On Computationally-Sound Distributed Certification and Efficient Local Decision

ANONYMOUS AUTHOR(S)

In local distributed algorithms, nodes are traditionally allowed to have unbounded computational power. This makes the model incomparable with centralized notions of efficient computations such as P and NP. In this paper, we study computationally-bounded distributed local decision and ask what can be achieved by computationally-efficient local algorithms and provers.

The contributions of this work are twofold. First, we study distributed certification, where we wish to certify that a distributed network satisfies some property, or that a distributed algorithm has produced correct output. To that end, a *prover* assigns to each node of the network a certificate, and the nodes then interact amongst themselves to verify the proof. We introduce the notion of *computationally-sound distributed certification*, where instead of requiring perfect soundness against any prover, we require only that a *computationally-efficient* prover must not be able to convince the network of a false statement, except with negligible probability. We show that under certain cryptographic assumptions, any property in NP can be certified using a polylogarithmic number of bits by a global prover that knows the entire network, and any computationally-efficient distributed algorithm can be certified by an efficient distributed prover that produces certificates of polylogarithmic length in the algorithm's local computation time, round complexity, and message size.

Next, we study the effect of restricting the nodes themselves to be computationally efficient. We introduce the classes PolyLOCAL and NPolyLOCAL of polynomial-time local decision and nondeterministic polynomial-time local decision, respectively, and compare them to the centralized complexity classes P and NP, and to the distributed classes LOCAL and NLOCAL, which correspond to local deterministic and nondeterministic decision, respectively. We show that when the size of the network is not known to the nodes, PolyLOCAL $\subsetneq$ LOCAL $\cap$ P; that is, there exists a problem that can be decided by an inefficient local algorithm and also by a poly-time centralized algorithm, but not by a poly-time local algorithm. When the size of the network is known, an unconditional separation of PolyLOCAL from LOCAL $\cap$ P would imply that P $\neq$ NP; however, we are still able to show that PolyLOCAL $\subsetneq$ LOCAL $\cap$ P assuming that injective one-way functions exist. When nondeterminism is introduced, the distinction vanishes, and NPolyLOCAL = NLOCAL $\cap$ NP.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Additional Key Words and Phrases: datasets, neural networks, gaze detection, text tagging

1 INTRODUCTION

In distributed graph algorithms, the typical complexity measures that one tries to minimize are related to communication and synchronization: we aim to construct algorithms that use few synchronized communication rounds, and send a small number of messages, each as short as possible. There is a rich body of literature on the power of such algorithms: from the LOCAL model, where communication rounds are the key resource, to CONGEST, where bandwidth is the main bottleneck, and many other combinations and variations. Distributed algorithms are typically allowed to have unbounded *local* computational power, with each network node able to compute any Turing-computable function for free (e.g., [31, 32]), or sometimes even any function at all (e.g., [51]). This puts the theory of local decision on completely different footing from classical centralized notions of efficiency, such as P and NP, and makes them impossible to compare.

In this paper, we study *computationally-bounded* distributed local decision, and ask what can be achieved by computationally-efficient local algorithms and provers. We show that computational restrictions can be helpful for prover-assisted distributed certification, but on the other hand, when there is no prover, computational restrictions do limit the power of local algorithms beyond what one might expect.

Computationally-sound distributed proofs. In distributed certification (also known as *proof labeling schemes* [47] or *locally checkable proofs* [36]), our goal is to certify some property of the network: for example, one might wish to certify that the output of a distributed algorithm is correct, or that the network graph has some desirable property. To facilitate this goal, we enlist the help of a *prover*, which provides each node with a short certificate; the nodes exchange their certificates with one another (or more generally, carry out some efficient verification procedure) and decide whether to accept or reject. The proof is considered to be *accepted* if all nodes accept. While many useful properties can be certified using short certificates, some problems are known to require very long certificates and a lot of communication between the nodes, up to $\Omega(n^2)$ bits [36].

The area of distributed certification has so far stood apart from the rich theory of centralized decision problems (e.g., P vs. NP) and delegated computation. In the centralized setting, under cryptographic assumptions, a computationally-bounded prover can present a weak verifier with a short proof that convinces the verifier of a statement of the form "$x \in \mathcal{L}$", where $\mathcal{L}$ is any language in P [20, 43], or, under stronger assumptions, in NP [14, 38, 49]. This is called a *computationally-sound proof*, or a *succinct argument* for $\mathcal{L}$. The key is that rather than requiring perfect soundness against any prover, we require only that the verifier not be fooled by any computationally-bounded prover, except with negligible probability. In return, we get much shorter proofs. If the argument is non-interactive, it is called a *succinct non-interactive argument* (SNARG), and if it also has the property that whenever the prover convinces the verifier, one can extract an NP-witness from the prover, then the argument is a *succinct non-interactive argument of knowledge* (SNARK).

We ask whether the same can be done in the distributed setting, and show that the answer is *yes*, under standard cryptographic assumptions in the case of language in P, or under somewhat strong assumptions for languages in NP. We define *succinct distributed arguments*, which are computationally-sound (non-interactive) proofs in the distributed network setting, and show:

Theorem 1.1. *Let $\mathcal{L}$ be a language on graphs.*

(1) *If $\mathcal{L} \in$ P, assuming SNARGs for P and collision-resistant hash function exist,*[1] *there is a succinct distributed argument for $\mathcal{L}$ with certificates of length* $\mathrm{polylog}(n)$.
(2) *If $\mathcal{L} \in$ NP, assuming SNARKs for NP and collision-resistant hash functions exist, there is a distributed argument for $\mathcal{L}$ using certificates of length* $\mathrm{polylog}(n)$.

Certifying executions of efficient distributed algorithms. One of the main motivations for studying distributed certification is fault-tolerance and self-stabilization: to cope with a dynamic and fault-prone environment, it is useful to be able to identify when the network is in an illegal state, so that we can undertake actions to correct the problem. Proof labeling schemes were originally defined at least in part with this motivation in mind [47]. One property that is very interesting to certify is whether output that was previously produced by a distributed algorithm $\mathcal{D}$ is still up-to-date: if executed in the current network, would $\mathcal{D}$ still produce the same output? It was shown in [47] that if $\mathcal{D}$ runs in r rounds, and every node sends at most m messages of b bits per round, then the execution of $\mathcal{D}$ can be certified using certificates of length $O(rmb)$ bits per node[2] by storing the entire history of messages sent at each node. Unfortunately, these certificates can be very long when the algorithm uses many rounds or messages.

With this motivation in mind, we show that a distributed algorithm that runs in polynomial number of rounds, message size and local computation time can *certify its own execution*, using

[1]We introduce these primitives in Section 2 and 3, and discuss concrete hardness assumptions under which they are known to exist in Appendix A and C.

[2]In [47] the scheme given is for certifying any property $\mathcal{P}$ that can be *verified* by a distributed algorithm that accepts or rejects at each node. The property "the algorithm produces the given output" can certainly be verified by running the algorithm itself and examining its output.

certificates of polylogarithmic length at each node, and incurring an additive overhead to the running time that is linear in the diameter of the graph.

Theorem 1.2 (Informal). *Let $\mathcal{D}$ be a distributed algorithm that runs in $T = \text{poly}(n)$ rounds and sends messages of length $\text{poly}(n)$. Assuming SNARKs for* NP *and a certain type of collision-resistant hash functions exist, there is a distributed argument of length $\text{polylog}(n)$ certifying $\mathcal{D}$'s execution, where the prover is an efficient distributed algorithm running in $O(T + \text{diam}(G))$ rounds and sending messages of $\text{polylog}(n)$ bits.*

Computationally-bounded local decision. The power of local decision algorithms has been extensively studied, under many variations (e.g., [31, 32, 51], and many others), perhaps the most famous of which is the LOCAL model. In all cases (to our knowledge), the algorithm is allowed unbounded local computational power, and as a result, deterministic local decision is incomparable with the usual notion of computational efficiency, the class P of polynomial-time algorithms. To bridge this gap, we define the class $\mathsf{PolyLOCAL}(t)$ of local distributed algorithms that run in $t(n)$ synchronous rounds, and require local computation time $\text{poly}(n)$ at every node. (The size of the network is not necessarily known to the nodes; we consider both options.)

What is the power of algorithms in $\mathsf{PolyLOCAL}(t)$? Clearly, such algorithms cannot decide languages that are not in P, nor can they decide languages that are not in $\mathsf{LOCAL}(t)$ (i.e., decidable in $t(n)$ rounds with no computational restrictions). But can they decide every language in $\mathsf{LOCAL}(t) \cap \mathsf{P}$? It turns out that the answer is "probably not", but whether or not we can prove it unconditionally depends on whether the nodes know the size of the network, and thus know *how long* they are allowed to run. Let $\mathsf{LOCAL}^{[\mathsf{n}]}$, $\mathsf{PolyLOCAL}^{[\mathsf{n}]}$ be variants of LOCAL and PolyLOCAL (resp.) where nodes know the size of the network. Then we can show:

Theorem 1.3. *We have:*

(1) $\mathsf{PolyLOCAL}(o(n)) \subsetneq \mathsf{LOCAL}(o(n)) \cap \mathsf{P}$;[3]
(2) *If* $\mathsf{PolyLOCAL}^{[\mathsf{n}]}(o(n)) \neq \mathsf{LOCAL}^{[\mathsf{n}]}(o(n)) \cap \mathsf{P}$, *then* $\mathsf{P} \neq \mathsf{NP}$*; and*
(3) *Assuming injective one-way functions exist,*[4] $\mathsf{PolyLOCAL}^{[\mathsf{n}]}(o(n)) \subsetneq \mathsf{LOCAL}^{[\mathsf{n}]}(o(n)) \cap \mathsf{P}$.

When we introduce nondeterminism, the distinction disappears:

Theorem 1.4. *Let* $\mathsf{NLOCAL}(t)$, $\mathsf{NPolyLOCAL}(t)$ *be the classes of languages decidable by nondeterministic $t(n)$-round algorithms with unbounded or, resp., polynomially-bounded local computation time. Then* $\mathsf{NPolyLOCAL}(t) = \mathsf{NLOCAL}(t) \cap \mathsf{NP}$.

Organization. The remainder of the paper is organized as follows. In Sections 2 and 3 we review the relevant background, discuss some of the cryptographic primitives we use, and give formal definitions for some of them; some definitions are deferred to Appendix A. In Section 4 we define succinct distributed arguments, and show how to construct them for NP-languages and for languages in P (Theorem 1.1). In Section 5 we construct the distributed prover of Theorem 1.2. Finally, in Section 6, we discuss the power of computationally-efficient distributed algorithms, and prove Theorem 1.3. Many of the proofs, as well as pseudocode for the constructions in Sections 4 and 5, are deferred to the appendix.

2 BACKGROUND AND RELATED WORK

Distributed certification. Although its roots trace back to work in self-stabilization, the field of distributed certification was formally initiated in [47], which introduced *proof labeling schemes*,

[3] A preliminary version of part (1) of Theorem 1.3 appeared in the brief announcement [6].
[4] A *one-way function* is a function that is easy to compute, but hard to invert. See Section 6 for the details.

and showed several constructions and impossibility results, among them the scheme for certifying spanning trees which is used in the current paper (and is a central building block for many certification schemes). Many variants of the basic model have been studied, featuring different communication constraints for the verifiers (e.g.,[29, 52, 54]), allowing randomization or interaction with the prover (e.g., [11, 46, 50]), and studying other settings; we refer to the excellent survey [27] for a comprehensive overview. To our knowledge, in all prior work, the prover and the verifier have unbounded local computational power.

In [25], the authors consider locally-restricted proof labeling schemes, where the prover itself is a (computationally-unbounded) local algorithm; however, the proof is required to be sound against any prover, not just a local one.

Local distributed decision. Local distributed algorithms have received an enormous amount of attention from the community, and local distributed decision in particular. Over the past decade there has been a significant effort towards building a complexity theory for the area: for example, in [32], the authors study the classes LD, BPLD and NLD of languages decidable by deterministic, randomized, or nondeterministic local algorithms,[5] relate them to one another, and prove (among other results) that combining randomization and nondeterminism allows a constant-round local algorithm to decide any language. We refer to the survey [28] for an overview of the area of local decision. Again, to our knowledge, in prior work the local computation power of the nodes is always unbounded.

Computationally sound proofs. The idea of a proof system that is only sound against adversaries with bounded computational power was introduced by Micali [49], who gave an implementation based on an earlier interactive protocol by Kilian [45]. Since Micali's work extensive research has been made on obtaining succinct, non-interactive arguments (SNARGs) in more realistic models, such as the *Common Reference String* (CRS) model (see Section 3 and Appendix A). Very recently SNARGs for all languages in P have been constructed from standard cryptographic assumptions [20, 41, 43, 44]. In the case of NP, [33] presented a substantial barrier to constructing SNARGs from standard hardness assumptions, and indeed, all known constructions of SNARGs use *knowledge assumptions*, which are considered nonstandard.

Knowledge assumptions capture the intuition that an algorithm whose output implicitly relates to some hard-to-compute value must *obtain* that value along the computation.[6] Under such assumptions, a SNARG candidate becomes even stronger—it becomes a *SNARG of knowledge*, a SNARK: under the knowledge assumption, whenever the prover manages to convince the verifier to accept, we can extract from the prover a *witness*.[7] The ability to extract a witness is useful for composing SNARKs with other primitives, and we use it for this purpose in Sections 4.2, 5. Despite the barrier of [33] on constructing SNARKs from standard cryptographic assumptions, they are nevertheless used on some blockchains, including Ethereum [1] and others [2–4].

[5]In [32] and much of the follow-up work, the outputs of the nodes are not allowed to depend on the identifiers in the network, and there is particular emphasis on the role of identifiers and their effect on expressive power. In this work we do not restrict the way that identifiers may be used; for this reason we use the notation LOCAL, NLOCAL instead of LD, NLD to denote the languages decidable by local deterministic and nondeterministic algorithms, respectively.

[6]For example, the knowledge-of-exponent assumption [21] essentially asserts that given a cyclic group G of prime order, a generator g of G, and an element $h = g^a$ for some $a \in [|G|]$, if we want to compute a pair (c, y) such that $c^a = y$, we must *compute the exponent a*, which is believed to be hard (this is the *discrete log assumption*).

[7]E.g., in the knowledge-of-exponent example, we can extract the exponent a.

3 PRELIMINARIES

Local distributed algorithms. We study the *local decision* model of [32],[8] where we have an unknown communication network G and an input assignment $x : V(G) \to \{0,1\}^*$. The pair (G, x) is called a *configuration*, and we use n to denote the size of the graph ($n = |V(G)|$). A *distributed language* $\mathcal{L}$ is a set of configurations. The *locality radius* of a distributed algorithm is $t : \mathbb{N} \to \mathbb{N}$ if all nodes halt within $t(n)$ rounds in networks of size n. We let $N_G(v)$ denote the neighborhood of node v in G, omitting the subscript G when the graph is clear from the context.

