L} & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, \pi) = 1 & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \wedge\ [\hat{n}, \hat{C}, \hat{\pi}] & (G, x, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ \hat{n} = n & \\ \wedge\ \forall v \in V : \mathsf{HT.Verify}(\mathsf{hk}, C, i_v, \mathsf{h}_{i_v}, \rho_v) = 1 & \end{array}\right] \geq \epsilon(\lambda).$$

Let $\mathcal{I}$ be the event that the prover gives all nodes distinct indices in the range $\{1, \ldots, n\}$. If all nodes receive (some) distinct indices, and all nodes accept, then event $\mathcal{I}$ occurs, as each node verifies that its index is no greater than the claimed size $\hat{n}$ of the graph, and we have already established that $\hat{n} = n$ if all nodes accept. We now show that if all nodes accept, then with overwhelming probability, all nodes receive distinct indices.

Recall that the nodes have unique identifiers. Therefore, whenever we have $\mathsf{HT.Verify}(\mathsf{hk}, C, i_v, \mathsf{h}_{i_v}, \rho_v) = 1$ at all nodes v, if there exist two nodes $v \neq u \in V(G)$ such that $i_v = i_u$, then there exist two distinct values $(v, x(v), N(v)) \neq (u, x(u), N(v))$, with hash values $\mathsf{h}_v = \mathsf{HT.Hash}(\mathsf{hk}, (v, x(v), N(v)))$, and $\mathsf{h}_u = \mathsf{HT.Hash}(\mathsf{hk}, (u, x(u), N(v)))$ and an index $i = i_u = i_v$ such that $\mathsf{HT.Verify}(\mathsf{hk}, C, i, h_v, \rho_v) = 1$ but also $\mathsf{HT.Verify}(\mathsf{hk}, C, i, h_u, \rho_u) = 1$. So, either $\mathsf{h}_v = \mathsf{h}_u$ even though $(v, x(v), N(v)) \neq (u, x(u), N(v))$, or $\mathsf{HT.Verify}(\mathsf{hk}, C, i, h_v, \rho_v) = \mathsf{HT.Verify}(\mathsf{hk}, C, i, h_u, \rho_u) = 1$ even though $h_v \neq h_u$. Therefore, by the *collision-resistance with respect to opening property* of the HT family, the last equation implies there exists a negligible function $\mu_1(\cdot)$ such that for every two nodes, $v \neq u \in V(G)$, for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} (G,x) \notin \mathcal{L} & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, \pi) = 1 & \\ \wedge\ [\hat{n}, \hat{C}, \hat{\pi}] & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \wedge\ \hat{n} = n & (G, x, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ \forall v \in V : & \\ \quad \mathsf{HT.Verify}(\mathsf{hk}, C, i_v, (v, x(v), N(v)), \rho_i) = 1 & \\ \wedge\ i_v = i_u & \end{array}\right] \leq \mu_1(\lambda),$$

and by a union bound over the nodes, we get,

$$\Pr\left[\begin{array}{l|l} (G,x) \notin \mathcal{L} & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, \pi) = 1 & \\ \wedge\ [\hat{n}, \hat{C}, \hat{\pi}] & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \wedge\ \hat{n} = n & (G, x, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ \forall v \in V : & \\ \quad \mathsf{HT.Verify}(\mathsf{hk}, C, i_v, (v, x(v), N(v)), \rho_i) = 1 & \\ \wedge\ \mathcal{I} & \end{array}\right] \geq \epsilon(\lambda) - n \cdot \mu_1(\lambda).$$

Whenever $\mathcal{I}$ occurs, there is a node whose index is n, and this node checks that

$$\mathsf{HT.Verify}(\mathsf{hk}, C, n+1, \perp, \rho_{n+1}) = 1.$$

Since we assumed that $M_{\mathcal{L}}$ does not read past the symbol $\perp$, the event described in the last equation implies that $\hat{C}$ opens to h, which then at each location i opens to $L_{G,x}[i]$ in all of the locations that $M_{\mathcal{L}}$ reads from. So, by the *extended completeness* property of the RAM SNARG (see Definition 3.11), we have that one can construct a SNARG proof π_0 for the true statement $L_{G,x} \notin \mathcal{L}$, using $\hat{C}$ as the digest. Formally, there exists an algorithm $\mathcal{P}'$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} (G,x) \notin \mathcal{L} & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, \pi) = 1 & \\ \wedge\ [\hat{n}, \hat{C}, \hat{\pi}] & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \wedge\ \hat{n} = n & (G,x,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ \forall v \in V: & \pi_0 \leftarrow \mathcal{P}'(\mathsf{SNARG.crs}, G, x, C) \\ \quad \mathsf{HT.Verify}(\mathsf{hk}, C, i_v, (v, x(v), N(v)), \rho_i) = 1 & \\ \wedge\ \mathcal{I} & \\ \wedge\ \mathsf{SNARG}.\mathcal{V}(\mathsf{SNARG.crs}, C, 0, \pi_0) = 1 & \end{array}\right] \geq \epsilon(\lambda) - \mu_1(\lambda).$$

Finally, the event that all nodes accept implies that at node 1 (in particular, but also at every other node) the SNARG proof $\mathsf{SNARG}.\pi = \hat{\pi}$ is accepted by $\mathsf{SNARG}.\mathcal{V}$ with the digest C and the output 1 (for the statement "$M_{\mathcal{L}}(L_{G,x}) = 1$"). We get:

$$\Pr\left[\begin{array}{l|l} (G,x) \notin \mathcal{L} & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, \pi) = 1 & \\ \wedge\ [\hat{n}, \hat{C}, \hat{\pi}] & \\ \wedge\ \hat{n} = n & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \wedge\ \forall v \in V: & (G,x,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad \mathsf{HT.Verify}(\mathsf{hk}, C, i_v, (v, x(v), N(v)), \rho_i) = 1 & \pi_0 \leftarrow \mathcal{P}'(\mathsf{SNARG.crs}, G, x, C) \\ \wedge\ \mathcal{I} & \\ \wedge\ \mathsf{SNARG}.\mathcal{V}(\mathsf{SNARG.crs}, C, 0, \pi_0) = 1 & \\ \wedge\ \mathsf{SNARG}.\mathcal{V}(\mathsf{SNARG.crs}, C, 1, \hat{\pi}) = 1 & \end{array}\right] \geq \epsilon(\lambda) - \mu_1(\lambda) - \mu_2(\lambda).$$

a contradiction to the soundness of the SNARG, which completes the proof of Claim 4.5. □

This concludes the proof of Theorem 4.3. □

5 Distributed Merkle Trees

In this section we define our notion of a distributed Merkle tree, and show how to construct it from collision-resistant hash functions. A distributed Merkle tree (DMT) is a hash tree that represents a commitment to values held by the nodes of a distributed network: each node v initially holds a collection of values $\{x_{v\to u}\}_{u\in N(v)}$, one value $x_{v\to u}$ for each neighbor of v, and we commit to all the values $\{x_{v\to u}\}_{v\in V(G), u\in N(v)}$. The DMT can be constructed by an efficient distributed algorithm, requiring $O(D)$ synchronized rounds in networks of diameter D, where in each round nodes exchange messages of length $\text{poly}(\lambda, \log n)$. At the end of the execution, each node v holds both the root of the DMT and a succinct opening to each of the values $x_{v\to u}$ that it held originally. DMTs will be used for our construction of LVD-SNARGs with distributed prover in Section 6.

Syntax. An efficient distributed Merkle tree DMT is associated with a recursive hash family with local openings

$$\mathsf{MT} = (\mathsf{MT.Gen}, \mathsf{MT.Hash}, \mathsf{MT.Open}, \mathsf{MT.Verify})$$

and consists of the following algorithms:

$\mathsf{Gen}(1^\lambda) \to \mathsf{hk}$. A randomized algorithm that takes as input the security parameter λ and outputs a hash key $\mathsf{hk} = \mathsf{MT.Gen}(1^\lambda)$.

$\mathsf{DistMake}(\mathsf{hk}; G; x) \to \{(\mathsf{val}_v, \mathsf{rt}_v, I_v, \rho_v, \beta_v)\}_{v \in V(G)}$. A distributed algorithm that executes in a distributed network G, with all nodes receiving the same hash key hk, and each node $v \in V(G)$ initially holding a collection of inputs $x(v) = \{x_{v \to u}\}_{u \in N(v)}$ (one input $x_{u \to v}$ for each neighbor $u \in N(v)$). The output at each node v consists of:

- A hash value val_v, which is the same at all nodes,
- A local MT-root rt_v,[16]
- An index $I_v \in \{0,1\}^*$,
- An opening path ρ_v, and
- A set β_v of tuples $(I_{v \to u}, \rho_{v \to u})$ of index and opening path for every neighbor $u \in N(v)$.

Definition 5.1 (DMT). *A* DMT *is required to satisfy the following properties:*

Well-formedness.

- *All nodes $v \in V(G)$ output the same value* val_v*,*
- *All indices I_v are of length $c \cdot \lceil \log n \rceil$, for some constant c,*
- *All indices $I_{v \to u}$ are of length $\lceil \log \Delta \rceil$, where Δ is the maximum degree in G.*[17]

MT**-functionality.** *Fix a hash key* hk*, a network G of size n and input assignment to it $x : V(G) \to \{0,1\}^*$, where for every $v \in V(G)$, $x(v) = \{x_{v \to u}\}_{u \in N(v)}$, such that for every edge $\{v,u\} \in E(G)$, $x_{v \to u} \in \{0,1\}^\ell$. Let*

$$\left\{(\mathsf{val}_v, \mathsf{rt}_v, I_v, \rho_v, p_v, \hat{F}_v, \beta_v)\right\}_{v \in V(G)} = \mathsf{DistMake}(\mathsf{hk}, G, x),$$

where $\beta_v = \{I_{v \to u}, \rho_{v \to u}\}_{u \in N(v)}$. For each directed edge (v,u), let $\mathsf{Index}(v,u) = I_v || I_{v \to u}$*, and* $\mathsf{Opening}(v,u) = \rho_v || \rho_{v \to u}$*. We say that the* DMT *satisfies* MT-functionality *if for every such output, there exists a constant c and a vector $\vec{x}$ of length at most $\leq 2^{c \cdot \lceil \log n \rceil + \lceil \log \Delta \rceil + \lceil \log \ell \rceil}$ such that:*

- *For every $v \in V(G)$ and $u \in N(v)$ we have $\vec{x}_{\mathsf{Index}(v,u)} = x_{v \to u}$,*
- *For every $v \in V(G)$,* $\mathsf{val}_v = \mathsf{MT.Hash}(\mathsf{hk}, \vec{x})$*,*
- *For every $v \in V(G)$ and $u \in N(v)$ we have: $(\vec{x}_{v \to u}, \mathsf{Opening}(v,u)) = \mathsf{MT.Open}(\mathsf{hk}, \vec{x}, \mathsf{Index}(v,u))$.*

Efficiency. *At each node, the local computation executed by* $\mathsf{DistMake}$ *runs in time* $\mathrm{poly}(\lambda, n, m)$.

[16]Throughout this section and the sequel, we use both val and rt to denote MT-values, which are also themselves MT-roots (the construction is recursive). We use val to denote a "final" value, the root of the entire network, which is later exposed to the algorithm using the DMT; we typically use rt for intermediate values handled inside the distributed Merkle.

[17]This is the indices length assuming that nodes know the maximum degree of the graph. If they do not have access to the maximum degree, they use n instead as an upper bound and the indices $I_{v \to u}$ are of length $\lceil \log n \rceil$, accordingly.

Low round complexity and low communication complexity. $\mathsf{DistMake}$ *runs in $O(D)$ synchronized communication rounds on networks of diameter D, and uses messages of length* $\mathrm{poly}(\lambda, \log n)$.

5.1 Construction from Recursive Hash Families with Local Openings

Theorem 5.2. *For every recursive hash family with local openings, there exists a respective distributed Merkle tree.*

Theorem 5.2, alongside with Theorem 3.7 implies that DMTs exists assuming either the discrete log assumption or the LWE assumption.

The rest of this section is devoted to proving Theorem 5.2. Let MT be a recursive hash family with local openings:

$$\mathsf{MT} = (\mathsf{MT.Gen}, \mathsf{MT.Hash}, \mathsf{MT.Open}, \mathsf{MT.Verify}).$$

As implied by the syntax, the algorithm $\mathsf{DMT.Gen}$ is simply $\mathsf{MT.Gen}$.

In Section 5.1.1 we describe the algorithm $\mathsf{DMT.DistMake}$, and in Section 5.1.2 prove its security.

5.1.1 The Algorithm $\mathsf{DistMake}$

Given input (hk, G, x), with each node v holding $\{x_{v\to u}\}_{u\in N(v)}$, the distributed algorithm $\mathsf{DistMake}$ executes the following stages.

Stage 1: local hash tree. For every $v \in V(G)$, Let $\vec{x}_v$ be the Δ-long array containing the values $\{x_{v\to u}\}_{u\in N(v)}$ held by node v, ordered by the port number of the neighbor $u \in N(v)$ at node v, and padded up to binary representation length $\lceil \log \Delta \rceil$ with $\perp$. (each entry of $\vec{x}_v$ contains an ℓ-size string). For each node v and neighbor $u \in N(v)$, let $I_{v\to u}$ be a binary representation of the port number of u at v, padded up to length $\lceil \log \Delta \rceil$.

Each node v computes

$$\mathsf{rt}_v = \mathsf{MT.Hash}(\mathsf{hk}, \vec{x}_v),$$

as well as the opening

$$\rho_{v\to u} \leftarrow \mathsf{MT.Open}(\mathsf{hk}, \vec{x}_v, I_{v\to u}),$$

for each neighbor $u \in N(v)$. Recall that an MT can be extended naturally to handle arrays of strings instead of bit vectors (see Remark 3.3), and here this is exactly the case.

We let

$$\beta_v \leftarrow \{(I_{v\to u}, \rho_{v\to u})\}_{u\in N(v)}\,.$$

Stage 2: spanning-tree computation. The nodes jointly compute a spanning tree $ST(G)$ of the network (as shown in [Pel00]), storing at every node v the parent $p_v \in N(v)$ of v and the children $C_v \subseteq N(v)$ of v. In the sequel we denote by v_0 the root of the spanning tree.

Stage 3: convergecast of hash-tree forests. In this stage we compute the global hash tree up the spanning tree ST, with each node v merging some or all of the hash-trees received from its children, and sending the result upwards in the form of a set of MT-roots annotated with height information.

Formally, each node v computes and sends upwards a set F_v of pairs (rt, h), where rt is an MT-root, and $h \in \mathbb{N}$.[18]

[18]To simplify the presentation, we assume throughout this section that every value $x_{v\to u}$ is unique, and that there are no collisions in the underlying hash-tree MT; otherwise F_v may be a multiset. Strictly speaking, this should be resolved by assigning every MT-root a unique identifier, and using triplets of the form $(\mathsf{uid}, \mathsf{rt}, h)$ throughout, instead of (rt, h) alone. Since the nodes of the network do have unique identifiers, it is easy to have them assign a uid field for every pair (rt, h) that they generate.

Upon receiving $\{F_c\}_{c\in C_v}$ from its children (or without waiting to receive anything, in the case of leaves, where $C_v = \emptyset$), node v computes a forest S_v, where each node is a pair (rt, h) (as above, rt is an MT-root and $h \in \mathbb{N}$):

1. Initially, $S_v = \{(\mathsf{rt}_v, 0)\} \cup \bigcup_{c\in C_v} F_c$.

2. As long as there remain two distinct tree roots $(\mathsf{rt}_0, h), (\mathsf{rt}_1, h)$ with the same value h in S_v, we create a new root node rt with value $h+1$, and place the two trees under this root. The new root rt is created by invoking $\mathsf{rt} \leftarrow \mathsf{MT.Hash}(\mathsf{hk}, (\mathsf{rt}_0, \mathsf{rt}_1))$. We then add $(\mathsf{rt}, h+1)$ to S_v, with the node (rt_0, h) as its left child and the node (rt_1, h) as its right child.

3. If v is the root of the spanning tree, we proceed as follows: while there remains more than one tree in S_v, let $h = \min\{h : \exists \mathsf{rt}.\,(h, \mathsf{rt}) \in S_v\}$ be the smallest height of any tree in S_v.

    - If there are two trees in S_v that have height h, then we merge them, as above.
    - Otherwise, we create a new "dummy" root rt' by invoking $\mathsf{rt}' \leftarrow \mathsf{MT.Hash}(\mathsf{hk}, \perp)$, and add (rt', h) to S_v as a leaf.

At the end of this process, if node v is not the root, then node v sends upwards the set F_v comprising the roots of the trees in S_v.

When the process completes, at the root v_0 of the spanning tree, S_{v_0} is a tree (the root cannot halt while S_{v_0} contains more than one tree). Let (rt, h) be the root of this tree. Node v_0 sets $\mathsf{val} \leftarrow \mathsf{rt}$, the root of the global hash-tree.

Stage 4: computing hash-tree indices and openings. In this stage we proceed down the spanning tree, forwarding the global root val downwards. In addition, as we move down the tree, each node v computes for each child $c \in C_v$ an annotated root-set,

$$\hat{F}_c = \{(\mathsf{rt}, I, \rho) : \exists h.\,(\mathsf{rt}, h) \in F_c\},$$

where every root rt appearing in F_c is annotated with the index I leading to rt from the global root val, and the corresponding opening path ρ. This is computed as follows: first, the root v_0 of the spanning tree sets $\hat{F}_{v_0} = \{(\mathsf{val}, \varepsilon, \varepsilon)\}$ (where ϵ is the empty string). Then, we proceed down the tree, starting from the root downwards, and each node v, upon receiving $\hat{F}_v$ from its parent (or computing it itself, in the case of the root v_0), computes an annotated version $\hat{S}_v$ of S_v as follows. For each node (rt, h) in S_v, in order of decreasing height:

1. If (rt, h) is a tree root in S_v (i.e., it does not have a parent), then $(\mathsf{rt}, h) \in F_v$, and therefore there is a corresponding annotated node $(\mathsf{rt}, I, \rho) \in \hat{F}_v$. We add (rt, I, ρ) to $\hat{S}_v$.

2. Otherwise, (rt, h) has a parent (rt_p, h_p) in S_v, which has already been processed since we proceed by decreasing height. Let $(\mathsf{rt}_p, I_p, \rho_p)$ be the annotated node corresponding to the parent in $\hat{S}_v$, and let (rt', h) be the sibling of (rt, h) in S_v. The index corresponding to rt is set to $I \leftarrow I_p||d$, where $d = 0$ if (rt, h) is the left child of (rt_p, h_p), and $d = 1$ if (rt, h) is the right child. The opening corresponding to rt is set to $\rho \leftarrow \rho_p||\mathsf{rt}'$. We add (rt, I, ρ) to $\hat{S}_v$ as the child of $(\mathsf{rt}_p, I_p, \rho_p)$ in whichever direction (rt, h) was.

Afterwards, node v extracts for each child $c \in C_v$ the set $\hat{F}_c = \left\{(\mathsf{rt}, I, \rho) \in \hat{S}_v : \exists h.\,(\mathsf{rt}, h) \in F_c\right\}$, and sends it to node c, along with the global root val.

For its own local root, node v extracts $(\mathsf{rt}_v, I_v, \rho_v)$ from $\hat{S}_v$ (i.e., it finds the node in $\hat{S}_v$ where the MT-root is rt_v, and takes the index and opening from that node).

Outputs. The final output at node v is $(\mathsf{val}, \mathsf{rt}_v, I_v, \rho_v, \beta_v)$.

5.1.2 Correctness of the Construction

We proceed to prove that DistMake implements MT-functionality.

In Stage 3, an easy induction on the steps of computing $\{S_v\}_{v \in V(G)}$ shows the following structural lemma:

Lemma 5.3. *For every $v \in V(G)$, the directed graph S_v is a forest, whose leaves are:*

- *The set $\{(\mathsf{rt}_v, 0)\} \cup \bigcup_{c \in C_v} F_c$, if v is not the root of $ST(G)$;*
- *The set above, with the possible addition of nodes of the form $(\mathsf{MT.Hash}(\mathsf{hk}, \perp), h)$ for some $h \geq 0$, if v is the root of $ST(G)$.*

In addition, S_v contains at most one tree of height h for any $h \in \mathbb{N}$.

Now let S be the directed graph obtained by taking the non-disjoint union of all the forests $\{S_v\}_{v \in V(G)}$. Recall that v_0 denotes the root of the spanning tree that the network forms. We state and prove the following lemma.

Lemma 5.4. *S has the following properties:*

1. *For every node $v \in V(G)$, there is exactly one node $(\mathsf{rt}_v, 0)$ in S, and it has in-degree zero.*
2. *The only other nodes with in-degree zero in S are of the form $(\mathsf{MT.Hash}(\mathsf{hk}, \perp), h)$ for some $h \geq 0$, and all such nodes appear only in S_{v_0}, where v_0 is the root of the spanning tree.*
3. *Every (rt, h) in S either has in-degree zero and out-degree one, or it has in-degree two and out-degree zero or one, and in this case its two in-neighbors are of the form $(\mathsf{rt}_0, h-1)$ and $(\mathsf{rt}_1, h-1)$, where $\mathsf{rt} = \mathsf{MT.Hash}(\mathsf{hk}, (\mathsf{rt}_0, \mathsf{rt}_1))$.*
4. *S is a rooted tree.*
5. *For every $v \neq v_0$ and node (rt, h) in S_v, the height of (rt, h) in S is exactly h, and the leaves in the subtree of (rt, h) in S are exactly the leaves $(\mathsf{rt}_u, 0)$ such that u is a descendant of v in $ST(G)$.*
6. *Every $(\mathsf{rt}, h) \in S$ has height $h \leq \lceil \log n \rceil$.*

Proof. Property 1 follows from the fact that each node v adds the leaf $(\mathsf{rt}_v, 0)$ to its own forest S_v, and never removes it, adds in-neighbors to it, or changes its height. Nodes further up the spanning tree never add in-neighbors to MT-nodes they received from their children, only to MT-nodes they create themselves.

Property 2 follows from the fact that the only node that adds leaves to S_v beyond the leafs $\{(\mathsf{rt}_v, 0)\}_{v \in V(G)}$ that nodes create for themselves is the root v_0 of the spanning tree, and those leaves are indeed of the form $(\mathsf{MT.Hash}(\mathsf{hk}, \perp), h)$ for some $h \geq 0$.

Property 3 is shown by an easy induction up the tree and an inner induction on the process of computing S_v, using the fact that new non-leaf nodes are created only by taking two S_v-nodes (rt_0, h) and (rt_1, h), and adding a new node $(\mathsf{MT.Hash}(\mathsf{hk}, (\mathsf{rt}_0, \mathsf{rt}_1)), h+1)$ as their parent in S_v.

