

Distributed SNARGs Robust to Corrupt Verifiers

Elette Boyle
NTT Research and Reichman University
elette.boyle@ntt-research.com

Lalita Devadas
The Cooper Union
lalita.devadas@cooper.edu

Abstract

Distributed certification is a method for monitoring the correctness of a distributed system. The model consists of a centralized prover in addition to multiple verifiers lying on the nodes of a communication network, where the goal is to assert that the network satisfies a desired property. In doing so, the prover generates certificates for each verifier; the verifiers can then communicate in a small number of rounds, and accordingly accept or reject. The prover’s assertion is accepted if all verifiers accept.

A significant body of work has gone toward developing and understanding limitations of distributed certification schemes, predominantly in the setting of information-theoretic soundness, and recently with computational soundness, achieving a form of distributed (locally verifiable) Succinct Non-interactive Arguments (SNARGs) (Aldema Tshuva et al, TCC 2023). As is standard in the model, soundness holds in existing constructions assuming that all verifiers in the network are honest.

In this work, we introduce and explore the notion of *robust* distributed SNARGs (rdSNARGs) which retain (computational) soundness guarantees even when the cheating prover can *collude with some nodes in the network*. Addressing cheating verifiers presents several challenges. We construct rdSNARGs for any distributed language in P with succinct certificate size and communication from extended versions of RAM SNARGs (Kalai et al, STOC 2023), where the level of succinctness scales with the threshold of corrupt nodes. Complementarily, we demonstrate a lower bound showing that an rdSNARG with significantly smaller certificates and communication implies a SNARG for NP .

1 Introduction

Distributed certification [APSV91, KKP05] is a method for monitoring the correctness of a distributed system. A particularly important setting for distributed certification is that of network algorithms. In this model, the system consists of many processors connected by pairwise communication links, and the pattern of connectivity is represented by an undirected graph: vertices correspond to processors, and edges correspond to bidirectional communication channels. Unlike in many other computational models, the graph plays a dual role: it is both the object being reasoned about and the infrastructure through which the computation itself proceeds. The central idea of distributed certification is to equip the processors ahead of time with auxiliary data, called *certificates*, and later use these certificates to check whether the current global configuration satisfies the desired correctness condition. Once the certificates have been placed at the processors, the system can run a verification procedure in which processors exchange information and collectively determine whether the present state is legal. The network is declared to accept if every node accepts.

This framework captures a wide range of algorithmic tasks. Examples include constructing network structures such as spanning trees or sparse subgraphs, as well as problems involving routing, scheduling, and resource allocation (see, e.g., the textbook of [Pel00]). A defining feature of these algorithms is that each node begins with only limited knowledge—typically its own unique identifier, the identities of its immediate neighbors, and only a small amount of coarse global information such as the size or diameter of the network.

Any property on an n -node network can be asserted with certificate size $O(n^2)$, by simply including a description of the whole graph (here, parties check consistency with their local neighborhoods and verify that the graph satisfies the desired property). The goal of distributed certification is to minimize the cost of verification, including the certificate size, and communication required for verification (often assumed to be a single round of short message exchange). Since the system may not actually be in a correct state, the certificates must be treated as if they were supplied by a cheating prover attempting to justify the legality of the system even in the illegal case. In the classical setting, this corresponds to requiring information-theoretic soundness against an all-powerful cheating prover. Significant work has been performed showing upper and lower bounds in this domain, e.g., [KKP05, FF22, GS16, OPR17, PSP17, FFH⁺21, BSFO22, FHK12, FGKS13, BDFO17, FBP22, FMRT24, FPSP19, FMO⁺19, KOS18, NPY20, BKO22, GJKP26] among others; see also survey [Feu21].

Interestingly, even short certificates can enable dramatic efficiency gains. For example, certificates of size just $O(\log n)$ can enable parties to distributedly verify an alleged spanning tree, with a single communication round of $O(\log n)$ bits per network edge [KKP05]. Without certificates, this task is known to require $\tilde{\Omega}(\sqrt{n}/B)$ communication rounds, for B bits of communication per edge per round [DHK⁺11]. Further, generating a spanning tree from scratch requires $\Omega(D)$ communication rounds, for network diameter D .

However, one cannot hope for such savings universally. Similarly to certification systems for NP, distributed certification schemes with information-theoretic soundness face strong lower bounds. It is known that certain properties on networks of size n *require* certificates of maximal size $\Omega(n^2)$, even if the verification process can be randomized [GS16, FMO⁺19].

Computationally sound distributed certification. A recent work [ABC⁺23] provided the first exploration of distributed certification in the cryptographic, *computationally* sound setting. As in the case of succinct non-interactive arguments (SNARG), imposing a computational bound on the cheating prover and allowing for a trusted common reference string that the prover and the verifiers all have access to is enough to circumvent the aforementioned lower bounds. In their work, Aldema Tshuva et al. [ABC⁺23] construct a so-called locally verifiable distributed SNARG (dSNARG) for *any* network property in P with succinct certificates and a single round of short communication between the nodes.

More concretely, for a given network property—for instance, the claim that the local states of the nodes collectively describe a valid spanning tree—a dSNARG consists of two parts. First, a computationally bounded prover (who can see the entire network) assigns certificates to the nodes. Second, the nodes execute a distributed verification algorithm that uses these certificates to check whether the claimed property holds. In one such round, every node may carry out local computation, send possibly different messages to different neighbors, and receive the messages sent to it. At the end of verification, each node decides whether to accept or reject. As before, the network is declared to accept only when every node accepts; a single rejecting node is enough to signal that something

is wrong.