We assume that the nodes have unique identifiers, drawn from some large domain $\mathcal{U}$, and we typically assume that a UID from $\mathcal{U}$ can be represented using $O(\log n)$ bits.[9][10] We often conflate nodes with their UIDs. We assume that we have some linear ordering $\mathcal{U}$ of the UID space, that is, for any two UIDs $u \neq v$ from $\mathcal{U}$, either $u < v$ or $v < u$.

When we need to encode a graph G, we represent it as an adjacency list, $L(G) = (N(v_1), \ldots, N(v_n))$, where $N(v_i)$ is the neighborhood of node v_i. The nodes appear in $L(G)$ in order of their UIDs, $v_1 < \ldots < v_n$.

The local computation of each network node $v \in V(G)$ is represented by a Turing machine which takes as input the UID v, the neighborhood $N(v)$ of v in G and its input $x(v)$, and in each round, reads the messages received by v from a dedicated input tape, and writes the messages sent by v on a dedicated output tape. Eventually, the machine enters a halting state, which is either accepting or rejecting. A configuration (G, x) is *accepted* iff all nodes accept, and otherwise the configuration is *rejected*; a distributed algorithm D *decides* the distributed language $\mathcal{L}(D)$ of configurations (G, x) that are accepted when D is executed in (G, x).

On security parameters, succinctness and efficient provers. Throughout Sections 4 and 5, we use cryptographic primitives that are sound against adversaries whose running time is bounded, typically polynomially, as a function of a security parameter, $\lambda \in \mathbb{N}$. The *succinctness* of these primitives—that is, the encoding length of whatever object they produce (e.g., the length of a hash value, or a proof)—is $\mathrm{poly}(\lambda, \log n)$. To get proofs of length $\mathrm{polylog}(n)$, the security parameter we use is $\lambda = \log^c(n)$ for some $c > 1$; we are interested in adversaries whose running time is polynomial in n, which means they are sub-exponential in λ. To allow for such provers, our hardness assumptions require security not against a polynomial-time adversary but against a sub-exponential one. It is relatively common to assume sub-exponential hardness; for example, the learning-with-errors (LWE) problem is believed to be sub-exponentially hard TODO: citations .

In the sequel, whenever we say "efficient adversary/prover", we mean sub-exponential in λ and polynomial in n.

Common reference string (CRS) model. The cryptographic primitives that we use are proved sound in the *common reference string* (CRS) model, where we assume that a trusted setup phase has occurred, during which all parties get access to a *common reference string* drawn from a known distribution. For example, the CRS can be used to select a hash function. In the distributed prover of Section 5, the CRS can be generated by having every node v propose a random string r_v, and summing the strings up a spanning tree to produce $\mathrm{crs} = \oplus_{v \in V} r_v$, which is then disseminated to all the nodes. As long as a single node generates its random string honestly, the resulting crs will be uniformly random.

---

[8]Except that unlike [10, 32], we do not restrict the use of UIDs, as explained above.

[9]This assumption is not essential, as UIDs from a larger domain can be hashed down to $\{1, \ldots, n\}$ in our constructions.

[10]In Section 6, when we consider networks of unknown size, we do not make any assumptions on the encoding of the UIDs, and in fact our results continue to hold even in anonymous networks.

Hash functions. A hash function is accessed using two procedures, Gen and Hash: $\mathsf{Gen}(1^\lambda, \ell) \rightarrow \mathsf{hk}$ is a setup procedure that takes the security parameter λ (in unary) and the length ℓ of the values to be hashed, and returns a *hash key* hk; $\mathsf{Hash}(\mathsf{hk}, x)$ takes a hash key and a value x, and returns a hashed value.

s

Vector commitment schemes. A vector commitment consists of the following algorithms.

$\mathsf{Gen}(1^\lambda, q) \rightarrow \mathsf{crs}$*:* a randomized algorithm that takes as input the security parameter λ and the length q of the committed vector, and outputs a common reference string crs.

$\mathsf{Com}(\mathsf{crs}, m_1, \ldots, m_q) \rightarrow (c, \mathsf{aux})$*:* a deterministic algorithm that takes as input q messages $m_1, \ldots, m_q \in \mathcal{M}$ and a common reference string crs, and outputs a commitment string c together with auxiliary information aux.

$\mathsf{Open}(\mathsf{crs}, m, i, \mathsf{aux}) \rightarrow \Lambda_i$*:* a deterministic algorithm that takes the crs, a message m, an index i, and auxiliary information aux, and produces a proof Λ_i that m is the i^{th} committed message.

$\mathsf{Ver}(\mathsf{crs}, C, m, i, \Lambda) \rightarrow b$*:* the verification algorithm takes the crs, a message m, an index i, and a proof Λ, and outputs an acceptance bit.

TODO: def:VC is used more than once. So is def:snarg

Definition 3.1 (Vector Commitments)**.** A VC (Gen, Com, Open, Ver) is required to satisfy the following properties.

Completeness. For every messages sequence, $m_1, \ldots m_q$,

$$\Pr\left[\forall i \in [q] : \mathsf{Ver}(\mathsf{crs}, C, m_i, i, \Lambda_i) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, q) \\ C, aux \leftarrow \mathsf{Com}(\mathsf{crs}, m_1, \ldots, m_q) \\ \forall i \in [q] : \Lambda_i \leftarrow \mathsf{Open}(\mathsf{crs}, m_i, i, \mathsf{aux})) \end{array} \right] = 1$$

Position-Binding. For every $i \in [q]$, for any efficient adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\cdot)$ such that for every λ,

$$\Pr\left[\begin{array}{l} \mathsf{Ver}(\mathsf{crs}, C, m, i, \Lambda_i) = 1 \wedge \\ \mathsf{Ver}(\mathsf{crs}, C, m', i, \Lambda'_i) = 1 \end{array} \;\middle|\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, q) \\ (C, m, m', i, \Lambda, \Lambda'_i) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array} \right] \leq \epsilon(\lambda)$$

Succinctness. The length of the commitment c output from Com, and the length of the opening Λ_i, output from Open, are both bounded by $\mathrm{poly}(\lambda, \log q)$.

Succinct non-interactive arguments of knowledge (SNARKs). A SNARK consists of the following procedures:

- $\mathsf{Gen}(1^\lambda, \ell) \rightarrow \mathsf{crs}$: a setup procedure that takes a security parameter λ and an instance length ℓ, and generates a crs.
- $\mathcal{P}(\mathsf{crs}, x, w) \rightarrow \pi$: a prover algorithm that takes the crs, an instance x of length ℓ, and a witness w of length $\mathrm{poly}(\ell)$, and produces a proof π.
- $\mathcal{V}(\mathsf{crs}, x, \pi) \rightarrow \{0, 1\}$: a verifier algorithm takes the crs, an instance x of length ℓ, and a proof π, and returns 1 or 0 (accept or reject).

Definition 3.2 (Succinct Non-Interactive Argument for NP)**.** Let $\mathcal{L}$ be an NP language, with a verifying machine M ($x \in \mathcal{L} \Leftrightarrow \exists w : M(x, w) = 1$), and let λ be a security parameter. $(\mathsf{Gen}, \mathcal{V}, \mathcal{P})$ is a *Succinct Non-Interactive Argument for* $\mathcal{L}$ if it satisfies the following properties.

Completeness. For every x and w such that $M(x,w) = 1$,

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

Soundness. For any efficient prover $\mathcal{P}^*$, there exists a negligible function $\epsilon(\cdot)$, such that

$$\Pr\left[\ \mathcal{V}(\mathsf{crs}, x, \pi^*) = 1 \ \middle|\ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, \ell) \\ (x, \pi^*) \leftarrow \mathcal{P}^*(\mathsf{crs}) \end{array} \right] \leq \epsilon(\lambda)$$

Verifier Efficiency. $\mathcal{V}$ runs in time $\mathrm{poly}(\lambda, |\pi|) = \mathrm{poly}(\lambda, \log n)$.

Prover Efficiency. $\mathcal{P}$ runs in time $\mathrm{poly}(\lambda, n)$.

Note that the prover $\mathcal{P}^*$ *chooses* the statement x that it would like to prove, and it does so after seeing the crs. This is called *adaptive soundness*, and it is stronger than asserting that there does not exist any $x \notin \mathcal{L}$ that the prover can cause the verifier to accept with non-negligible probability (if there existed such an x, we could "hard-wire" it into the prover in the adaptive definition).

Finally, the SNARK is required to be *succinct*: the length of the proof π produced by $\mathcal{P}$ should be $\mathrm{poly}(\lambda, \log \ell)$, and the $\mathcal{V}$ procedure should run in time $\mathrm{poly}(\lambda, |\pi|) = \mathrm{poly}(\lambda, \log \ell)$.

4 SUCCINCT DISTRIBUTED ARGUMENTS

In this section we define *succinct distributed arguments* and show how to construct them for graph languages in NP. In Section 4.3 we give a construction for graph languages in P, which is similar in spirit but uses different cryptographic primitives and has a different soundness proof. In particular, it can be instantiated under *standard* cryptographic assumptions.

For simplicity, in this section and the next, we restrict attentiont to *graph languages*, where the nodes have no input. The definition and the constructions easily extend to the case where there are inputs (see Appendix B).

4.1 Defining Succinct Distributed Arguments

A succinct distributed argument 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.

$\mathcal{P}(\mathsf{crs}, G, w) \to \{\pi_v\}_{v \in V(G)}$*:* takes a crs obtained from $\mathsf{Gen}(1^\lambda, n)$, , a graph G on n nodes, and possibly a witness w of length $\mathrm{poly}(n)$ (which may be empty, e.g., for languages in P), and outputs a list of proofs $\{\pi(v)\}_{v \in V(G)}$, one for each node $v \in G$.

$\mathcal{V}(\mathsf{crs}, v, N(v), \pi(v), \pi(N(v))) \to \{0,1\}$*:* takes a common reference string crs obtained from $\mathsf{Gen}(1^\lambda, n)$, a UID v, a list of neighbors $N(v)$, a proof $\pi(v)$, and the proofs of the neighbors, $\pi(N(v)) = \{\pi(u) : u \in N(v)\}$,[11] and outputs an acceptance bit.

Definition 4.1. Let $\mathcal{L}$ and $\mathcal{R}$ be an NP language and a compatible relation on graphs, such that $G \in \mathcal{L}$ iff there exists a witness w such that $(G, w) \in \mathcal{R}$. A *succinct distributed argument* for $\mathcal{L}$ and $\mathcal{R}$, denoted $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$, satisfies the following properties:

Completeness. For $(G, w) \in \mathcal{R}$,

$$\Pr\left[\begin{array}{l} \forall v \in V(G): \\ \mathcal{V}(\mathsf{crs}, v, N(v), \pi(v), \pi(N(v))) = 1 \end{array} \middle|\ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \{\pi(v)\}_{v \in V(G)} \leftarrow \mathcal{P}(\mathsf{crs}, G, w) \end{array} \right] = 1.$$

[11]For simplicity, we follow the original design of proof labeling schemes [47], where neighbors only exchange their certificates with their immediate neighbors. The model can be generalized to allow for more general verification procedures.

Soundness. For any efficient algorithm $\mathcal{P}^*$ there exists a negligible function $\epsilon(\cdot)$ such that

$$\Pr\left[\begin{array}{l} G \notin \mathcal{L} \\ \wedge\ \forall v \in V(G): \\ \quad \mathcal{V}(\mathsf{crs}, v, N(v), \pi_v, \pi(N(v))) = 1 \end{array} \middle| \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ (G, \{\pi_v\}_{v \in V(G)}) \leftarrow \mathcal{P}^*(\mathsf{crs}, 1^\lambda, 1^n) \end{array}\right] \le \epsilon(\lambda).$$

Note that, as in the definition of SNARKs, the prover gets to choose the false statement it would like to prove after seeing the crs.

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

Verifier Efficiency. $\mathcal{V}$ runs in time $\mathrm{poly}(\lambda, |\pi|) = \mathrm{poly}(\lambda, \log n)$.

Prover Efficiency. $\mathcal{P}$ runs in time $\mathrm{poly}(\lambda, n)$.

4.2 Succinct Distributed Arguments for NP from SNARKS

In this section, we construct a distributed argument for graph languages in NP using Vector Commitments and SNARKs, proving the second part of Theorem 1.1. We give an overview of the construction and the details can be found in Appendix B.1.

The idea is simple: given a graph G represented as an adjacency list $AdjL = (N(v_1), \ldots, N(v_n))$, the prover provides all the nodes with the same proof, which consists of

- A vector commitment c to the adjacency list $AdjL$, and
- A SNARK proof π_{SNARK} proving that there exists an adjacency list $AdjL'$ whose vector commitment is c, such that the graph represented by $AdjL'$ is in $\mathcal{L}$.

Additionally, to convince the nodes that $AdjL' = AdjL$, the prover gives each node v_i an opening to the i-th coordinate of the vector commitment, allowing v_i to verify that the i-th coordinate of c opens to v_i's true neighborhood, $N(v_i)$. If all nodes succeed in their verification, and they all received the same commitment c, then c is indeed a commitment to the true graph G; the nodes then verify the SNARK proof π_{SNARK}, which convinces them that $G \in \mathcal{L}$.

One issue with this scheme is that the nodes do not initially have an ordering $v_1, \ldots, v_n$ of their UIDs, so each node does not know the coordinate of the vector commitment to which it is supposed to receive an opening. We resolve these issues by modifying the approach above slightly:

- Instead of committing to the adjacency list $AdjL = (N(v_1), \ldots, N(v_n))$, the prover commits to a list $L = ((v_1, N(v_1)), \ldots, (v_n, N(v_n)))$ that also includes the UIDs of the nodes, so that when a node opens a given coordinate, it can verify that its own UID appears there.
- To prevent the prover from committing to a graph that is larger than G, we ask the prover to prove a stronger property in the SNARK proof: it proves that there exists an adjacency list $AdjL'$ whose vector commitment is c, such that $AdjL'$ *is symmetric*, and the graph represented by $AdjL'$ *is connected* and satisfies $\mathcal{L}$.

If the prover now tries to commit to a list $AdjL'$ that is longer than the size of the real graph, then since the graph G', represented by $AdjL'$, is connected, the cut between the "fake nodes" $V(G') \backslash V(G)$ and the "real nodes" $V(G)$ includes some edge, $\{u, v\}$, where $u \in V(G)$ and $v \notin V(G)$. Since G' is symmetric, $v \in N_{G'}(u)$. Thus, when node u opens its coordinate in the vector commitment, it will see that its purported neighborhood there includes the "fake node" v, and it will reject.