Property 4 is implied by the previous properties. First, they imply that S is a directed forest, oriented upwards (in particular, it is acyclic by Property 3, as the heights are increasing on every directed path and every node has out-degree at most 1). To see that S is a single rooted tree, suppose for the sake of contradiction that it is not, and let (rt, h) and (rt', h') be the roots of two distinct trees in S. Let v and v' be the highest nodes in the spanning tree such that $(\mathsf{rt}, h) \in S_v$ and $(\mathsf{rt}', h') \in S_{v'}$. At least one of the nodes v and v' is not the root v_0, as v_0 does not halt while S_{v_0} contains more than one tree. Assume $v \neq v_0$; then, v has a parent $p_v \neq \perp$, and by choice of v, we have $(\mathsf{rt}, h) \notin S_{p_v}$. But this cannot be: node v includes the root (rt, h) in F_v, and thus, p_v adds it to S_{p_v}.

Property 5 is shown once again by an easy induction up the tree and an inner induction on the process of computing S_v, using the fact that nodes other than v_0 do not create new leafs, and only merge existing trees to form a new tree with height greater by one.

To see property 6, note first that at all nodes except the root of the spanning tree, it follows from the previous properties: if $v \neq v_0$, then for any $(\mathsf{rt}, h) \in S_v$, the subtree of (rt, h) in S contains only leafs of the form $(\mathsf{rt}_u, 0)$ where $u \in V(G) \setminus \{v_0\}$ (by Property 5), and there are at most $n-1$ such leaves. We have also noted that the branching factor in S is 2. This implies that $h \leq \lfloor \log(n-1) \rfloor$ for every $(\mathsf{rt}, h) \in S_v$.

Now consider the root of the spanning tree, and let S_0 be the value of S_{v_0} when the root completes Step 2 of the convergecast stage. Since no dummy nodes have been created yet, the same reasoning as above shows that S_0 contains trees of height at most $\log n$, and moreover, each tree in S_0 has a different height (otherwise Step 2 would not be complete). Let $S_1, \ldots, S_k$ be the intermediate values of S_{v_0} after each merge performed in Step 3 of the convergecast stage at the root, such that S_k is the final tree S_{v_0}. If Step 3 was skipped, meaning, $k = 0$ and $S_0 = S_k$, we have that S_0 contains exactly one binary tree of n leaves and height $\log n$ (which is an integer in this case). Otherwise, $S_0 \neq S_k$, and S_0 contains at least two trees, where each tree has at most $n-1$ leaves, and is of height at most $\lfloor \log(n-1) \rfloor$. Let $h^i_{\max}$ and $h^i_{\min}$ denote the maximum and minimum heights of trees in S_i, resp., (where $0 \leq i \leq k$). By induction on i, we claim that for every $0 \leq i < k$,

- $h^i_{\max} = h^0_{\max}$,
- If $i > 0$ then $h^i_{\min} = h^{i-1}_{\min} + 1$,
- S_i contains at most one tree of each height $h \neq h^i_{\min}$, and at most two trees of height $h^i_{\min}$.

The base of the induction is immediate, since we already know that S_0 contains at most one tree of any given height. For the induction step, suppose the claim holds for $i < k-1$, and consider the operation $i+1 < k$. Recall that S_{i+1} is obtained from S_i by choosing a tree of height $h^i_{\min}$, and merging it with either another tree of the same height $h^i_{\min}$ or with a dummy leaf. This eliminates all trees of height $h^i_{\min}$, since by the induction hypothesis there were at most two such trees, and creates a new tree of height $h^i_{\min} + 1$. Therefore, $h^{i+1}_{\min} = h^i_{\min} + 1$, and $h^{i+1}_{\max} = \max(h^i_{\max}, h^i_{\min} + 1)$.

In S_i we had at most one tree of every height except $h^i_{\min}$, and since we did not create any trees of height greater than $h^{i+1}_{\min} = h^i_{\min} + 1$, and we created one tree of height $h^{i+1}_{\min} = h^i_{\min} + 1$, in S_{i+1} we have at most one tree of every height except $h^{i+1}_{\min} = h^i_{\min} + 1$, and at most twos tree of height $h^{i+1}_{\min} = h^i_{\min} + 1$. Finally, assume for the sake of contradiction that $h^{i+1}_{\max} \neq h^0_{\max}$. By the induction hypothesis we know that $h^i_{\max} = h^0_{\max}$, so we have $h^{i+1}_{\max} \neq h^i_{\max}$. Recall also that $h^{i+1}_{\max} = \max(h^i_{\max}, h^i_{\min} + 1)$, so we must have $h^{i+1}_{\max} = h^i_{\min} + 1 > h^i_{\max}$, that is, $h^i_{\min} = h^i_{\max}$; namely, S_i contains only trees of the same height. Since S_i did not have more than two trees of the same height by the induction hypothesis, either S_i was already a single tree, or S_i comprised exactly two same-height trees, which were merged to form a single tree in S_{i+1}. But in both cases we would have ended Step 3 by the $i+1$ operation, which is not the case, since $i+1 < k$ and k is the final step.

To conclude the proof, consider the final tree S_k constructed in the last step, in the case where $S_k \neq S_0$ (and Step 3 is not skipped). In S_{k-1} there must be exactly two trees, and they must have the same height, otherwise, a single merge step would not suffice to form a single tree. By the claim above, $h^{k-1}_{\max} = h^0_{\max} \leq \lfloor \log(n-1) \rfloor$. The single tree in S_k has height $h^{k-1}_{\max} + 1 \leq \lfloor \log(n-1) \rfloor + 1 \leq \lceil \log n \rceil$. $\square$

In the sequel, let $\vec{y}$ be the vector whose length is the number of leaves in S, such that $\vec{y}[I_v] = \mathsf{rt}_v$ for every $v \in V(G)$, and the remaining values are $\perp$. The previous lemma implies that $\mathsf{val} = \mathsf{MT.Hash}(\mathsf{hk}, \vec{y})$. The following lemma is proved by an easy induction down the tree S, following immediately from the computation of the annotations in Stage 4:

Lemma 5.5. *Following Stage 4, for every node v we have:* $\mathsf{MT.Open}(\mathsf{hk}, \vec{y}, I_v) = \rho_v$.

Let $\vec{x}$ be the vector obtained from $\vec{y}$ by replacing each element rt_v where $v \in V(G)$ by the vector $\vec{x}_v$ (defined in Stage 1 above). The following lemma follows immediately from the properties of the underlying MT, together with the definition of the local openings and local indices:

Lemma 5.6. *Following stage 4, for every $v \in V(G)$ and $u \in N(v)$ we have:*

$$\mathsf{MT.Open}(\mathsf{hk}, \vec{x}, I_v || I_{v \to u}) = \rho_v || \rho_{v \to u}.$$

Finally, we note that indeed, $\mathsf{val} = \mathsf{MT.Hash}(\mathsf{hk}, \vec{x})$, as $\mathsf{rt}_v = \mathsf{MT.Hash}(\mathsf{hk}, \vec{x}_v)$ for every $v \in V(G)$ and we already observed that $\mathsf{val} = \mathsf{MT.Hash}(\mathsf{hk}, \vec{y})$. The remaining properties of the DMT are easy to see.

Well-formedness. The value output by all nodes is the value val sent down from the root. The index I_v is a path leading down to rt_v from the global root val, and since the global root is of height $O(\log n)$, the length of I_v is at most $O(\log n)$. Finally, for every neighbor $u \in N(v)$, the relative index $I_{v \to u}$ is simply the port number of u at v, which we assumed comprises $\lceil \log \Delta \rceil$ bits.

Efficiency. The computation at each node involves computing a polynomial number of hashes (since the entire hash-tree is of logarithmic height), as well as simple operations such as finding the minimum-height tree in a forest of polynomial size and concatenating strings of polylogarithmic length. These are all efficient operations.

Round and communication complexity. Let D be the diameter of the network. The algorithm $\mathsf{DistMake}$ runs in $O(D)$ rounds, since it requires the computation of a spanning tree [Pel00], convergecast up the tree, and broadcast down the tree. In each step, nodes send their parent or children at most $O(\log n)$ tree roots, openings and indices, since F_v contains at most one tree of every height at each node v, and the height is at most logarithmic. Each tree root or opening is of length $\mathrm{poly}(\lambda, \log n)$, by the succinctness of the underlying recursive hash; the indices are of polylogarithmic length by the well-formedness property above.

6 Locally Verifiable Distributed SNARGs with a Distributed Prover

In this section, we show how to construct $\mathsf{LVD\text{-}SNARG}$s, where the prover is itself a distributed algorithm. Let $\mathcal{L}$ be a distributed language such that $\mathcal{L} \in \mathsf{P}$. For the case of the distributed prover, instead of relying on the centralized machine that decides $\mathcal{L}$ in polynomial time, we rely on a distributed algorithm $\mathcal{D}$ that decides $\mathcal{L}$ in polynomial communication rounds, while sending polylogarithmic-size messages in each round. Since in $\mathrm{poly}(n)$ rounds, nodes can gather all of the information in one place (even if in each round they send only up to $\mathrm{polylog}(n)$ bits on each edge), the fact that $\mathcal{L} \in \mathsf{P}$ implies that such an algorithm $\mathcal{D}$ exists.

Moreover, as one of our motivations is creating self-proving distributed algorithms, we consider any (polynomial number of rounds, polylogarithmic message size) distributed algorithm $\mathcal{D}$, including the non-decisional case. We then think of the distributed language $\mathcal{L}$ as the set of all input-output tuples (x, y) (where both x and y are functions assigning strings to nodes of the graph), such that $\mathcal{D}(G, x) = y$. We then refer to this algorithm as *computing* the language $\mathcal{L}$.

The syntax for the distributed-prover $\mathsf{LVD\text{-}SNARG}$ remains the same as for the global prover, except for $\mathcal{P}$ being the following distributed algorithm.

$\mathcal{P}(\mathsf{crs}; G; x) \to (y, \pi)$. A distributed algorithm that runs in the network G, where all of the nodes have access to the common reference string crs obtained from Gen, and each node $v \in V(G)$ inputs $x(v)$, and outputs (1) an assignment of outputs $y : V(G) \to \{0,1\}^*$ of $\mathcal{D}$ when executed in G, and (2) an assignment of proofs $\pi : V(G) \to \{0,1\}^*$.

Respectively, we augment the efficiency, locality, and low communication complexity requirements as follows.

Definition 6.1. *A* distributed-prover $\mathsf{LVD\text{-}SNARG}$ *$(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ satisfies the properties from Definition 4.1 (where the completeness is with the distributed prover $\mathcal{P}$ and the soundness holds still for any global poly-size prover $\mathcal{P}^*$), and the following additional properties:*

Prover low rounds and communication complexity. *$\mathcal{P}$ runs in $O(D)$ communication rounds on networks of diameter D, where in each round, each node sends a message of length* $\text{poly}(\lambda, \log n)$ *to each neighbor.*

Prover efficiency. *At each node, the local computation executed by $\mathcal{P}$ runs in time* $\text{poly}(n)$.

We note that since the motivation is to construct algorithms that *verify their own execution*, in what follows we refer instead to any distributed algorithm, not necessarily a decision algorithm. This is equivalent: for any distributed language $\mathcal{L} \in \mathsf{P}$, there is a distributed algorithm that decides $\mathcal{L}$ in $O(n^2)$ synchronized rounds, using messages of length $O(\log n)$, by simply collecting the entire input configuration at one node (which does not need to be chosen in advance) and having that node locally decide membership in $\mathcal{L}$. (This is a simple folklore result in the $\mathsf{CONGEST}$ model of distributed network algorithms.)

In Section 6.1 we describe in more detail our model of the distributed algorithm, in Section 6.2 we present our construction of a distributed-prover $\mathsf{LVD\text{-}SNARG}$, and in Section 6.3 its analysis.

6.1 Modelling the Distributed Algorithm

To construct our $\mathsf{LVD\text{-}SNARG}$, we first model the distributed algorithm more explicitly. Let $\mathcal{D}$ be a distributed algorithm that runs in $R = R(n) = \text{poly}(n)$ rounds in networks of n nodes, sending messages of size $\text{polylog}(n)$, where each such round is divided into three phases:

1. Computing phase: each node performs some computation that takes up to $P = P(n) = \text{poly}(n)$ steps. At the end of this phase, the internal memory contains all of the messages that this node is about to send in the current round.
2. Sending phase: each node sends (possibly empty) messages to each of its neighbors, copied from the internal memory.
3. Receiving phase: each node reads the messages from its neighbors, sent in the previous phase and copies them into its own internal memory.

The last round is divided into only two phases: a computing phase and an output phase, where nodes produce their outputs.

The local computation of the distributed algorithm. Recall that $\mathcal{U}$ stands for the domain of unique IDs (UIDs) of the nodes (see Section 3.1). Let $M_{\mathcal{D}}$ be the Turing machine with the following tapes:

1. $\mathsf{Env} = (v, N)$: a read-only, random access input tape that contains a node UID $v \in \mathcal{U}$ and a neighborhood $N \subseteq \mathcal{U}$ (a list of UIDs).

2. In: a read-only, random access input tape.
3. Read: a read-only, random access input tape, that is divided into $|N|$ blocks.
4. Mem: a read-write, random access memory tape.
5. Write: an output tape that is divided into $|N|$ blocks.

For $n \in \mathbb{N}$, a graph G of n nodes, for every round $r \in [R]$ of $\mathcal{D}$ when executed on $(G; x)$, the machine $M_{\mathcal{D}}$ is such that for every $v \in V(G)$, if:

- Env contains $(v, N(v))$,
- In contains $x(v)$,
- Read contains the messages received from the nodes in $N(v)$ in round $r-1$, and
- Mem contains the internal memory of v at the end of round $r-1$,

then, at the end of $M_{\mathcal{D}}$'s execution:

- Mem contains the internal memory of v at the end of round r, and
- Write contains the messages that v sends to the nodes of $N(v)$ in round r.

Moreover, for the last round $r = R$, at the end of $M_{\mathcal{D}}$'s execution, Write contains the output $y(v)$ (ignoring the division into $|N|$ blocks).

In order to execute its computations (which include either reading or writing to some tape at each computation step), in addition to its tapes, $M_{\mathcal{D}}$ has a state st, which contains an instruction that may include a location to read from or write into in one of the tapes, a short working space, and an index, which will include both the round index (which does not change throughout $M_{\mathcal{D}}$'s execution) and the step index. In each computation step, $M_{\mathcal{D}}$ may read from a location in one of the tapes into the short state-memory, perform an arithmetic operation with that short memory or write into some location in one of the tapes. Let S be the size of st. We assume $S = \mathrm{polylog}(n)$. We define the following functions:

- $R_{\mathcal{D}}(\mathsf{st}) \to (\mathsf{TP}, j)$. The function that on state st, if st indicates to read from location j on tape $\mathsf{TP} \in \{\mathsf{Env}, \mathsf{In}, \mathsf{Mem}, \mathsf{Read}\}$, returns (TP, j). Otherwise, it returns $(\perp, \perp)$.
- $W_{\mathcal{D}}(\mathsf{st}) \to (\mathsf{TP}, j, b)$. The function that on state st, if st indicates to write the bit b in location j on tape $\mathsf{TP} \in \{\mathsf{Mem}, \mathsf{Write}\}$, returns (TP, j, b). Otherwise, it returns $(\perp, \perp, \perp)$.
- $T_{\mathcal{D}}(\mathsf{st}, b) \to (\mathsf{st}_+)$. The function that on state st and bit b, returns the state st_+ that the machine $M_{\mathcal{D}}$ moves to after being in st and, if st indicates reading, reading a bit b from the respective tape.

Let $P = P(n) = \mathsf{time}(M_{\mathcal{D}})$. We assume $P = \mathrm{poly}(n)$.

The communication phases in the distributed algorithm. The machine $M_{\mathcal{D}}$ describes the "next-round function" of each node during the execution of $\mathcal{D}$, but it does not describe the actual mechanism of sending and receiving messages between nodes. To remedy this, we describe here a mechanism that is *equivalent* to the execution of $\mathcal{D}$: we use $M_{\mathcal{D}}$ to describe the computing phase of each round, and we use the UIDs of the nodes to assume an order over the sending and receiving actions. Since in the actual model the sending and receiving happen in every round all at once, the order assumption is *without loss of generality.*

Let $\widetilde{n} = |\mathcal{U}|$ be the size of the UID domain and note that $\widetilde{n} = \mathrm{poly}(n)$. We denote v_j the node with ID j. The set $\{v_1, \ldots, v_{\widetilde{n}}\}$ contains all the nodes in the graph, and possibly some dummy nodes (as not all of the possible UIDs are actually in use).

For every tuple $(k, \ell) \in [\widetilde{n}]^2$, and for every $i \in [P]$, we define the following:

- $\mathsf{comp}(i) = i$.

- $\mathsf{send}(k,\ell) = P + \widetilde{n} \cdot (k-1) + \ell$.
- $\mathsf{recv}(k,\ell) = P + \widetilde{n}^2 + \widetilde{n} \cdot (k-1) + \ell$.

Moreover, we abuse this notation and use comp, send, and recv to also denote the following sets:

- $\mathsf{comp} = [P]$,
- $\mathsf{send} = \left\{P+1, \ldots, P+\widetilde{n}^2\right\}$,
- $\mathsf{recv} = \left\{P+\widetilde{n}^2+1, \ldots P+2\widetilde{n}^2\right\}$.

Let $T = P + 2\widetilde{n}^2$.

As described above, the distributed algorithm $\mathcal{D}$ runs in R rounds where each round is divided into three phases: computing, sending, and receiving. We now think of every round $r \in [R]$ as a sequence of $P + 2\widetilde{n}^2$ steps, denoted $(r,i) \in [R] \times [T]$ where the computing occurs in the steps such that $i \in \mathsf{comp}$, the sending occurs in steps such that $i \in \mathsf{send}$ and the receiving occurs in steps such that $i \in \mathsf{recv}$. The computing steps are simply computation steps of the machine $M_{\mathcal{D}}$. We assume without loss of generality that in $\mathcal{D}$, each node sends a message to each neighbor in every round; if not, it can send $\perp$ instead. For $k, \ell \in [\widetilde{n}]$, for every $r \in R$, if $\{v_k, v_\ell\}$ is an edge in the graph, then in step $(r, \mathsf{send}(k,\ell))$, a message is sent from v_k to v_ℓ, and in step $(r, \mathsf{recv}(k,\ell))$, that message is received by v_ℓ. For all sending steps but the last one in each round, we assume (for all nodes) that none of the tapes change, and in the last sending round, we assume these tapes are emptied, for the next round. In the receiving steps, we assume the Mem and Write tapes of the nodes do not change, but at the end of each step $(r, \mathsf{recv}(k,\ell))$, if $\{v_k, v_\ell\} \in E(G)$, and d_ℓ^k is the port number of v_ℓ in v_k, and d_k^ℓ is the port number of v_k in v_ℓ, we assume that the Read tape of v_ℓ is filled in its $d_k^{\ell\,\text{th}}$ location with what is written in the $d_\ell^{k\,\text{th}}$ location of the Write tape of node v_k.

For $k, \ell \in [\widetilde{n}]$, if $\{v_k, v_\ell\}$ is an edge in the graph, then for every $r \in R$, node v_k writes a message to v_ℓ in step $(r, \mathsf{send}(k,\ell)$, and v_ℓ reads that message in step $(r, \mathsf{recv}(k,\ell))$. If $\{v_k, v_\ell\}$ is not an edge in the graph, then in step $(r, \mathsf{send}(k,\ell))$ and in step $(r, \mathsf{recv}(k,\ell))$ nothing happens.

We assume without loss of generality that in $\mathcal{D}$, each node sends a message to each neighbor in every round; if not, it can send $\perp$ instead. The following describes the content of the tapes $\mathsf{Read}, \mathsf{Mem}, \mathsf{Write}$ during the execution:

- In the first step of the first round, all tapes are empty.
- In each round, for every computing step but the last, either only the Mem tape changes, or only the Write tape changes, or none of them change. In a last computing step of a round, the Read tape is emptied (in any non-last computing step the Read tape does not change).
- For each round, for every sending step but the last, none of the tapes change. In the last sending step of a round, the Write tape is emptied.
- For each round, for every receiving step indexed $i = (k,\ell)$, if $\{v_k, v_\ell\} \in E(G)$, and d_ℓ^k is the port number of v_ℓ in v_k, and d_k^ℓ is the port number of v_k in v_ℓ, then the Read tape of v_ℓ is filled in its $d_k^{\ell\,\text{th}}$ location with what was written in the $d_\ell^{k\,\text{th}}$ location of the Write tape of node v_k in the sending step $i' = (\ell, k)$.

Lastly, we use modular addition over the step numbers, and we may sometimes use, e.g., $(r, T{+}1)$ to denote step $(r+1, 1)$. That is, whenever we have (r,i) such that $i > T$, $(r,i) = (r+1, i-T)$.

6.2 Construction from DMT and seBARGs

We proceed to describe the LVD-SNARG algorithms $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$. In addition to distributed Merkle trees (see Section 5), in our construction, we use somewhere extractable batch arguments for index languages [CJJ21a] of the form

$$\{i \ : \ \exists w \text{ such that } C(i,w) = 1\}\,,$$

where C is a Boolean circuit; see Section 3.4 for further details.

Theorem 6.2. *Assume the existence of a* seBARG *for index languages and* DMT*s. Then, for every graph language* $\mathcal{L} \in \mathsf{P}$ *there exists an* $\mathsf{LVD\text{-}SNARG}$ *with a distributed prover.*

Theorem 6.2, alongside with Theorem 5.2 and Theorem 3.14 imply that $\mathsf{LVD\text{-}SNARG}$s with distributed prover exist assuming either LWE or DLIN or subexponential DDH.

6.2.1 The Generation Algorithm $\mathsf{Gen}(1^\lambda, n)$.

The generation algorithm needs to provide hash keys for one DMT scheme and for two HT schemes, but these can use the same hash key; in addition, common random strings for three separate seBARGs (we use the three separate setups to simplify the analysis). Therefore, on input $(1^\lambda, n)$, the algorithm Gen computes:

1. $\mathsf{hk} = \mathsf{DMT.Gen}(1^\lambda)$.
2. For a parameter $s \in \mathbb{N}$ that is specified below as part of the prover algorithm,
   (a) $(\mathsf{crs}_1, \mathsf{td}_1) \leftarrow \mathsf{seBARG.Gen}(1^\lambda, 2 \cdot R(n) \cdot T(n), 1^s, (1,1))$,
   (b) $(\mathsf{crs}_2, \mathsf{td}_2) \leftarrow \mathsf{seBARG.Gen}(1^\lambda, 2 \cdot R(n) \cdot T(n), 1^s, (1,1))$,
   (c) $(\mathsf{crs}_3, \mathsf{td}_3) \leftarrow \mathsf{seBARG.Gen}(1^\lambda, 2 \cdot R(n) \cdot T(n), 1^s, (1,1))$.