The dSNARG construction of [ABC⁺23] supports certificates of size $\text{poly}(\log n)$, together with a single round of $\text{poly}(\log n)$ bits of communication per edge, based on computationally sound RAM SNARGs [KPY19, CJJ22, KLVW23]. In turn, this can be based on Learning With Errors (LWE), $O(1)$ -LIN over bilinear maps, or Quadratic Residuosity (QR) and subexponential Decisional Diffie-Hellman (DDH) [KLVW23]. A large part of their work is in extending to a distributed-prover setting, where the nodes themselves generate the proof in a distributed manner, further optimized by [AO24].

Corrupt verifiers? One point to note is that these constructions all assume the verifiers themselves are *honest*. If a cheating prover is able to corrupt even one node in the network, soundness is no longer maintained. For example, recall the soundness guarantee only ensures that at least one node rejects if the network does not satisfy the property, so if the cheating prover corrupts that node and tells it to accept when it should reject, they can cheat with high probability. Even if there is a node who would have rejected and is not corrupted by the cheating prover, we cannot assume that it will still reject in this case. This is because this node may receive incorrect information from a corrupt neighbor during verification causing it to accept.

Nodes may also choose to collude with the cheating prover if it benefits them. Consider, for example, the problem of certifying that a leader was chosen fairly. A node may be willing to collude with the cheating prover if they agree to make it the leader. This completely destroys the soundness guarantee of the dSNARG in [ABC⁺23], leading to simple attacks. Of course, the problem only compounds when we consider the possibility that more than one node may be corrupted or colluding with the cheating prover.

1.1 Our Results

In this work, we introduce and explore the notion of locally verifiable distributed SNARGs which retain some soundness guarantee even when the cheating prover can *collude with some nodes in the network*, augmenting the model introduced in [ABC⁺23]. We refer to this as a *robust* (locally verifiable) distributed SNARG (rdSNARG). In this model, we cannot hope for the same notion of soundness or the same level of succinctness as is achievable with all honest verifiers, as we will see. Indeed, since the cheating prover can essentially decide itself (with no tie to the real network) if there exists an edge or not between any two corrupt nodes, there is a flavor of *nondeterminism* not present in the original model.

We formalize a notion of rdSNARGs, and then provide two types of results: 1) We construct rdSNARGs for any distributed language in P with $o(n^2)$ certificate size and communication from RAM SNARGs, where the level of succinctness scales with the threshold of corrupt nodes ; 2) We demonstrate a lower bound, showing that an rdSNARG with significantly smaller certificates and communication implies a SNARG for NP .

Defining *robust distributed SNARGs* (rdSNARG). Recall that even in the case of all honest verifiers, a computationally sound dSNARG must allow for a common reference string (CRS), to prevent attacks from a non-uniform adversary who has accepting certificates for a false property hardwired. Similarly, we must allow at least one round of communication between the verifiers, to prevent attacks where the adversary provides each node with a valid certificate for a network which satisfies the property and is consistent with their local view (but different from the real network).

In the presence of corrupt verifiers, one must make some additional assumptions about the network and corruptions to allow any chance for certifying general properties of the network. In particular, we need that the corruptions do not disconnect the network, i.e., that the honest nodes remain connected. This is the natural augmentation of the assumption in distributed certification (with all honest verifiers) that the network is connected. Without this assumption, we run into the same problem as in the setting without communication between the disconnected sets of verifiers, and cannot hope to certify anything about the complete honest view.

In general, one may consider additional restrictions or assumed partial information about the network held by the parties in advance.

Syntax. The syntax of our model is identical to the syntax in [ABC⁺23], except that the setup algorithm also takes as input a threshold on the number of corrupt nodes. We focus in this work on a centralized prover setting, which in the corrupt-node environment already presents significant challenges, and leave the goal of a distributed prover algorithm to future work. The rdSNARG thus consists of three algorithms:

- *A setup algorithm:* This is a centralized probabilistic polynomial time algorithm. Given the security parameter, the network size, and the number of corrupt nodes as input, the setup algorithm produces a common reference string.
- *A prover algorithm:* This is a centralized deterministic polynomial time algorithm. Given the common reference string and the network as input, the prover algorithm produces certificates for every node.
- *A verification algorithm:* This is a distributed deterministic polynomial time algorithm. Given the common reference string and their certificate, each node generates a message for each of its neighbors. After synchronously receiving messages from all of its neighbors, each node decides to accept or reject.

Completeness is also identical to [ABC⁺23]: for networks which satisfy the property, if the prover and verifier nodes perform their operations honestly, all parties will accept.¹

Soundness. Defining the soundness game for an rdSNARG is subtle, and an independent contribution of this work. We cannot simply translate the soundness notion for a dSNARG from [ABC⁺23], which is a natural extension of soundness for a (centralized) SNARG. In our setting, since the honest nodes do not know anything about edges/non-edges between corrupt nodes, we cannot hope for them to verify anything about these edges. We need to introduce a sort of *non-determinism* into the soundness guarantee of an rdSNARG for it to be achievable. In particular, all we can hope for is that if all honest verifiers accept, then there *exists* some satisfying network which is consistent with the honest view.

In the (nonadaptive) soundness game, the cheating prover must commit to which nodes it will corrupt before it sees the common reference string. After receiving the common reference string, the cheating prover provides certificates for the honest nodes, as well as the messages they would have received from any corrupt neighbors. The cheating prover wins the soundness game if all honest verifiers accept given the certificates and messages when there does not exist any satisfying network consistent with the honest view. We formalize this notion in [Definition 5.1](#).

¹One could also additionally aim to guarantee that all honest verifiers accept an honest prover's certificates even in the presence of malicious verifiers. This is orthogonal to the goal of robust soundness, and is not a focus of this work.

Construction of rdSNARG. Let n be the number of nodes in the network, and let t be the number of corrupted nodes. We construct a robust distributed SNARG with succinct certificates of size scaling with $\binom{t}{2}/(n-t)$, and succinct messages of size $\text{poly}(\lambda, \log n)$.