We remark that in the construction as presented above, the prover sends the same SNARK proof to every node, and all nodes verify it. This is not needed; for example, using an additional $O(\log n)$ bits, we can ask the prover to provide a spanning tree of the network [47], and have only the root receive the SNARK proof and verify it.

We briefly sketch the soundness proof for this construction (see Appendix B.1 for the full proof). Suppose we have an efficient prover $\mathcal{P}^*$ that generates "false statements" $G \notin \mathcal{L}$, together with certificates $\{\pi(v)\}_{v\in V(G)}$ that are accepted with non-negligible probability. The certificates include a commitment c to an adjacency list, and a SNARK proof π_{SNARK}. By the argument of knowledge property of the SNARK, we can extract a *witness* from $\mathcal{E}_{\mathcal{P}^*}$ in the form of an adjacency list $AdjL'$, which is supposed to match the commitment c and represent a symmetric and connected graph $G' \in \mathcal{L}$. Now there are two cases: if $AdjL'$ is *not* a proper witness—if it does not match the commitment c, or it does not represent a graph G' that is symmetric, connected, and in $\mathcal{L}$—then we have broken the *argument of knowledge* property of the SNARK, by extracting an improper witness for a statement that is accepted with non-negligible probability. On the other hand, if $AdjL'$ is a proper witness, then since $G \notin \mathcal{L}$, we know that $AdjL \neq AdjL'$ (where $AdjL$ is the adjacency list of G). We show that this means we have broken the position-binding property of the vector commitment, by proving that every coordinate of $AdjL'$ is opened by some node of G, which verifies that its UID and its neighborhood are correctly represented. The prover is thus able to fool at least one node v into accepting a commitment to an incorrect value, which differs from $(v, N(v))$.

4.3 Succinct Distributed Arguments for P from RAM SNARGs

We give a very high-level sketch of our construction for graph languages in P, which does not rely on knowledge assumptions; the details appear in Appendix B.2.

The construction uses a primitive called a *flexible RAM SNARG for* P [42, 43], whose precise definition we defer to Appendix A.4. In general, a SNARG is used to prove statements of the form "$M(x) = b$", where M is a deterministic polynomial-time Turing machine, x is an input, and $b \in \{0, 1\}$ indicates whether M accepts or rejects the input.[12] The key property of the RAM SNARG of [42, 43] is that the prover and the verifier are actually not given the instance x, but only a *hash* of x, called a *digest*, which is much shorter than the input x itself. Roughly speaking, the prover then proves the statement "the input whose digest is d is accepted/rejected by M".

Of course, this is not well-defined: since the digest x is much shorter than x, there may be *many* inputs that have the same digest as x—some of them may be accepted by M, and some may be rejected. What does it mean to prove that "the input whose digest is d is accepted/rejected by M", when this input is not unique? This is resolved in [42] by re-defining the soundness of the SNARG: we now require only that an efficient adversary should not be able to find a digest d and two proofs π_0, π_1, such that π_0 convinces the verifier that M rejects, and π_1 convinces the verifier that M accepts (both with respect to "some input" whose digest is d). This soundness definition suffices for our purposes here.

The digest we use is a vector commitment c to the network graph, which the prover computes and gives to all the nodes, as in the previous section. In addition, the prover computes a RAM SNARG proof for the statement "the graph whose vector commitment is c is accepted by M", where M is a deterministic Turing machine that decides the graph language $\mathcal{L} = \mathcal{L}(M)$ that we would like to certify. As before, each node opens its entry in the vector commitment and verifies that its neighborhood is correctly represented, and then the nodes verify the SNARG proof.

The soundess proof for the new construction is quite different from the previous one: before, we relied on the proof-of-knowledge property, which allowed us to *extract* from a cheating prover $\mathcal{P}^*$ a concrete graph $G' \neq G$ that has the same vector commitment as G, and argue that $\mathcal{P}^*$ breaks the position-binding property of the vector commitment. A SNARG does not have the proof-of-knowledge property, so even if the prover $\mathcal{P}^*$ has successfully convinced all nodes to accept a graph $G \notin \mathcal{L}$, this does not mean we can use $\mathcal{P}^*$ to find a graph $G' \neq G$ that has the same vector commitment as G. To get around this issue, we require an additional property from the vector

[12]Since P is closed under complement, we can prove both membership in $\mathcal{L}(M)$ and non-membership in $\mathcal{L}(M)$.

commitment, which essentially asserts that for any given vector $m = (m_1, \ldots, m_q)$, there is only one commitment c that opens to m_i at every position i:

Definition 4.2 (Inverse Collision-Resistance). A VC $(\text{Gen}, \text{Com}, \text{Open}, \mathcal{V})$ is *Inverse Collision-Resistant* if for any efficient adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\cdot)$, such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \forall i : \mathcal{V}(crs, C^*, m, i) = 1 & crs \leftarrow Gen(1^\lambda, q) \\ \wedge C^* \neq C & C^*, \{(m_i, \lambda_i)\}_{i\in[q]} = \mathcal{A}(crs) \\ & C \leftarrow \text{Com}(crs, m_1, \ldots, m_q) \end{array}\right] \leq \epsilon(\lambda).$$

In Appendix A we show that a succinct inverse collision-resistant VC can be implemented from a collision-resistant hash function using a Merkle tree [48].

To conclude our sketch of the soundness proof, suppose a cheating prover $\mathcal{P}^*$ is able to find a graph $G \notin \mathcal{L}(M)$ that is accepted by all the nodes with non-negligible probability. Since $G \notin \mathcal{L}(M)$, we know that $M(G) = 0$, and we can compute the vector commitment c of G, and an honest SNARG proof π_0 for the statement "the graph whose vector commitment is c is rejected by M". However, since $\mathcal{P}^*$ was able to convince all nodes to accept, it has found a proof π_1 that convinces them that "the graph whose vector commitment is c' is accepted by M", where c' is *also* a vector commitment to G (otherwise, some node would open its entry in c', see that its neighborhood is not represented correctly, and reject). By the inverse collision-resistance property of the commitment, there can only be one commitment that opens correctly at all nodes, and therefore $c = c'$. But this violates the soundness of the RAM SNARG, as we have now found a digest (c) and two proofs π_0, π_1, both of which convince the verifiers, but they prove opposite statements.

5 CERTIFYING EXECUTIONS OF COMPUTATIONALLY-EFFICIENT DISTRIBUTED ALGORITHMS

In this section we construct a succinct distributed argument for certifying the execution of polynomial-time distributed algorithm, where the prover is itself distributed; essentially, the distributed algorithm certifies its own execution, using an additional $O(\text{diam}(G))$ rounds.

Fix a distributed algorithm, represented by a deterministic Turing machine D that executes at every node. The distributed language we would like to certify is the language $\mathcal{L}_D$ consisting of all configurations (G, x, y), where G is the network graph, $x : V(G) \to \mathcal{X}$ is an input assignment to the nodes, and $y : V(G) \to \mathcal{Y}$ is the output stored at the nodes, such that when D is executed at every node of G with input assignment x, each node $v \in V(G)$ produces the output $y(v)$. We construct a distributed prover for the statement "$(G, x, y) \in \mathcal{L}_D$".

To simplify the presentation, we assume here that there is no input x, and that D is a *decision* algorithm, so that the output we want to certify is $y(v) = 1$ at all nodes. (See Appendix B for the general case.)

Overview of the construction. Certifying the execution of the distributed algorithm D essentially amounts to certifying a collection of "local" statements, each asserting that at a specific node $v \in V(G)$, the algorithm D indeed produces the output $y(v) = 1$. The prover at node v can record the local computation at node v as D executes, and use a SNARG or a SNARK to certify that it is correct: for example, it can certify that incoming messages are handled correctly (according to D), outgoing messages are produced correctly, and eventually, the output of v is indeed $y(v) = 1$. The main obstacle is verifying consistency across the nodes: how can we be sure that incoming messages recorded at node v were indeed sent by v's neighbors, and that v's outgoing messages are indeed received by v's neighbors?

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 $\{v, u\}$; on the other side 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.

Our solution is instead to compute a hash $h(m)$ for every message m sent or received by node v, but store only a *sum* of the hashes: we separate outgoing messages from incoming messages, and store two sums, $s_{out}(v) = \sum_{\text{outgoing } m} h(m)$ and $s_{in}(v) = \sum_{\text{incoming } m} h(m)$. To certify consistency across all the nodes, we compute a spanning tree of the network, and store at every tree node u the partial sums in its subtree,

$$S_{out}(u) = \sum_{v \in u\text{'s subtree}} s_{out}(v), \qquad S_{in}(u) = \sum_{v \in u\text{'s subtree}} s_{in}(v).$$

In particular, at the root r of the tree, we store the full sums:

$$S_{out}(r) = \sum_{v \in V(G)} s_{out}(v), \qquad S_{in}(r) = \sum_{v \in V(G)} s_{in}(v).$$

The root then compares the two sums, and verifies that they are equal, which means that every message sent was indeed received, and vice-versa.

Since we compare *sums* of hashed values rather than directly comparing hashed values to one another, our construction requires the following property, which we call *Sum-Collision-Resistance*; it is a plausible strengthening of standard collision-resistance (see discussion in Appendix A). TODO: add + subsection of appendix

Definition 5.1 (Sum-Collision-Resistant Hash (SCRH))**.** A hash family (Gen, Hash) is considered *sum-collision-resistant* if for any probabilistic poly-time adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\cdot)$, such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \sum_{m \in M} \mathsf{Hash}(\mathsf{hk}, m) = \sum_{m' \in M'} \mathsf{Hash}(\mathsf{hk}, m') \\ M \neq M' \end{array} \middle| \begin{array}{l} \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ (M, M') \leftarrow \mathcal{A}(\mathsf{hk}, 1^\lambda, n) \end{array}\right] \leq \epsilon(\lambda).$$

Detailed description of the construction. In the sequel, fix an SCRH, (SCRH.Gen, SCRH.Hash).

We represent a message by $msg = (r, \{u, v\}, m)$, where $r \in \mathbb{N}$ is the round number in which the message was sent, $\{u, v\} \in E$ is the edge on which the message was sent, and $m \in \{0, 1\}^*$ is the contents of the message. It is important that this representation of a message does not indicate whether the message was sent by u and received by v or vice-versa, as our construction relies on hashing messages and verifying that every (hashed) incoming message has a corresponding (hashed) outgoing message.

The consistency of the local computation at a specific node is captured by a language $\mathcal{D}$, which consists of all triplets $(\mathsf{hk}, I(v), W(v))$ such that:

- hk is a hash key obtained by calling SCRH.Gen,
- $I(v) = (v, N(v), s_{in}(v), s_{out}(v))$, where $v \in \mathcal{U}$ is the UID of a node, $N(v) \in \mathcal{U}^*$ is the neighborhood of the node, and $s_{in}(v), s_{out}(v)$ are hash sums;
- $W(v) = (msgout(v), msgin(v))$ consists of two sets of messages;
- $(\mathsf{hk}, I(v), W(v)) \in \mathcal{D}$ iff when the distributed algorithm D is executed at a node with UID v and neighbors $N(v)$, and the incoming messages at node v are $msgin(v)$, then node v sends the messages $msgout(v)$ and accepts (i.e., outputs 1), and furthermore,

$$s_{in} = \sum_{msg \in msgin} \mathsf{SCRH.Hash}(\mathsf{hk}, msg), \qquad s_{out} = \sum_{msg \in msgout} \mathsf{SCRH.Hash}(\mathsf{hk}, msg). \tag{5.1}$$

We think of $W(v) = (msgout(v), msgin(v))$ as a *witness* to the correct computation at node v.

Since the algorithm D is itself a polynomial-time Turing machine, and the SCRH can be computed in polynomial time, there is a polynomial-time Turing machine M that decides the language $\mathcal{D}$. Fix a SNARK (SNARK.Gen, SNARK.$\mathcal{P}$, SNARK.$\mathcal{V}$, SNARK.$\mathcal{E}$) for the language of pairs (hk, I) satisfying $\exists W = (msgout, msgin) : M$ accepts (hk, I, W).

The distributed prover at each node v computes the following certificate $\pi(v)$:

- The hash-sums $s_{out}(v), s_{in}(v)$.
- A SNARK proof $\pi_{\text{SNARK}}(v)$, certifying that there exists a witness $W(v) = (msgout(v), msgin(v))$ such that $(\mathsf{hk}, I(v), W(v)) \in \mathcal{D}$.

In addition, the distributed prover computes a spanning tree of the network in $O(\text{diam}(G))$ rounds, and stores at each node v the parent $p(v)$ of v (or $\perp$, if v is the root), and a spanning-tree certificate [47] consisting of the UID of the root and the distance of v from the root. Finally, by convergecast up the tree, the distributed prover computes and stores at v the partial sums

$$S_{out}(v) = s_{out}(v) + \sum_{u \in \text{children}(v)} S_{out}(u), \qquad S_{in}(v) \leftarrow s_{in}(v) + \sum_{u \in \text{children}(v)} S_{in}(u).$$

6 ON POLYNOMIAL-TIME LOCAL DISTRIBUTED ALGORITHMS

TODO: refer to appendix: is this the right version?? In this section we investigate the power of computationally-bounded local decision algorithms: we define complexity classes for languages decidable by such algorithms, and study their relationship to the class of languages that can be decided by local algorithms with unbounded local computational power, and to the complexity class P. On a high level, our main result is that combining the requirements for locality and computational efficiency in one algorithm is more restrictive than requiring that the language be decidable by one algorithm that is local, and also by another algorithm that is computationally efficient.

6.1 Definitions

Fix an input domain $\mathcal{X}$, and a UID space $\mathcal{U}$, and let $C = C(\mathcal{X}, \mathcal{U})$ be the set of all configurations (G, x) with inputs $x : V(G) \to \mathcal{X}$ drawn from $\mathcal{X}$, and UIDs drawn from $\mathcal{U}$. We let $\mathcal{B}^t$ be the set of all t-neighborhoods that appear in C: $\mathcal{B}^t = \left\{ N^t_{G,x}(v) : (G,x) \in C, v \in V(G) \right\}$, where $N^t_{G,x}(v)$ denotes the t-neighborhood of v, including the UIDs and the inputs of the nodes in the t-neighborhood.

We model a *t-local decision algorithm* as a mapping $\mathcal{A} : \mathcal{B}^t \to \{0, 1\}$, which outputs a Boolean value (accept / reject). We require that $\mathcal{A}$ be a computable function. As usual, a configuration is accepted by $\mathcal{A}$ iff when $\mathcal{A}$ is executed at every node, it outputs 1 everywhere.

Definition 6.1 (The classes LOCAL, PolyLOCAL)**.** A distributed language $\mathcal{L}$ is in the class $\mathsf{LOCAL}(t(\cdot))$ if it can be decided in graphs of size n by a $t(n)$-local decision algorithm $\mathcal{A}$. If in addition the algorithm $\mathcal{A}$ can be computed by a Turing machine that runs in time $\text{poly}(n)$ in graphs of size n, then $\mathcal{L}$ is in the class $\mathsf{PolyLOCAL}(t(\cdot))$.