The output is $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_1, \mathsf{crs}_2, \mathsf{crs}_3)$. Note that when using $\mathsf{seBARG.Gen}$, we use the index $(1,1)$ as the hidden binding index arbitrarily, and we do not use the trapdoors td_1, td_2, and td_3.

6.2.2 The Prover Algorithm $\mathcal{P}(\mathsf{crs}; G; x)$.

First, the prover executes the distributed algorithm $\mathcal{D}$, and documents it. Then, the proving process consists of five stages:

1. An internal stage where nodes commit to messages;
2. A distributed stage where nodes compute a DMT;
3. A distributed but local (i.e., one-round) stage where nodes get auxiliary information from their neighbors;
4. An internal stage where nodes commit to message indices;
5. Another internal stage, where nodes construct the seBARG proofs.

Stage 0: Documenting the distributed algorithm. The network jointly executes the distributed algorithm $\mathcal{D}$, where $M_{\mathcal{D}}$ is the "next-round-machine." Each node constructs R tables, such that the r^{th} table contains the contents of the node's tapes and registers in every computation step of the execution of $M_{\mathcal{D}}$ in round r. Overall, these are R tables, each containing P rows of size $\text{poly}(n) + S$.

Stage 1: Internal hash values. For every round r and every step i of the machine $M_{\mathcal{D}}$'s execution in round r, for every node $v \in V(G)$, let $\mathsf{Read}_{r,i}(v)$, $\mathsf{Mem}_{r,i}(v)$, and $\mathsf{Write}_{r,i}(v)$ be the contents of the tapes Read, Mem, and Write of the machine $M_{\mathcal{D}}$ in step i, when executed in node v in round r, as taken from the table formed in the last stage.

Each node v computes:

- $\mathsf{hEnv}(v) = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Env}) = \mathsf{HT.Hash}(\mathsf{hk}, (v, N(v)))$.
- $\mathsf{hIn}(v) = \mathsf{HT.Hash}(\mathsf{hk}, x(v))$.
- $\mathsf{hOut}(v) = \mathsf{HT.Hash}(\mathsf{hk}, y(v))$.

In addition, for every round r and internal computation step i, node v computes:

- $\mathsf{hRead}_{r,i}(v) = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Read}_{r,i}(v))$.
- $\mathsf{hMem}_{r,i}(v) = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Mem}_{r,i}(v))$.
- $\mathsf{hWrite}_{r,i}(v) = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Write}_{r,i}(v))$.

Let $\mathsf{hRead}(v)$, $\mathsf{hMem}(v)$ and $\mathsf{hWrite}(v)$ be the arrays that on location (r, i) contain $\mathsf{hRead}_{r,i}(v)$, $\mathsf{hMem}_{r,i}(v)$ and $\mathsf{hWrite}_{r,i}(v)$ respectively. In addition, each v computes:

- $\mathsf{val}_v^{\mathsf{Read}} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{hRead}(v))$.
- $\mathsf{val}_v^{\mathsf{Mem}} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{hMem}(v))$.
- $\mathsf{val}_v^{\mathsf{Write}} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{hWrite}(v))$.

Let $\mathsf{st}_{r,i}(v)$ be the state of node v in the i^{th} step of the r^{th} execution of $M_{\mathcal{D}}$, and let $\mathsf{st}(v)$ be the array that on location (r, i) contains $\mathsf{st}_{r,i}(v)$. Each node computes:

- $\mathsf{val}_v^{\mathsf{st}} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{st}(v))$

For each edge $(v, u) \in E(G)$, let $\mathsf{msg}_{v,u}$ be the vector that contains, at each index r, the message sent by node v to node u in round r.

Each node v also computes for every $u \in N(v)$:

- $\mathsf{rt}_{v,u} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{msg}_{v,u})$ and $\mathsf{rt}_{u,v} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{msg}_{u,v})$.

Note that $\mathsf{rt}_{v,u}$ is computed by both v and u. Let f_{rt} be the function mapping for each node $v \in V(G)$ the set of hash values for its outgoing messages: $f_{\mathsf{rt}}(v) = \{\mathsf{rt}_{v,u}\}_{u \in N(v)}$.

Stage 2: Constructing the DMT. The network executes $\mathsf{DMT.DistMake}$, using the input $\mathsf{rt}_{v,u}$ for each node $v \in V$ and each neighbor $u \in N(v)$. Formally, the input to $\mathsf{DMT.DistMake}$ is given by:

$$(\mathsf{hk}; G; f_{\mathsf{rt}}(v)) .$$

We note that even though each node holds two DMT hash values for each edge ($\mathsf{rt}_{v,u}$ and $\mathsf{rt}_{u,v}$), only $\mathsf{rt}_{v,u}$ is used as v's input to the DMT. The other value, $\mathsf{rt}_{u,v}$, will be used as part of node u's input. Nevertheless, node v still needs to hold on to $\mathsf{rt}_{u,v}$ for now; it will discard it at the end of the proving stage.

Let the output of $\mathsf{DistMake}$ on node v be:

$$(\mathsf{val}^{\mathsf{msg}}, \mathsf{rt}_v, I_v, \rho_v, \beta_v).$$

Stage 3: Obtaining auxiliary information. Each node v parses $\beta_v = \{(I_{v\to u}, \rho_{v\to u})\}_{u \in N(v)}$, and to each neighbor $u \in N(v)$, sends $I_{v,u} = I_v \parallel I_{v\to u}$ and $\rho_{v,u} = \rho_v \parallel \rho_{v\to u}$, and receives $(I_{u,v}, \rho_{u,v})$. Let $\mathsf{Ind}^{\mathsf{out}}(v)$ and $\mathsf{Ind}^{\mathsf{in}}(v)$ be the vectors such that $\mathsf{Ind}^{\mathsf{out}}(v)[I_{v\to u}] = I_{v,u}$ and $\mathsf{Ind}^{\mathsf{in}}(v)[I_{v\to u}] = I_{u,v}$.

Stage 4: Committing to port numbers and indices of the neighbors. Each node $v \in V(G)$ constructs the vector $\mathsf{Port}(v)$ such that for every $k \in \mathcal{U}$ with $v_k \in N(v)$, it sets $\mathsf{Port}(v)[k] = I_{v \to v_k}$. Further for $z \in \mathcal{U}$ such that $v_z \notin N(v)$, it sets $\mathsf{Port}(v)[z] = \bot$. Each node then computes the following hash values:

- $\mathsf{val}_v^{\mathsf{in}} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Ind}^{\mathsf{in}}(v))$,
- $\mathsf{val}_v^{\mathsf{out}} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Ind}^{\mathsf{out}}(v))$,
- $\mathsf{val}_v^{\mathsf{Port}} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Port}(v))$.

Stage 5: seBARGs. Each node v computes three seBARG proofs, for the following statements, using crs_1, crs_2, and crs_3, for the circuit C_v defined below.

Let $\mathsf{st}_{\mathsf{start}}$ be the initial state of $M_{\mathcal{D}}$. Denote by empty an empty array, and by hempty a hash of an empty array, where the size of the array will be clear from the context (e.g, usually this will be the degree of a certain node). For every $v \in V(G)$, let

$$\mathsf{Inf}_v = \left(v, \mathsf{hEnv}(v), \mathsf{hIn}(v), \mathsf{hOut}(v), \mathsf{val}_v^{\mathsf{st}}, \mathsf{val}_v^{\mathsf{Read}}, \mathsf{val}_v^{\mathsf{Mem}}, \mathsf{val}_v^{\mathsf{Write}}, \mathsf{val}_v^{\mathsf{in}}, \mathsf{val}_v^{\mathsf{out}}, \mathsf{val}_v^{\mathsf{Port}}, \mathsf{val}^{\mathsf{msg}}\right),$$

and let C_v be the circuit that has Inf_v hard-wired and on input $((r,i),w)$ where $(r,i) \in [R] \times [T]$ and

$$w = \begin{pmatrix} \mathsf{st}, \rho^{\mathsf{st}}, \mathsf{st}_+, \rho_+^{\mathsf{st}}, \mathsf{hRead}, \rho^{\mathsf{hRead}}, \mathsf{hRead}_+, \rho_+^{\mathsf{hRead}}, \\ \mathsf{hMem}, \rho^{\mathsf{hMem}}, \mathsf{hMem}_+, \rho_+^{\mathsf{hMem}}, \mathsf{hWrite}, \rho^{\mathsf{hWrite}}, \mathsf{hWrite}_+, \rho_+^{\mathsf{hWrite}}, \\ b, \rho^{\mathsf{TP}}, d, \rho^{\mathsf{Port}}, I, \rho^{\mathsf{ind}} m, \rho^m \end{pmatrix},$$

verifies the following:

1. The following hash openings:
   (a) $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{st}}, (r,i), \mathsf{st}, \rho^{\mathsf{st}}) = 1$.
   (b) $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{st}}, (r,i+1), \mathsf{st}_+ \rho_+^{\mathsf{st}}) = 1$.
   (c) $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{Read}}, (r,i), \mathsf{hRead}, \rho^{\mathsf{hRead}}) = 1$.
   (d) $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{Read}}, (r,i+1), \mathsf{hRead}_+, \rho_+^{\mathsf{hRead}}) = 1$.
   (e) $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{Mem}}, (r,i), \mathsf{hMem}, \rho^{\mathsf{hMem}}) = 1$.
   (f) $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{Mem}}, (r,i+1), \mathsf{hMem}_+, \rho_+^{\mathsf{hMem}}) = 1$.
   (g) $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{Write}}, (r,i), \mathsf{hWrite}, \rho^{\mathsf{hWrite}}) = 1$.
   (h) $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{Write}}, (r,i+1), \mathsf{hWrite}_+, \rho_+^{\mathsf{hWrite}}) = 1$.
2. If $i \in \mathsf{comp}$, then
   (a) If $i = 1$, then $\mathsf{st} = \mathsf{st}_{\mathsf{start}}$.
   (b) If $r = 1$, then $\mathsf{hRead} = \mathsf{hempty}$. If additionally, $i = 1$, then $\mathsf{hWrite} = \mathsf{hMem} = \mathsf{hempty}$.
   (c) If $i < P$, then $\mathsf{hRead}_+ = \mathsf{hRead}$; otherwise (if $i = P$), $\mathsf{hRead}_+ = \mathsf{hempty}$.
   (d) If $R_{\mathcal{D}}(\mathsf{st}) = (\mathsf{TP}, j)$ for some tape $\mathsf{TP} \in \{\mathsf{In}, \mathsf{Mem}, \mathsf{Read}\}$ and a location j, then: $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{hTP}, j, b, \rho^{\mathsf{TP}}) = 1$.
   (e) If $W_{\mathcal{D}}(\mathsf{st}) = (\bot, \bot, \bot)$, then $\mathsf{hMem} = \mathsf{hMem}_+$ and $\mathsf{hWrite} = \mathsf{hWrite}_+$.
   (f) If $W_{\mathcal{D}}(\mathsf{st}) = (\mathsf{Mem}, j, b')$, then:
      i. $\mathsf{HT.WVerify}(\mathsf{hk}, \mathsf{hMem}, \mathsf{hMem}_+, j, b', \rho^{\mathsf{TP}}) = 1$.
      ii. $\mathsf{hWrite}_+ = \mathsf{hWrite}$.
   (g) If $W_{\mathcal{D}}(\mathsf{st}) = (\mathsf{Write}, j, b')$, then:

i. $\mathsf{HT.WVerify}(\mathsf{hk}, \mathsf{hWrite}, \mathsf{hWrite}_+, j, b', \rho^{\mathsf{TP}}) = 1$.

ii. $\mathsf{hMem}_+ = \mathsf{hMem}$.

(h) $T_{\mathcal{D}}(\mathsf{st}, b) = \mathsf{st}_+$

(i) $d = I = m = \rho^{\mathsf{Port}} = \rho^m = \rho^{\mathsf{ind}} = \perp$.

3. If $i \in \mathsf{send}$, then:

(a) $\mathsf{st}_+ = \mathsf{st}$, $\mathsf{hRead}_+ = \mathsf{hRead}$ and $\mathsf{hMem}_+ = \mathsf{hMem}$. Moreover, if $i \leq P + \widetilde{n}^2$, then $\mathsf{hWrite}_+ = \mathsf{hWrite}$.

(b) If $r < R$ and $i = P + \widetilde{n}^2$, then $\mathsf{hWrite}_+ = \mathsf{hempty}$.

(c) If $r = R$, then $\mathsf{hWrite} = \mathsf{hWrite}_+ = \mathsf{hOut}$

(d) $b = \rho^{\mathsf{Read}} = \rho^{\mathsf{Mem}} = \rho^{\mathsf{Write}} = \perp$.

(e) Let $k = \left\lfloor \frac{(i-P)}{\widetilde{n}} \right\rfloor + 1$, and let $\ell = (i - P) \bmod \widetilde{n}$. $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{Port}}, \ell, d, \rho^{\mathsf{Port}}) = 1$. If $v = v_k$ and $d \neq \perp$, then additionally:

i. $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{hWrite}, d, m, \rho^{\mathsf{TP}}) = 1$,

ii. $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{out}}, d, I, \rho^{\mathsf{ind}}) = 1$,

iii. $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}^{\mathsf{msg}}, I||r, m, \rho^m) = 1$,

(f) If $i = P + \widetilde{n}^2$, then $\mathsf{hWrite}_+ = \mathsf{hempty}$.

4. If $i \in \mathsf{recv}$, then:

(a) $\mathsf{st}_+ = \mathsf{st}$, $\mathsf{hMem}_+ = \mathsf{hMem}$, and $\mathsf{hWrite}_+ = \mathsf{hWrite}$.

(b) $b = \rho^{\mathsf{Read}} = \rho^{\mathsf{Mem}} = \rho^{\mathsf{Write}} = \perp$.

(c) Let $k = \left\lfloor \frac{i-P-\widetilde{n}^2}{\widetilde{n}} \right\rfloor + 1$, and $\ell = (i - P - \widetilde{n}^2) \bmod \widetilde{n}$.
$\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{Port}}, k, d, \rho^{\mathsf{Port}}) = 1$. If $v = v_\ell$ and $d \neq \perp$, then additionally:

i. $\mathsf{HT.WVerify}(\mathsf{hk}, \mathsf{hRead}, \mathsf{hRead}_+, d, m, \rho^{\mathsf{TP}}) = 1$.

ii. $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}_v^{\mathsf{in}}, d, I, \rho^{\mathsf{ind}}) = 1$.

iii. $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}^{\mathsf{msg}}, I||r, m, \rho^m) = 1$.

and otherwise, (if $v \neq v_\ell$ or $d = \perp$), $\mathsf{hRead} = \mathsf{hRead}_+$.

For every $(r, i) \in [R] \times [T]$ and every node $v \in V(G)$, the prover computes the following values, that will later be used as part of witnesses to C_v. To avoid lengthy notations, in what follows, the node v is fixed; when describing components of the (r, i) witness of a node $v \in V(G)$ (for example, the component st from above, for that particular witness), we'll denote it with subscript r, i (for example, $\mathsf{st}_{r,i}$), but without something specific for v, (like $\mathsf{st}_{r,i}(v)$).

1. Compute the following values and openings for $j \in \{i, i+1\}$:

(a) $(\mathsf{st}_{r,j}, \rho_{r,j}^{\mathsf{st}}) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{st}(v), (r, j))$.

(b) $(\mathsf{hRead}_{r,j}, \rho_{r,j}^{\mathsf{hRead}}) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{hRead}(v), (r, j))$.

(c) $(\mathsf{hMem}_{r,j}, \rho_{r,j}^{\mathsf{hMem}}) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{hMem}(v), (r, j))$.

(d) $(\mathsf{hWrite}_{r,j}, \rho_{r,j}^{\mathsf{hWrite}}) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{hWrite}(v), (r, j))$.

2. If $i \in \mathsf{comp}$:

(a) Let $(\mathsf{TP}, j) = R_{\mathcal{D}}(\mathsf{st}_{r,i})$.
If $\mathsf{TP} \neq \perp$ and $j \neq \perp$, then compute $\rho_{r,i}^{\mathsf{TP}}$ by $(b_{r,i}, \rho_{r,i}^{\mathsf{TP}}) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{TP}_{r,i}(v), j)$.

(b) Let $(\mathsf{TP}, j, b_{r,i}) = W_{\mathcal{D}}(\mathsf{st}_{r,i})$.
If $\mathsf{TP} \neq \bot$, $j \neq \bot$ and $b_{r,i} \neq \bot$, then compute $\rho^{\mathsf{TP}}_{r,i}$ by $(\mathsf{hTP}_{r,i+1}, \rho^{\mathsf{TP}}_{r,i}) = \mathsf{HT.WOpen}(\mathsf{hk}, \mathsf{TP}_{r,i}(v), j, b_{r,i})$.

(c) $d = I = m = \rho^{\mathsf{Port}} = \rho^{m} = \rho^{\mathsf{ind}} = \bot$.

3. If $i \in \mathsf{send}$:

(a) $b = \rho^{\mathsf{Read}} = \rho^{\mathsf{Mem}} = \rho^{\mathsf{Write}} = \bot$.

(b) Let k and ℓ be such that $i = \mathsf{send}(k, \ell)$. If $v = v_k$, and $v_\ell \in N(v)$, then compute:

i. $(d, \rho^{\mathsf{Port}}) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{Port}_v, \ell)$,

ii. $(I_{r,i}, \rho^{\mathsf{ind}}_{r,i}) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{Ind}^{\mathsf{out}}(v), d)$, and

iii. $(m, \rho^{\mathsf{TP}}) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{Write}_{r,i}(v), d)$.

iv. $m = \mathsf{msg}_{v,v_\ell}[r]$.

v. $\rho^{m}_{r,i} = \rho_{v,v_\ell}$

4. If $i \in \mathsf{recv}$:

(a) $b = \rho^{\mathsf{Read}} = \rho^{\mathsf{Mem}} = \rho^{\mathsf{Write}} = \bot$.

(b) Let k and ℓ be such that $i = \mathsf{recv}(k, \ell)$. If $v = v_\ell$, and $v_k \in N(v)$, then compute:

i. $(d, \rho^{\mathsf{Port}}) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{Port}_v, k)$,

ii. $(I_{r,i}, \rho^{\mathsf{ind}}_{r,i}) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{Ind}^{\mathsf{in}}(v), d)$, and

iii. $(m, \rho^{\mathsf{TP}}) = \mathsf{HT.Open}(\mathsf{hk}, \mathsf{Read}_{r,i}(v), d)$.

iv. $m_{r,i} = \mathsf{msg}_{v_k,v}[r]$.

v. $\rho^{m}_{r,i} = \rho_{v_k,v}$.

The prover then obtains the following (r, i)-witness for every $(r, i) \in [R] \times [T]$,

$$w = \left(\begin{array}{c} \mathsf{st}_{r,i}, \rho^{\mathsf{st}}_{r,i}, \mathsf{st}_{r,i+1}, \rho^{\mathsf{st}}_{r,i+1}, \mathsf{hRead}_{r,i}, \rho^{\mathsf{hRead}}_{r,i}, \mathsf{hRead}_{r,i+1}, \rho^{\mathsf{hRead}}_{r,i+1}, \\ \mathsf{hMem}_{r,i}, \rho^{\mathsf{hMem}}_{r,i}, \mathsf{hMem}_{r,i+1}, \rho^{\mathsf{hMem}}_{r,i+1}, \mathsf{hWrite}_{r,i}, \rho^{\mathsf{hWrite}}_{r,i}, \mathsf{hWrite}_{r,i+1}, \rho^{\mathsf{hWrite}}_{r,i+1}, \\ b_{r,i}, \rho^{\mathsf{TP}}_{r,i}, d_{r,i}, \rho^{\mathsf{Port}}_{r,i}, I_{r,i}, \rho^{\mathsf{ind}}_{r,i}, m_{r,i}, \rho^{m}_{r,i} \end{array} \right),$$

and uses $\mathsf{seBARG}.\mathcal{P}$ to compute the following proofs:

1. $\mathsf{seBARG}.\pi^1_v = \mathsf{seBARG}.\mathcal{P}\left(\mathsf{crs}_1, C, \{w_{r,i}\}_{R[n]\times[T(n)]}\right)$.

2. $\mathsf{seBARG}.\pi^2_v = \mathsf{seBARG}.\mathcal{P}\left(\mathsf{crs}_2, C, \{w_{r,i}\}_{R[n]\times[T(n)]}\right)$.

3. $\mathsf{seBARG}.\pi^3_v = \mathsf{seBARG}.\mathcal{P}\left(\mathsf{crs}_3, C, \{w_{r,i}\}_{R[n]\times[T(n)]}\right)$.

Finally, the prover outputs at each node $v \in V(G)$:

$$\pi(v) = \left(\mathsf{Inf}_v, \mathsf{rt}_v, I_v, \rho_v, \mathsf{seBARG}.\pi^1_v, \mathsf{seBARG}.\pi^2_v, \mathsf{seBARG}.\pi^3_v\right).$$

6.2.3 The Verification Algorithm $\mathcal{V}(\mathsf{crs}, G, x, \pi)$.

The verification algorithm consists of three stages, plus a parsing stage that we refer to as Stage 0. In Stage 1, nodes obtain auxiliary information from their neighbors, echoing Stage 3 of the honest prover; in Stage 2, nodes verify the committed indices (computed in Stage 4 of the honest prover); in Stage 3, the nodes verify the seBARG proofs (computed in Stage 5 of the honest prover).

Stage 0: Parsing. Each node parses $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_1, \mathsf{crs}_2, \mathsf{crs}_3)$ and

$$\pi(v) = \Big(\mathsf{Inf}_v, \mathsf{rt}_v, I_v, \rho_v, \mathsf{seBARG}.\pi_v^1, \mathsf{seBARG}.\pi_v^2, \mathsf{seBARG}.\pi_v^3\Big).$$

where

$$\mathsf{Inf}_v = \Big(v, \mathsf{hEnv}(v), \mathsf{hIn}(v), \mathsf{hOut}(v), \mathsf{val}_v^{\mathsf{st}}, \mathsf{val}_v^{\mathsf{Read}}, \mathsf{val}_v^{\mathsf{Mem}}, \mathsf{val}_v^{\mathsf{Write}}, \mathsf{val}_v^{\mathsf{out}}, \mathsf{val}_v^{\mathsf{in}}, \mathsf{val}^{\mathsf{msg}}(v)\Big).$$

Stage 1: Verifying the DMT hash value and obtaining indices of neighbors. Each node v sends to each neighbor $u \in N(v)$ it's value of $\mathsf{val}^{\mathsf{msg}}(v)$, and the index $I_{v,u} = I_v \parallel I_{v\to u}$, and receives from neighbor u $\mathsf{val}^{\mathsf{msg}}(u)$ and the index $I_{u,v}$, accordingly. If for some $u \in N(v), \mathsf{val}^{\mathsf{msg}}(v) \neq \mathsf{val}^{\mathsf{msg}}(u)$, reject.