As noted above, we will require some additional restrictions and that the parties hold some additional information about the network in advance. In our constructions, we will also assume that every corrupt node has an honest neighbor, so the total honest view has a “grip” on all the nodes in the network. Lastly, we will assume that all nodes know the size of the network. Without either of these assumptions, our construction can only hope to attain a weaker soundness guarantee, as honest parties cannot prevent the cheating prover from inventing “ghost” nodes who are only connected to the corrupt nodes, since the honest nodes collectively have no idea if these nodes exist or not. An interesting and challenging goal would be to derive the same result with one of the two assumptions removed.

Our constructions use as ingredients *fully local somewhere-extractable hash families* and *flexible RAM SNARGs*, which are both known from [LWE](#), $O(1)$ -LIN over bilinear maps, or [QR](#) and subexponential DDH [[KLVW23](#)]. A RAM SNARG allows certifying a property of an input so that the certificate can be verified against a *hash* of the input, rather than the entire input. Recent works showed how to achieve notions of soundness for RAM SNARGs even when the verifier does not actually know the entire input. When instantiated with a fully local somewhere-extractable hash family, the size of the RAM SNARG certificate does not need to grow with the size of the hash value. Though we often think about a hash value as being succinct, in this setting it may need to be large to encode part of the input which the verifier does not hold.

In fact, the soundness notion we need from the flexible RAM SNARG is an interpolation of the partial input soundness notion in [[KLVW23](#)] and the older, incomparable soundness notion in [[KPY19](#), [CJJ22](#)], which is not implied in a black-box way from existing constructions. For this reason, we provide in this work a generalization of the constructions from [[CJJ22](#), [KLVW23](#)], modified to handle being associated with multiple hash families at once, and prove that it satisfies the soundness notion we need. We refer to [Section 4](#) for the construction and proof.

Theorem 1.1 (Informal - rdSNARG Construction). *Based on standard cryptographic assumptions (see above) there exists a rdSNARG for distributed languages in P on n nodes with t corruptions, with certificates of size $\lceil \binom{t}{2}/(n-t) \rceil \cdot \text{poly}(\lambda, \log n)$ and one round of node-to-node messages of size $\text{poly}(\lambda, \log n)$.*

We refer to [Section 2.1](#) for a high-level overview and [Section 5.3](#) for the detailed construction and analysis.

Lower bound on rdSNARGs. Complementarily, we also show that an rdSNARG for distributed languages in P with certificates asymptotically smaller than what we achieve in our construction would in fact imply a SNARG for hard languages in NP . This may seem quite surprising, since in general SNARGs for deterministic computation do not imply SNARGs for non-deterministic computation. However, the transformation utilizes the non-deterministic flavor of soundness for rdSNARGs, which guarantees if all honest verifiers accept, then there *exists* some satisfying network which is consistent with the honest view.

This flavor of soundness is reminiscent of SNARGs of proximity for P [[KR15](#), [CJW25](#)], where the verifier has oracle access to the input (and will only make a sublinear number of queries), and wishes to certify that the input is *close* to some language. These works also show an implication

from a SNARG of proximity for P to a SNARG for NP . Our lower bound is essentially a much simpler version of the communication lower bound in these works.

Interestingly, our transformation is able to construct a SNARG for any NP language if the corresponding witness length is at least (approximately) square root of the input length. We stress that the tools we use in our construction are *not* known to imply a SNARG for these NP languages.

Theorem 1.2 (Informal - Lower Bound). *An rdSNARG for distributed languages in P on n nodes which is robust to t corruptions, has node-to-node messages of size $o(t)$, and has certificates of size $o\left(\binom{t}{2}/(n-t)\right)$ implies a SNARG for NP languages where the witness length is asymptotically larger than the square root of the input length.*

Morally, the lower bound states that if the honest nodes do not receive at least as much information as is included in the part of the network that they know nothing about (this corresponds to the NP witness), then the rdSNARG for P is in fact a SNARG for the corresponding NP language. Since each honest node must receive some certificate of at least 1 bit, the bound becomes vacuous in the regime where the number of honest nodes is greater than the number of bits of information they do not know. In this regime, the honest nodes always receive *more* information than is included in the part of the network that they know nothing about, no matter how succinct the certificates are. We note that our construction does in fact achieve succinct certificates and succinct messages for values of t which the lower bound does not address.

We refer to [Section 2.2](#) for a technical overview of our lower bound, and [Section 6](#) for details.

1.2 Related Work

Distributed certification. [\[KKP05\]](#) introduced *proof labeling schemes*, the classical model of distributed certification (though it was already implicit in prior work on self-stabilization algorithms). In this model, the centralized, computationally unbounded prover generates labels for each node, which they exchange with their neighbors before deciding to accept or reject (based on their own certificate, their neighbors' certificates, and their knowledge of their neighborhood). There should exist certificates which cause all verifiers to accept for exactly the set of networks which satisfy the property being certified.

This model is quite restrictive, but relaxations which allow randomization, multiple rounds of communication between the verifiers, or interaction with the prover have also been studied in the following years. We refer to [\[Feu21\]](#) for a comprehensive survey of work in this area.

Error-sensitive proof labeling schemes. The augmentation of this model in [\[FF22\]](#) appears to be related to our setting. [\[FF22\]](#) introduces *error-sensitive proof labeling schemes*, which have the same syntax as proof labeling schemes, with the additional soundness property that the number of nodes who reject should be proportional to the distance between the current states in the network and the closest legal state.

These schemes, as with all information-theoretic distributed certification schemes, also face communication lower bounds which we can circumvent using cryptography. Furthermore, they do not consider the problem of whether the cheating prover can predict which nodes will reject (though some of their lower bounds involve doing such a matching).