We are interested in algorithms that run in a sublinear number of rounds: let $\mathsf{LOCAL} = \bigcup_{t(\cdot) \in o(n)} \mathsf{LOCAL}(t(\cdot))$, and let $\mathsf{PolyLOCAL} = \bigcup_{t(\cdot) \in o(n)} \mathsf{PolyLOCAL}(t(\cdot))$. Note that, as usual in the area of local decision, the local algorithm may not know the size n of the network; nevertheless, as external observers, we can study the dependence of the algorithm's locality radius and its local running time on n.

6.2 Unconditional Separation of PolyLOCAL from LOCAL ∩ P

By definition we have PolyLOCAL ⊆ LOCAL, as every PolyLOCAL-algorithm is also an LOCAL-algorithm. It is also easy to see that PolyLOCAL ⊆ P: if every node of the network computes its decision in $\mathrm{poly}(n)$ time, then a poly-time centralized Turing machine can simulate the local algorithm at every node, and accept iff all nodes accept. Together we have that PolyLOCAL $\subseteq$ LOCAL $\cap$ P. Our first result shows that the containment is strict.

High-level overview. To separate PolyLOCAL from LOCAL$\cap$P, we use a variation on the language ITER, which was used in [10] to separate Π_1^{local} from LOCAL. We call our variation ITER-BOUND.

The idea is to construct a language of paths, where the center node is given a Turing machine M, two inputs $a, b \in \{0,1\}^*$, and a bound s; the goal is to decide whether M halts on both a and b within at most s computation steps, and accepts either a or b (or both). The bound s may be much larger than the length of the input (it is encoded in binary), so an efficient algorithm cannot afford to run M for s steps and check whether it accepts a or b, but a local algorithm with unbounded computation time can do so, and therefore ITER-BOUND $\in$ LOCAL. To make the task solvable for a polynomial-time centralized Turing machine, we add additional annotations (in the form of inputs to the nodes): on the left side of the path, from the center outwards, we write the sequence of configurations that M goes through in its computation on a, until it halts; on the right side of the path we do the same for b. This makes it possible for a poly-time Turing machine to simply examine the computation sequence of M, make sure it is legal (i.e., it matches the transition function of M), and verify that at either the left or the right side of the path (or both) we have an accepting configuration of M. Thus, ITER-BOUND $\in$ P.

Finally, we prove that an algorithm that is both local and efficient cannot decide the language ITER-BOUND: intuitively, this is because it can neither afford to run M for s steps, nor can it "see" the endpoints of the path to verify that at least one of them has an accepting configuration. The formal proof shows that if there existed a PolyLOCAL-algorithm for ITER-BOUND then we could use it to decide in polynomial time a language that is not in P.

Detailed construction. Let M be a Turing machine, and let $a, b \in \{0,1\}^*$ be strings such that M halts on input a and on input b. We define a configuration $C^{n_L,n_R}(M,a,b,s) = (G,x)$, as follows:

- G is a path of the form $u_{n_L}, \ldots, u_1, v, w_1, \ldots, w_{n_R}$, consisting of a *pivot node* $v \in V(G)$, a left sub-path $L = u_{n_L}, \ldots, u_1$, and a right sub-path $R = u_1, \ldots, u_{n_R}$.
- The input of the pivot node v is $x(v) = (0, \langle M \rangle, a, b, s)$, where $\langle M \rangle$ is the encoding of the Turing machine M.
- For each node $u_i \in L$ on the left sub-path, the input of u_i is given by $u_i = (i, \langle M \rangle, M_{a,i})$, where $M_{a,i}$ is the configuration of M after i steps executing with input a (recall that the configuration of a Turing machine consists of the contents of the tape, the location of the tape head, and the current state). To avoid the clash in terminology, we refer to configurations of Turing machines as *TM-configurations.*
- Similarly, for each node $w_i \in R$ on the right sub-path, we have $x(w_i) = (i, \langle M \rangle, M_{b,i})$.

We simplify the notation somewhat by writing $C^n(M,a,b,s) = C^{n,n}(M,a,b,s)$, and $C(M,a,b,s) = C^s(M,a,b,s)$. Given a configuration $C^{n_L,n_R}(M,a,b,s) = (G,x)$ as defined above, we say that a node $u \in V(G)$ is *r-central* if the distance of u from the pivot is at most r.

The language ITER-BOUND consists of all configuration $C^{n_L,n_R}(M,a,b,s)$ such that the TM-configurations written at the end of both sub-paths are both halting, $s \geq \max(n_L, n_R)$, and M accepts a or b (or both).

As we explained above, it is not difficult to see that ITER-BOUND can be decided by a local algorithm, and is also in P:

Claim 6.2. ITER-BOUND $\in$ LOCAL $\cap$ P.

Next we show that ITER-BOUND is not decidable by a polynomial-time local algorithm:

Claim 6.3. ITER-BOUND $\notin$ PolyLOCAL.

Proof. Suppose for the sake of contradiction that there is a PolyLOCAL-algorithm $\mathcal{A}$ that decides ITER-BOUND, and let $t > 0$ be its locality radius. Let $\mathcal{L} \in \mathsf{DTIME}(2^n) \setminus \mathsf{P}$ be some language that is Turing-decidable in time $O(2^n)$ but not in polynomial time, and such that $\epsilon \notin \mathcal{L}$ (here and in the sequel, ϵ denotes the empty word). Such a language exists by the Time Hierarchy Theorem [39]. We claim that using the PolyLOCAL-algorithm $\mathcal{A}$ that decides ITER-BOUND, we can construct a polynomial-time Turing machine that decides $\mathcal{L}$, a contradiction.

Let M be a $\mathsf{DTIME}(2^n)$-time Turing machine that decides $\mathcal{L}$, and let $f \in O(2^n)$ be a function bounding the running time of M on inputs of length n. Given input $z \in \{0,1\}^*$, let $C_z = C(M, \epsilon, z, f(|z|))$ be the configuration that encodes the runs of M on ϵ (on the left sub-path) and on z (on the right sub-path) until M halts, using sub-paths of length $f(|z|)$. Since we assume that $\epsilon \notin \mathcal{L}$, we have $C_z \in$ ITER-BOUND iff $z \in \mathcal{L}$.

We define a poly-time Turing machine M' for $\mathcal{L}$ as follows: on input $z \in \{0,1\}^*$, M' constructs the configuration $C'_z := C^{2t}(M, \epsilon, z, f(|z|))$, which is essentially the central portion of C_z, including only $2t$ nodes to the left and to the right of the pivot (a total of $4t + 1$ nodes). Next, M' simulates the local algorithm $\mathcal{A}$ on all the nodes of C'_z. Finally, M' accepts iff $\mathcal{A}$ outputs 1 at all t-central nodes of C'_z (ignoring the outputs of the other nodes).

It is not difficult to verify that the running time of M' is polynomial in $|z|$, in the description length of M (which is constant), and in $t = o(n)$. To show that M' indeed decides $\mathcal{L}$, suppose first that $z \in \mathcal{L}$. Then $C_z \in$ ITER-BOUND by construction, and therefore $\mathcal{A}$ must output 1 at all nodes of C_z. But this means that all t-central nodes in C'_z must also accept: for each t-central node u in C'_z, the t-local view of u is the same in C_z and in C'_z, because C'_z is obtained from C_z by removing only nodes at distance greater than t from u. Since the output of u depends only on its t-local view, and we know that u accepts in C_z, it must also accept in C'_z Thus, M' accepts z.

Now suppose that $z \notin \mathcal{L}$. In this case, $C_z \notin$ ITER-BOUND, because in C_z the two inputs encoded in x are both rejected by M (as $\epsilon, z \notin \mathcal{L}$). We claim that at least one t-central node of C_z must reject; as above, this means that the same node also rejects in C'_z, causing M' to reject z.

Suppose for the sake of contradiction that all t-central nodes of C_z accept. However, since $C_z \notin$ ITER-BOUND, we know that some node of C_z rejects; let u be such a node. The distance of u from the pivot v must be greater than t, since we assumed that no t-central node rejects. Now fix some string $a \in \mathcal{L}$ (which must exist, as $\emptyset \in \mathsf{P}$ and we assumed $\mathcal{L} \notin \mathsf{P}$), and let $C_{a,z} = C(M, a, z, f(\max(|a|, |z|)))$ be the configuration encoding the runs of M on a (on the left sub-path) and on z (on the right sub-path), using paths of length $f(\max(|a|, |z|)$, so that M halts on both. Since $a \in \mathcal{L}$, we have $C_{a,z} \in$ ITER-BOUND, and thus all nodes must accept $C_{a,z}$. This includes node u. However, since u is at distance greater than t from the pivot, the t-local view of u is the same in $C_{a,z}$ and in C_z; thus, u also accepts in C_z, a contradiction. □

6.3 Separation of PolyLOCAL$^{[n]}$ From LOCAL$^{[n]}$ $\cap$ P Assuming Injective One-Way Functions

In the previous section we showed that LOCAL $\cap$ P $\not\subseteq$ PolyLOCAL, but our proof used the fact that the nodes do not know the size of the graph, and therefore their output when the graph is a short path is the same as their output on a long path, provided their local neighborhood stays the same. We now ask whether the separation continues to hold if nodes do know the size of the network: let LOCAL$^{[n]}$, PolyLOCAL$^{[n]}$ be variants of LOCAL, PolyLOCAL (resp.), where nodes receive the size n of the graph as part of their input. Is it still true that LOCAL$^{[n]}$ $\cap$ P $\not\subseteq$ PolyLOCAL$^{[n]}$?

Perhaps surprisingly, even though we are considering deterministic computation models, the answer turns out to be related to whether or not P = NP: we prove that LOCAL$^{[n]}$$\cap$P $\not\subseteq$ PolyLOCAL$^{[n]}$

implies $\mathsf{P} \neq \mathsf{NP}$, and conversely, under the plausible assumption that injective one-way functions exist,[13] we can still show that $\mathsf{LOCAL}^{[n]} \cap \mathsf{P} \nsubseteq \mathsf{PolyLOCAL}^{[n]}$.

A *one-way function family* is a family $\{f_n\}_{n \in \mathbb{N}}$, where $f_n : \{0,1\}^n \to \{0,1\}^{m(n)}$ for some $m(n) \geq n$, such that given an image $y \in \{0,1\}^{m(n)}$, it is difficult to find a pre-image x such that $f_n(x) = y$ (we refer to [34] for the formal definition, as it is not needed here). It is known that every one-way function has a *hard-core predicate* [35], a Boolean predicate that can be computed in poly-time from $x \in \{0,1\}^n$, but is hard to compute given only $f_n(x)$:

Definition 6.4 (Hard-core predicate). A family of predicates $\{b_n : \{0,1\}^n \to \{0,1\}\}_{n \in \mathbb{N}}$ computable in poly-time is called a *hard-core* of a family of functions $\left\{f_n : \{0,1\}^n \to \{0,1\}^{m(n)}\right\}$ (where $m(n) \geq n$) if for every probabilistic, polynomial-time (PPT) algorithm $\mathcal{A}$, there is a negligible function $\epsilon(\cdot)$ such that for all sufficiently large n we have $\Pr\left[\mathcal{A}(f(z)) = b(z) \mid z \leftarrow U_n\right]$, where U_n denotes the uniform distribution on $\{0,1\}^n$.

Proof of Theorem 1.3, part (3). Fix a family $\mathcal{F} = \{f_n\}_{n \in \mathbb{N}}$ of injective one-way functions, where $f_n : \{0,1\}^n \to \{0,1\}^{m(n)}$ for $m(n) \geq n$, and let $\{b_n\}_{n \in \mathbb{N}}$ be a family of hard-core predicates for $\mathcal{F}$. Consider a distributed language $\mathcal{L}$, which includes all configurations $C_z = (G, x_z)$ for $z \in \{0,1\}^*$, where G is a path $v_0, \ldots, v_{n-1}$ of length $n = |z|$, and the input assignment x_z is given by $x_z(v_0) = (0, b_n(z))$, $x_z(v_{n-1}) = (n-1, z)$, and $x_z(v_i) = (i, f_n(z))$ for every $0 < i < n-1$. We claim that $\mathcal{L} \in \mathsf{LOCAL}^{[n]} \cap \mathsf{P}$, but $\mathcal{L} \notin \mathsf{PolyLOCAL}^{[n]}$.

To decide membership in $\mathcal{L}$ using a local algorithm with unbounded computation time, the first node on the path can simply invert f_n to compute z (recall that f_n is injective), and then use z to compute $b_n(z)$ and compare it against its input. In addition, the other path nodes need to verify that their input is locally consistent with $\mathcal{L}$ (e.g., they are indexed properly).

To decide membership in $\mathcal{L}$ using a polynomial-time centralized algorithm, we can simply read z off of the last node on the path, compute both $f_n(z)$ and $b_n(z)$, and verify that the input is consistent with $f_n(z)$ and $b_n(z)$.

Now suppose for the sake of contradiction that $\mathcal{L} \in \mathsf{PolyLOCAL}^{[n]}$, and let A be a t-local efficient algorithm for $\mathcal{L}$, for some $t = o(n)$. Then for every sufficiently large n, we can break the hard-core predicate b_n using the following adversary $\mathcal{B}$: given input $w = f_n(z)$ for some $z \in \{0,1\}^n$, the adversary constructs the first $2t$ nodes of the configuration $C' = (G, x')$, where G is a path of length n, and x' is identical to x_z, except that the inpu of the first node is $(0,0)$ (since the adversary does not know $b_n(z)$). Note that the adversary does not need to know z for this, because z is only given to the last node on the path, and $t < n$; the adversary only needs to know $f_n(z)$, which it is given. The adversary simulates the first $2t$ nodes in C', and if the first t nodes in the first version accept, it outputs "0"; otherwise it outputs "1".

We claim that our adversary correctly computes $b_n(z)$ for all $z \in \{0,1\}^n$. Given $w \in \{0,1\}^{m(n)}$, there is a unique $z \in \{0,1\}^n$ such that $w = f_n(z)$, because f is injective. If $b_n(z) = 0$, then the t-neighborhood of each of the first t nodes in the configuration C' constructed by the adversary is identical to their view in the "true" configuration $C_z = (G, x_z)$. Since $C_z \in \mathcal{L}$, all nodes must accept, and in particular the first t nodes do; therefore the first t nodes also accept in C'. Now suppose that $b_n(z) = 1$. Then the configuration C', of which the adversary constructed the first t nodes, is not in $\mathcal{L}$; some node must reject in C'. Furthermore, one of the first t nodes must reject in C': suppose they do not, and let v_j be some node that rejects, with $j > t$. In the "true" configuration $C_z = (G, x_z)$, the t-neighborhood of node v_j is the same as in (G_n, x_z'), because the only difference between the two configurations is the input of the first node, which is at distance greater than t

[13] This is stronger than assuming that $\mathsf{P} \neq \mathsf{NP}$, because if $\mathsf{P} = \mathsf{NP}$ then every function is easy to invert.

from v_j. But this means that v_j also rejects in $(G_n, x_z) \in \mathcal{L}$, contradicting the correctness of the local algorithm. We conclude that at least one of the first t nodes must reject, and therefore our adversary outputs “1”.