Stage 4: Verifying hash values. After computing $\{I_{v,u}\}_{u\in N(v)}$ and obtaining $\{I_{u,v}\}_{u\in N(v)}$ from its neighbors in the previous stage, each node v verifies that:

1. $\mathsf{hEnv}_v = \mathsf{HT.Hash}(\mathsf{hk}, (v, N(v)))$.
2. $\mathsf{hIn}_v = \mathsf{HT.Hash}(\mathsf{hk}, x(v))$.
3. $\mathsf{hOut}_v = \mathsf{HT.Hash}(\mathsf{hk}, y(v))$
4. $\mathsf{val}_v^{\mathsf{in}} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Ind}^{\mathsf{in}}(v))$, where $\mathsf{Ind}^{\mathsf{in}}(v)$ is the vector that has $\mathsf{Ind}^{\mathsf{in}}(v)[I_{v\to u}] = I_{u,v}$ for each $u \in N(v)$.
5. $\mathsf{val}_v^{\mathsf{out}} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Ind}^{\mathsf{out}}(v))$, where $\mathsf{Ind}^{\mathsf{out}}(v)$ is the vector that has $\mathsf{Ind}^{\mathsf{out}}(v)[I_{v\to u}] = I_{v,u}$ for each $u \in N(v)$.
6. $\mathsf{val}^{\mathsf{Port}}(v) = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Port}_v)$, where Port_v is the vector that for every $d \in [\deg(v)]$, if u is the d^{th} neighbor of v ($N(v)[d] = u$), then $\mathsf{Port}_v[I_u] = d$. For every j such that j is not a DMT-index of one of v's neighbors, $\mathsf{Port}_v[j] = \bot$.

If either of the above is not true, node v rejects.

Stage 3: Verifying the seBARGs. Each node v verifies that:

- $\mathsf{seBARG.Verify}(\mathsf{crs}_1, C_v, \mathsf{seBARG}.\pi_v^1) = 1$,
- $\mathsf{seBARG.Verify}(\mathsf{crs}_2, C_v, \mathsf{seBARG}.\pi_v^2) = 1$,
- $\mathsf{seBARG.Verify}(\mathsf{crs}_3, C_v, \mathsf{seBARG}.\pi_v^3) = 1$,

and rejects if any of the above is not true.

6.3 Analysis of the Construction

We now proceed to show that our construction satisfies all of the properties of a distributed-prover $\mathsf{LVD\text{-}SNARG}$.

Completeness. Follows naturally from the MT-functionality property of the DMT, the opening completeness property of the underlying MT family and from the completeness property of the seBARG.

Succinctness. Recall that for each node v,

$$\pi(v) = \Big(\mathsf{Inf}_v, \mathsf{rt}_v, I_v, \rho_v, \mathsf{seBARG}.\pi_v^1, \mathsf{seBARG}.\pi_v^2, \mathsf{seBARG}.\pi_v^3\Big).$$

We go over the components of the proof and see they are all of size at most $\mathrm{poly}(\lambda, \log n)$.

- $\mathsf{Inf}_v = \Big(v, \mathsf{hEnv}(v), \mathsf{hIn}(v), \mathsf{hOut}(v), \mathsf{val}_v^{\mathsf{st}}, \mathsf{val}_v^{\mathsf{Read}}, \mathsf{val}_v^{\mathsf{Mem}}, \mathsf{val}_v^{\mathsf{Write}}, \mathsf{val}_v^{\mathsf{out}}, \mathsf{val}_v^{\mathsf{in}}, \mathsf{val}^{\mathsf{msg}}(v)\Big)$: this is the node's UID, and HT roots of vectors of size $\mathrm{poly}(n)$, and so, they are of size $\mathrm{poly}(\lambda, \log n)$.
- rt_v: also an HT-root of a $\mathrm{poly}(n)$-size vector, so this is of size $\mathrm{poly}(\lambda, \log n)$.
- I_v: by the well-formedness property of the DMT, I_v is of size $O(\log n)$.
- ρ_v: by the MT-functionality property of the DMT, ρ_v is a substring of ρ, where $(b, \rho) = \mathsf{MT}.\mathsf{Open}(\mathsf{hk}, X, I_v || I_{v \to u})$, for a vector X of length at most $2^{O(\log n)} = O(n)$, so, by the succinctness property of the underlying MT, we get that it is of size at most $\mathrm{poly}(\lambda, \log n)$.
- $\mathsf{seBARG}.\pi_v^1$, $\mathsf{seBARG}.\pi_v^2$, and $\mathsf{seBARG}.\pi_v^3$: by the seBARG succinctness property, these are of size $\mathrm{poly}(\lambda, |w|, \log(n))$ where $|w|$ is the size of one witness, which is made of components which are all of size $\mathrm{poly}(\lambda, \log n)$. So, we have that the size of the seBARG proof is $\mathrm{poly}(\lambda, \log n)$.

Verifier locality and low communication. Each node $v \in V(G)$ sends one message to each neighbor $u \in N(v)$ in the verification process, and that message is $I_{v,u}$, which is of size $O(\log n)$ (see succinctness proof).

Verifier efficiency. The round complexity of the verifier is 1. The local computation time is $\mathrm{poly}(\lambda, |\pi(v)|, |x(v)|, \deg(v))$, as follows from the respective properties of the HT family, the DMT and the seBARG.

Prover low rounds and communication complexity. In terms of communication, all the prover does is run $\mathsf{DMT}.\mathsf{DistMake}$, and additionally, each node sends one more message to each of its neighbors, of length $\mathrm{poly}(\lambda, \log n)$. So, this follows from the corresponding property of the DMT.

Prover efficiency. At each node, the local computation of the prover includes executions of HT algorithms for up to $\mathrm{poly}(n)$-size inputs, the local computation required by the algorithm $\mathsf{DMT}.\mathsf{DistMake}$, executions of $\mathsf{seBARG}.\mathcal{P}$, and some more parsing operations of inputs of size $\mathrm{poly}(n)$. So, this follows from the corresponding properties of the HT family, DMT and seBARG scheme.

In the remainder of this section, we prove the soundness of our construction of an LVD-SNARG.

6.4 Proof of Soundness

Suppose for the sake of contradiction that there exists a poly-size adversary $\mathcal{P}^*$, a non-negligible function $\alpha(\cdot)$, and a polynomial $n = n(\lambda)$ such that for every $\lambda \in \mathbb{N}$, [19]

$$\Pr\left[\begin{array}{l} \mathcal{D}(G,x) \neq y \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 \end{array} \middle| \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \end{array}\right] \geq \alpha(\lambda).$$

[19] The actual negation of the soundness requirement says that *for every negligible function* $\epsilon(\cdot)$ *there is some* $\lambda \in \mathbb{N}$ such that the above probability is greater than $\epsilon(\lambda)$. This implies that the above probability, as a function of λ, is non-negligible, which then implies the above statement.

For a configuration (G,x) and a node $v \in V(G)$, let $\widetilde{\mathsf{Env}}(G,v) = (v, N(v))$, $\widetilde{\mathsf{Out}}(G,v) = \mathcal{D}(G,x)(v)$ and for every step $(r,i) \in [R] \times [T]$, let $\widetilde{\mathsf{Write}}_{r,i}(G,v)$, $\widetilde{\mathsf{Mem}}_{r,i}(G,v)$, and $\widetilde{\mathsf{Read}}_{r,i}(G,v)$ be the *actual* content of the tapes Write, Mem, and Read, resp., in the beginning of step (r,i). For every round $r \in [R]$, let

$$W_r(G,x,v) = \left(x(v), \widetilde{\mathsf{Env}}(G,v), \widetilde{\mathsf{Write}}_{r,1}(G,v), \widetilde{\mathsf{Mem}}_{r,1}(G,v), \widetilde{\mathsf{Read}}_{r,1}(G,v)\right).$$

For a non-final round, we denote: $M_{\mathcal{D}}(W_r(G,x,v)) = W_{r+1}(G,x,v)$, and for the final round, we denote: $M_{\mathcal{D}}(W_R(G,x,v)) = \mathcal{D}(G,x)(v)$. Therefore,

$$\mathcal{D}(G,x) \neq y \iff \exists v^* : \ M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*).$$

So, the above statement is equivalent to the following one.

$$\Pr\left[\begin{array}{l} \exists v^* \in V(G): \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 \end{array} \middle| \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \end{array}\right] \geq \alpha(\lambda). \tag{6.1}$$

Notation.

- For an index $i \in [T]$, we denote by $j(i) \in [T]$ the following index:
  - If $i \in \mathsf{comp}$ (that is, $i \leq P$), $j(i) = i$.
  - If $i \in \mathsf{send}$ (that is, $P < i \leq P + \widetilde{n}^2$), $j(i) = P+1$.
  - If $i \in \mathsf{recv}$ (that is, $i > P + \widetilde{n}^2$), let $k, \ell \in [\widetilde{n}]$ be indices such that $i = \mathsf{recv}(k,\ell)$; then, $j(i) = \mathsf{send}(k,\ell)$. Or, in other words, $j(i) = i - \widetilde{n}^2$.
- Throughout the proof, we define and use different versions of the algorithm Gen, in which seBARG.Gen is applied with different binding indices. For steps (r_1,i_1), (r_2,i_2), and (r_3,i_3), let $\mathsf{Gen}_{(r_1,i_1),(r_2,i_2),(r_3,i_3)}$ be the algorithm that on input $(1^\lambda, n)$, outputs $(\mathsf{crs}, \mathsf{td})$, where $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_1, \mathsf{crs}_2, \mathsf{crs}_3)$ and $\mathsf{td} = (\mathsf{td}_1, \mathsf{td}_2, \mathsf{td}_3)$, where:
  - $\mathsf{hk} = \mathsf{HT.Gen}(1^\lambda, n)$.
  - For every $b \in \{1,2,3\}$, $(\mathsf{crs}_b, \mathsf{td}_b) = \mathsf{seBARG.Gen}(1^\lambda, n, (r_b, i_b))$.

  We often refer to specific instantiations of $\mathsf{Gen}_{(r_1,i_1),(r_2,i_2),(r_3,i_3)}$ by different, shorter names, that are defined along the proof.
- Throughout the proof, we sometimes write "for every node $u \in V(G)$, the probability of some event happening for u is as follows" where the graph G is not yet chosen. This is an abuse of notation of the longer following claim: "for every $t \in [n]$, whenever a graph G of size n is chosen, the following holds for the t^{th} node of that graph".

Utilizing the Index Hiding property. Throughout the proof, we use the index hiding property of the seBARG scheme in a somewhat non-direct way; following [CJJ21b, KLVW23], we use a supposedly stronger version of it, stated formally in the following lemma, which is proved in Appendix A to be implied by the original index hiding property.

Lemma 6.3. *Let* $(\mathsf{Gen}, \mathcal{P}, \mathcal{V}, \mathcal{E})$ *be a somewhere extractable batch argument. For every poly-size algorithm* $\mathcal{M}$*, polynomial* k *and a poly-time algorithm* $\mathcal{A}$*, there exists a negligible function* $\text{negl}(\cdot)$ *such that for every pair of indices* $(i_0, i_1) \in [k]^2$*,* $\lambda \in \mathbb{N}$*, we have:*

$$\Pr\left[\ \mathcal{M}(\mathsf{crs}, z) = 1 \ \middle|\ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_1) \\ z \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right]$$
$$\geq \Pr\left[\ \mathcal{M}(\mathsf{crs}, z) = 1 \ \middle|\ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_0) \\ z \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] - \text{negl}(\lambda).$$

In what follows, whenever we refer to the index hiding property, we in fact refer to Lemma 6.3.

Parsing. Here and throughout, we refer to the network G that the adversary outputs. For each $v \in V(G)$, parse

$$\pi(v) = \Big(\mathsf{Inf}_v, \mathsf{rt}_v, I_v, \rho_v, \pi_v^1, \pi_v^2, \pi_v^3\Big),$$

where

$$\mathsf{Inf}_v = \Big(v, \mathsf{hEnv}(v), \mathsf{hIn}(v), \mathsf{hOut}(v), \mathsf{val}_v^{\mathsf{st}}, \mathsf{val}_v^{\mathsf{Read}}, \mathsf{val}_v^{\mathsf{Mem}}, \mathsf{val}_v^{\mathsf{Write}}, \mathsf{val}_v^{\mathsf{in}}, \mathsf{val}_v^{\mathsf{out}}, \mathsf{val}_v^{\mathsf{Port}}, \mathsf{val}^{\mathsf{msg}}(v)\Big).$$

Soundness violation yields contradiction. For each step (r,i), let $\mathsf{Gen}_{r,i}$ be $\mathsf{Gen}_{(r,i),(r,j(i)),(1,1)}$. By the *index hiding* property of the seBARG, Equation (6.1) implies that there exists a negligible function $\delta(\cdot)$ such that for every step $(r,i) \in [R] \times [T]$ and every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \exists v^* \in V(G): \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 \end{array} \middle| \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{r,i}(1^\lambda, n) \\ (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \end{array}\right] \geq \alpha(\lambda) - \delta(\lambda). \tag{6.2}$$

We proceed to prove that the soundness violation contradicts the collision-resistance property of the HT family. We begin by stating the following lemma, which asserts that for any triplet of steps, the probability of extracting witnesses for all nodes corresponding to one of the three steps such that the witness is *rejected* by the circuit C_v at *some* node v is negligible. The lemma is then proved (quite simply) in Section 6.4.1.

Lemma 6.4 (Extraction at all nodes)**.** *There exists a negligible function $\nu(\lambda)$ such that for every three steps $(r_1,i_1),(r_2,i_2),(r_3,i_3) \in [R] \times [T]$, every $b \in \{1,2,3\}$, and every $\lambda \in \mathbb{N}$:*

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 \\ \wedge\ \exists v \in V(G): \\ \quad C_v((r_b, i_b), w^b_{r_b,i_b}(v)) = 0 \end{array} \middle| \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{(r_1,i_1),(r_2,i_2),(r_3,i_3)}(1^\lambda, n) \\ (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \{w^b_{r_1,i_1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_b, \pi_v^b)\}_{v \in V(G)} \end{array}\right] \leq n \cdot \nu(\lambda).$$

We proceed to prove that Equation (6.1) yields contradiction. Equation (6.2) together with Lemma 6.3 (extraction at all nodes) implies that there exists a negligible function $\mu(\cdot)$ such that for every step $(r,i) \in [R] \times [T]$ and every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \exists v^* \in V(G): \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 \\ \wedge\ \forall v \in V(G): \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 \\ \quad \wedge\ C_v((r,j(i)), w^1_{r,j(i)}(v)) = 1 \end{array} \middle| \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{r,i}(1^\lambda, n) \\ (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi_v^1)\}_{v \in V(G)} \\ \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi_v^2)\}_{v \in V(G)} \end{array}\right] \tag{6.3}$$
$$\geq \alpha(\lambda) - \delta(\lambda) - n \cdot 2\mu(\lambda).$$

where we subtract $2 \cdot n \cdot \mu(\lambda)$ instead of simply $n \cdot \mu(\lambda)$ since the equation above uses the Lemma 6.3 (extraction at all nodes) twice.

For each $v \in V(G)$, for each $(r,i) \in [R] \times [T]$, and for each $c \in \{1,2,3\}$, parse

$$w^c_{r,i}(v) = \left(\begin{array}{c} \mathsf{st}^c_{r,i}(v), \rho^{\mathsf{st},c}_{r,i}(v), \mathsf{st}^c_{r,i+1}(v), \rho^{\mathsf{st},c}_{r,i+1}(v), \mathsf{hRead}^c_{r,i}(v), \rho^{\mathsf{hRead},c}_{r,i}(v), \\ \mathsf{hRead}^c_{r,i+1}(v), \rho^{\mathsf{hRead},c}_{r,i+1}(v), \mathsf{hMem}^c_{r,i}(v), \rho^{\mathsf{hMem},c}_{r,i}(v), \mathsf{hMem}^c_{r,i+1}(v), \rho^{\mathsf{hMem},c}_{r,i+1}(v), \\ \mathsf{hWrite}^c_{r,i}(v), \rho^{\mathsf{hWrite},c}_{r,i}(v), \mathsf{hWrite}^c_{r,i+1}(v), \rho^{\mathsf{hWrite},c}_{r,i+1}(v) \\ b^c_{r,i}(v), \rho^{\mathsf{TP},c}_{r,i}(v), d^c_{r,i}(v), \rho^{\mathsf{Port},c}_{r,i}(v), I^c_{r,i}(v), \rho^{\mathsf{ind},c}_{r,i}(v), m^c_{r,i}(v), \rho^{m,c}_{r,i}(v) \end{array}\right),$$

and for every node v and step (r,i), let

$$\left(\widetilde{\mathsf{st}}_{r,i}(v), \widetilde{\mathsf{hRead}}_{r,i}(v), \widetilde{\mathsf{hMem}}_{r,i}(v), \widetilde{\mathsf{hWrite}}_{r,i}(v), \widetilde{b}_{r,i}(v), \widetilde{d}_{r,i}(v), \widetilde{I}_{r,i}(v), \widetilde{m}_{r,i}(v)\right)$$

be the *actual* state, hash of the content of the respective tapes, read bit, port number, message index, and message for node v in step (r,i), when executing $\mathcal{D}$ on the graph G. Also, for $y = \mathcal{D}(G,x)$, these are some of the components of the $(r,i)^{\text{th}}$ witness produced by the honest prover for the proof of $\mathcal{D}(G,x) = y$.

Given a node v, a step (r,i) and a witness $w^c_{r,i}(v)$, let $\textsc{AlmostReal}[v, w^c_{r,i}]$, be the following event:

$$\begin{aligned}
& \mathsf{st}^c_{r,i}(v) = \widetilde{\mathsf{st}}_{r,i}(v) \\
\wedge\quad & \mathsf{hMem}^c_{r,i}(v) = \widetilde{\mathsf{hMem}}_{r,i}(v) \\
\wedge\quad & \mathsf{hWrite}^c_{r,i}(v) = \widetilde{\mathsf{hWrite}}_{r,i}(v) \\
\wedge\quad & \mathsf{hRead}^c_{r,i}(v) = \widetilde{\mathsf{hRead}}_{r,i}(v) \\
\wedge\quad & b^c_{r,i}(v) = \widetilde{b}_{r,i}(v) \\
\wedge\quad & d^c_{r,i}(v) = \widetilde{d}_{r,i}(v) \\
\wedge\quad & I^c_{r,i}(v) = \widetilde{I}_{r,i}(v),
\end{aligned}$$

and let $\textsc{Real}[v, w^c_{r,i}]$ be the event:

$$\textsc{AlmostReal}[v, w^c_{r,i}] \quad \wedge \quad m^c_{r,i}(v) = \widetilde{m}_{r,i}(v).$$

Let $\beta(\cdot) = \alpha(\cdot) - \delta(\cdot) - n \cdot 2\mu(\cdot)$. We proceed by proving by induction that Equation (6.3) also implies that with a noticeable probability, not only the extracted witnesses for (r,i) and $(r,j(i))$ are accepted by the circuit C_v for every node v, but also the events $\textsc{Real}[v, w^1_{r,i}]$ and $\textsc{Real}[v, w^2_{r,j(i)}]$ hold. More formally, we claim there exists a negligible function $\xi(\cdot)$ such that for every (r,i) and every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l}
\exists v^* \in V(G): & \\
\quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\
\wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{r,i}(1^\lambda, n) \\
\wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\
\quad C_v((r,i), w^1_{r,i}(v)) = 1 & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v \in V(G)} \\
\quad \wedge\ \textsc{Real}[v, w^1_{r,i}] & \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v \in V(G)} \\
\quad \wedge\ C_v((r,j(i)), w^2_{r,j(i)}(v)) = 1 & \\
\quad \wedge\ \textsc{Real}[v, w^2_{r,j(i)}] &
\end{array}\right] \tag{6.4}$$
$$\geq \beta(\lambda) - n \cdot ((r-1) \cdot T + i) \cdot \xi(\lambda).$$

Equation (6.4) for the last step, by the definition of C_{v^*}, implies the following event:

$$\begin{aligned}
& y(v^*) \neq \widetilde{\mathsf{Out}}(G, v^*) = \widetilde{\mathsf{Write}}_{R,1}(G, v^*) \\
\wedge & \left(\begin{array}{l} \mathsf{HT.Hash}(\mathsf{hk}, \widetilde{\mathsf{Write}}_{R,1}(G, v^*)) = \widetilde{\mathsf{hWrite}}_{R,1}(v^*) \\ = \mathsf{hWrite}^1_{R,1}(v^*) = \mathsf{hOut}_{v^*} = \mathsf{HT.Hash}(\mathsf{hk}, y(v^*)) \end{array}\right)
\end{aligned}$$

with probability at least

$$\beta(\lambda) - n \cdot R \cdot T \cdot \xi(\lambda) = \alpha(\lambda) - \delta(\lambda) - n \cdot 2\mu(\lambda) - n \cdot R \cdot T \cdot \xi(\lambda)$$

which for a negligible function $\xi(\cdot)$, is non-negligible in λ, in contradiction to the *collision resistance* property of the HT family.[20] It remains to prove that Equation (6.4) does hold for every (r,i) and every $\lambda \in \mathbb{N}$ as stated in the following lemma.

[20] Here we refer to regular collision resistance (as in Definition 3.1, the definition of a CRH), as opposed to collision resistance with respect to opening or to writing. This property is explicit in Definition 3.1, and implicit in Definition 3.2 and Definition 3.4, the definition of an HT family, since it is implied by the combination of opening completeness and collision resistance with respect to opening.

Type of i	*Computing*					*Sending*					*Receiving*					*Computing*			
i	1	⇨2	⇨3	⇨4	⇨5	⇨6	⇨7	⇨8	⇨9	⇨10	⇨11	⇨12	⇨13	⇨14	⇨15	⇨1	⇨2	⇨3	⇨...
j(i)	1	⇩2	⇩3	⇩4	⇩5	⇩6	⇨6	⇨6	⇨6	⇨6	⬆⇨6	⬆⇨7	⬆⇨8	⬆⇨9	⬆⇨10	⇩1	⇩2	⇩3	⇩...
r	1															2			

Figure 6.1: In this table, each step i of round r (here, $r = 1$) is coupled with its respective step $j(i)$. In the induction step, we infer from a statement on $(r, i-1)$ and $(r, j(i-1))$, a statement on (r, i) and $(r, j(i))$. The grey arrows denote the use of Lemma 6.5 (double extraction), the white arrows denote the use of Lemma 6.6 (sliding extraction window), and the blue arrows denote the use of the collision-resistance properties of the HT family. As shown in the table, for computing steps and for the first sending step, we infer from $i-1$ to i using Lemma 6.6 (sliding extraction window), and from i to $j(i)$ using Lemma 6.5 (double extraction). For the rest of the sending steps, we infer from $i-1$ to i using Lemma 6.6 (sliding extraction window), and from $j(i-1)$ to $j(i)$ using Lemma 6.5 (double extraction). The most interesting steps are the receiving steps, where we first infer from $j(i-1)$ to $j(i)$ using Lemma 6.6 (sliding extraction window), and then by using both the weak version of Lemma 6.6 (sliding extraction window) and the collision resistance properties of the HT family, we infer i from $j(i)$ and $i-1$ *together*.