Distributed SNARGs. As described above, [ABC⁺23] is the starting place for our model and our construction, though it does not immediately give any soundness guarantee in our model. In their scheme, the certificates and the messages exchanged during verification have length $\text{poly}(\lambda, \log n)$. Their construction uses as ingredients a collision-resistant hash function and a RAM SNARG.

In addition to their construction for a centralized prover (like our setting), they also provide a construction with a distributed prover. Follow up works [AO24, AO26] show how to improve succinctness of their distributed arguments and how to achieve incrementally verifiable computation in the distributed setting.

Arguments of proximity. The works of [KR15, CJW25] construct arguments of proximity, a closely related notion. In an argument of proximity, the verifier has query access to the input, and wishes to certify that it is close to a language by querying a sublinear number of bits. On the surface, this seems very similar to our setting, if we think about their verifier as choosing to play the role of a random node in the network. The primary difference is that their verifier can query *any* set of bits of the input, so every bit of the input can be included in many different query sets. For us, every bit of the input (an edge or non-edge) can only be included in two different query sets. This allows us to show a much simpler version of the communication lower bounds in these works.

Proofs on distributed data. A line of work beginning in [BBC⁺19] study a related setting of a centralized prover asserting claims $x \in \mathcal{L}$ to a collection of verifiers who each hold pieces of the corresponding input x . However, these works assume a fully connected communication graph between all verifiers, thus addressing different challenges than those faced in the partially connected graph setting of distributed certification.

1.3 Future Directions

Our work leaves several compelling directions for future investigation.

- Can our construction ideas be extended to obtain robust distributed SNARGs with a *distributed* prover, in the sense achieved by [ABC⁺23, AO24] in the honest-verifier setting? Even defining such notion requires care, given the malicious actions of a subset of parties. The constructions in [ABC⁺23, AO24] begin by having the verifiers distributedly construct a spanning tree of the network; translating even this first step to the setting of corrupt verifiers poses a challenge.
- Is it possible to remove assumed knowledge of the network size? In [ABC⁺23], they use a classical protocol from [KKP05] to certify the size of the network with perfect (not computational) soundness. However, soundness of this protocol breaks down completely in the setting with corrupt nodes, with no clear fix. Even with the assumption that every corrupt node has an honest neighbor, as made in our construction, it is unclear how to leverage this in the context of size certification, since the honest nodes do not know which nodes are corrupt, and so cannot catch corrupt nodes claiming the existence of “ghost” nodes.
- Our lower bound demonstrates that any rdSNARG with succinctness beyond the $\binom{t}{2}/(n-t)$ certificate size we achieve from standard cryptographic assumptions implies a form of SNARG for NP. For example, for the case of $t \in \Theta(n)$, this corresponds to $o(n)$ -size certificates. On the other hand, given SNARGs for NP, can one *obtain* rdSNARGs with highly succinct (eg, $O(\log n)$ -size) certificates? Such a construction is straightforward given the stronger object

of succinct non-interactive arguments of *knowledge* (SNARK), in which a valid witness can be extracted from any successful prover. However, it is not clear how to proceed without such extraction. We might also hope to extend our lower bound to show an implication to a SNARK for NP (see [Remark 6.1](#) for more discussion).

- More specifically, our lower bound implication yields SNARGs for any language in NP in which the witness size w is at least $w \in \omega(\sqrt{m})$ in the input size m . How limiting is this requirement? Is it in fact easier in general to construct SNARGs for such languages, and do they face all of the same barriers as SNARGs for all of NP?

2 Technical Overview

2.1 Positive Results

Construction for honest verifiers. As a starting point, let us recall the construction from [\[ABC⁺23\]](#). Their construction uses as ingredients a collision-resistant hash family with local openings and a RAM SNARG. A RAM SNARG is a certificate for some property which can be verified against a *hash* of the input (rather than the entire input, as in the usual notion of SNARG). If the certificate *and* openings to every bit of the input verify against the hash, then soundness states that the input must satisfy the property.²

In [\[ABC⁺23\]](#), to certify a property of the network, the prover hashes a description of the entire network and computes a RAM SNARG with respect to the hash certifying that the network satisfies the property. Each node receives the hash value, which they exchange with their neighbors to ensure that the prover provides the same hash value to all nodes in the network. Each node also receives the RAM SNARG certificate and an opening from the hash value to their neighborhood, both of which they can verify locally against the hash value.

Nodes accept if all checks pass, that is, if 1) they receive the same hash value from their neighbors as from the prover, 2) the RAM SNARG certificate is valid, and 3) the opening to their neighborhood is valid. Thus, if every node accepts, that means an opening to every bit of the input, i.e., every neighborhood in the network description, is verified by some node, so we can conclude by RAM SNARG soundness that the network satisfies the property.

Warmup construction for corrupted verifiers. Unfortunately, in the setting where some verifiers may be corrupted, the corrupt verifiers may accept without performing the checks that they are supposed to. This means that there will be some bits of the input, i.e., the neighborhoods of corrupt nodes, which are not checked by anyone, even when all honest verifiers accept. It may be that no such completion of corrupt neighborhoods exists for which the full graph satisfies the desired property, in which case the prover should be rejected.

The simplest solution is to have each node send their neighborhood and its opening to their neighbors, so that the honest neighbors of each corrupt node (recall we assume that at least one such honest neighbor exists) can perform the check that the corrupt node would have performed if they were honest. In particular, they can verify the opening from the hash value to the neighborhood.

²This is a nonstandard version of RAM SNARG soundness which relies on the *extended completeness* property defined in [\[ABC⁺23\]](#).