We have now shown that $\mathcal{B}(f_n(z)) = b_n(z)$ for all sufficiently large n and $z \in \{0,1\}^n$. This implies that b_n is not a hard-core predicate, as it contradicts Definition 6.4. $\square$

REFERENCES

[1] [n. d.]. What are zero-knowledge proofs? https://ethereum.org/en/zero-knowledge-proofs/.

[2] [n. d.]. What are zk-SNARKs? https://z.cash/technology/zksnarks/.

[3] [n. d.]. What Is ZK-STARK? https://starkware.co/stark/.

[4] [n. d.]. zk-SNARKs for the World. https://research.protocol.ai/sites/snarks/.

[5] William Aiello, Sandeep N Bhatt, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. 2000. 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*. 463–474.

[6] Eden Aldema Tshuva and Rotem Oshman. 2022. Brief Announcement: On Polynomial-Time Local Decision. In *PODC*. ACM, 48–50.

[7] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. 1998. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)* 45, 3 (1998), 501–555.

[8] Sanjeev Arora and Shmuel Safra. 1998. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM (JACM)* 45, 1 (1998), 70–122.

[9] László Babai, Lance Fortnow, Leonid A Levin, and Mario Szegedy. 1991. Checking computations in polylogarithmic time. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*. 21–32.

[10] Alkida Balliu, Gianlorenzo D’Angelo, Pierre Fraigniaud, and Dennis Olivetti. 2018. What can be verified locally? *J. Comput. System Sci.* 97 (2018), 106–120.

[11] Mor Baruch, Pierre Fraigniaud, and Boaz Patt-Shamir. 2015. Randomized proof-labeling schemes. In *Symposium on Principles of Distributed Computing (PODC)*. ACM, 315–324.

[12] Mihir Bellare and Phillip Rogaway. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*. 62–73.

[13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2012. 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*. 326–349.

[14] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2013. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 111–120.

[15] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. 2013. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography Conference*. Springer, 315–333.

[16] Sean Bowe, Ariel Gabizon, and Matthew D. Green. 2017. A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. Cryptology ePrint Archive, Paper 2017/602. https://eprint.iacr.org/2017/602 https://eprint.iacr.org/2017/602.

[17] Ran Canetti and Marc Fischlin. 2001. Universally composable commitments. In *Annual International Cryptology Conference*. Springer, 19–40.

[18] Dario Catalano and Dario Fiore. 2013. Vector commitments and their applications. In *International Workshop on Public Key Cryptography*. Springer, 55–72.

[19] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. 2021. Non-interactive batch arguments for np from standard assumptions. In *Annual International Cryptology Conference*. Springer, 394–423.

[20] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. 2021. SNARGs for P from LWE. *IACR Cryptol. ePrint Arch.* 2021 (2021), 808.

[21] Ivan Damgård. 1992. Towards Practical Public Key Systems Secure Against Chosen Ciphertext attacks. In *Advances in Cryptology — CRYPTO ’91*. 445–456.

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

[23] Giovanni Di Crescenzo and Helger Lipmaa. 2008. Succinct NP proofs from an extractability assumption. In *Conference on Computability in Europe*. Springer, 175–185.

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

pdf (2004).
[25] Yuval Emek, Yuval Gil, and Shay Kutten. 2022. Locally Restricted Proof Labeling Schemes. In *36th International Symposium on Distributed Computing (DISC 2022)*, Vol. 246. 20:1–20:22.
[26] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. 1991. Approximating clique is almost NP-complete. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*. IEEE Computer Society, 2–12.
[27] Laurent Feuilloley. 2021. Introduction to local certification. *Discrete Mathematics and Theoretical Computer Science* 23, 3 (2021). https://doi.org/10.46298/dmtcs.6280
[28] Laurent Feuilloley and Pierre Fraigniaud. 2016. Survey of Distributed Decision. (2016). https://hal.archives-ouvertes.fr/hal-01331880
[29] Laurent Feuilloley, Pierre Fraigniaud, Juho Hirvonen, Ami Paz, and Mor Perry. 2021. Redundancy in distributed proofs. *Distributed Comput.* 34, 2 (2021), 113–132.
[30] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*. Springer, 186–194.
[31] Pierre Fraigniaud, Mika Göös, Amos Korman, and Jukka Suomela. 2013. What can be decided locally without identifiers?. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*. ACM, New York, NY, USA, 157–165.
[32] Pierre Fraigniaud, Amos Korman, and David Peleg. 2013. Towards a complexity theory for local distributed computing. *Journal of the ACM (JACM)* 60, 5 (2013), 1–26.
[33] Craig Gentry and Daniel Wichs. 2011. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*. 99–108.
[34] Oded Goldreich. 2001. *The Foundations of Cryptography - Volume 1: Basic Techniques.* Cambridge University Press.
[35] O. Goldreich and L. A. Levin. 1989. A Hard-Core Predicate for All One-Way Functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing (STOC '89)*. 25–32.
[36] Mika Göös and Jukka Suomela. 2016. Locally Checkable Proofs in Distributed Computing. *Theory Comput.* 12, 1 (2016), 1–33.
[37] Jens Groth. 2010. Short pairing-based non-interactive zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 321–340.
[38] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 305–326.
[39] Juris Hartmanis and Richard E Stearns. 1965. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.* 117 (1965), 285–306.
[40] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. 2022. SNARGs for P from Sub-exponential DDH and QR. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 520–549.
[41] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. 2021. 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*. 708–721.
[42] Yael Kalai and Omer Paneth. 2016. Delegating RAM computations. In *Theory of Cryptography Conference*. Springer, 91–118.
[43] Yael Tauman Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. 2022. Boosting Batch Arguments and RAM Delegation. Cryptology ePrint Archive, Paper 2022/1320. https://eprint.iacr.org/2022/1320 https://eprint.iacr.org/2022/1320.
[44] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. 2019. How to delegate computations publicly. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 1115–1124.
[45] Joe Kilian. 1992. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 723–732.
[46] Gillat Kol, Rotem Oshman, and Raghuvansh R. Saxena. 2018. Interactive Distributed Proofs. In *Symposium on Principles of Distributed Computing (PODC)*. 255–264.
[47] Amos Korman, Shay Kutten, and David Peleg. 2005. Proof labeling schemes. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*. 9–18.
[48] Ralph C Merkle. 1989. A certified digital signature. In *Conference on the Theory and Application of Cryptology*. Springer, 218–238.
[49] Silvio Micali. 2000. Computationally sound proofs. *SIAM J. Comput.* 30, 4 (2000), 1253–1298.
[50] Moni Naor, Merav Parter, and Eylon Yogev. 2020. The Power of Distributed Verifiers in Interactive Proofs. In *Symposium on Discrete Algorithms (SODA)*, Shuchi Chawla (Ed.). 1096–115.
[51] Moni Naor and Larry Stockmeyer. 1995. What Can be Computed Locally? *SIAM J. Comput.* 24, 6 (1995), 1259–1277.

[52] Rafail Ostrovsky, Mor Perry, and Will Rosenbaum. 2017. Space-Time Tradeoffs for Distributed Verification. In *International Colloquium on Structural Information and Communication Complexity*. Springer, 53–70.
[53] Omer Paneth and Rafael Pass. 2022. Incrementally Verifiable Computation via Rate-1 Batch Arguments. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1045–1056.
[54] Boaz Patt-Shamir and Mor Perry. 2017. Proof-labeling schemes: broadcast, unicast and in between. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 1–17.

Appendices

A FURTHER BACKGROUND AND DISCUSSION ON THE CRYPTOGRAPHIC PRIMITIVES USED

A.1 Brief Introduction to Computationally Sound Proof Systems

Computationally Sound Proofs. The idea of a proof system that its soundness only holds for adversaries of bounded computational power (also known as an argument) was introduced by Micali in [49], and was based on an earlier interactive protocol of Kilian ([45]) and a variant of the Fiat-Shamir paradigm [30]. Kilian's protocol is based on the following idea. The verifier choose a key to a hash function from a collision-resistant hash family, and send it to the prover. The prover then uses a Merkle Tree [48] induced by the hash function to commit to a polynomial Probabilistically Checkable Proof (PCP) [7–9, 26], using the instance and the witness (that the prover has, and the verifier does not have), and send the commitment to the verifier. Next, the verifier chooses the queries for the PCP, and the prover then sends answers to the queries along with an opening of the Merkle tree, proving that these indeed where the values in the queries' indices in the PCP. TODO: Rotem: capitalized/uncapitalized ro and crs models?

The Random Oracle Model. A Random Oracle is simply a deterministic function that is chosen randomly from all possible functions (from a certain input length to a certain output length), that provide an oracle access: the parties cannot hold a full description of the function, but they can query it and get answers to their queries, in $O(1)$ steps per quary. Though this is not a realistic model, it is very useful in both theory and practice [12]. In the random oracle model, Micali used the paradigm of Fiat and Shamir [30], to lose the interaction in Kilian's protocol, and by that, to obtain a computationally sound proof (which later has been called a SNARG).

SNARGs in the CRS model. Since Micali's seminal work, there have been several attempts [5, 13, 23, 24, 37] to obtain succinct non-interactive arguments (SNARGs) in a more realistic model, but still different from the plain model: the common reference string (CRS) model.[14] In the CRS model, it is assumed that all parties have access to a string generated in a trusted manner. The Common Reference String model is a generalization of the Common *Random* String model, where the reference string is taken from the uniform distribution (in the common *reference* string model, it may be taken from any arbitrary distribution). These two versions are in fact equal in power [17]. The main difference between the ROM and CRS model is that proofs in the ROM are *heuristic*, since the actual protocol instantiation uses a hash function that is blatantly NOT a random oracle. In contrast, proofs in the CRS model have a standard reduction-based proof of security, and so are not heuristic. The main advantage of the CRS model over the Random Oracle model is that security is standard, and doesn't rely on a heuristic belief system that the real protocol that uses a standard hash function is secure. The main disadvantage is that the CRS needs to be generated somehow, and in the absence of a trusted setup, this is not trivial. An example where CRS is used in reality is

[14] Some works, such as [14] did construct SNARGs (in fact, SNARKs) in the plain model.

the Zcash protocol: there, the CRS is generated by a Multi-Party Computation (MPC) protocol [16]. Whenever at least one participant is honest, then the setup is trust-worthy.

SNARGs from knowledge assumptions and SNARKs. To this day (to our knowledge), in every construction of a SNARG in the CRS (or in the plain) model, in order to prove its soundness, some knowledge assumption (or: knowledge *extractability*) was assumed. Knowledge assumptions capture the intuition that any algorithm whose output is related to a certain value that is hard to compute (for instance, a convincing proof, that is related to an NP-witness), must obtain that value along the computation. This assumption is non-falsifiable, meaning, one cannot define a game where at the end of the game, we could efficiently decide whether the assumption was broken or not. This is unlike hardness assumptions, where the design of such a game is very easy to design TODO: word . Under such assumptions, the SNARG candidate becomes stronger: it becomes a SNARG of knowledge — a SNARK; instead of only being sound, we can promise (under the knowledge assumption), that any prover that manages to convince the verifier, knows a witness. This is useful for composing such arguments with other primitives, and in particular, this is useful for our constructions.

In [33], a substantial barrier to proving the soundness of a SNARG for NP under falsifiable assumptions was presented; it was shown that SNARGs[15] cannot be proven secure by a reduction to *any* falsifiable assumption if the reduction is black-box in the adversary's code. Since, many works were either focused on what can be done without knowledge assumptions, which include, for example, SNARGs for deterministic computation [20, 41, 44], batch arguments for NP [19, 22, 40, 43], and incrementally verifiable computation [53], or focused on refining the knowledge assumptions used and generalizing them [13].

A.2 Vector Commitments, Merkle Trees, and Collision Resistant Hash Families

Vector Commitments. Vector Commitments are defined formally in [18], where they are constructed in a way that makes them more succinct than what we required in this work: they show VCs TODO: initials for VCs, SNARGs, etc with commitment and opening length that depends only on the security parameter, and is completely *independent* of the input size.

For our use, a more classic form of VCs suffices; a Merkle Tree [48] induced by a CRH satisfies our completeness, position-binding TODO: capitalize? , and succinctness requirements. TODO: Change hash definition to family?

Definition A.1 (Collision-Resistant Hash (CRH))**.** A hash family (Gen, Hash) is considered *collision-resistant*, if for any efficient adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\cdot)$, such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\ h(x_1) = h(x_2)\ \middle|\ \begin{array}{l} hk \leftarrow Gen(1^n, 1^\lambda) \\ x_1, x_2 \leftarrow \mathcal{A}(hk) \end{array}\right] \leq \epsilon(\lambda)$$

Merkle Trees. We describe Merkle Trees informally. Let $h : \{0,1\}^2k \longrightarrow \{0,1\}^k$ be a function. The root of the Merkle Tree induced by h on an input $x = \in \{0,1\}^{n \times 2k}$, $rt_h(x)$ is computed in $\log k$ iterations as follows. In every iteration, the current sequence (starting when the current sequence is the input) is divided into k-bit blocks, and h is evaluated on every pair of adjacent odd and even indexed blocks, to obtain the new sequence, which its length is half the length of the previous sequence. In the end, we get a value of length $O(k \log n)$. This is the commitment to x. To open a commitment in position i, which is the ith $2k$-bit block of x, the committer only has to show one node in every level of the commitment tree (except for the lowest level, where it has to show two nodes). A verifier can verify that in each level, h was evaluated correctly.

[15] The result of [33] is specific for *adaptive* soundness.

If h is taken from a CRH family, then this scheme is a position-binding VC, since in order to open the same position in two different ways, one must find a collision.

Moreover, for our use in Section 4.3 TODO: verify all labels and references and capitalized Sections , where we also require the VC be *inverse collision-resistant*, it is essential we use Merkle Trees. This is because Merkle Trees are based on a *deterministic* function, and the process of verifying an opening is based on evaluating the same function that is used for the commitment. In order to obtain two different commitments and full openings of them (that is, opening for every position to each commitment), one needs to find two different *outputs* for the same *input* of a deterministic function, which is impossible. Note that in the definition of the inverse collision-resistance property, we allowed this to happen with a negligible probability, because nothing more is necessary for our proof, but in fact, when instantiating the VC by a Merkle Tree, we can promise that such "inverse collisions" are never found.