Lemma 6.5. *There exists a negligible function $\xi(\cdot)$ such that Equation (6.4) holds for every $(r, i) \in [R] \times [T]$, for every $\lambda \in \mathbb{N}$.*

We prove Lemma 6.5 by induction over the steps. Recall that for every $r < R$, we have $(r, T+1) = (r+1, T)$, so we prove the base case $(r, i) = (1, 1)$ and then the induction step: from $(r, i-1)$ to (r, i), which in the case of $i - 1 = T$, would be (r, T) to $(r+1, 1)$. We now give an overview of the induction proof.

Induction proof overview. In what follows, we refer to a witness $w^c_{r,i}(v)$ as *almost real* (resp., *real*) if the event $\textsc{AlmostReal}[v, w^c_{r,i}]$ (resp., $\textsc{Real}[v, w^c_{r,i}]$), holds for it. To prove the induction claim, we first notice the relationship between step (r, i) and step $(r, j(i))$, and show the following two lemmas:

- Lemma 6.5 (double extraction), which states that for every step (r, i), the probability of extracting two witnesses for the same step (r, i) from two of the seBARG proofs at all nodes, such that the first witness is real at all nodes and the second witness is not real in at least one node, is negligible.
- Lemma 6.6 (sliding extraction window), which states that for every step (r, i), the probability of extracting two witnesses from two of the seBARG proofs at all nodes, one for step $(r, i-1)$, and the other for step (r, i), such that the first witness is real at all nodes and the second witness is not almost real in at least one node, is negligible. Moreover, for a step (r, i) such that $i \in \mathsf{comp} \cup \mathsf{send}$, the probability of extracting such two witnesses such that the first is real at all nodes and the second is not *real* in at least one node, is also negligible.

The latter couple of lemmas allow us to infer the induction step from $(r, i-1)$ to (r, i), by using different versions of Gen (that is, with different binding indices). See Figure 6.1 for an illustration of the steps and the use of Lemmas 6.6 and 6.7. To move between different versions of Gen, we use the index hiding property of the seBARG scheme, at its "enhanced" formalization presented in Lemma 6.3.

We now overview our inference procedure for computing steps, sending steps, and receiving steps.

- For $i \in \mathsf{comp} \cup \{P+1\}$, we have $j(i) = i$ (see Observation 6.10). On such steps, we can infer Equation (6.4) for (r, i) from the same equation for $(r, i-1)$ by the following hybrid steps:

  1. By the *index hiding* property, we can replace $\mathsf{Gen}_{r,i-1}$ with $\mathsf{Gen}_{(r,i),(r,i-1),(r,j(i-1))}$, and by Lemma 6.3 (extraction at all nodes), extract the respective witnesses, and claim that they are all accepted by the circuit C_v for each node v, with the respective indices. Then, we use Lemma 6.6 (sliding extraction window) (applied on (r, i) and $(r, i-1)$) in its strong version (for the case of $i \in \mathsf{comp} \cup \mathsf{send}$), to claim that the extracted $w^1_{r,i}(v)$ is real.
  2. By the *index hiding* property, we can replace $\mathsf{Gen}_{(r,i),(r,i-1),(r,j(i-1))}$ with $\mathsf{Gen}_{(r,i),(r,j(i)),(r,j(i-1))}$, and by Lemma 6.3 (extraction at all nodes), extract the respective witnesses and claim that they are all accepted by the circuit C_v for each node v, with the respective indices. Then, we use Lemma 6.5 (double extraction) (as $(r, j(i)) = (r, i)$) to claim that the extracted $w^2_{r,j(i)}(v)$ is real.
  3. Finally, we use the *index hiding* property of the seBARG again to replace $\mathsf{Gen}_{(r,i),(r,j(i)),(r,j(i-1))}$ with $\mathsf{Gen}_{r,i}$.

- For $i \in \mathsf{send} \setminus \{P+1\}$, we have $j(i) = j(i-1)$. So, we can go through the same steps as for $i \in \mathsf{comp} \cup \{P+1\}$, where in step 2, we still use Lemma 6.5 (double extraction), but with the justification that $(r, j(i)) = (r, j(i-1))$ instead of $(r, j(i)) = (r, i)$.

- The case of $i \in \mathsf{recv}$ is the most complicated one, as it treats the communication between different nodes. Nevertheless, we observe that $j(i-1)$ either equals to $j(i)$ (in the case where $i = P + \widetilde{n}^2 + 1$, the first receiving step), or equals to $j(i) - 1$ (for $i > P + \widetilde{n}^2 + 1$) (see Observation 6.12). So, the following steps allow us to infer Equation (6.4) for (r, i) from the same equation for $(r, i-1)$:

  1. By the *index hiding* property, we can replace $\mathsf{Gen}_{r,i-1}$ with $\mathsf{Gen}_{(r,j(i)),(r,i-1),(r,j(i-1))}$, and by Lemma 6.3 (extraction at all nodes), extract the respective witnesses, and claim that they are all accepted by the circuit C_v for each node v, with the respective indices. We next divide into cases according to i, to claim that the extracted $w^1_{r,j(i)}(v)$ is real:

     – For the case of $i = P + \widetilde{n}^2 + 1$, we use Lemma 6.5 (double extraction) (as $(r, j(i)) = (r, j(i-1))$).

     – For the case of $i > P + \widetilde{n}^2 + 1$, we use the strong version of Lemma 6.6 (sliding extraction window) (applied on $(r, j(i))$ and $(r, j(i-1))$, as $j(i) - 1 = j(i-1)$). Note that we can still use the strong version of Lemma 6.6 (sliding extraction window) here as $j(i-1), j(i) \in \mathsf{comp} \cup \mathsf{send}$, even though $i \in \mathsf{recv}$.

  2. By the *index hiding* property, we can replace $\mathsf{Gen}_{(r,j(i)),(r,i-1),(r,j(i-1))}$ with $\mathsf{Gen}_{(r,i),(r,j(i)),(r,i-1)}$, and by Lemma 6.3 (extraction at all nodes), extract the respective witnesses and claim that they are all accepted by the circuit C_v for each node v, with the respective indices. Then, we use the weak version of Lemma 6.6 (sliding extraction window) for (r, i) and $(r, i-1)$ to claim that the extracted $w^1_{r,i}(v)$ is almost real for every $v \in V(G)$. In order to claim it is not only almost real but actually *real*, we let $k, \ell \in [\widetilde{n}]$ be such that $i = \mathsf{recv}(k, \ell)$, and observe that the interesting case (where $\widetilde{m}_{r,i}$ is not $\perp$) is when $u = v_\ell$ and $v_k \in N(v_\ell)$. In that case, $j(i) = \mathsf{recv}(k, \ell)$, and the event that $w^1_{r,j(i)}(v_k)$ is real and $w^2_{r,i}$ is almost real, implies that the indices $I^1_{r,i}(v_\ell)$ and $I^2_{r,j(i)}(v_k)$ documented in the respective witnesses equal the real ones, and as for the real indices, since $i = \mathsf{recv}(k, \ell)$ and $j(i) = \mathsf{send}(k, \ell)$, we have that $I^1_{r,j(i)}(v_k) = I^2_{r,i}(v_\ell)$. So, we can use the collision resistance with respect

to opening property of the HT family to get $m^1_{r,i}(v_k) = m^2_{r,j(i)}(v_\ell)$, with probability close to that of the latter event. Since $w^2_{r,j(i)}(v_k)$ is real, this implies (with the same probability) that $m^1_{r,i}(v_\ell) = \widetilde{m}_{r,i}(v_\ell)$, which implies $w^2_{r,i}$ is also real.

3. Finally, we use the index hiding property of the seBARG again to replace $\mathsf{Gen}_{(r,i),(r,j(i)),(r,j(i-1))}$ with $\mathsf{Gen}_{r,i}$.

Before proceeding to the induction proof, we state formally Lemmas 6.6 and 6.7, mentioned in the proof overview. Their proofs appear after the proof of Lemma 6.5, in Section 6.4.1.

Lemma 6.6 (Double extraction). *For every $(r,i) \in [R] \times [T]$, let $\mathsf{Gen}^{\mathsf{double}}_{r,i,c_1,c_2}(1^\lambda, n) \to (\mathsf{crs}, \mathsf{td})$ where $c_1 \neq c_2 \in \{1,2,3\}$ be a function $\mathsf{Gen}_{(r_1,i_1),(r_2,i_2),(r_3,i_3)}$ where $r_{c_1} = r_{c_2} = r$, $i_{c_1} = i_{c_2} = i$, and $(r_{c_3}, i_{c_3}) \in [R] \times [T]$.*

Then, there exists a negligible function $\nu(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\begin{array}{l|l} \forall v \in V(G): & \\ \quad C_v((r,i), w^{c_1}_{r,i}(v)) = 1 & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{double}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \quad \wedge \textsc{Real}[v, w^{c_1}_{r,i}] & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad \wedge C_v((r,i), w^{c_2}_{r,i}(v)) = 1 & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v\in V(G)} \\ \wedge \exists u \in V(G): & \{w^{c_2}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v\in V(G)} \\ \quad \neg\textsc{Real}[v, w^{c_2}_{r,i}] & \end{array}\right] \leq n \cdot \nu(\lambda).$$

Lemma 6.7 (Sliding extraction window). *For every $(r,i) \in ([R] \times [T]) \setminus \{(1,1)\}$, let $\mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \to (\mathsf{crs}, \mathsf{td})$ where $c_1 \neq c_2 \in \{1,2,3\}$ be a function $\mathsf{Gen}_{(r_1,i_1),(r_2,i_2),(r_3,i_3)}$ where $r_{c_1} = r_{c_2} = r$, $i_{c_1} = i-1$, $i_{c_2} = i$, and $(r_{c_3}, i_{c_3}) \in [R] \times [T]$.*

Then, there exists a negligible function $\nu(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\begin{array}{l|l} \forall v \in V(G): & \\ \quad \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & \\ \quad C_v((r,i-1), w^{c_1}_{r,i-1}(v)) = 1 & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \quad \wedge \textsc{Real}[v, w^{c_1}_{r,i-1}] & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad \wedge C_v((r,i), w^{c_2}_{r,i}(v)) = 1 & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v\in V(G)} \\ \wedge \exists u \in V(G): & \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v\in V(G)} \\ \quad \neg\textsc{AlmostReal}[u, w^{c_2}_{r,i}] & \end{array}\right] \leq n \cdot \nu(\lambda).$$

Moreover, if $i \in \mathsf{comp} \cup \mathsf{send}$, then there exists a negligible function $\nu_2(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\begin{array}{l|l} \forall v \in V(G): & \\ \quad \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & \\ \quad C_v((r,i-1), w^{c_1}_{r,i-1}(v)) = 1 & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \quad \wedge \textsc{Real}[v, w^{c_1}_{r,i}] & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad \wedge C_v((r,i), w^{c_2}_{r,i}(v)) = 1 & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v\in V(G)} \\ \wedge \exists u \in V(G): & \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v\in V(G)} \\ \quad \neg\textsc{Real}[u, w^{c_2}_{r,i}] & \end{array}\right] \leq n \cdot \nu(\lambda).$$

Base Case.

Claim 6.8. *Equation (6.4) holds for $r = i = 1$.*

Proof. For every node $v \in V(G)$, by the definition of C_v, for every $c \in \{1,2,3\}$, we have that $C_v((1,1), w^c_{1,1}(v)) = 1$ implies: $\mathsf{st}^c_{1,1}(v) = \mathsf{st}_{\mathsf{start}} = \widetilde{\mathsf{st}}_{1,1}(v)$, and $\mathsf{hRead}^c_{1,1}(v) = \mathsf{hMem}^c_{1,1}(v) = \mathsf{hWrite}^c_{1,1}(v) = \mathsf{hempty}$, which in turn implies $\mathsf{hRead}^c_{1,1}(v) = \widetilde{\mathsf{hRead}}_{1,1}(v)$,

$\mathsf{hMem}^c_{1,1}(v) = \widetilde{\mathsf{hMem}}_{1,1}(v)$, and $\mathsf{hWrite}^c_{1,1} = \widetilde{\mathsf{hWrite}}_{1,1}(v)$. Moreover, since $1 \in \mathsf{comp}$, we also have $\widetilde{d}_{1,1}(v) = \widetilde{I}_{1,1}(v) = \widetilde{m}_{1,1}(v) = \bot$, so $C_v((1,1), w^c_{1,1}(v)) = 1$ also implies $d^c_{1,1} = \widetilde{d}_{1,1}(v)$, $I^c_{1,1} = \widetilde{I}_{1,1}(v)$, $m^c_{1,1} = \widetilde{m}_{1,1}(v)$. Additionally, if $R_{\mathcal{D}}(\mathsf{st}_{\mathsf{start}}) = (\bot,\bot)$ then $\widetilde{b}_{1,1}(v) = \bot$ and $C_v((1,1), w^c_{1,1}(v)) = 1$ also implies $b^c_{1,1} = \widetilde{b}_{1,1}(v)$. However, if $R_{\mathcal{D}}(\mathsf{st}_{\mathsf{start}}) = (\mathsf{TP}, j)$ for $\mathsf{TP} \in \{\mathsf{Env}, \mathsf{In}, \mathsf{Mem}, \mathsf{Read}\}$, then $C_v((1,1), w^c_{1,1}(v)) = 1$ implies $\mathsf{HT}.\mathsf{Verify}(\mathsf{hk}, \mathsf{hTP}^c_{1,1}, j, b^c_{1,1}, \rho^{c,\mathsf{TP}}_{1,1}) = 1$, which then implies by the *opening completeness* and *collision resistance with respect to opening* properties of the HT family, that there exists a negligible function $\xi_1(\cdot)$ such that for every node $u \in V(G)$ and every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v \in V(G)} \\ \quad \wedge\ C_v((r,j(i)), w^2_{r,j(i)}(v)) = 1 & \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v \in V(G)} \\ \wedge \left(\begin{array}{l} \quad \neg\textsc{Real}[v, w^1_{r,i}] \\ \vee\ \neg\textsc{Real}[v, w^2_{r,j(i)}] \end{array}\right) & \end{array}\right] \leq \xi_1(\lambda).$$

So, by a union bound over the nodes, the above equation implies

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & \\ \wedge\ \forall v \in V(G): & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{r,i}(1^\lambda, n) \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad \wedge\ C_v((r,j(i)), w^2_{r,j(i)}(v)) = 1 & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v \in V(G)} \\ \wedge\ \exists u \in V(G): & \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v \in V(G)} \\ \quad \left(\begin{array}{l} \quad \neg\textsc{Real}[v, w^1_{r,i}] \\ \vee\ \neg\textsc{Real}[v, w^2_{r,j(i)}] \end{array}\right) & \end{array}\right] \leq n \cdot \xi_1(\lambda).$$

which implies Equation (6.4) for $r = i = 1$ for any function $\xi(\cdot) \geq \xi_1(\cdot)$. □

(r,i)-induction step. By the induction hypothesis, we have:

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{r,i-1}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,i-1), w^1_{r,i-1}(v)) = 1 & \{w^1_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v \in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^1_{r,i-1}] & \{w^2_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v \in V(G)} \\ \quad \wedge\ C_v((r,j(i-1)), w^2_{r,j(i-1)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,j(i-1)}] & \end{array}\right]$$
$$\geq \beta(\lambda) - ((r-1)T + (i-1))\xi(\lambda). \tag{6.5}$$

Let $\gamma(\cdot) = \beta(\lambda) - ((r-1)T + (i-1))\xi(\lambda)$. We continue to prove the induction step by dividing into cases: the case where $i \in \mathsf{comp}$ or $i \in \mathsf{send}$, and the case of $i \in \mathsf{recv}$.

Computing and sending steps.

Claim 6.9. *For every $r \in [R]$, $i \in \mathsf{comp} \cup \mathsf{send}$, if Equation (6.4) holds for $(r, i-1)$ then it also holds for (r,i).*

Proof. Denote by $\mathsf{Gen}'_{r,i}$ the function $\mathsf{Gen}_{(r,i),(r,i-1),(r,j(i-1))}$. By the *index hiding* property of the seBARG and by the symmetry and independence between crs_1, crs_2, and crs_3, Equation (6.5) implies that there exists a negligible function $\zeta_1(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}'_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,i-1), w^2_{r,i-1}(v)) = 1 & \{w^2_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,i-1}] & \{w^3_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r,j(i-1)), w^3_{r,j(i-1)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,j(i-1)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \zeta_1(\lambda).$$

By Lemma 6.3 (extraction at all nodes), the above equation implies there exists a negligible function $\zeta_2(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}'_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r,i-1), w^2_{r,i-1}(v)) = 1 & \{w^2_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,i-1}] & \{w^3_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r,j(i-1)), w^3_{r,j(i-1)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,j(i-1)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \zeta_1(\lambda) - n \cdot \zeta_2(\lambda).$$

By Lemma 6.6 (sliding extraction window) and by the definition of $\mathsf{Gen}'_{r,i}$, since $(r,i) \in \mathsf{comp} \cup \mathsf{send}$, the above equation implies there exists a negligible function $\zeta_3(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & \\ \wedge\ \forall v \in V(G): & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}'_{r,i}(1^\lambda, n) \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad \wedge\ \textsc{Real}[v, w^1_{r,i}] & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r,i-1), w^2_{r,i-1}(v)) = 1 & \{w^2_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,i-1}] & \{w^3_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r,j(i-1)), w^3_{r,j(i-1)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,j(i-1)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \zeta_1(\lambda) - n \cdot (\zeta_2(\lambda) + \zeta_3(\lambda)).$$

Denote by $\mathsf{Gen}''_{r,i}$ the function $\mathsf{Gen}_{(r,i),(r,j(i)),(r,j(i-1))}$. By *index hiding* property of the seBARG, the above equation implies that there exists a negligible function $\zeta_4(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}''_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^1_{r,i}] & \{w^3_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r, j(i-1)), w^3_{r,j(i-1)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,j(i-1)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \zeta_1(\lambda) - n\cdot(\zeta_2(\lambda) + \zeta_3(\lambda)) - \zeta_4(\lambda).$$

By Lemma 6.3 (extraction at all nodes), the above equation implies there exists a negligible function $\zeta_5(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}''_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^1_{r,i}] & \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r, j(i)), w^2_{r,j(i)}(v)) = 1 & \{w^3_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r, j(i-1)), w^3_{r,j(i-1)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,j(i-1)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \zeta_1(\lambda) - n\cdot(\zeta_2(\lambda) + \zeta_3(\lambda) + \zeta_5(\lambda)) - \zeta_4(\lambda). \tag{6.6}$$

Observation 6.10. For every $i \in \mathsf{comp}$, we have $j(i) = i$, and for every $i \in \mathsf{send}$, we have $j(i) = P+1$. This means that for every $i \in \mathsf{comp} \cup \{P+1\}$, we have $j(i) = i$, and for every $i \in \mathsf{send} \setminus \{P+1\}$, we have $j(i) = j(i-1)$.

By Observation 6.10, together with Lemma 6.5 (double extraction), either for $c_1 = 1$ and $c_2 = 2$ or for $c_1 = 3$ and $c_2 = 2$, Equation (6.6) implies there exists a negligible function $\zeta_6(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & \\ \wedge\ \forall v \in V(G): & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}''_{r,i}(1^\lambda, n) \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad \wedge\ \textsc{Real}[v, w^1_{r,i}] & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r, j(i)), w^2_{r,j(i)}(v)) = 1 & \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,j(i)}] & \{w^3_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r, j(i-1)), w^3_{r,j(i-1)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,j(i-1)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \zeta_1(\lambda) - n\cdot(\zeta_2(\lambda) + \zeta_3(\lambda) + \zeta_5(\lambda) + \zeta_6(\lambda)) - \zeta_4(\lambda).$$

By the *index hiding* property of the seBARG, the above equation implies that there exists a negligible function $\zeta_7(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v \in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^1_{r,i}] & \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v \in V(G)} \\ \quad \wedge\ C_v((r,j(i)), w^2_{r,j(i)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,j(i)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \zeta_1(\lambda) - n \cdot (\zeta_2(\lambda) + \zeta_3(\lambda) + \zeta_5(\lambda) + \zeta_6(\lambda)) - \zeta_4(\lambda) - \zeta_7(\lambda).$$

This concludes the proof for the computing and sending steps for any $\xi(\cdot)$ that satisfies

$$n \cdot \xi(\cdot) \geq \zeta_1(\cdot) + n \cdot \zeta_2(\cdot) + n \cdot \zeta_3(\cdot) + \zeta_4(\cdot) + n \cdot \zeta_5(\cdot) + n \cdot \zeta_6(\cdot) + \zeta_7(\cdot). \qquad \square$$

Receiving steps.

Claim 6.11. *For every $r \in [R]$, $i \in \mathsf{recv}$, if Equation (6.4) holds for $(r, i-1)$ then it also holds for (r, i).*

Proof. Denote by $\mathsf{Gen}^*_{r,i}$ the function $\mathsf{Gen}_{(r,j(i)),(r,i-1),(r,j(i-1))}$. By the *index hiding* property of the seBARG, and symmetry and independence between crs_1, crs_2, and crs_3, there exists a negligible function $\eta_1(\cdot)$ such that for every $\lambda \in \mathbb{N}$, the induction hypothesis (Equation (6.5)) implies that the following holds:

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^*_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,i-1), w^2_{r,i-1}(v)) = 1 & \{w^2_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v \in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,i-1}] & \{w^3_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v \in V(G)} \\ \quad \wedge\ C_v((r,j(i-1)), w^3_{r,j(i-1)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,j(i-1)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \eta_1(\lambda).$$

By Lemma 6.3 (extraction at all nodes), the above implies that there exists a negligible function $\eta_2(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^*_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,j(i)), w^1_{r,j(i)}(v)) = 1 & \{w^1_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v \in V(G)} \\ \quad C_v((r,i-1), w^2_{r,i-1}(v)) = 1 & \{w^2_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v \in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,i-1}] & \{w^3_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v \in V(G)} \\ \quad \wedge\ C_v((r,j(i-1)), w^3_{r,j(i-1)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,j(i-1)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \eta_1(\lambda) - n \cdot \eta_2(\lambda). \tag{6.7}$$

Observation 6.12. Since this is a receiving step, let $k, \ell \in [\widetilde{n}]$ be such that $i = \mathsf{recv}(k, \ell)$.