Of course, the corrupt nodes can claim any neighborhood they want in their messages to their honest neighbors. We cannot guarantee anything about the edges/non-edges between corrupt nodes, since no honest node knows anything about them, i.e., they are not part of the honest view. But the soundness we are hoping for from our rdSNARG does not require us to guarantee anything about these edges. We simply need to ensure that there exists some network which satisfies the property and is consistent with the honest view.

What saves us here is that now, as before, an opening to every bit of the input, i.e., every neighborhood in the network description, is verified by some node, so we can rely on RAM SNARG soundness. Instead of simply having the RAM SNARG certify that the input satisfies the property, we can have it simultaneously certify that the network description has undirected edges, i.e., if node v appears in the neighborhood of node u , then node u also appears in the neighborhood of node v . This means that if the RAM SNARG certificate is valid, the corrupt nodes must have claimed neighborhoods which are consistent with the honest view. Here the “input” is not necessarily the real network, but is *some* network which is consistent with the honest view, as we argued above.

This construction is sound even when some verifiers are corrupted, which is what we wanted! However, the messages sent between nodes are no longer succinct: they grow with the degree of the network, which in the worst case could be $\Theta(n)$ communication per edge. As it turns out, we can in fact achieve soundness against corrupt verifiers *without* requiring the nodes to send large messages.

Construction with succinct messages. To obtain succinct node-to-node messages, we will need a different notion of soundness for the RAM SNARG. The soundness notion we will need is an interpolation between the notion described above and *partial-input soundness*, introduced in [KLW23], which considers the setting when the RAM SNARG is computed with respect to a *somewhere-extractable* hash [HW15, OPWW15]. A somewhere-extractable hash key can be sampled to be *extractable* (and also statistically binding) on a subset of the bits of the string it hashes. These bits can be extracted from the resulting hash value (computed using this hash key) using a trapdoor which is sampled alongside the hash key. The key itself also hides the identity of the extractable set.

Partial-input soundness [KLW23] states that if the RAM SNARG is computed with respect to a somewhere-extractable hash of the input, then verifying the certificate against the hash value is sufficient to guarantee the input satisfies the property, *without* verifying any openings against the hash value. However, the notion in [KLW23] requires that the hash value should be extractable on *all* bits of the input which are relevant to whether or not the property is true, i.e., all bits which are touched by a RAM machine which decides the property.

This soundness notion is not directly helpful for us, since in our case the property of the input we want to certify may depend on *every* bit of the input. If we were to sample a somewhere-extractable hash key to be extractable on all bits of the input, the resulting hash value (which must be included in each node’s certificate) would be as large as the description of the network. This is exactly what we were trying to avoid by using cryptography.

To get a construction which is sound but has certificates smaller than the network description, we need an interpolation of the two soundness notions defined above: the first being “opening-type” soundness, which requires openings to every bit of the input, and the second being “extractability-type” soundness, which requires that all relevant input bits can be extracted from the hash value. Essentially, we want the flexibility to use opening-type soundness for some bits of the input and extractability-type soundness for the other bits of the input. In particular, we want the guarantee that the input satisfies the property if 1) the RAM SNARG certificate verifies against the somewhere-

extractable hash of the input, and 2) openings to all bits of the input which the hash value is *not* extractable on also verify against the hash value. In [Section 4](#) we show that the construction in [\[KLVW23\]](#) in fact satisfies this notion of soundness.

This is helpful because it allows us to sample the hash key to be extractable on *only* the bits of the hash input (i.e., the network description) that correspond to edges/non-edges between *corrupt* nodes. All other bits of the input correspond to the honest view of the network, which the honest nodes can verify openings to.

To get the succinctness we claim, we actually need to split this hash value, which has size $\binom{t}{2}$, between all of the $n - t$ honest nodes. We can do this by including a separate hash key for every node in the CRS. Each key will be of appropriate size to support extraction of $\binom{t}{2}/(n - t)$ bits. The prover computes the hash value and openings for each node according to their hash key. In the security proof, honest parties' hash keys will be sampled to support extraction of a different set of the network description, collectively enabling extraction of all $\binom{t}{2}$ bits describing edges/non-edges between corrupt nodes.

However, we still need the nodes to be able to ensure global consistency of the hash the prover provides across the network, even though now the prover is computing a separate hash value using a separate hash key for each node. To this end, we will also have the prover provide the same succinct collision-resistant hash of the network to every node, just as in the prior construction. The RAM SNARG certificate will be computed with respect to both the somewhere-extractable hash *and* the succinct hash.

Putting everything together, in our construction the CRS consists of a collision-resistant hash key, a CRS for a RAM SNARG, and a somewhere-extractable hash key for each node, which in the security analysis we will program to be extractable on exactly the bits which correspond to the edges/non-edges between corrupt nodes. Each certificate consists of a succinct hash value, a somewhere-extractable hash value, an opening to the node's neighborhood, and a RAM SNARG computed with respect to both hash values. During verification, nodes exchange the succinct hash value to ensure consistency, and locally verify the opening to their neighborhood and the RAM SNARG.

Note that for small values of t , i.e., bounded by approximately $\sqrt{n}$, this construction has succinct certificates along with succinct messages.

2.2 Lower Bound

To complement our positive results, we show that an rdSNARG which has certificates which are asymptotically smaller than what the above construction achieves in fact implies a SNARG for any language in NP with witness size at least square-root the input size.

Our lower bound is quite simple. Given a succinct rdSNARG as above, we construct a SNARG for such an NP language $\mathcal{L}$.

Consider encoding an instance of length N and a witness of length M for $\mathcal{L}$ into a network as follows. First, we pick a network size n and a corruption threshold t so that $\binom{t}{2} = \Theta(M)$, $\binom{n-t-1}{2} > N$, and $n - t < \binom{t}{2}$. Note that this third requirement is the source of the restriction for this result that we must have $M = \omega(\sqrt{N})$.