A.3 SNARKs for NP

There is an abundance of constructions of SNARKs. In this work, we refer to the construction in [15], but it could be replaced by any *publically verifiable* SNARK construction, with or without pre-processing. We refer to [14, 15] for more details on public vs. private verifiability and preprocessing in SNARKs. From the construction in [15], we get SNARKs for NP from the knowledge-of-exponent assumption in bilinear groups. TODO: Copy it?

A.4 RAM SNARGs for Deterministic Computation

For a polynomial-time Turing machine M, we would like to have a way of verifying that the execution of M on input x was executed correctly, and in particular, the output is correct. We would like to do that more efficiently than simulating the computation, and more importantly, without having access to the entire input. RAM Delegation allows us to do so. In general, a RAM Machine is a deterministic Turing machine that has random access to memory that is much longer (mostly, exponentially longer) than its local state, and a RAM SNARG is a SNARG that proves that a RAM machine indeed outputs a certain output, without having the verifier simulate the entire execution (that requires access to a long memory). A RAM SNARG is associated with a *digest* algorithm, that processes the long input into a much shorter string that the verifier can read. In [43], this definition is extended to *Flexible SNARGs for RAM*, which is a RAM SNARG where the digest can be implemented by any hash family, and the SNARG is sound if that hash family has local openings.

Let M be a RAM machine. A flexible RAM SNARG for M is associated with a hash family with local opening[16]

$$\mathsf{HT} = \mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Open}, \mathsf{HT.Ver}$$

and consists of the following algorithms.[17]

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

[16]For the use in [43], and for our use, a succinct vector commitment satisfies all the required properties of the hash with local openings, where $\mathsf{HT.Gen} = \mathsf{VC.Gen}$, $\mathsf{HT.Hash} = \mathsf{VC.Com}$, $\mathsf{HT.Open} = \mathsf{VC.Open}$, $\mathsf{HT.Ver} = \mathsf{VC.Ver}$

[17]In [43], they include in the SNARG definition also the digestion algorithm, that uses a key generated by Gen and applies $\mathsf{HT.Hash}(x)$ to obtain d. Since this is fully defined by the rest of the algorithms mentioned here, we omit it from the definition.

$\mathcal{P}(\mathsf{crs}, x) \to b, \pi$: [18] takes a common reference string crs obtained from $\mathsf{Gen}(1^\lambda, T)$, an instance $x \in \{0,1\}^\ell$, and outputs a bit $b = M(x)$ and a proof π.

$\mathcal{V}(\mathsf{crs}, d, b, \pi) \to \{0,1\}$: takes a common reference string crs, a digest if memory d, an output bit b, and proof π, and outputs an acceptance bit.

Definition A.2 (Flexible RAM SNARGs). Eden: I might have over-copied here Let M be a RAM machine. A Flexible RAM SNARG for M associated with a hash family with local opening $\mathsf{HT} = (\mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Open}, \mathsf{HT.Ver})$ satisfies the following properties.

Completeness. There exists a negligible function $\epsilon(\cdot)$, such that for every $\lambda, n \in \mathbb{N}$ such that $n \le T(n) \le 2^\lambda$, and every $x \in \{0,1\}^n$ such that M on x halts after $T(n)$ steps,

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

Soundness. [19] For any efficient adversarial prover $\mathcal{P}^*$ and a polynomial $T = T(\lambda)$, there exists a negligible function $\epsilon(\cdot)$, such that

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

Succinctness. The length of the proof output of $\mathcal{P}$ is $\mathrm{poly}(\lambda, \log n, \log T)$.

Verifier Efficiency. $\mathcal{V}$ runs in time $\mathrm{poly}(\lambda, |\pi|) = \mathrm{poly}(\lambda, \log n, \log T)$

B PSEUDOCODE AND CORRECTNESS PROOFS

B.1 Succinct Distributed Arguments for NP from SNARKS

Detailed construction. Let $\mathcal{L}$ be an NP-language on graphs, and let $V_\mathcal{L}$ be a polynomial-time Turing machine such that:

$$G \in \mathcal{L} \Leftrightarrow \exists w \in \{0,1\}^{\mathrm{poly}(|G|)} \text{ such that } V_\mathcal{L}(L(G), w) = 1.$$

Fix a vector commitment $(\mathsf{VC.Gen}, \mathsf{VC.Com}, \mathsf{VC.Open}, \mathsf{VC}.\mathcal{V})$. We define the following NP language.[20][21]

$$\mathcal{L}^{\mathrm{com}} = \left\{ (c, 1^n, \mathsf{crs}) \middle| \exists L, w : \begin{array}{l} L = (L_1, \ldots, L_m) \text{ is an adjacency list} \\ \exists \mathsf{aux} : \\ \mathsf{VC.Com}(\mathsf{crs}, L_1, \ldots, L_m) = (c, \mathsf{aux}) \\ \wedge\ V_\mathcal{L}(L, w) = 1 \\ \wedge \text{the graph represented by } L \text{ is symmetric and connected} \end{array} \right\}$$

Now fix a SNARK $(\mathsf{SNARK.Gen}, \mathsf{SNARK}.\mathcal{P}, \mathsf{SNARK}.\mathcal{V}, \mathsf{SNARK}.\mathcal{E})$ for $\mathcal{L}^{\mathrm{com}}$. The succinct distributed argument $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ for $\mathcal{L}$ is defined as follows.

Claim B.1. $\mathsf{Gen}, \mathcal{P}, \mathcal{V}$ *is a succinct distributed argument.*

[18] In [43], the input x is divided into a short explicit input x_{exp} that the verifier has, and a long input the verifier doesn't have x_{imp}. Since it is not required for the SNARG's properties that x_{exp} be non-empty, and we only use the node's input outside the SNARG itself, we omit x_{exp} from the definition.

[19] In [43], it is shown that this soundness notion can be replaced by a different one, called *Partial Input Soundness*. We do not require it.

[20] We give here 1^n as part of the input since for a succinct commitment scheme, c should be much shorter than 1^n, whereas the witness size is bounded from below by n^2.

[21] aux is existentially quantified, but a polynomial-time verifying machine for $\mathcal{L}^{\mathrm{com}}$ would not need aux as part of the witness, since VC.Com is polynomial-time.

Fig. B.1a. The Setup Procedure of Section 4.2: $\mathsf{Gen}(1^\lambda, n)$

1: $\mathsf{crs}_{\mathrm{VC}} \leftarrow \mathsf{VC.Gen}(1^\lambda, n)$
2: $\mathsf{crs}_{\mathrm{SNARK}} \leftarrow \mathsf{SNARK.Gen}(1^\lambda, 1^n)$
3: Output: $\mathsf{crs} = (\mathsf{crs}_{\mathrm{VC}}, \mathsf{crs}_{\mathrm{SNARK}})$

Fig. B.1b. The Prover of Section 4.2 : $\mathcal{P}(\mathsf{crs}, G, w)$

1: Parse $\mathsf{crs} = (\mathsf{crs}_{\mathrm{VC}}, \mathsf{crs}_{\mathrm{SNARK}})$
2: Let $v_1, \ldots, v_n$ be the nodes of G, sorted by UID
3: Let $L \leftarrow (L_1, \ldots, L_n)$, where $L_i = (v_i, N(v_i))$ for each $i \in [n]$
4: $(c, \mathsf{aux}) \leftarrow \mathsf{VC.Com}(\mathsf{crs}_{\mathrm{VC}}, L_1, \ldots, L_n)$
5: Compute for every i: $\Lambda_i \leftarrow \mathsf{VC.Open}(\mathsf{crs}_{\mathrm{VC}}, L_i, i, \mathsf{aux})$
6: Compute $\pi_{\mathrm{SNARK}} \leftarrow \mathsf{SNARK}.\mathcal{P}(\mathsf{crs}_{\mathrm{SNARK}}, c, (L, w))$
7: Output $\{\pi(v_i)\}_{v_i \in V(G)}$, where for every $v_i \in V(G)$: $\pi(v_i) = (c, i, \Lambda_i, \pi_{\mathrm{SNARK}})$

Fig. B.1c. The Verifier of Section 4.2 : $\mathcal{V}(\mathsf{crs}, N(v), v, \pi(v))$

1: Parse $\mathsf{crs} = (\mathsf{crs}_{\mathrm{VC}}, \mathsf{crs}_{\mathrm{SNARK}})$
2: Parse $\pi = (c, i, \Lambda_i, \pi_{\mathrm{SNARK}})$
3: Verify that for every neighbor $u \in N(v)$: $c(u) = c$ and $\pi_{\mathrm{SNARK}}(u) = \pi_{\mathrm{SNARK}}$ (otherwise output 0)
4: Output 1 if the following holds:
   - $\mathsf{VC.Ver}(\mathsf{crs}_{\mathrm{VC}}, c, (v, N(v)), i, \Lambda_i) = 1$
   - $\mathsf{SNARK}.\mathcal{V}(\mathsf{crs}_{\mathrm{SNARK}}, c, \pi) = 1$

Proof. Perfect completeness and succinctness follow immediately from the perfect completeness and the succinctness of the VC and the SNARK. We now prove soundness.

Assume towards contradiction that there exists an efficient prover $\mathcal{P}^*$ and a non-negligible function $\alpha(\cdot)$, such that the following holds with probability at least $\alpha(\lambda)$.

$$\Pr\left[\begin{array}{l} G \notin \mathcal{L} \\ \wedge\ \forall v \in V(G) : \mathcal{V}(\mathsf{crs}, N_G(v), v, \pi_v) = 1 \end{array} \middle| \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ (G, \{\pi_v\}_{v \in V(G)}) \leftarrow \mathcal{P}^*(\mathsf{crs}, 1^\lambda, 1^n) \end{array}\right]. \tag{B.1}$$

First, note that since all nodes verify that they agree with their neighbors on the vector commitment c and SNARK proof; if all nodes accept, then the prover gave the same values to all nodes. We assume this from now on.

We use $\mathcal{P}^*$ to construct an efficient adversary $\mathcal{A}$ that breaks either the SNARK or the VC. The adversary $\mathcal{A}$ proceeds as follows:

- Given $\mathsf{crs}, 1^\lambda, n$, it first uses $\mathcal{P}^*(\mathsf{crs}, 1^\lambda, n)$ to obtain a graph G and certificates $\{\pi(v)\}_{v \in V(G)}$.
- From $\pi(v)$ (for an arbitrary v, since they all agree), the adversary extracts the vector commitment c and SNARK proof π_{SNARK}.
- The adversary extracts the NP-witness $(L^*, w) \leftarrow \mathsf{SNARK}.\mathcal{E}_{\mathcal{P}^*}(\mathsf{crs}, c, \pi_{\mathrm{SNARK}})$ from the SNARK proof.
- If (L^*, w) is not a valid witness for the membership $(c, 1^n, \mathsf{crs}) \in \mathcal{L}^{\mathrm{com}}$, then the adversary has broken adaptive proof of knowledge property, as $\mathsf{SNARK}.\mathcal{V}(\mathsf{crs}_{\mathrm{SNARK}}, c, \pi) = 1$.

- Otherwise, L^* is an adjacency list, $L = (L_1, \dots, L_m)$ (for some m which is not necessarily equal to n), such that $\mathsf{VC.Com}(\mathsf{crs}, L_1, \dots, L_m) = (c, \mathsf{aux})$, $V_{\mathcal{L}}(L, w) = 1$, and the graph G' represented by L is symmetric and connected. In particular, since $v_{\mathcal{L}}(L, w) = 1$, we have $G' \in \mathcal{L}$. Thus, whenever $G \notin \mathcal{L}$, we must have $G' \neq G$, in other words, L is not the adjacency list of G.
  For each $v \in V$, let $i(v)$ be the index appearing in v's certificate, $\pi(v) = (c, i(v), \Lambda(v), \pi_{\text{SNARK}})$. There are two cases:
  - For some node $v \in V(G)$ we have $L_{i(v)} \neq (v, N(v))$. But node v verifies that entry $i(v)$ of c opens to its true neighborhood, i.e., $\mathsf{VC.Ver}(\mathsf{crs}_{\mathsf{VC}}, c, (v, N(v)), i(v), \Lambda) = 1$, and so the adversary has broken the binding property of the vector commitment.
  - For every $v \in V(G)$ we have $L_{i(v)} = (v, N(v))$. This implies that $i(v) \neq i(v')$ for every $v \neq v' \in V(G)$, and therefore $\left|\{i(v)\}_{v \in V(G)}\right| = n$. We claim that in this case $G = G'$, that is, L is the true adjacency list of G, contradicting our assumption that it is not.
    If $|L| = |V(G)|$, then $\left|\{i(v)\}_{v \in V(G)}\right| = n$ and $L_{i(v)} = (v, N(v))$ for every $v \in V$, the list does match G. Thus, assume that $|L| \neq |V(G)|$.
    It must be that $|L| > |V(G)|$, as we already said that $\left|\{i(v)\}_{v \in V(G)}\right| = n$. Thus, there is some entry (u, N) in L, such that u is not a node of G. Since G' is connected, there is an edge $\{w, w'\}$ in the cut between $V(G') \setminus V(G)$ and $V(G)$, with $w \in V(G)$ and $w' \notin V(G)$. We know that $L_{i(w)} = (w, N(w))$, but $w' \notin N(w)$ (since $w' \notin V(G)$); this is a contradiction, since we assumed that L represents G'.

□

B.2 Succinct Distributed Arguments for P from RAM SNARGs

In this section we use Flexible RAM SNARGs to construct a succinct distributed argument for P. Such RAM SNARGs are defined w.r.t some hash family with local opening. For our use, that hash family will be a succinct vector commitment, which already satisfies all of the hash family with local openings requirements. For our use, we also require that the vector commitment has the following property.[22]

Definition B.2 (Inverse Collision-Resistance)**.** A VC $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open}, \mathcal{V})$ is *Inverse Collision-Resistant* if for any efficient adversary $\mathcal{A}$, there exists a negligible function $\epsilon(\cdot)$, such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \forall i : \mathcal{V}(crs, C^*, m, i) = 1 \\ \wedge C^* \neq C \end{array} \middle| \begin{array}{l} crs \leftarrow Gen(1^\lambda, q) \\ C^*, \{(m_i, \lambda_i)\}_{i \in [q]} = \mathcal{A}(crs) \\ C \leftarrow \mathsf{Com}(crs, m_1, \dots, m_q) \end{array}\right] \leq \epsilon(\lambda)$$

Theorem B.3. *Let $\mathcal{L}$ be a graph language, such that $\mathcal{L} \in$ P. Assuming Flexible RAM SNARGs for* P *and Inverse Collision-Resistant VC exist, there is a succinct distributed argument for $\mathcal{L}$.*

Let $\mathcal{L}$ be a language on graphs that is decidable in polynomial time, given the entire graph as input, and let $M_{\mathcal{L}}$ be the Turing machine that decides it:

$$G \in \mathcal{L} \Leftrightarrow M_{\mathcal{L}}(L(G)) = 1$$

Fix a vector commitment $(\mathsf{VC.Gen}, \mathsf{VC.Com}, \mathsf{VC.Open}, \mathsf{VC.Ver})$ that is inverse collision-resistant and a RAM SNARG $(\mathsf{SNARG.Gen}, \mathsf{SNARG}.\mathcal{P}, \mathsf{SNARG}.\mathcal{V})$ for $M_{\mathcal{L}}$, corresponding to the vector commitment as the hash with local opening. The succinct distributed argument for $\mathcal{L}$, $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$, is defined as follows.