- If $i = P + \widetilde{n}^2 + 1 = \mathsf{recv}(1,1)$, then $j(i) = \mathsf{send}(1,1) = P + 1$. Moreover, $i - 1 = P + \widetilde{n}^2 = \mathsf{send}(\widetilde{n}, \widetilde{n})$, so, $j(i-1) = \mathsf{send}(1,1)$ and in particular $j(i) = j(i-1)$.
- Otherwise, we have that $i - 1 \geq P + \widetilde{n}^2 + 1$ and $(r, i-1)$ is also a receiving step which means that $j(i) - 1 = j(i-1)$:
  - If $\ell > 1$ then $i - 1 = \mathsf{recv}(k, \ell - 1)$ and
  $$j(i) - 1 = \mathsf{send}(k,\ell) - 1 = \widetilde{n} \cdot (k-1) + \ell - 1 = \mathsf{send}(k, \ell-1) = j(i-1).$$
  - If $\ell = 1$ then $i - 1 = \mathsf{recv}(k-1, \widetilde{n})$ and
  $$j(i) - 1 = \mathsf{send}(k,1) - 1 = \widetilde{n} \cdot (k-1) + 1 - 1 = \widetilde{n} \cdot (k-2) + \widetilde{n} = \mathsf{send}(k-1, \widetilde{n}) = j(i-1).$$

For $i = P + \widetilde{n}^2 + 1$, by Observation 6.12 and Lemma 6.5 (double extraction), Equation (6.7) implies that there exists a negligible function $\nu_1(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^*_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r, j(i)), w^1_{r,j(i)}(v)) = 1 & \{w^1_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v \in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^1_{r,j(i)}] & \{w^2_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v \in V(G)} \\ \quad \wedge\ C_v((r, i-1), w^2_{r,i-1}(v)) = 1 & \{w^3_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v \in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,i-1}] & \\ \quad \wedge\ C_v((r, j(i-1)), w^3_{r,j(i-1)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,j(i-1)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \eta_1(\lambda) - n \cdot \eta_2(\lambda) - n \cdot \nu_1(\lambda).$$

For $i > P + \widetilde{n}^2 + 1$, by Observation 6.12 and Lemma 6.6 (sliding extraction window), since $j(i) - 1 = j(i-1)$, and $j(i) \in \mathsf{recv}$, Equation (6.7) implies there exists a negligible function $\nu_2(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^*_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r, j(i)), w^1_{r,j(i)}(v)) = 1 & \{w^1_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v \in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^1_{r,j(i)}] & \{w^2_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v \in V(G)} \\ \quad \wedge\ C_v((r, i-1), w^2_{r,i-1}(v)) = 1 & \{w^3_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v \in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,i-1}] & \\ \quad \wedge\ C_v((r, j(i-1)), w^3_{r,j(i-1)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,j(i-1)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \eta_1(\lambda) - n \cdot \eta_2(\lambda) - n \cdot \nu_2(\lambda).$$

Let $\eta_3(\cdot) = \max(\nu_1(\cdot), \nu_2(\cdot))$. So, the last couple of equations imply $\eta_3(\cdot)$ is a negligible function such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \begin{array}{l} \exists v^* \in V(G): \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) \\ \land\ \mathcal{V}(\mathsf{crs};G;x,y,\pi)=1 \\ \land\ \forall v \in V(G): \\ \quad C_v((r,j(i)),w^1_{r,j(i)}(v))=1 \\ \quad \land\ \textsc{Real}[v,w^1_{r,j(i)}] \\ \quad \land\ C_v((r,i-1),w^2_{r,i-1}(v))=1 \\ \quad \land\ \textsc{Real}[v,w^2_{r,i-1}] \\ \quad \land\ C_v((r,j(i-1)),w^3_{r,j(i-1)}(v))=1 \\ \quad \land\ \textsc{Real}[v,w^3_{r,j(i-1)}] \end{array} & \begin{array}{l} (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^*_{r,i}(1^\lambda,n) \\ (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \{w^1_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1,\pi^1_v)\}_{v\in V(G)} \\ \{w^2_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2,\pi^2_v)\}_{v\in V(G)} \\ \{w^3_{r,j(i-1)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3,\pi^3_v)\}_{v\in V(G)} \end{array} \end{array}\right]$$
$$\geq \gamma(\cdot) - \eta_1(\lambda) - n\cdot(\eta_2(\lambda) + \eta_3(\lambda)).$$

Denote by $\mathsf{Gen}^{**}_{r,i}$ the function $\mathsf{Gen}_{(r,i),(r,j(i)),(r,i-1)}$. By the *index hiding* property of the seBARG and symmetry and independence between crs_1, crs_2, and crs_3, the above equation implies that there exists a negligible function $\eta_4(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \begin{array}{l} \exists v^* \in V(G): \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) \\ \land\ \mathcal{V}(\mathsf{crs};G;x,y,\pi)=1 \\ \land\ \forall v \in V(G): \\ \quad C_v((r,j(i)),w^2_{r,j(i)}(v))=1 \\ \quad \land\ \textsc{Real}[v,w^2_{r,j(i)}] \\ \quad \land\ C_v((r,i-1),w^3_{r,i-1}(v))=1 \\ \quad \land\ \textsc{Real}[v,w^3_{r,i-1}] \end{array} & \begin{array}{l} (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^{**}_{r,i}(1^\lambda,n) \\ (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2,\pi^2_v)\}_{v\in V(G)} \\ \{w^3_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3,\pi^3_v)\}_{v\in V(G)} \end{array} \end{array}\right]$$
$$\geq \gamma(\cdot) - \eta_1(\lambda) - n\cdot(\eta_2(\lambda) + \eta_3(\lambda)) - \eta_4(\lambda).$$

By Lemma 6.3 (extraction at all nodes), the above equation implies there exists a negligible function $\eta_5(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \begin{array}{l} \exists v^* \in V(G): \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) \\ \land\ \mathcal{V}(\mathsf{crs};G;x,y,\pi)=1 \\ \land\ \forall v \in V(G): \\ \quad C_v((r,i),w^1_{r,i}(v))=1 \\ \quad \land\ C_v((r,j(i)),w^2_{r,j(i)}(v))=1 \\ \quad \land\ \textsc{Real}[v,w^2_{r,j(i)}] \\ \quad \land\ C_v((r,i-1),w^3_{r,i-1}(v))=1 \\ \quad \land\ \textsc{Real}[v,w^3_{r,i-1}] \end{array} & \begin{array}{l} (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^{**}_{r,i}(1^\lambda,n) \\ (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1,\pi^1_v)\}_{v\in V(G)} \\ \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2,\pi^2_v)\}_{v\in V(G)} \\ \{w^3_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3,\pi^3_v)\}_{v\in V(G)} \end{array} \end{array}\right]$$
$$\geq \gamma(\cdot) - \eta_1(\lambda) - n\cdot(\eta_2(\lambda) + \eta_3(\lambda) + \eta_5(\lambda)) - \eta_4(\lambda).$$

By Lemma 6.6 (sliding extraction window) for a receiving step, the above equation implies there exists a negligible function $\eta_6(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G) : & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^{**}_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G) : & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{AlmostReal}[v, w^1_{r,i}] & \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r,j(i)), w^2_{r,j(i)}(v)) = 1 & \{w^3_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,j(i)}] & \\ \quad \wedge\ C_v((r,i-1), w^3_{r,i-1}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,i-1}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \eta_1(\lambda) - n\cdot(\eta_2(\lambda) + \eta_3(\lambda) + \eta_5(\lambda) + \eta_6(\lambda)) - \eta_4(\lambda).$$

Let $k, \ell \in [\widetilde{n}]$ be such that $i = \mathsf{recv}(k,\ell)$. For every node $u \neq v_\ell$, we have $\widetilde{m}_{r,i}(u) = \bot$, and in that case $C_u((r,i), w^1_{r,i}(u)) = 1$ implies $m^1_{r,i}(u) = \widetilde{m}_{r,i}(u)$.

For v_ℓ, if $v_k \notin N(v_\ell)$ then $\widetilde{m}_{r,i}(v_\ell) = \bot$ as well, and the event

$$C_{v_\ell}((r,i), w^1_{r,i}(v_\ell)) = 1 \quad \wedge \quad d^1_{r,i}(v_\ell) = \widetilde{d}_{r,i}(v_\ell)$$

implies $m^1_{r,i}(v_\ell) = \widetilde{m}_{r,i}(v_\ell)$.

For $\{v_k, v_\ell\} \in E(G)$, we observe that $j(i) = \mathsf{recv}(k,\ell)$. The event:

$$I^1_{r,i}(v_\ell) = \widetilde{I}_{r,i}(v_\ell) \quad \wedge \quad I^2_{r,j(i)}(v_k) = \widetilde{I}_{r,j(i)}(v_k)$$

implies $I^1_{r,i}(v_\ell) = I^2_{r,j(i)}(v_k)$ (as the true index v_ℓ reads from in step (r,i) *is* the true index v_k writes to in step $j(i)$, that is, $\widetilde{I}_{r,i}(v_\ell) = \widetilde{I}_{r,j(i)}(v_k)$). Since $\textsc{AlmostReal}[v, w^1_{r,i}]$ implies both $d^1_{r,i}(v_\ell) = \widetilde{d}_{r,i}(v_\ell)$ and $I^1_{r,i}(v_\ell) = \widetilde{I}_{r,i}(v_\ell)$, and $\textsc{Real}[v_k, w^1_{r,j(i)}]$ implies $I^1_{r,j(i)}(v_k) = \widetilde{I}_{r,j(i)}(v_k)$, by the *collision resistance with respect to opening* property of the HT family, since C_{v_k} on input $w^2_{r,j(i)}(v_k)$ verifies that $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}^{\mathsf{msg}}(v_k), I^2_{r,j(i)}(v_k), m^2_{r,j(i)}(v_k)) = 1$, C_{v_ℓ} on input $w^1_{r,i}$ verifies that $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}^{\mathsf{msg}}(v_\ell), I^1_{r,i}(v_\ell), m^1_{r,i}(v_\ell)) = 1$, and $\mathcal{V}(\mathsf{crs}; G; x, y, \pi)$ on nodes v_k, v_ℓ verifies that $\mathsf{val}^{\mathsf{msg}}(v_k) = \mathsf{val}^{\mathsf{msg}}(v_k)$, the last equation implies there exists a negligible function $\eta_7(\lambda)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G) : & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & \\ \wedge\ \forall v \in V(G) : & \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^{**}_{r,i}(1^\lambda, n) \\ \quad \wedge\ \textsc{AlmostReal}[v, w^1_{r,i}] & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad \wedge\ C_v((r,j(i)), w^2_{r,j(i)}(v)) = 1 & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,j(i)}] & \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r,i-1), w^3_{r,i-1}(v)) = 1 & \{w^3_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,i-1}] & \\ \wedge\ \forall u \neq v_\ell(r,i) : & \\ \quad m^1_{r,i}(u) = \widetilde{m}_{r,i}(u) & \\ \wedge\ m^1_{r,i}(v_\ell) = m^2_{r,j(i)}(v_k) & \end{array}\right]$$
$$\geq \gamma(\cdot) - \eta_1(\lambda) - n\cdot(\eta_2(\lambda) + \eta_3(\lambda) + \eta_5(\lambda) + \eta_6(\lambda)) - \eta_4(\lambda) - \eta_7(\lambda).$$

Additionally, since v_ℓ reads on (r,i) what v_k writes on step $(r,j(i))$ (that is, $\widetilde{m}_{r,i}(v_\ell) = \widetilde{m}_{r,j(i)}(v_k)$), the event:

$$m^2_{r,j(i)}(v_k) = \widetilde{m}_{r,j(i)}(v_k) \quad \wedge \quad m^1_{r,i}(v_\ell) = m^2_{r,j(i)}(v_k)$$

implies that $m^1_{r,i}(v_\ell) = \widetilde{m}_{r,i}(v_\ell)$. So, since $\textsc{Real}[v_k, w^2_{r,j(i)}]$ implies $m^2_{r,j(i)}(v_k) = \widetilde{m}_{r,j(i)}(v_k)$, by the definition of $\textsc{AlmostReal}[v, w^1_{r,i}]$, the above equation actually implies:

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^{**}_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^1_{r,i}] & \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r,j(i)), w^2_{r,j(i)}(v)) = 1 & \{w^3_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_3, \pi^3_v)\}_{v\in V(G)} \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,j(i)}] & \\ \quad \wedge\ C_v((r,i-1), w^3_{r,i-1}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^3_{r,i-1}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \eta_1(\lambda) - n\cdot(\eta_2(\lambda) + \eta_3(\lambda) + \eta_5(\lambda) + \eta_6(\lambda)) - \eta_4(\lambda) - \eta_7(\lambda).$$

By the *index hiding* property of the seBARG, the above equation implies there exists a negligible function $\eta_9(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \exists v^* \in V(G): & \\ \quad M_{\mathcal{D}}(W_R(G,x,v^*)) \neq y(v^*) & \\ \wedge\ \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}_{r,i}(1^\lambda, n) \\ \wedge\ \forall v \in V(G): & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r,i), w^1_{r,i}(v)) = 1 & \{w^1_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_1, \pi^1_v)\}_{v\in V(G)} \\ \quad \wedge \textsc{Real}[v, w^1_{r,i}] & \{w^2_{r,j(i)}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_2, \pi^2_v)\}_{v\in V(G)} \\ \quad \wedge\ C_v((r,j(i)), w^2_{r,j(i)}(v)) = 1 & \\ \quad \wedge\ \textsc{Real}[v, w^2_{r,j(i)}] & \end{array}\right]$$
$$\geq \gamma(\cdot) - \eta_1(\lambda) - n\cdot(\eta_2(\lambda) + \eta_3(\lambda) + \eta_5(\lambda) + \eta_6(\lambda)) - \eta_4(\lambda) - \eta_7(\lambda) - \eta_8(\lambda) - \eta_9(\lambda).$$

This concludes the proof for the reading case for any $\xi(\cdot)$ that satisfies

$$n\cdot\xi(\cdot) \geq \eta_1(\cdot) + n\cdot\eta_2(\cdot) + n\cdot\eta_3(\cdot) + \eta_4(\cdot) + n\cdot\eta_5(\cdot) + n\cdot\eta_6(\cdot) + \eta_7(\cdot) + \eta_8(\cdot) + \eta_9(\cdot). \quad \square$$

Lemma 6.5 – the induction claim that implies contradiction given the soundness violation, is now proved by Claims 6.8, 6.9 and 6.11 and setting

$$\xi(\cdot) = \max\left(\xi_1, \sum_{k=1}^{7}\zeta_k, \sum_{k=1}^{8}\eta_k\right).$$

6.4.1 Proofs of Lemmas 6.4, 6.6 and 6.7

We now provide the proofs of Lemma 6.3 (extraction at all nodes), Lemma 6.5 (double extraction) and Lemma 6.6 (sliding extraction window).

Proof of Lemma 6.3 (extraction at all nodes). Fix steps (r_1, i_1), (r_2, i_2), and (r_3, i_3). Parse $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_1, \mathsf{crs}_2, \mathsf{crs}_3)$ and $\mathsf{td} = (\mathsf{td}_1, \mathsf{td}_2, \mathsf{td}_3)$. For every $b \in \{1,2,3\}$, by the *somewhere argument of knowledge* property of the seBARG scheme, applied for crs_b, there exists a negligible function $\nu_b(\cdot)$, such that for every $\lambda \in \mathbb{N}$, for every node $u \in V(G)$:

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}_{(r_1,i_1),(r_2,i_2),(r_3,i_3)}(1^\lambda, n) \\ \wedge\ C_u((r_b, i_b), w^b_{r_b,i_b}(u)) = 0 & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ & \{w^b_{r_b,i_b}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_b, \pi^b_v)\}_{v\in V(G)} \end{array}\right] \leq \nu_b(\lambda).$$

Set $\nu(\cdot) = \max\Big(\nu_1(\cdot), \nu_2(\cdot), \nu_3(\cdot)\Big)$ Note that since $\nu_1(\cdot), \nu_2(\cdot), \nu_3(\cdot)$ are all negligible, then so is $\nu(\cdot)$. The above implies that for every $\lambda \in \mathbb{N}$ and every $u \in V(G)$,

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{(r_1,i_1),(r_2,i_2),(r_3,i_3)}(1^\lambda, n) \\ \wedge\ C_u((r_b, i_b), w^b_{r_b,i_b}(u)) = 0 & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ & \{w^b_{r_b,i_b}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_b, \pi^b_v)\}_{v\in V(G)} \end{array}\right] \leq \nu(\lambda).$$

By a union bound over the nodes, the above implies that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}_{(r_1,i_1),(r_2,i_2),(r_3,i_3)}(1^\lambda, n) \\ \wedge\ \exists v \in V(G): & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad C_v((r_b, i_b), w^b_{r_b,i_b}(v)) = 0 & \{w^b_{r_b,i_b}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_b, \pi^b_v)\}_{v\in V(G)} \end{array}\right] \leq n \cdot \nu(\lambda),$$

as desired. □

Proof of Lemma 6.5 (double extraction). For every $(r, i) \in [R] \times [T]$ and node $u \in V(G)$, let $\textsc{Diff}^{\mathsf{double}}[u, w^{c_1}_{r,i}, w^{c_2}_{r,i}]$ be the following event:

$$\begin{array}{ll} & \mathsf{st}^{c_1}_{r,i}(u) \neq \mathsf{st}^{c_2}_{r,i}(u) \\ \vee & \mathsf{hRead}^{c_1}_{r,i}(u) \neq \mathsf{hRead}^{c_2}_{r,i}(u) \\ \vee & \mathsf{hMem}^{c_1}_{r,i}(u) \neq \mathsf{hMem}^{c_2}_{r,i}(u) \\ \vee & \mathsf{hWrite}^{c_1}_{r,i}(u) \neq \mathsf{hWrite}^{c_2}_{r,i}(u) \\ \vee & b^{c_1}_{r,i}(u) \neq b^{c_2}_{r,i}(u) \\ \vee & d^{c_1}_{r,i}(u) \neq d^{c_2}_{r,i}(u) \\ \vee & I^{c_1}_{r,i}(u) \neq I^{c_2}_{r,i}(u) \\ \vee & m^{c_1}_{r,i}(u) \neq m^{c_2}_{r,i}(u) \end{array}$$

Recall that for every $(r, i) \in [R] \times [T]$, $\mathsf{Gen}^{\mathsf{double}}_{r,i,c_1,c_2}(1^\lambda, n) \to (\mathsf{crs}, \mathsf{td})$ where $c_1 \neq c_2 \in \{1, 2, 3\}$ is a function $\mathsf{Gen}_{(r_1,i_1),(r_2,i_2),(r_3,i_3)}$ where $r_{c_1} = r_{c_2} = r$, $i_{c_1} = i_{c_2} = i$, and $(r_{c_3}, i_{c_3}) \in [R] \times [T]$. For every $u \in V(G)$, by the definition of $\{C_v\}_{v\in V(G)}$ and by the *collision resistance with respect to opening* property of the HT family, there exists a negligible function $\nu_u(\cdot)$, such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \forall v \in V(G): & \\ \quad C_v((r, i), w^{c_1}_{r,i}(v)) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{double}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \quad \wedge\ \textsc{Real}[v, w^{c_1}_{r,i}] & (G, x, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad \wedge\ C_v((r, i), w^{c_2}_{r,i}(v)) = 1 & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v\in V(G)} \\ \wedge\ \textsc{Diff}^{\mathsf{double}}[u, w^{c_1}_{r,i}, w^{c_2}_{r,i}] & \{w^{c_2}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v\in V(G)} \end{array}\right] \leq \nu_u(\lambda).$$

By the definition of the events $\textsc{Real}[u, w^{c_1}_{r,i}], \textsc{Real}[u, w^{c_2}_{r,i}]$, for every $u \in V(G)$, the event

$$\textsc{Real}[u, w^{c_1}_{r,i}] \quad \wedge \quad \neg\textsc{Real}[u, w^{c_2}_{r,i}]$$

implies the event $\textsc{Diff}^{\mathsf{double}}[u, w^{c_1}_{r,i}, w^{c_2}_{r,i}]$. So, the above equation implies for every $\lambda \in \mathbb{N}$, for every $u \in V(G)$,

$$\Pr\left[\begin{array}{l|l} \forall v \in V(G): & \\ \quad C_v((r, i), w^{c_1}_{r,i}(v)) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{double}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \quad \wedge\ \textsc{Real}[v, w^{c_1}_{r,i}] & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad \wedge\ C_v((r, i), w^{c_2}_{r,i}(v)) = 1 & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v\in V(G)} \\ \wedge\ \neg\textsc{Real}[u, w^{c_2}_{r,i}] & \{w^{c_2}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v\in V(G)} \end{array}\right] \leq \nu_u(\lambda).$$

Since a maximum of negligible functions is a negligible function, and by a union bound over the nodes, the above equation implies there exists a negligible function $\nu(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \forall v \in V(G): & \\ \quad C_v((r,i), w^{c_1}_{r,i}(v)) = 1 & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{double}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \quad \wedge\ \textsc{Real}[v, w^{c_1}_{r,i}] & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \quad \wedge\ C_v((r,i), w^{c_2}_{r,i}(v)) = 1 & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v\in V(G)} \\ \wedge\ \exists u \in V(G): & \{w^{c_2}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v\in V(G)} \\ \quad \neg\textsc{Real}[u, w^{c_2}_{r,i}] & \end{array}\right] \le n \cdot \nu(\lambda).$$

This concludes the proof of Lemma 6.5 (double extraction). $\square$

Proof of Lemma 6.6 (sliding extraction window). For every $(r,i) \in [R] \times [T] \setminus \{(1,1)\}$, and a node $u \in V(G)$, let $\textsc{Diff}^{\mathsf{slide}}[u, w^1_{r,i}, w^2_{r,i-1}]$ be the following event:

$$\begin{array}{ll} & \mathsf{st}^{c_1}_{r,i}(u) \neq \mathsf{st}^{c_2}_{r,i}(u) \\ \vee & \mathsf{hRead}^{c_1}_{r,i}(u) \neq \mathsf{hRead}^{c_2}_{r,i}(u) \\ \vee & \mathsf{hMem}^{c_1}_{r,i}(u) \neq \mathsf{hMem}^{c_2}_{r,i}(u) \\ \vee & \mathsf{hWrite}^{c_1}_{r,i}(u) \neq \mathsf{hWrite}^{c_2}_{r,i}. \end{array}$$

Recall that for every $(r,i) \in [R] \times [T] \setminus \{(1,1)\}$, $\mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \to (\mathsf{crs},\mathsf{td})$ where $c_1 \neq c_2 \in \{1,2,3\}$ is a function $\mathsf{Gen}_{(r_1,i_1),(r_2,i_2),(r_3,i_3)}$ where $r_{c_1} = r_{c_2} = r$, $i_{c_1} = i$, $i_{c_2} = i-1$ and $(r_{c_3}, i_{c_3}) \in [R] \times [T]$. For every $u \in V(G)$, by the definition of $\{C_v\}_{v\in V(G)}$, and by the *collision resistance with respect to opening* property of the HT family, there exists a negligible function $\nu_{u,1}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} C_u((r,i), w^{c_1}_{r,i}(u)) = 1 & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ \textsc{Real}[u, w^{c_2}_{r,i-1}] & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v\in V(G)} \\ \wedge\ \textsc{Diff}^{\mathsf{slide}}[u, w^1_{r,i}, w^2_{r,i-1}] & \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v\in V(G)} \end{array}\right] \le \nu_{u,1}(\lambda). \tag{6.8}$$

Fix a node $u \in V(G)$. We now continue by dividing into cases according to i.