- Assign the first $n - t - 1$ nodes to be “honest” and encode the instance in the edges/non-edges between them.

- Assign the last t nodes to be “corrupt” and encode the witness in the edges/non-edges between them. We have $\binom{t}{2}$ bits with which to encode the witness, and we will in fact need this to be a tight bound on the witness length.
- Assign the $(n - t)$ th node to be “honest” and add edges from this node to all other nodes. Thanks to this connector node, we now know that our assumptions about the network corruptions (namely, that the honest nodes are connected and that every corrupt node has an honest neighbor) are true.

We define a distributed language $\mathcal{L}' \in \mathcal{P}$ to consist of networks which encode valid instance-witness pairs of $\mathcal{L}$. To generate a SNARG for an instance-witness pair, the SNARG prover encodes them into a network and runs the distributed SNARG prover to obtain certificates for each node. The SNARG proof consists of the $n - t$ certificates corresponding to the honest nodes, and the t messages that the corrupt nodes would have sent to their (single) honest neighbor during the verification stage. Soundness of the SNARG follows immediately from soundness of the rdSNARG.

As long as the node-to-node messages for the rdSNARG are asymptotically smaller than t and the certificates are asymptotically smaller than $\binom{t}{2}/(n - t)$ (the size of the certificates in our construction), the total SNARG proof size is asymptotically smaller than the witness size of $\binom{t}{2}$ bits.

3 Preliminaries

Notation. We use PPT to denote probabilistic polynomial-time, and denote the set of all positive integers up to n as $[n] := \{1, \dots, n\}$. For any finite set S , $x \leftarrow S$ denotes a uniformly random element x from the set S . Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element x drawn from distribution $\mathcal{D}$.

3.1 Modeling Distributed Networks

In this section, we give a more formal overview of our network model; this model is standard in the area of distributed network algorithms (see, e.g., the textbook [Pel00], and much of this section is taken from [ABC⁺23]).

A distributed network is modeled as an undirected,³ connected⁴ graph $G = (V, E)$, where the nodes V of the network are the processors participating in the computation, and the edges E represent bidirectional communication links between them.

We assume that V consists of a lexicographically ordered list of unique identifiers (UID), one for each node v in the network. For convenience we sometimes conflate the unique identifier of a node v with the vertex v representing v in the network graph. We assume that the UID is represented by a logarithmic number of bits in the size of the graph.

For a node $v \in V$, we denote by $N_G(v)$ (or by $N(v)$ if G is clear from context) the neighborhood of v in the graph G . We assume that the neighborhood $N(v)$, i.e., the UIDs of every neighbor $u \in N(v)$, is known to node v before the verification stage; the node does not necessarily need to

³It is fairly standard to assume an undirected graph topology, i.e., bidirectional communication links.

⁴We consider only connected networks, since in disconnected networks one can never hope to carry out any computation involving more than one connected component. As we will see later, even when we allow some nodes to be corrupted, we still require that the all honest nodes remain connected.

have it stored in memory at the beginning of the verification stage, but it should be able to generate them at verification time (e.g., by probing its neighborhood, opening communication sessions with its neighbors one after the other; or, in the case of a wireless network, by running a neighbor-discovery protocol).⁵ We do *not* assume that the UUIDs of the other nodes in the network (i.e., which are not neighbors of v) are known to v before the verification stage.

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

In this work, we will consider distributed languages in P , which for us means that a *centralized* algorithm which can see the entire configuration can decide membership in the language efficiently, i.e., in time polynomial in the size of the network. We will work with distributed languages where the network is represented via an adjacency list (as in [ABC⁺23]), as well as distributed languages where the network is represented via an adjacency matrix. We note that since a centralized algorithm which can see the entire network can convert between these representations efficiently, the choice of representation does not change the class of languages we are working with.

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

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

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

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

We frequently abuse notation by writing a sequence of values or mappings instead of a single one for α or β (respectively), e.g., when we write that the input to a distributed algorithm is $(a, b; G; x, y)$,

⁵As we will see later, we assume this is the case even if some of v 's neighbors are corrupted during verification.

we mean that every node $v \in V(G)$ is initially given $a, b, x(v), y(v)$, and the algorithm executes in the network described by the graph G .

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

3.2 Hash Family with Local Opening

In this section we recall the definition of a hash family with local opening [Mer88].⁶

Definition 3.1. A hash family (HT) with succinct local opening consists of the following algorithms:

$\text{Gen}(1^\lambda, N) \rightarrow \text{hk}$. This is a probabilistic poly-time algorithm that takes as input the security parameter 1^λ and the input length N and outputs a key $\text{hk} \in \{0, 1\}^{\text{poly}(\lambda, \log N)}$.

$\text{Hash}(\text{hk}, x) \rightarrow \text{h}$. This is a deterministic poly-time algorithm that takes as input a hash key hk and an input $x \in \{0, 1\}^N$ and outputs a hash value $\text{h} \in \{0, 1\}^{\text{poly}(\lambda, \log N)}$.

$\text{Open}(\text{hk}, x, j) \rightarrow (b, \rho)$. This is a deterministic poly-time algorithm that takes as input a hash key hk , an input $x \in \{0, 1\}^N$ and an index $j \in [N]$, and outputs a bit $b \in \{0, 1\}$ and an opening $\rho \in \{0, 1\}^{\text{poly}(\lambda, \log N)}$.

$\text{Verify}(\text{hk}, \text{h}, j, b, \rho) \rightarrow 0/1$. This is a deterministic poly-time algorithm that takes as input a hash key hk , a hash value h , an index $j \in [N]$, a bit $b \in \{0, 1\}$ and an opening $\rho \in \{0, 1\}^{\text{poly}(\lambda, \log N)}$, and outputs 1 (accept) or 0 (reject).