We now prove the following statement, from which Theorem B.3 follows.

[22] A succinct, inverse collision-resistant VC can be instanciated by a Merkle Tree [48].

Fig. B.2a. The Setup Procedure of Section 4.3: $\mathsf{Gen}(1^\lambda, n)$

Compute: $\mathsf{crs}_{\mathsf{VC}} = \mathsf{VC.Gen}(1^\lambda, n)$
Compute: $\mathsf{crs}_{\mathsf{SNARG}} = \mathsf{SNARG.Gen}(1^\lambda, 1^n)$
Output: $\mathsf{crs} = (\mathsf{crs}_{\mathsf{VC}}, \mathsf{crs}_{\mathsf{SNARG}})$

Fig. B.2b. The Prover of Section 4.3 : $\mathcal{P}(\mathsf{crs}, G)$

Parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{VC}}, \mathsf{crs}_{\mathsf{SNARG}})$
Represent G as an adjacency list $L = L_1, \dots, L_n$.
for each $i \in [n]$ **do**
    Set $v \in V(G)$ to be the node with the i smallest identifier.
    Set $i_v = i$, $L_{i_v} = (v, N_G(v_i))$
end for
Compute $(d, \mathsf{aux}) = \mathsf{VC.Com}(\mathsf{crs}_{\mathsf{VC}}, L_1, \dots, L_n)$
Compute for every i: $\Lambda_i = \mathsf{VC.Open}(\mathsf{crs}_{\mathsf{VC}}, L_i, i, aux)$
Compute $\pi_{\mathsf{SNARG}}, b = \mathsf{SNARG}.\mathcal{P}(\mathsf{crs}_{\mathsf{SNARG}}, L)$
Output $\{\pi_v\}_{v \in V(G)}$, where for every $v \in V(G)$: $\pi_v = (d, i_v, \Lambda_{i_v}, \pi_{\mathsf{SNARG}})$

Fig. B.2c. The Verifier of Section 4.3 : $\mathcal{V}(\mathsf{crs}, N, v, \pi)$

1: Parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{VC}}, \mathsf{crs}_{\mathsf{SNARG}})$
2: Parse $\pi = (c, i, \Lambda_i, \pi_{\mathsf{SNARG}})$
3: Verify that $\forall u \in N$, $d(u) = d$ and $\pi_{\mathsf{SNARK}}(u) = \pi_{\mathsf{SNARK}}$ (otherwise output 0)
4: Output 1 if the following holds:
   - $\mathsf{VC.Ver}(\mathsf{crs}_{\mathsf{VC}}, d, (v, N), i, \Lambda_i) = 1$
   - $\mathsf{SNARG}.\mathcal{V}(\mathsf{crs}_{\mathsf{SNARG}}, d, \pi) = 1$

Claim B.4. $\mathsf{Gen}, \mathcal{P}, \mathcal{V}$ *is a succinct distributed argument.*

Proof. Completeness and succinctness follow immediately from the completeness and the succinctness of the VC and the SNARG. We proceed to the proof of soundness. Assume towards contradiction that there exists an efficient prover $\mathcal{P}^*$ and a non-negligible function $\alpha(\cdot)$ such that:

$$\Pr\left[\begin{array}{l} G \notin \mathcal{L} \\ \wedge\ \forall v \in V(G) : \mathcal{V}(crs, N_G(v), v, \pi_v) = 1 \end{array} \middle| \begin{array}{l} crs \leftarrow \mathsf{Gen}(1^\lambda, n) \\ G, \{\pi_v\}_{v \in V(G)} \leftarrow \mathcal{P}^*(crs, 1^\lambda, 1^n) \end{array}\right] \geq \alpha(\lambda) \quad \text{(B.2)}$$

First, note that since all nodes verify the consistency of d and π_{SNARG} with their neighbors,(in Step 3), if all nodes accept, then the prover gave the same commitment (digest), and the same SNARG proof π_{SNARG} to all of the nodes.

We use $\mathcal{P}^*$ to construct an efficient adversary $\mathcal{A}$ that breaks either one of the properties of the SNARG, or one of the properties of the VC. Let $G, \{\pi_v\}_{v \in V(G)}$ be the graph and the proofs outputted from $\mathcal{P}^*(\mathsf{crs}, 1^\lambda, n)$, and let $\mathsf{crs}_{\mathsf{VC}}, \mathsf{crs}_{\mathsf{SNARG}}$ be the parsed reference strings from crs. On input $\mathsf{crs}, 1^\lambda, n$, $\mathcal{A}$ outputs d, d^*, π_0, π_1, where:

- d^* is the commitment in π_v that is consistent across all $v \in V(G)$.
- $d, \mathsf{aux} = \mathsf{VC.Com}(\mathsf{crs}_{\mathsf{VC}}, L(G))$. ($\mathcal{A}$ doesn't output aux but we refer it later)
- $\pi_0 = \mathsf{SNARG}.\mathcal{P}(\mathsf{crs}_{\mathsf{SNARG}}, L(G))$.

For every $v \in V(G)$, let $\Lambda_{i_v} = \text{VC.Open}(\text{crs}_{\text{VC}}, L_{i_v}, i_v, \text{aux})$. We define the following events.

- Let *VCCompBreak* be the event that $d \neq d^*$ and there exists some $v \in V(G)$ such that VC.Ver rejects $(\text{crs}_{\text{VC}}, d, (v, N_G(v)), i\Lambda_{i_v})$.
- Let *VCICRBreak* be the event that $d \neq d^*$ VC.Ver accepts $(\text{crs}_{\text{VC}}, d, (v, N_G(v)), i\lambda_{i_v})$.
- Let *SNARGCompBreak* be the event that SNARG.$\mathcal{V}$ rejects $(\text{crs}_{\text{SNARG}}, d, 0, \pi_0)$.
- Let *SNARGSoundBreak* be the event that $d = d^*$ and SNARG.$\mathcal{V}$ accepts $(\text{crs}_{\text{SNARG}}, d, 1, \pi_1)$ and $(\text{crs}_{\text{SNARG}}, d, 0, \pi_0)$.

Whenever the event in B.2 occurs, one of the following must hold:

- $d \neq d^*$, so either *VCCompBreak* or *VCICRBreak* occur.
- $d = d^*$, so either *SNARGCompBreak* or *SNARGSoundBreak* occur.

Since the event in B.2 happens with probability at least $\alpha(\lambda)$, one of the events *VCCompBreak*, *VCICRBreak*, *SNARGSoundBreak*, *SNARGCompBreak* happens with probability at least $\alpha(\lambda)/4$, which is also a non-negligible function of λ, and so, $\mathcal{A}$ breaks at least one of the following: the VC's completeness property, the VC's position-binding property, the SNARG's completeness property, the SNARG's soundness property. Completeness, succinctness, and verifier efficiency follow naturally from the primitives' properties. □

B.3 Certifying Executions of Computationally-Efficient Distributed Algorithms

In the general case where we have inputs $x : V(G) \to \mathcal{X}$ and outputs $y : V(G) \to \mathcal{Y}$, the consistency of the local computation at a specific node is captured by the language $\mathcal{D}$, which consists of all triplets $(\text{hk}, I(v), W(v))$ such that:

- hk is a hash key obtained by calling SCRH.Gen,
- $I(v) = (v, x(v), N(v), y(v), s_{in}(v), s_{out}(v))$, where $v \in \mathcal{U}$ is the UID of a node, $x(v) \in \mathcal{X}$ is the input of the node, $N(v) \in \mathcal{U}^*$ is the neighborhood of the node, $y(v) \in \mathcal{Y}$ is an output value, and $s_{in}(v), s_{out}(v)$ are hash sums;
- $W(v) = (msgout(v), msgin(v))$ consists of two sets of messages;
- $(\text{hk}, I(v), W(v)) \in \mathcal{D}$ iff when the distributed algorithm D is executed at a node with UID v, input $x(v)$ and neighbors $N(v)$, and the incoming messages at node v are $msgin(v)$, the node produces output $y(v)$ and sends the messages $msgout(v)$, and furthermore,

$$s_{in} = \sum_{msg \in msgin} \text{SCRH.Hash}(\text{hk}, msg), \qquad s_{out} = \sum_{msg \in msgout} \text{SCRH.Hash}(\text{hk}, msg). \tag{B.3}$$

Let $G = (V, E)$ be a graph of size n, and let $\ell = \text{poly}(n)$ be the maximum encoding length of a message sent by D in graphs of size n.[23]

Fig. B.3a. The Setup Procedure of Section 5: $\text{Gen}(1^\lambda, n)$

1: $\text{hk} \leftarrow \text{SCRH.Gen}(1^\lambda, \ell(n))$
2: $\text{crs}_{\text{SNARK}} \leftarrow \text{SNARK.Gen}(1^\lambda, n')$ // n' is the encoding length of $(\text{hk}, I(v))$ for a single vertex v in graphs of size n
3: output $(\text{hk}, \text{crs}_{\text{SNARK}})$

Claim B.5. Gen, $\mathcal{P}, \mathcal{V}$ *is a succinct distributed argument for* $\mathcal{L}_D$

[23]Recall that the encoding of a message consists of the round number, the edge on which it is sent, and the message contents; for an algorithm that runs on polynomial rounds and sends polynomially-long messages, the encoding of a message is polynomial in n.

Fig. B.3b. The Distributed Prover of Section 5: $\mathcal{P}(\mathsf{crs}, (G, x, y))$

The prover is the following distributed algorithm, executed jointly by the nodes of G.

Parse crs = $(\mathsf{hk}, \mathsf{crs}_{\text{SNARK}})$ // All nodes must know the CRS
Compute the messages $\{msgout(v), msgin(v)\}_{v \in V(G)}$ sent and received during the execution of D at each $v \in V(G)$ // This can be done while D is executing
for each $v \in V(G)$ **do** // In parallel
  $s_{out}(v) \leftarrow \sum_{msg \in msgout(v)} \mathsf{SCRH.Hash}(\mathsf{hk}, msg)$
  $s_{in}(v) \leftarrow \sum_{msg \in msgin(v)} \mathsf{SCRH.Hash}(\mathsf{hk}, msg)$
  $I(v) \leftarrow (v, x(v), N_G(v), y(v), s_{in}(v), s_{out}(v))$
  $W(v) \leftarrow (msgout(v), msgin(v))$
  $\pi_{\text{SNARK}}(v) \leftarrow \mathcal{P}(crs, (\mathsf{hk}, I(v)), W(v))$
end for
Compute a spanning tree T of G, of height $\leq 2\text{diam}(G)$
$r \leftarrow$ the root of T
$d(r) \leftarrow 0, p(r) \leftarrow \perp$
By broadcast down the tree, for each node v with parent u, set $p(v) \leftarrow u, d(v) \leftarrow d(u) + 1$
By convergecast up the tree, for each node v set $S_{out}(v) \leftarrow s_{out}(v) + \sum_{u \in \text{children}(v)} S_{out}(u)$, $S_{in}(v) \leftarrow s_{in}(v) + \sum_{u \in \text{children}(v)} S_{in}(u)$
Output $\pi(v) = (p(v), d(v), r, s_{out}(v), s_{in}(v), S_{out}(v), S_{in}(v), \pi_{\text{SNARK}}(v))$ at each node v

Fig. B.3c. The Verifier of Section 5

The verifier at node v, with setup crs = $(\mathsf{hk}, \mathsf{crs}_{\text{SNARK}})$ and certificate

$$\pi(v) = (p(v), d(v), r(v), s_{out}(v), s_{in}(v), S_{out}(v), S_{in}(v), \pi_{\text{SNARK}}(v)),$$

verifies the following conditions:

$r(v) = r(u)$ for every $u \in N(v)$
if $r(v) = v$ **then**
  $d(v) = 0$
  $S_{out}(v) = S_{in}(v)$
else
  $p(v) \in N(v)$
  $d(v) = d(p(v)) + 1$
end if
$S_{out}(v) = s_{out}(v) + \sum_{u \in N(v): p(u) = v} S_{out}(u)$
$S_{in}(v) = s_{in}(v) + \sum_{u \in N(v): p(u) = v} S_{in}(u)$
$\mathsf{SNARK}.\mathcal{V}(\mathsf{crs}_{\text{SNARK}}, (\mathsf{hk}, (v, x(v), N(v), y(v), s_{in}(v), s_{out}(v))), \pi_{\text{SNARK}}(v)) = 1.$

Proof. Suppose for the sake of contradiction that there is an efficient adversary $\mathcal{A}$ such that for some non-negligible function $\alpha(\cdot)$ and for all sufficiently large n, we have

$$\Pr\left[\begin{array}{l|l} G \notin \mathcal{L} & \\ \wedge\ \forall v \in V(G) : & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n) \\ \mathcal{V}(\mathsf{crs}, v, (x(v), y(v)), & (G, \{\pi(v)\}_{v \in V(G)}) \leftarrow \mathcal{P}^*(\mathsf{crs}, 1^\lambda, 1^n) \\ \qquad N(v), \pi(v), \pi(N(v))) = 1 & \end{array}\right] \geq \alpha(\lambda).$$

We use $\mathcal{A}^*$ to construct an efficient adversary $\mathcal{A}'$ that breaks either the SNARK or the hash. $\mathcal{A}'$ works as follows: After obtaining $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, n)$, where $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_{\text{SNARK}})$, we execute $\mathcal{A}^*$ to obtain a graph G, along with certificates $\{\pi(v)\}_{v \in V(G)}$ for the nodes of V. From the certificates, $\mathcal{A}'$ extracts:

- A collection of hash-sums $\{s_{in}(v), s_{out}(v)\}_{v \in V}$.
  As usual, let us denote $I(v) = (v, x(v), N(v), y(v), s_{in}(v), s_{out}(v))$ for each $v \in V(G)$.
- A collection of SNARK proofs $\{\pi_{\text{SNARK}}(v)\}_{v \in V}$.
- Recall that the SNARK proof $\pi_{\text{SNARK}}(v)$ is for the statement "$\exists W(v) : (\mathsf{hk}, I(v), W(v)) \in \mathcal{D}$". Using the extraction algorithm $\mathsf{SNARK}.\mathcal{E}\mathcal{A}^*(\mathsf{crs}_{\text{SNARK}}, (\mathsf{hk}, I(v))$ of the SNARK, we extract a witness for each node v in the form of message sets, $W(v) = (\mathit{msgin}(v), \mathit{msgout}(v))$.