Case 1: $i \in \mathsf{comp}$. By the definition of C_u, the event:

$$b^{c_2}_{r,i-1}(u) = \widetilde{b}_{r,i-1}(u) \quad \wedge \quad C_u((r,i-1), w^{c_2}_{r,i-1}) = 1$$

implies that $\mathsf{st}^{c_2}_{r,i}(u) = \widetilde{\mathsf{st}}_{r,i}(u)$, and in particular, the event

$$\textsc{Real}[u, w^{c_2}_{r,i}] \quad \wedge \quad C_u((r,i-1), w^{c_2}_{r,i-1}) = 1$$

implies that $\mathsf{st}^{c_2}_{r,i}(u) = \widetilde{\mathsf{st}}_{r,i}(u)$. Further, since the event $\mathsf{st}^{c_1}_{r,i}(u) \neq \mathsf{st}^{c_2}_{r,i}(u)$ implies the event $\textsc{Diff}^{\mathsf{slide}}[u, w^1_{r,i}, w^2_{r,i-1}]$, we actually have that the event

$$C_u((r,i-1), w^{c_2}_{r,i-1}) = 1 \quad \wedge \quad \textsc{Real}[u, w^{c_2}_{r,i}] \quad \wedge \quad \mathsf{st}^{c_1}_{r,i}(u) \neq \widetilde{\mathsf{st}}_{r,i}(u)$$

implies $\textsc{Diff}^{\mathsf{slide}}[u, w^1_{r,i}, w^2_{r,i-1}]$. So, Equation (6.8), implies that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} C_u((r,i), w^{c_1}_{r,i}(u)) = 1 & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ \textsc{Real}[u, w^{c_2}_{r,i-1}] & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v\in V(G)} \\ \wedge\ \mathsf{st}^{c_1}_{r,i}(u) \neq \widetilde{\mathsf{st}}_{r,i}(u) & \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v\in V(G)} \end{array}\right] \le \nu_{u,1}(\lambda).$$

If $\widetilde{\mathsf{st}}_{r,i-1}(u)$ does not instruct to write (that is, $W_{\mathcal{D}}(\widetilde{\mathsf{st}}_{r,i-1}(u)) = (\perp,\perp,\perp)$, then $C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1$ implies that hRead, hMem, and hWrite do not change in $w^{c_2}_{r,i-1}(u)$ between $(r,i-1)$ and (r,i). In this case, the event

$$\neg\textsc{Diff}^{\mathsf{slide}}[u, w^1_{r,i}, w^2_{r,i-1}] \quad\wedge\quad C_u((r,i-1), w^{c_2}_{r,i-1}) = 1 \quad\wedge\quad \textsc{Real}[u, w^{c_2}_{r,i}]$$

implies that for every $\mathsf{TP} \in \{\mathsf{Write}, \mathsf{Mem}, \mathsf{Read}\}$:

$$\mathsf{hTP}^{c_1}_{r,i}(u) = \mathsf{hTP}^{c_2}_{r,i}(u) = \mathsf{hTP}^{c_2}_{r,i-1}(u) = \widetilde{\mathsf{hTP}}_{r,i-1}(u) = \widetilde{\mathsf{hTP}}_{r,i}(u) \tag{6.9}$$

If $\widetilde{\mathsf{st}}_{r,i-1}(u)$ does instruct to make a change to one of the tapes, let $(\mathsf{TP}', j, b') = W_{\mathcal{D}}(\widetilde{\mathsf{st}}_{r,i-1}(u))$. For any $\mathsf{TP} \in \{\mathsf{Read}, \mathsf{Mem}, \mathsf{Write}\} \setminus \{\mathsf{TP}'\}$, the same event implies Equation (6.9). For TP', instead of Equation (6.9), we get the following assertion:

$$\begin{gathered}\mathsf{hTP}'^{c_1}_{r,i}(u) = \mathsf{hTP}'^{c_2}_{r,i}(u)\\ \wedge\ \mathsf{HT.WVerify}\Big(\mathsf{hk}, \mathsf{hTP}'^{c_2}_{r,i-1}(u), \mathsf{hTP}'^{c_2}_{r,i}(u), j, b', \rho^{\mathsf{TP},c_2}_{r,i}(u)\Big) = 1\\ \wedge\ \mathsf{hTP}'^{c_2}_{r,i-1}(u) = \widetilde{\mathsf{hTP}'}_{r,i-1}(u),\end{gathered}$$

which in turn implies the event

$$\mathsf{HT.WVerify}\Big(\mathsf{hk}, \widetilde{\mathsf{hTP}}_{r,i-1}(u), \mathsf{hTP}^{c_1}_{r,i}(u), j, b', \rho^{\mathsf{TP},c_2}_{r,i}(u)\Big) = 1. \tag{6.10}$$

Equation (6.9) (applicable when $W_{\mathcal{D}}(\widetilde{\mathsf{st}}_{r,i-1}(u)) = (\perp,\perp,\perp)$ and for $\mathsf{TP} \in \{\mathsf{Read}, \mathsf{Mem}, \mathsf{Write}\} \setminus \{\mathsf{TP}'\}$ when $(\mathsf{TP}', j, b) = W_{\mathcal{D}}(\widetilde{\mathsf{st}}_{r,i-1}(u))$) along with Equation (6.10) (for TP in the latter case) imply, by the *completeness* and the *collision resistance with respect to writing* properties of the HT family, that there exists a negligible function $\nu_{u,2}(\lambda)$ such that for every $\lambda \in \mathbb{N}$

$$\Pr\left[\begin{array}{l|l} C_u((r,i), w^{c_1}_{r,i}(u)) = 1 & \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \wedge\ \textsc{Real}[u, w^{c_2}_{r,i-1}] & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ \neg\textsc{Diff}^{\mathsf{slide}}[u, w^1_{r,i}, w^2_{r,i-1}] & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v \in V(G)} \\ \wedge\ \mathsf{hTP}^{c_1}_{r,i}(u) \neq \widetilde{\mathsf{hTP}}_{r,i}(u) & \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v \in V(G)} \end{array}\right] \le \nu_{u,2}(\lambda).$$

If $\widetilde{\mathsf{st}}_{r,i}(u)$ instructs to read, then let $\mathsf{TP}, j = R_{\mathcal{D}}(\widetilde{\mathsf{st}}_{r,i}(u))$. By the definition of C_u, $C_u((r,i), w^{c_1}_{r,i}(u)) = 1$ implies that $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{hTP}_{r,i}(u), j, b^{c_1}_{r,i}(u), \rho^{\mathsf{TP},c_1}_{r,i}(u)) = 1$. So, the event

$$\begin{aligned} & \mathsf{st}^{c_1}_{r,i}(u) = \widetilde{\mathsf{st}}_{r,i}(u)\\ \wedge\quad & \mathsf{hWrite}^{c_1}_{r,i}(u) = \widetilde{\mathsf{hWrite}}_{r,i}(u)\\ \wedge\quad & \mathsf{hMem}^{c_1}_{r,i}(u) = \widetilde{\mathsf{hMem}}_{r,i}(u)\\ \wedge\quad & \mathsf{hRead}^{c_1}_{r,i}(u) = \widetilde{\mathsf{hRead}}_{r,i}(u)\\ \wedge\quad & C_u((r,i), w^{c_1}_{r,i}(u)) = 1 \end{aligned}$$

implies that $\mathsf{HT.Verify}(\mathsf{hk}, \widetilde{\mathsf{hTP}}_{r,i}(u), j, b^{c_1}_{r,i}(u), \rho^{\mathsf{TP},c_1}_{r,i}(u)) = 1$. Which in turn, by the *opening completeness* and *collision resistance with respect to opening* properties of the HT family, implies there exists a negligible function $\nu_{u,3}(\cdot)$ such that for every $\lambda \in \mathbb{N}$

$$\Pr\left[\begin{array}{l|l} C_u((r,i), w^{c_1}_{r,i}(u)) = 1 & \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 & \\ \wedge\ \textsc{Real}[u, w^{c_2}_{r,i-1}] & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \wedge\ \mathsf{st}^{c_1}_{r,i}(u) = \widetilde{\mathsf{st}}_{r,i}(u) & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ \mathsf{hWrite}^{c_1}_{r,i}(u) = \widetilde{\mathsf{hWrite}}_{r,i}(u) & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v\in V(G)} \\ \wedge\ \mathsf{hMem}^{c_1}_{r,i}(u) = \widetilde{\mathsf{hMem}}_{r,i}(u) & \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v\in V(G)} \\ \wedge\ \mathsf{hRead}^{c_1}_{r,i}(u) = \widetilde{\mathsf{hRead}}_{r,i}(u) & \\ \wedge\ b^{c_1}_{r,i}(u) \neq \widetilde{b}_{r,i}(u) & \end{array}\right] \leq \nu_{u,3}(\lambda).$$

For a computing step, it holds that $\widetilde{d}_{r,i}(u) = \widetilde{I}_{r,i}(u) = \widetilde{m}_{r,i}(u) = \bot$, and the event $C_u((r,i), w^{c_1}_{r,i}(u)) = 1$ implies

$$d^{c_1}_{r,i}(u) = I^{c_1}_{r,i}(u) = m^{c_1}_{r,i}(u) = \bot = \widetilde{d}_{r,i}(u) = \widetilde{I}_{r,i}(u) = \widetilde{m}_{r,i}(u).$$

So, to conclude, the above equations together imply that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} C_u((r,i), w^{c_1}_{r,i}(u)) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 & (G,x,y,\pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ \textsc{Real}[u, w^{c_2}_{r,i-1}] & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v\in V(G)} \\ \wedge\ \neg\textsc{Real}[u, w^{c_1}_{r,i}] & \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v\in V(G)} \end{array}\right] \tag{6.11}$$
$$\leq \nu_{u,1}(\lambda) + \nu_{u,2}(\lambda) + \nu_{u,3}(\lambda).$$

Case 2: $i \in \mathsf{send} \cup \mathsf{recv}$**.** In this part of the proof, we refer to a sending or a receiving step as "a communication step." In a communication step, we have that $\widetilde{\mathsf{st}}_{r,i}(u) = \widetilde{\mathsf{st}}_{r,i-1}(u)$. So, by the definition of C_u, $C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1$ implies $\mathsf{st}^{c_2}_{r,i}(u) = \mathsf{st}^{c_1}_{r,i-1}(u)$, and the event

$$\textsc{Real}[u, w^{c_2}_{r,i-1}] \quad \wedge \quad C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 \quad \wedge \quad \mathsf{st}^{c_1}_{r,i}(u) \neq \widetilde{\mathsf{st}}_{r,i}(u)$$

implies

$$\mathsf{st}^{c_1}_{r,i}(u) \neq \widetilde{\mathsf{st}}_{r,i}(u) = \widetilde{\mathsf{st}}_{r,i-1}(u) = \mathsf{st}^{c_2}_{r,i-1}(u) = \mathsf{st}^{c_2}_{r,i}(u)$$

which implies $\textsc{Diff}^{\mathsf{slide}}[u, w^1_{r,i}, w^2_{r,i-1}]$.

Since this is a communication step, we also have, for any $i \in \mathsf{send} \cup \mathsf{recv} \setminus \{P + \widetilde{n}^2 + 1\}$, that neither of the tapes Write and Mem change between $(r,i-1)$ to (r,i). For $i = P + \widetilde{n}^2 + 1$, we have that $\widetilde{\mathsf{hWrite}}_{r,i}(u) = \mathsf{hempty}$ and by the definition of C_u, we have that $C_u((r,i-1), w^{c_1}_{r,i-1}(u)) = 1$ implies $\mathsf{hWrite}^{c_2}_{r,i}(u) = \widetilde{\mathsf{hWrite}}_{r,i}(u)$. So, similarly to the states, we have that for $\mathsf{TP} \in \{\mathsf{Write}, \mathsf{Mem}\}$, the event

$$\textsc{Real}[u, w^{c_2}_{r,i-1}] \quad \wedge \quad C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 \quad \wedge \quad \mathsf{hTP}^{c_1}_{r,i}(u) \neq \widetilde{\mathsf{hTP}}_{r,i}(u)$$

also implies $\textsc{Diff}^{\mathsf{slide}}[u, w^1_{r,i}, w^2_{r,i-1}]$.

For a sending step $i \in \mathsf{send} \setminus \{P+1\}$, the above applies also for $\mathsf{TP} = \mathsf{Read}$. For $i = P+1$, similarly to the case of $i = P + \widetilde{n}^2 + 1$ for hWrite, we have that $\widetilde{\mathsf{hRead}}_{r,i}(u) = \mathsf{hempty}$ and by the definition of C_u, we have that $C_u((r,i-1), w^{c_1}_{r,i-1}(u)) = 1$ implies $\mathsf{hRead}^{c_2}_{r,i}(u) = \widetilde{\mathsf{hRead}}_{r,i}(u)$, so overall the argument above does applies for $\mathsf{TP} = \mathsf{Read}$ for any sending step.

For a receiving step ($i \in \mathsf{recv}$), the following event

$$\mathsf{hRead}^{c_2}_{r,i-1}(u) = \widetilde{\mathsf{hRead}}_{r,i-1}(u) \quad \wedge \quad m^{c_2}_{r,i-1} = \widetilde{m}_{r,i-1}(u) \quad \wedge \quad C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1$$

implies $\mathsf{hRead}^{c_2}_{r,i}(u) = \widetilde{\mathsf{hRead}}_{r,i}(u)$. So, the event

$$\textsc{Real}[u, w^{c_2}_{r,i-1}] \quad \wedge \quad C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 \quad \wedge \quad \mathsf{hRead}^{c_1}_{r,i}(u) \neq \widetilde{\mathsf{Read}}_{r,i}(u)$$

implies $\textsc{Diff}^{\mathsf{slide}}[u, w^1_{r,i}, w^2_{r,i-1}]$ also for the case of $i \in \mathsf{recv}$.

In addition, for any communication step, for every $u \in V(G)$ it holds that $\widetilde{b}_{r,i}(u) = \bot$, and by the definition of C_u, we have that $C_u((r,i), w^{c_1}_{r,i}(u)) = 1$ implies $b^{c_1}_{r,i}(u) = \widetilde{b}_{r,i}(u)$. Therefore, Equation (6.8) implies that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} C_u((r,i), w^{c_1}_{r,i}(u)) = 1 & \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 & \\ \wedge\ \textsc{Real}[u, w^{c_2}_{r,i-1}] & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \wedge \left(\begin{array}{l} \mathsf{st}^{c_1}_{r,i}(u) \neq \widetilde{\mathsf{st}}_{r,i}(u) \\ \vee\ \mathsf{hWrite}^{c_1}_{r,i}(u) \neq \widetilde{\mathsf{hWrite}}_{r,i}(u) \\ \vee\ \mathsf{hMem}^{c_1}_{r,i}(u) \neq \widetilde{\mathsf{hMem}}_{r,i}(u) \\ \vee\ \mathsf{Read}^{c_1}_{r,i}(u) \neq \widetilde{\mathsf{hRead}}_{r,i}(u) \\ \vee\ b^{c_1}_{r,i}(u) \neq \widetilde{b}_{r,i}(u) \end{array}\right) & \begin{array}{l} (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v \in V(G)} \\ \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v \in V(G)} \end{array} \end{array}\right] \leq \nu_{u,1}(\lambda).$$

For $i \in \mathsf{send}$, let $k, \ell \in [\widetilde{n}]$ be such that $i = \mathsf{send}(k, \ell)$. For every $u \neq v_k$, we have that $\widetilde{d}_{r,i}(u) = \widetilde{I}_{r,i}(u) = \bot$, so in that case, by the definition of C_u, $C_u((r,i), w^{c_1}_{r,i}) = 1$ implies both $d^{c_1}_{r,i}(u) = \widetilde{d}_{r,i}(u)$ and $I^{c_1}_{r,i}(u) = \widetilde{I}_{r,i}(u)$. Similarly, for $i \in \mathsf{recv}$, let $k, \ell \in [\widetilde{n}]$ be such that $i = \mathsf{recv}(k, \ell)$. For every $u \neq v_\ell$, we have that $\widetilde{d}_{r,i}(u) = \bot$, so in that case, by the definition of C_u, it holds that $C_u((r,i), w^{c_1}_{r,i}) = 1$ implies $d^{c_1}_{r,i}(u) = \widetilde{d}_{r,i}(u)$.

It remains to handle the case where $u = v_k$ for $i = \mathsf{send}(k, \ell)$ and the case where $u = v_\ell$ for $i = \mathsf{recv}(k, \ell)$. By the definition of $\mathcal{V}$, $\mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1$ implies that $\mathsf{val}^{\mathsf{Port}}_u = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Port}(u))$. In the sending case, for $u = v_k$, we have $\widetilde{d}_{r,i}(u) = \mathsf{Port}(u)[\ell]$, and by the definition of the circuit C_u, the event $C_u((r,i), w^{c_1}_{r,i}(u)) = 1$ implies $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}^{\mathsf{Port}}_u, \ell, d^{c_1}_{r,i}, \rho^{\mathsf{Port},c_1}_{r,i}) = 1$. In the receiving case, for $u = v_\ell$, we have $\widetilde{d}_{r,i}(u) = \mathsf{Port}(u)[k]$, and by the definition of the circuit C_u, the event $C_u((r,i), w^{c_1}_{r,i}(v_\ell)) = 1$ implies $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}^{\mathsf{Port}}_u, k, d^{c_1}_{r,i}, \rho^{\mathsf{Port},c_1}_{r,i}) = 1$

So, by the *opening completeness* and *collision resistance with respect to opening* properties of the HT family, there exists a negligible function $\nu_{u,4}$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ C_u((r,i), w^{c_1}_{r,i}(u)) = 1 & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v \in V(G)} \\ \wedge\ d^{c_1}_{r,i}(u) \neq \widetilde{d}_{r,i}(u) & \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v \in V(G)} \end{array}\right] \leq \nu_{u,4}(\lambda).$$

Similarly, by the definition of $\mathcal{V}$, it holds that $\mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1$ implies that $\mathsf{val}^{\mathsf{in}}_u = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Ind}^{\mathsf{in}}(u))$ and that $\mathsf{val}^{\mathsf{out}}_u = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{Ind}^{\mathsf{out}}(u))$.