A HT family ($\text{Gen}, \text{Hash}, \text{Open}, \text{Verify}$) is required to satisfy the following:

Opening completeness. For any polynomial $N = N(\lambda)$, there exists a negligible function $\text{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}$, any $x \in \{0, 1\}^N$, and any index $j \in [N]$,

$$\Pr \left[\begin{array}{l} b = x_j \\ \wedge \text{Verify}(\text{hk}, \text{h}, j, b, \rho) = 1 \end{array} : \begin{array}{l} \text{hk} \leftarrow \text{Gen}(1^\lambda, N), \\ \text{h} = \text{Hash}(\text{hk}, x), \\ (b, \rho) = \text{Open}(\text{hk}, x, j) \end{array} \right] = 1 - \text{negl}(\lambda).$$

Collision resistance w.r.t. opening. For any poly-size adversary $\mathcal{A}$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} \text{Verify}(\text{hk}, \text{h}, j, 0, \rho_0) = 1 \\ \wedge \text{Verify}(\text{hk}, \text{h}, j, 1, \rho_1) = 1 \end{array} : \begin{array}{l} \text{hk} \leftarrow \text{Gen}(1^\lambda, N), \\ (\text{h}, j, \rho_0, \rho_1) \leftarrow \mathcal{A}(\text{hk}) \end{array} \right] = \text{negl}(\lambda).$$

Theorem 3.2 (Merkle [Mer88]). *Assuming the existence of a collision resistant hash family there exists a hash family with succinct local opening (according to Definition 3.1).*

⁶In what follows we use the notation HT to denote a hash family with local opening, where HT symbolizes a Hash Tree construction. We emphasize that we are not restricted to such a construction, and use this notation only to give the reader an example to have in mind.

Remark 3.1 (Opening multiple locations). One can extend the **Open** algorithm to take a set of indices $J \subseteq N$, as opposed to taking a single index $j \in [N]$, in the natural way. Namely,

$$\text{Open}(\text{hk}, x, J) = (\text{Open}(\text{hk}, x, j))_{j \in J}$$

Similarly, one can extend the **Verify** algorithm to verify a set of openings, as opposed to a single opening.

For [Construction 4.2](#), we will also need a hash family with local opening and efficient write operations.

Definition 3.3. A hash family (HTW) with succinct local opening and efficient write operations consists of the following algorithms:

Gen($1^\lambda, N$) $\rightarrow$ **hk**. This is a probabilistic poly-time algorithm that takes as input the security parameter 1^λ and the input length N and outputs a key $\text{hk} \in \{0, 1\}^{\text{poly}(\lambda, \log N)}$.

Hash(**hk**, x) $\rightarrow$ (**tree**, **rt**). This is a deterministic poly-time algorithm that takes as input a hash key **hk** and an input $x \in \{0, 1\}^N$ and outputs a hash tree **tree** and a root **rt** $\in \{0, 1\}^{\text{poly}(\lambda, \log N)}$.

Read(**hk**, **tree**, j) $\rightarrow$ (b, ρ). This is a deterministic poly-time algorithm that takes as input a hash key **hk**, a hash tree **tree**, and an index $j \in [N]$, and outputs a bit $b \in \{0, 1\}$ and an opening $\rho \in \{0, 1\}^{\text{poly}(\lambda, \log N)}$.

Write(**hk**, **tree**, j, b) $\rightarrow$ (**tree'**, **rt'**, ρ). This is a deterministic poly-time algorithm that takes as input a hash key **hk**, a hash tree **tree**, an index $j \in [N]$, and a bit b , and outputs a new tree **tree'**, a new root **rt'**, and an opening $\rho \in \{0, 1\}^{\text{poly}(\lambda, \log N)}$.

VerRead(**hk**, **rt**, j, b, ρ) $\rightarrow$ 0/1. This is a deterministic poly-time algorithm that takes as input a hash key **hk**, a root **rt**, an index $j \in [N]$, a bit $b \in \{0, 1\}$ and an opening $\rho \in \{0, 1\}^{\text{poly}(\lambda, \log N)}$, and outputs 1 (accept) or 0 (reject).

VerWrite(**hk**, **rt**, j, b, rt' , ρ) $\rightarrow$ 0/1. This is a deterministic poly-time algorithm that takes as input a hash key **hk**, a root **rt**, an index $j \in [N]$, a bit $b \in \{0, 1\}$, a new root **rt'**, and an opening $\rho \in \{0, 1\}^{\text{poly}(\lambda, \log N)}$, and outputs 1 (accept) or 0 (reject).

A HTW family (**Gen**, **Hash**, **Read**, **Write**, **VerRead**, **VerWrite**) is required to satisfy the following:

Read completeness. For any polynomial $N = N(\lambda)$, any $\lambda \in \mathbb{N}$, any $x \in \{0, 1\}^N$, and any index $j \in [N]$,

$$\Pr \left[\begin{array}{l} b = x_j \\ \wedge \text{VerRead}(\text{hk}, \text{rt}, j, b, \rho) = 1 \end{array} : \begin{array}{l} \text{hk} \leftarrow \text{Gen}(1^\lambda, N), \\ (\text{tree}, \text{rt}) = \text{Hash}(\text{hk}, x), \\ (b, \rho) = \text{Read}(\text{hk}, \text{tree}, j) \end{array} \right] = 1.$$

Read soundness. For any poly-size adversary $\mathcal{A}$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} \text{VerRead}(\text{hk}, \text{rt}, j, 0, \rho_0) = 1 \\ \wedge \text{VerRead}(\text{hk}, \text{rt}, j, 1, \rho_1) = 1 \end{array} : \begin{array}{l} \text{hk} \leftarrow \text{Gen}(1^\lambda, N), \\ (\text{rt}, j, \rho_0, \rho_1) \leftarrow \mathcal{A}(\text{hk}) \end{array} \right] = \text{negl}(\lambda).$$