Let ST be the event that the verificiation of the spanning tree distances, root and partial sums succeeds at all nodes. An easy induction on the height of the tree shows that when this event occurs we have

$$S_{in}(r) = \sum_{v \in V} s_{in}(v), \qquad S_{out}(r) = \sum_{v \in V} s_{out}(v), \tag{B.4}$$

where $r \in V$ is the the unique node specified in all the certificates as the root of the spanning tree. From now on, we condition on this event, and refer to r as "the root". Recall also that as part of the partial sum verification, the root verifies that

$$S_{in}(r) = S_{out}(r). \tag{B.5}$$

We claim that if $G \notin \mathcal{L}$, whenever all nodes accept, one of the following events must occur:

- *HashCheat*: the two collections of messages given by $\mathit{Out} = \{\mathit{msg} \in \mathit{msgout}(v) : v \in V\}$ and by $\mathit{In} = \{\mathit{msg} \in \mathit{msgin}(v) : v \in V\}$ are not equal, but they break the SCRH property by having $\sum_{m \in \mathit{Out}} \mathsf{SCRH.Hash}(\mathsf{hk}, m) = \sum_{m' \in \mathit{In}} \mathsf{SCRH.Hash}(\mathsf{hk}, m')$; or
- *SnarkCheat*: at some node v we have $\mathsf{SNARK}.\mathcal{V}(\mathsf{crs}_{\text{SNARK}}, (\mathsf{hk}, I(v)), \pi_{\text{SNARK}}(v)) = 1$ but M rejects $(\mathsf{hk}, I(v), W(v))$, violating the argument of knowledge property of the SNARK.

To see this, suppose that all nodes accept, but neither event occurs. We divide into cases:

- Suppose that M accepts $(\mathsf{hk}, I(v), W(v))$ at all $v \in V(G)$ (i.e., *SnarkCheat* has not occurred), and the message collections *Out* and *In* are not equal. Recall that M verifies (B.3), and that the event ST on which we are conditioning implies (B.4), (B.5). But (B.3) and (B.4) together imply that

  $$S_{in}(r) = \sum_{m \in \mathit{In}} \mathsf{SCRH.Hash}(\mathsf{hk}, m), \qquad S_{out}(r) = \sum_{m \in \mathit{Out}} \mathsf{SCRH.Hash}(\mathsf{hk}, m),$$

  and (B.5) implies that the two hash-sums are equal, $S_{in}(r) = S_{out}(r)$; thus, *HashCheat* has occurred.
- Now suppose that the message collections *Out* and *In* *are* equal, but $G \notin \mathcal{L}_D$. Then there is some node $v \in V$ such that $y(v)$ is not the output of the distributed algorithm D at node v when executed in G.
  Let us say that a message $\mathit{msg} = (r, \{u, w\}, m)$ is *correct* if it would be sent in the execution of D in G. There are two cases:
  - All messages in $W(v) = (\mathit{msgin}(v), \mathit{msgout}(v))$ are correct. In this case, $M(\mathsf{hk}, I(v), W(v))$ rejects, as it is not true that v produces the output $y(v)$ (which appears in $I(v)$) when D is executed in G.
  - Some message in $W(v) = (\mathit{msgin}(v), \mathit{msgout}(v))$ is not correct. In this case, let t be the first round such that $\mathit{In} = \mathit{Out}$ includes some incorrect round-t message $\mathit{msg} = (t, \{u, v\}, m)$, and let u be the node such that $\mathit{msg} \in \mathit{msgout}(u)$. Then M rejects

(hk, $I(u)$, $W(u)$): at round t, if fed the messages in $msgin(u)$ up to round $t-1$ (which are all correct), it is not true that u sends msg (as this message is incorrect).

In both cases, M rejects (hk, $I(u)$, $W(u)$), but we assumed that all nodes accept, and therefore $SnarkCheat(v)$ has occurred.

We conclude that one of the events *HashCheat*, *SnarkCheat* occurs with probability at least $\alpha(\lambda)/2$, as $\mathcal{A}$ generates $G \notin \mathcal{L}$ and certificates that all nodes accept with probability at least $\alpha(\lambda)$, and whenever this occurs, at least one of the events *HashCheat*, *SnarkCheat* occurs.

Since $\alpha(\cdot)$ is non-negligible, $\alpha(\cdot)/2$ is also non-negligible. If *HashCheat* occurs with probability at least $\alpha(\lambda)/2$, this violates the SCRH property of the hash function; and if *SnarkCheat* occurs with probability at least $\alpha(\lambda)/2$, this violates the argument of knowledge property of the SNARK. □

Theorem B.6. *TODO: fill in — this is a placeholder for the full theorem about the distributed prover*

C CONCRETE INSTANTIATIONS FOR THE CONSTRUCTIONS

We discuss several possibilities for instantiating the primitives used in our constructions. Throughout the first part of this work (Section 4 and Section 5), we use several cryptographic primitives, some of them quite strong. Here we refine them and show from which concrete assumptions each of them can be constructed. TODO: CRH is not so concrete

Theorem C.1. *Let $\mathcal{L}$ be a graph language, such that $\mathcal{L} \in \mathsf{NP}$. Assuming Collision Resistant Hash Families,*

(1) *There is a succinct* interactive *distributed argument with 4 rounds of communication for $\mathcal{L}$.*
(2) *In the Random Oracle model, there is a succinct non-interactive distributed argument for $\mathcal{L}$.*
(3) *In the Common Reference String model, assuming* Knowledge-of-Exponent in Bilinear Groups, *there is a succinct non-interactive distributed argument for $\mathcal{L}$.*

From Theorem 1.1 and [48], we get that part (1) follows from [45], part (2) follows from [49] and part (3) follows from [15].

Theorem C.2. *Let $\mathcal{L}$ be a graph language, such that $\mathcal{L} \in \mathsf{P}$. Assuming Collision Resistant Hash Families, there is a succinct distributed argument for $\mathcal{L}$, assuming either*

(1) *The $O(1)-\mathsf{LIN}$ assumption on a pair of cryptographic groups with efficient bilinear map, or*
(2) *A combination of the sub-exponential* DDH *assumption and the* QR *assumption.*

As shown in [43], Flexible RAM SNARGs exist under any of these assumptions. Together with theorem B.3, this implies corollary **??**.

Theorem C.3. *Let $\mathcal{D}$ be a distributed algorithm that runs in $T = \mathrm{poly}(n)$ rounds and sends messages of length* $\mathrm{poly}(n)$. *Assuming Sum-Collision-Resistant Hash Families,*

(1) *There is a succinct* interactive *distributed argument with 4 rounds of communication for $\mathcal{L}$.*
(2) *In the Random Oracle model, there is a succinct non-interactive distributed argument for $\mathcal{L}$.*
(3) *In the Common Reference String model, assuming* Knowledge-of-Exponent in Bilinear Groups, *there is a succinct non-interactive distributed argument for $\mathcal{L}$.*

Where in all of the cases above, the prover of the distributed argument has a distributed implementation.

D MISSING PROOFS FROM SECTION 6

D.1 If $\mathsf{LOCAL}^{[n]} \cap \mathsf{P} \not\subseteq \mathsf{PolyLOCAL}^{[n]}$, Then $\mathsf{P} \neq \mathsf{NP}$

In Section 6, we showed the separation of PolyLOCAL from LOCAL ∩ P by extensive use of the fact that the nodes in an PolyLOCAL algorithm do not know the size of a graph, and thus do not know how much time they are allowed to run. We would have wanted to prove the same without using this fact, that is, in the case where the nodes do know how much time they are allowed to run. Let $\mathsf{LOCAL}^{[\mathsf{n}]}$ and $\mathsf{PolyLOCAL}^{[\mathsf{n}]}$, be the variants of LOCAL and PolyLOCAL (resp.) where nodes have the size of the graph as part of their input.

In what follows, we demonstrate that proving this statement unconditionally would be *very* unexpected. However, in Section 6.3, we prove this statement conditioned on the existence of injective one-way functions.

Theorem D.1. *If* $\mathsf{LOCAL}^{[\mathsf{n}]} \cap \mathsf{P} \not\subseteq \mathsf{PolyLOCAL}^{[\mathsf{n}]}$*, then* $\mathsf{P} \neq \mathsf{NP}$.

Proof. Assume that $\mathsf{P} = \mathsf{NP}$, and let us show that every language $\mathcal{L} \in \mathsf{LOCAL}^{[\mathsf{n}]} \cap \mathsf{P}$ is also in $\mathsf{PolyLOCAL}^{[\mathsf{n}]}$.

Let $\mathcal{L}$ be a distributed language that is decided by some t-local algorithm $A \in \mathsf{LOCAL}^{[\mathsf{n}]}$, and also by a poly-time Turing machine M. We first modify A by restricting it so that it accepts only t-neighborhoods that can be embedded in some instance in $\mathcal{L}$, by defining a t-local algorithm A' where

$$A'(B) = 1 \qquad \Leftrightarrow \qquad A(B) = 1 \text{ and } \exists \tilde{G} \in \mathcal{L}\; \exists u \in \tilde{G} : B = N^t_{\tilde{G}}(u)$$

Algorithms A and A' decide the same distributed language, $\mathcal{L}$: every instance $\tilde{G}$ accepted by A' is also accepted by A, since whenever $A'(B) = 1$ we also have $A(B) = 1$. For the other direction, if an instance $\tilde{G}$ is accepted by A, then all nodes accept under A; also, $\tilde{G} \in \mathcal{L}$, and therefore every t-neighborhood of $\tilde{G}$ can be embedded in some instance in $\mathcal{L}$. Therefore all nodes accept $\tilde{G}$ under A'.

Now let $\mathcal{H}$ be the following node-language:

$$\mathcal{H} = \left\{ B \in \mathcal{B}^{t,\mathsf{n}} : \text{there is some } \tilde{G} \in \mathcal{L} \text{ and a vertex } u \in \tilde{G} \text{ such that } B = N^t_{\tilde{G}}(u) \right\}.$$

We first show that the node-language decided by A' is exactly $\mathcal{H}$: first, suppose that $B \in \mathcal{H}$. Then there is some $\tilde{G} \in \mathcal{L}$ and a vertex $u \in \tilde{G}$ such that $B = N^t_{\tilde{G}}(u)$. Since A' decides the distributed language $\mathcal{L}$, when we run A' in $\tilde{G}$ all nodes must accept, and therefore $A'(B) = A'(B^t_{\tilde{G}}(u)) = 1$. For the other direction, suppose that A' accepts some t-neighborhood B. Then in particular, by definition of A', there exists an instance $\tilde{G} \in \mathcal{L}$ and a node $u \in \tilde{G}$ such that $B = N^t_{\tilde{G}}(u)$. This implies that $B \in \mathcal{H}$.

To conclude the proof, we observe that $\mathcal{H} \in \mathsf{NP}$: it is decided by a poly-time Turing machine that takes the input B and witness $\tilde{G}$, and accepts iff M accepts $\tilde{G}$ and there is some node $u \in \tilde{G}$ that has $N^t_{\tilde{G}}(u) = B$. (Recall that every node of $\tilde{G}$ is annotated with 1^n, where n is the size of $\tilde{G}$; thus, the encoding length of $\tilde{G}$ is polynomial in the encoding length of the annotated t-neighborhood B.) Since we assumed that $\mathsf{P} = \mathsf{NP}$, this implies that $\mathcal{H} \in \mathsf{P}$ as well. But $\mathcal{H}$ is the node-language of A', and A' decides $\mathcal{L}$, so this implies that $\mathcal{L} \in \mathsf{PolyLOCAL}^{[\mathsf{n}]}$, as desired. □

D.2 NPolyLOCAL = NLOCAL ∩ NP

In this section, we show that the distinction between the local-polynomial classes and the intersection of the local and polynomial classes vanishes when introducing non-determinism. Let NLOCAL and NPolyLOCAL be the non-deterministic variants of LOCAL and PolyLOCAL (resp.). In the case of non-deterministic decision, it no longer matters whether the nodes know the network size, because one can always provide them with the size through their certificates, along with a proof that the size is correct, using a spanning tree [47]. In fact, the certificate can include the *entire graph*. For this reason, when we have unique identifiers, nodes can verify that the certificate describes the network graph accurately, allowing them to locally decide whatever can be decided by a centralized algorithm; therefore in this case $\mathsf{NP} \subseteq \mathsf{NLOCAL}$. When we do not have unique identifiers, it is not possible to verify that the certificates describe the network graph, but only that the network graph is a *lift* of the graph given in the certificates; this suffices for our purposes, because every NLOCAL language is closed under lifts [31].

THEOREM D.2. *Either with or without unique identifiers, we have* $\mathsf{NPolyLOCAL} = \mathsf{NLOCAL} \cap \mathsf{NP}$.

PROOF. We prove the case without identifiers, since it is the more general case.

The inclusion $\mathsf{NPolyLOCAL} \subseteq \mathsf{NLOCAL} \cap \mathsf{NP}$ is easy to see, as an NPolyLOCAL-algorithm is in particular an NLOCAL-algorithm, and it can also be efficiently simulated by a polynomial-time Turing machine that is given all the nodes' certificates.

To see the other direction of the inclusion, let $\mathcal{L} \in \mathsf{NLOCAL} \cap \mathsf{NP}$, let A be a t-local algorithm for $\mathcal{L}$, and let M be an NP-verifier for $\mathcal{L}$. We construct the following NPolyLOCAL-algorithm, B: in a configuration (G, x) on n nodes, we give to each node a certificate $c(v) = (i, (G', x'), w)$, where

- $i \in \{1, \ldots, n\}$ is an index,
- G' and x' represent the configuration (G, x), using $\{1, \ldots, n\}$ as the vertices,
- w is an NP-witness, such that M accepts $((G', x'), w)$.[24]

The nodes locally verify that

- Their t-neighborhood in G' is isomorphic to their true neighborhood in G, using the indices provided in the certificates as the isomorphism; and x' correctly describes their input, again using the index.
- M accepts $((G', x'), w)$.

As shown in [31], the first part of the verification passes iff (G', x') is a t-lift of (G, x), and NLOCAL-languages are closed under lift. The completeness of B follows from this fact. Soundness follows as well: if all nodes accept, then (G', x') is a t-lift of (G, x); since M accepts $((G', x'), w)$, we have $(G', x') \in \mathcal{L}$, which implies that $(G, x) \in \mathcal{L}$ as well. □

[24]Technically, (G', x') is a lift of (G, x), and therefore $(G', x') \in \mathcal{L}$ iff $(G, x) \in \mathcal{L}$.