For the sending case, for $u = v_k$ it holds that $\widetilde{I}_{r,i}(u) = \mathsf{Ind}^{\mathsf{out}}(u)[\widetilde{d}_{r,i}(u)]$, and by the definition of C_u, $C_u((r,i), w^{c_1}_{r,i}(u)) = 1$ implies $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}^{\mathsf{out}}_u, d^{c_1}_{r,i}(u), I^{c_1}_{r,i}(u), \rho^{\mathsf{Ind},c_1}_{r,i}) = 1$. So, the event

$$C_u((r,i), w^{c_1}_{r,i}(u)) = 1 \quad \wedge \quad d^{c_1}_{r,i}(u) = \widetilde{d}_{r,i}(u)$$

implies $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}^{\mathsf{out}}_u, \widetilde{d}_{r,i}(u), I^{c_1}_{r,i}(u), \rho^{\mathsf{Ind},c_1}_{r,i}) = 1$. For the receiving case, for $u = v_\ell$ it holds that $\widetilde{I}_{r,i}(u) = \mathsf{Ind}^{\mathsf{in}}(u)[\widetilde{d}_{r,i}(u)]$, and by the definition of C_u, $C_u((r,i), w^{c_1}_{r,i}(u)) = 1$ implies $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}^{\mathsf{in}}_u, d^{c_1}r, iu, I^{c_1}_{r,i}(u), \rho^{\mathsf{Ind},c_1}_{r,i}(u)) = 1$. So, the event

$$C_u((r,i), w^{c_1}_{r,i}(u)) = 1 \quad \wedge \quad d^{c_1}_{r,i}(u) = \widetilde{d}_{r,i}(u)$$

implies that $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{val}^{\mathsf{in}}_u, \widetilde{d}_{r,i}(u), I^{c_1}_{r,i}(u), \rho^{\mathsf{Ind},c_1}_{r,i}(u)) = 1$. Therefore, by the *opening completeness* and the *collision resistance with respect to opening* properties of the HT family, there exists a negligible function $\nu_{u,5}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & \\ C_u((r,i), w^{c_1}_{r,i}(u)) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ d^{c_1} = \widetilde{d}_{r,i}(v_k) & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v \in V(G)} \\ \wedge\ I^{c_1}_{r,i}(u) \neq \widetilde{I}_{r,i}(u) & \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v \in V(G)} \end{array}\right] \leq \nu_{u,5}(\lambda).$$

For $i \in \mathsf{send}$, for every $u \neq v_k$, we have $\widetilde{m}_{r,i}(u) = \bot$, so by the definition of C_u, we have $C_u((r,i), w^{c_1}_{r,i}(u)) = 1$ implies $m^{c_1}_{r,i}(u) = \widetilde{m}_{r,i}(u)$. For $u = v_k$, we have $C_u((r,i), w^{c_1}_{r,i}(u)) = 1$ implies $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{hWrite}^{c_1}_{r,i}(u), d^{c_1}_{r,i}(u), m^{c_1}_{r,i}(u), \rho^{m,c_1}_{r,i}(u)) = 1$. So, the event

$$\mathsf{hWrite}^{c_1}_{r,i}(u) = \widetilde{\mathsf{hWrite}}_{r,i}(u) \quad \wedge \quad d^{c_1}_{r,i}(u) = \widetilde{d}_{r,i}(u) \quad \wedge \quad C_u((r,i), w^{c_1}_{r,i}(u)) = 1$$

implies $\mathsf{HT.Verify}(\mathsf{hk}, \widetilde{\mathsf{hWrite}}_{r,i}(u), \widetilde{d}_{r,i}(u), m^{c_1}_{r,i}(u), \rho^{m,c_1}_{r,i}(u)) = 1$. So, by the *opening completeness* and *collision resistance with respect to opening* properties of the HT family, there exists a negligible function $\nu_{6,u}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & \\ C_u((r,i), w^{c_1}_{r,i}(u)) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ \mathsf{hWrite}^{c_1}_{r,i}(u) = \widetilde{\mathsf{hWrite}}_{r,i}(u) & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v \in V(G)} \\ \wedge\ d^{c_1}_{r,i}(u) = \widetilde{d}_{r,i}(u) & \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v \in V(G)} \\ \wedge\ m^{c_1}_{r,i}(u) \neq \widetilde{m}_{r,i}(u) & \end{array}\right] \leq \nu_{6,u}(\lambda).$$

We now observe that overall the event

$$\begin{gathered} \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 \\ \wedge\ C_u((r,i), w^{c_1}_{r,i}(u)) = 1 \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 \\ \wedge\ \textsc{Real}[u, w^{c_2}_{r,i}] \end{gathered}$$

together with $\neg\textsc{Real}[u, w^{c_1}_{r,i}]$ for $i \in \mathsf{send}$ and together with the event $\neg\textsc{AlmostReal}[u, w^{c_1}_{r,i}]$ for $i \in \mathsf{recv}$, imply at least one of the following:

1. $\textsc{Diff}^{\mathsf{slide}}[u, w^1_{r,i}, w^2_{r,i-1}]$,
2. $d_{r,i}(u) \neq \widetilde{d}_{r,i}(u)$,
3. $d_{r,i}(u) = \widetilde{d}_{r,i}(u)$ and $I^{c_1}_{r,i}(u) \neq \widetilde{I}_{r,i}(u)$, or
4. (only for the sending case:) $\mathsf{hWrite}^{c_1}_{r,i}(u) = \widetilde{\mathsf{hWrite}}_{r,i}(u)$ and $I^{c_1}_{r,i}(u) = \widetilde{I}_{r,i}(v)$ and $m^{c_1}_{r,i}(u) \neq \widetilde{m}_{r,i}(u)$,

So, by the last equations and by Equation (6.8), we have for $i \in \mathsf{send}$, for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ C_u((r,i), w^{c_1}_{r,i}(u)) = 1 & (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 & \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v \in V(G)} \\ \wedge\ \textsc{Real}[u, w^{c_2}_{r,i-1}] & \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v \in V(G)} \\ \wedge\ \neg\textsc{Real}[u, w^{c_1}_{r,i}] & \end{array}\right] \tag{6.12}$$
$$\leq \nu_{1,u}(\lambda) + \nu_{4,u}(\lambda) + \nu_{5,u}(\lambda) + \nu_{6,u}(\lambda).$$

and for $i \in \mathsf{recv}$, for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}; G; x, y, \pi) = 1 \\ C_u((r,i), w^{c_1}_{r,i}(u)) = 1 \\ \wedge\ C_u((r,i-1), w^{c_2}_{r,i-1}(u)) = 1 \\ \wedge\ \textsc{Real}[u, w^{c_2}_{r,i-1}] \\ \wedge\ \neg\textsc{AlmostReal}[u, w^{c_1}_{r,i}] \end{array} \middle| \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}^{\mathsf{slide}}_{r,i,c_1,c_2}(1^\lambda, n) \\ (G, x, y, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}) \\ \{w^{c_1}_{r,i}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_1}, \pi^{c_1}_v)\}_{v \in V(G)} \\ \{w^{c_2}_{r,i-1}(v) \leftarrow \mathsf{seBARG}.\mathcal{E}(\mathsf{td}_{c_2}, \pi^{c_2}_v)\}_{v \in V(G)} \end{array}\right] \tag{6.13}$$
$$\leq \nu_{1,u}(\lambda) + \nu_{4,u}(\lambda) + \nu_{5,u}(\lambda).$$

Since the maximum of negligible functions is a negligible function, and by a union bound over the nodes, we get the desired, for

$$\nu(\cdot) = \nu_1(\cdot) + \max(\nu_2(\cdot) + \nu_3(\cdot), \nu_4(\cdot) + \nu_5(\cdot) + \nu_6(\cdot)). \qquad \square$$

A Using the index hiding property

Let $\mathsf{seBARG} = (\mathsf{Gen}, \mathcal{P}, \mathcal{V}, \mathcal{E})$ be a somewhere extractable BARG for an index language with circuit size s. The following discussion focuses on Gen and the *index hiding* property.

Recall the following experiment that defines the index hiding property. Given a security parameter λ, a polynomial k, and a polynomial-size adversary $\mathcal{A}$:

- The adversary $\mathcal{A}$ chooses indices (i_0, i_1) given 1^λ.
- A random bit $b \leftarrow \{0,1\}$ is chosen.
- The generation algorithm samples a common reference string with a trapdoor for the index i_b: $\mathsf{crs}, \mathsf{td} \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_b)$.
- $\mathcal{A}$ attempts to guess b given the reference string: $b' \leftarrow \mathcal{A}(\mathsf{crs})$.

We say $\mathcal{A}$ wins if $i_0, i_1 \in [k(\lambda)]$ (which could be achieved trivially), and $\mathcal{A}$ correctly guesses b: $b' = b$.

The index hiding property guarantees that for every $\mathcal{A}, k, s$, there exists a negligible function $\mathrm{negl}(\cdot)$ such that $\mathcal{A}$'s advantage in this experiment, relative to k and λ, is at most $\mathrm{negl}(\lambda)$ beyond random guessing ($\frac{1}{2}$).

In this work, as in previous works such as [CJJ21b, KLVW23], we use a supposedly stronger property that involves an additional "party": a polynomial-size algorithm $\mathcal{M}$. The alternative experiment is defined by k, $\mathcal{A}$, an index i, and a security parameter λ, and proceeds as follows:

- The generation algorithm samples a common reference string with a trapdoor for the index i: $\mathsf{crs}, \mathsf{td} \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i)$.
- Given the reference string, $\mathcal{A}$ outputs a string z: $z \leftarrow \mathcal{A}(\mathsf{crs})$.
- $\mathcal{M}$ either accepts or rejects: $b \leftarrow \mathcal{M}(\mathsf{crs}, z)$.

We say that $\mathcal{A}$ wins if $\mathcal{M}$ accepts.

We aim to guarantee that for every k and $\mathcal{A}$, there exists a negligible function $\epsilon(\cdot)$ such that for any pair of indices (i_0, i_1) and a security parameter, as long as $i_0, i_1 \in [k(\lambda)]$, merely replacing i_1 with i_0 does not allow $\mathcal{A}$ to gain an advantage greater than $\epsilon(\lambda)$ in this latter experiment.

There are two main differences between the original index-hiding property and this enhanced notion. The first difference is that we require one negligible function to bound the advantage for all pairs of indices, instead of potentially having a different function for each pair. The second difference is in the definition of $\mathcal{A}$ winning: it now involves a general polynomial-size algorithm accepting the output, instead of $\mathcal{A}$ merely guessing a bit. This second difference (which, in a way, makes the notion weaker) is what enables the guarantee, as the adversary can now verify if it has won, whereas previously, an adversary capable of determining this would have broken the property.

When we use this property in our work, $\mathcal{M}$ could sometimes simply be the seBARG verifier, but sometimes verify a bit stronger notion, for instance, both that the verifier accepts, and that some other, possibly unrelated, check pass; for example, this could be the seBARG circuit accepting some witness extracted from some other proof.

In what follows we formalize the discussed notion and prove that it follows from the original index-hiding property.

Notation. Let $\mathcal{M}(\mathsf{crs}, z) \to b$ be a polynomial-time algorithm that inputs a crs, and a string z and returns a bit b.

For every polynomial k, poly-size algorithm $\mathcal{A}$, an index i and a security parameter $\lambda \in \mathbb{N}$, denote:

$$P_{\mathcal{M}}(k, \mathcal{A}, i, \lambda) = \Pr\left[\mathcal{M}(\mathsf{crs}, z) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i) \\ z \leftarrow \mathcal{A}(\mathsf{crs}) \end{array} \right].$$

We now state Lemma 6.3 using the notation:

Lemma A.1 (Lemma 6.3, restated)**.** *For every poly-size algorithm $\mathcal{M}$, polynomial k and a poly-time algorithm $\mathcal{A}$, there exists a negligible function $\epsilon(\cdot)$ such that for every pair of indices (i_0, i_1) and every security parameter $\lambda \in \mathbb{N}$, if $i_0, i_1 \in [k(\lambda)]$, we have:*

$$P_{\mathcal{M}}(k, A, i_1, \lambda) \geq P_{\mathcal{M}}(k, A, i_0, \lambda) - \epsilon(\lambda).$$

Proof of Lemma 6.3. Assume the lemma does not hold. Fix $\mathcal{M}$, k, and $\mathcal{A}$ such that for every negligible function $\epsilon(\cdot)$, there exists an index pair (i_0, i_1) and a security parameter $\lambda \in \mathbb{N}$ such that $i_0, i_1 \in [k(\lambda)]$, and

$$P_{\mathcal{M}}(k, \mathcal{A}, i_1, \lambda) < P_{\mathcal{M}}(k, \mathcal{A}, i_0, \lambda) - \epsilon(\lambda). \tag{A.1}$$

Let $\mathcal{A}'$ be the following adversary:

- On input 1^λ, $\mathcal{A}'$ outputs (i_0^*, i_1^*) such that

$$(i_0^*, i_1^*) = (i_0^*(\lambda), i_1^*(\lambda)) = \arg \max_{(i_0, i_1) \in [k(\lambda)]^2} P_{\mathcal{M}}(k, \mathcal{A}, i_0, \lambda) - P_{\mathcal{M}}(k, \mathcal{A}, i_1, \lambda).$$

- On input crs, $\mathcal{A}'$ simulates $\mathcal{A}(\mathsf{crs})$, to obtain $z \leftarrow \mathcal{A}(\mathsf{crs})$, and checks whether $\mathcal{M}(\mathsf{crs}, z) = 1$. If so, it outputs 0. Otherwise, it outputs 1.

Let $\epsilon(\cdot)$ be a negligible function, and let (i'_0, i'_1) and $\lambda' \in \mathbb{N}$ be a pair of indices and a security parameter satisfying Equation (A.1) for the negligible function $\epsilon'(\cdot) = 2\epsilon(\cdot)$. We now show that as a distinguisher in the original index-hiding experiment, $\mathcal{A}'$ has an advantage larger than $\epsilon(\lambda)$ for some $\lambda \in \mathbb{N}$ (in particular, for $\lambda = \lambda'$), contradicting the index-hiding property of the seBARG.

The following is $\mathcal{A}'$'s probability of distinguishing for every $\lambda \in \mathbb{N}$:

$$\begin{aligned}
&\Pr\left[\begin{array}{l} i_0^*, i_1^* \in [k] \\ \mathcal{A}'(\mathsf{crs}) = b \end{array} \;\middle|\; \begin{array}{l} (i_0^*, i_1^*) \leftarrow \mathcal{A}'(1^\lambda) \\ b \leftarrow \{0,1\} \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_b) \end{array} \right] \\
&= \frac{1}{2} \Pr\left[\begin{array}{l} i_0^*, i_1^* \in [k] \\ \mathcal{A}'(\mathsf{crs}) = b \end{array} \;\middle|\; \begin{array}{l} (i_0^*, i_1^*) \leftarrow \mathcal{A}'(1^\lambda) \\ b = 0 \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_b) \end{array} \right] \\
&+ \frac{1}{2} \Pr\left[\begin{array}{l} i_0^*, i_1^* \in [k] \\ \mathcal{A}'(\mathsf{crs}) = b \end{array} \;\middle|\; \begin{array}{l} (i_0^*, i_1^*) \leftarrow \mathcal{A}'(1^\lambda) \\ b = 1 \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_b) \end{array} \right].
\end{aligned}$$

We now note that by $\mathcal{A}'(1^\lambda)$'s choice of (i_0^*, i_1^*), and by the definition of $\mathcal{A}'(\mathsf{crs})$,

$$\begin{aligned}
&\Pr\left[\begin{array}{c} i_0^*, i_1^* \in [k] \\ \mathcal{A}'(\mathsf{crs}) = b \end{array} \,\middle|\, \begin{array}{l} (i_0^*, i_1^*) \leftarrow \mathcal{A}'(1^\lambda) \\ b = 0 \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_b^*) \end{array}\right] \\
&= \Pr\left[\ \mathcal{A}'(\mathsf{crs}) = 0 \ \middle|\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_0^*(\lambda))\ \right] \\
&= \Pr\left[\ \mathcal{M}(\mathsf{crs}, \pi) = 0 \ \middle|\ \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_0^*(\lambda)) \\ \pi \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \\
&= P_{\mathcal{M}}(k, \mathcal{A}, i_0^*(\lambda), \lambda).
\end{aligned}$$

On the other hand,

$$\begin{aligned}
&\Pr\left[\begin{array}{c} i_0^*, i_1^* \in [k] \\ \mathcal{A}'(\mathsf{crs}) = b \end{array} \,\middle|\, \begin{array}{l} (i_0^*, i_1^*) \leftarrow \mathcal{A}'(1^\lambda) \\ b = 1 \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_b) \end{array}\right] \\
&= \Pr\left[\ \mathcal{M}(\mathsf{crs}, \pi) = 0 \ \middle|\ \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i_1^*(\lambda)) \\ \pi \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \\
&= 1 - P_{\mathcal{M}}(k, \mathcal{A}, i_1^*(\lambda), \lambda).
\end{aligned}$$

By $\mathcal{A}'(1^\lambda)$'s choice of (i_0^*, i_1^*), we have for every security parameter $\lambda \in \mathbb{N}$ and every $(i_0, i_1) \in [k(\lambda)]^2$,

$$P_{\mathcal{M}}(k, \mathcal{A}, i_0^*(\lambda), \lambda) - P_{\mathcal{M}}(k, \mathcal{A}, i_1^*(\lambda), \lambda) > P_{\mathcal{M}}(k, \mathcal{A}, i_0, \lambda) - P_{\mathcal{M}}(k, \mathcal{A}, i_1, \lambda)$$

By applying the latter for $\lambda = \lambda'$ and $(i_0, i_1) = (i_0', i_1')$ defined earlier, we get

$$P_{\mathcal{M}}(k, \mathcal{A}, i_0^*(\lambda'), \lambda') - P_{\mathcal{M}}(k, \mathcal{A}, i_1^*(\lambda'), \lambda') \geq P_{\mathcal{M}}(k, \mathcal{A}, i_0', \lambda') - P_{\mathcal{M}}(k, \mathcal{A}, i_1', \lambda') > 2\epsilon(\lambda').$$

So, overall, we have:

$$\begin{aligned}
&\Pr\left[\begin{array}{c} i_0^*, i_1^* \in [k] \\ \mathcal{A}'(\mathsf{crs}) = b \end{array} \,\middle|\, \begin{array}{l} (i_0^*, i_1^*) \leftarrow \mathcal{A}'(1^{\lambda'}) \\ b \leftarrow \{0, 1\} \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^{\lambda'}, k, 1^s, i_b) \end{array}\right] \\
&= \frac{1}{2} \cdot P_{\mathcal{M}}(k, \mathcal{A}, i_0^*(\lambda), \lambda) + \frac{1}{2}(1 - P_{\mathcal{M}}(k, \mathcal{A}, i_1^*(\lambda), \lambda)) \\
&= \frac{1}{2} + \frac{1}{2}(P_{\mathcal{M}}(k, \mathcal{A}, i_0^*(\lambda), \lambda) - P_{\mathcal{M}}(k, \mathcal{A}, i_1^*(\lambda), \lambda)) > \frac{1}{2} + \epsilon(\lambda'),
\end{aligned}$$

as required. □

Remark A.2. We remark that the latter proof works for *non-uniform adversaries*, as the index in use by the reduction depends on the security parameter λ, and we cannot assume that there is some efficient way to find it given λ. This means that the latter cannot be adapted as-is for a case where the underlying index-hiding property is only guaranteed against uniform adversaries.

Acknowledgments

We would like to thank Omer Paneth and Rafael Pass for fruitful and illuminating discussions.

References

[ABOR00] William Aiello, Sandeep N Bhatt, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Fast verification of any remote procedure call: Short witness-indistinguishable one-round proofs for NP. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 463–474, 2000.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.

[AO24] Eden Aldema Tshuva and Rotem Oshman. On polynomial time local decision. In *27th International Conference on Principles of Distributed Systems (OPODIS 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2024.

[APV91] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction. In *Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 268–277, 1991.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349, 2012.

[BDFO18] Alkida Balliu, Gianlorenzo D'Angelo, Pierre Fraigniaud, and Dennis Olivetti. What can be verified locally? *Journal of Computer and System Sciences*, 97:106–120, 2018.

[BFO22] Yoav Ben Shimon, Orr Fischer, and Rotem Oshman. Proof labeling schemes for reachability-related problems in directed graphs. In *Structural Information and Communication Complexity: 29th International Colloquium, SIROCCO 2022, Paderborn, Germany, June 27–29, 2022, Proceedings*, pages 21–41. Springer, 2022.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 474–482, 2017.

[BKK$^+$18] Saikrishna Badrinarayanan, Yael Tauman Kalai, Dakshita Khurana, Amit Sahai, and Daniel Wichs. Succinct delegation for low-space non-deterministic computation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 709–721, 2018.

[BKO22] Aviv Bick, Gillat Kol, and Rotem Oshman. Distributed zero-knowledge proofs over networks. In *SODA*, pages 2426–2458. SIAM, 2022.

[CGJ$^+$23] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In *Proceedings of the 43rd Annual International Cryptology Conference, CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 635–668. Springer, 2023.

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In *Proceedings of the 41st Annual International Cryptology Conference, CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 394–423. Springer, 2021.

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for P from LWE. In *62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 68–79, 2021.

[Dam87] Ivan Bjerre Damgård. Collision free hash functions and public key signature schemes. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 203–216. Springer, 1987.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-np and applications. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1057–1068. IEEE, 2022.

[DL08] Giovanni Di Crescenzo and Helger Lipmaa. Succinct NP proofs from an extractability assumption. In *Proceedings of the 4th Conference on Computability in Europe, CiE 2008*, volume 5028 of *LNCS*, pages 175–185. Springer, 2008.

[DLN$^+$04] Cynthia Dwork, Michael Langberg, Moni Naor, Kobbi Nissim, and Omer Reingold. Succinct proofs for NP and spooky interactions. *Unpublished manuscript, available at http://www. cs. bgu. ac. il/˜ kobbi/papers/spooky_ sub_crypto. pdf*, 2004.

[FBP22] Laurent Feuilloley, Nicolas Bousquet, and Théo Pierron. What can be certified compactly? compact local certification of MSO properties in tree-like graphs. In *PODC*, pages 131–140. ACM, 2022.

[Feu21] Laurent Feuilloley. Introduction to local certification. *Discrete Mathematics and Theoretical Computer Science*, 23(3), 2021.

[FFH$^+$21] Laurent Feuilloley, Pierre Fraigniaud, Juho Hirvonen, Ami Paz, and Mor Perry. Redundancy in distributed proofs. *Distributed Comput.*, 34(2):113–132, 2021.

[FGKS13] Pierre Fraigniaud, Mika Göös, Amos Korman, and Jukka Suomela. What can be decided locally without identifiers? In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 157–165, New York, NY, USA, 2013. ACM.

[FHK12] Pierre Fraigniaud, Magnús M Halldórsson, and Amos Korman. On the impact of identifiers on local decision. In *International Conference On Principles Of Distributed Systems*, pages 224–238, Berlin, Heidelberg, 2012. Springer.

[FMO$^+$19] Pierre Fraigniaud, Pedro Montealegre, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. On distributed Merlin-Arthur decision protocols. In *SIROCCO*, volume 11639 of *LNCS*, pages 230–245. Springer, 2019.

[FMRT22] Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. A meta-theorem for distributed certification. In *Proceedings of the 29th International Colloquium on Structural Information and Communication Complexity, SIROCCO 2022*, volume 13298 of *LNCS*, pages 116–134, 2022.

[FPP19] Pierre Fraigniaud, Boaz Patt-Shamir, and Mor Perry. Randomized proof-labeling schemes. *Distributed Computing*, 32:217–234, 2019.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.

[GGH11] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Collision-free hashing from lattice problems. *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation: In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 30–39, 2011.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security, ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, 2010.

[GS16] Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory Comput.*, 12(1):1–33, 2016.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 99–108, 2011.

[HR18] Justin Holmgren and Ron Rothblum. Delegating computations with (almost) minimal time and space overhead. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 124–135. IEEE, 2018.

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC '21*, pages 708–721. ACM, 2021.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732, 1992.

[KKP05] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 9–18, 2005.

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1545–1552, 2023.

[KOS18] Gillat Kol, Rotem Oshman, and Raghuvansh R. Saxena. Interactive distributed proofs. In *Symposium on Principles of Distributed Computing (PODC)*, pages 255–264, 2018.

[KP98] Shay Kutten and David Peleg. Fast distributed construction of small k-dominating sets and applications. *Journal of Algorithms*, 28:27, 1998.

[KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In *Proceedings of the 14th International Theory of Cryptography Conference, TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 91–118, 2016.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 1115–1124. ACM, 2019.

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In *Symposium on Theory of Computing Conference, STOC'13*, pages 565–574. ACM, 2013.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *Symposium on Theory of Computing, STOC'14*, pages 485–494. ACM, 2014.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[Mer89] Ralph C. Merkle. A certified digital signature. In *Proceedings of the 9th Annual International Cryptology Conference, CRYPTO '89*, volume 435 of *LNCS*, pages 218–238. Springer, 1989.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[NPY20] Moni Naor, Merav Parter, and Eylon Yogev. The power of distributed verifiers in interactive proofs. In Shuchi Chawla, editor, *Symposium on Discrete Algorithms (SODA)*, pages 1096–115, 2020.

[OPR17] Rafail Ostrovsky, Mor Perry, and Will Rosenbaum. Space-time tradeoffs for distributed verification. In *International Colloquium on Structural Information and Communication Complexity*, pages 53–70. Springer, 2017.

[Pel00] David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* Society for Industrial and Applied Mathematics, USA, 2000.

[PP17] Boaz Patt-Shamir and Mor Perry. Proof-labeling schemes: broadcast, unicast and in between. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 1–17. Springer, 2017.

[SHK$^+$12] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed Verification and Hardness of Distributed Approximation. In *SIAM Journal on Computing (special issue of STOC 2011)*, November 2012.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography Conference*, pages 1–18. Springer, 2008.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *Proceedings of the 42nd Annual International Cryptology Conference, CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 433–463. Springer, 2022.