Write completeness. For any polynomial $N = N(\lambda)$, any $\lambda \in \mathbb{N}$, any $x \in \{0, 1\}^N$, any index $j \in [N]$, and any bit b ,

$$\Pr \left[\begin{array}{l} (\text{tree}', \text{rt}') \text{ are correctly updated} \\ \wedge \text{VerWrite}(\text{hk}, \text{rt}, j, b, \text{rt}', \rho) = 1 \end{array} : \begin{array}{l} \text{hk} \leftarrow \text{Gen}(1^\lambda, N), \\ (\text{tree}, \text{rt}) = \text{Hash}(\text{hk}, x), \\ (\text{tree}', \text{rt}') = \text{Write}(\text{hk}, \text{tree}, j, b) \end{array} \right] = 1.$$

Write soundness. For any poly-size adversary $\mathcal{A}$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} \text{rt}_0 \neq \text{rt}_1 \\ \wedge \text{VerWrite}(\text{hk}, \text{rt}, j, b, \text{rt}_0, \rho_0) = 1 \\ \wedge \text{VerWrite}(\text{hk}, \text{rt}, j, b, \text{rt}_1, \rho_1) = 1 \end{array} : \begin{array}{l} \text{hk} \leftarrow \text{Gen}(1^\lambda, N), \\ (\text{rt}, j, b, \text{rt}_0, \rho_0, \text{rt}_1, \rho_1) \leftarrow \mathcal{A}(\text{hk}) \end{array} \right] = \text{negl}(\lambda).$$

Theorem 3.4 (Merkle [Mer88]). *Assuming the existence of a collision resistant hash family there exists a hash family with succinct local opening and efficient write operations (according to Definition 3.3).*

3.3 Fully Local Somewhere-Extractable Hash Family

In this section we recall the definitions of a somewhere-extractable hash (SEH) family [HW15, OPWW15] and a fully local somewhere-extractable hash (flSEH) family [DGKV22, KLVW23]. At a high level, a somewhere-extractable hash family allows for sampling the hash key with a trapdoor so that the trapdoor can be used to *extract* a subset of bits of the string that was hashed from the hash value. A *fully local* somewhere-extractable hash family additionally has the property that local openings to any bit of the string that was hashed do not grow with the number of bits which can be extracted from the hash value.

Definition 3.5 (SEH). A somewhere-extractable hash family (SEH) consists of algorithms

$$(\text{Gen}, \text{Hash}, \text{Open}, \text{Verify}, \text{Extract})$$

where Hash , Open , Verify have the same syntax as those of a hash family with local opening (Definition 3.1), and Gen and Extract have the following syntax:

$\text{Gen}(1^\lambda, N, i) \rightarrow (\text{hk}, \text{vk}, \text{td})$. This is a probabilistic poly-time setup algorithm that takes as input a security parameter 1^λ in unary, a message length N , and an index $i \in [N]$. It outputs a key pair (hk, vk) along with trapdoor td .

$\text{Extract}(\text{td}, \text{h}) \rightarrow b$. This is a deterministic poly-time extraction algorithm that takes as input a hash value h and a trapdoor td , and outputs a bit b .

An SEH hash family $(\text{Gen}, \text{Hash}, \text{Open}, \text{Verify}, \text{Extract})$ is required to satisfy the following properties:

Index hiding. For any poly-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} \mathcal{A}_2(\text{st}, \text{hk}) = b \\ \text{hk}, \text{vk}, \text{td} \leftarrow \text{Gen}(1^\lambda, N, i_b) \end{array} : \begin{array}{l} (N, i_0, i_1, \text{st}) \leftarrow \mathcal{A}_1(1^\lambda), \\ b \leftarrow \{0, 1\} \end{array} \right] \leq \frac{1}{2} + \text{negl}(\lambda).$$

Opening completeness. For any polynomial $N = N(\lambda)$, there exists a negligible function $\text{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}$, any indices $i, j \in [N]$, and any $x \in \{0, 1\}^N$,

$$\Pr \left[\begin{array}{l} b = x_j \\ \wedge \text{Verify}(\text{vk}, \text{h}, j, b, \rho) = 1 \end{array} : \begin{array}{l} (\text{hk}, \text{vk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, i), \\ \text{h} = \text{Hash}(\text{hk}, x), \\ (b, \rho) = \text{Open}(\text{hk}, x, j), \end{array} \right] = 1 - \text{negl}(\lambda).$$

We note that this property is almost identical to that in [Definition 3.1](#), where the above definition quantifies over all $i, j \in [N]$, whereas [Definition 3.1](#) quantified only over $j \in [N]$ (the extraction index i did not exist there).

Somewhere extractability w.r.t. opening. For any poly-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} b \neq \text{Extract}(\text{td}, \text{h}) \\ \wedge \text{Verify}(\text{vk}, \text{h}, i, b, \rho) = 1 \end{array} : \begin{array}{l} (N, i, \text{st}) \leftarrow \mathcal{A}_1(1^\lambda) \\ (\text{hk}, \text{vk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, i), \\ (\text{h}, b, \rho) \leftarrow \mathcal{A}_2(\text{st}, \text{hk}) \end{array} \right] \leq \text{negl}(\lambda).$$

Remark 3.2. The index-hiding property and the somewhere extractability w.r.t. opening property of a SEH hash family together imply collision resistance w.r.t. opening as in [Definition 3.1](#).

Remark 3.3 (m -SEH). Note that any SEH hash family (as defined in [Definition 3.5](#)) can be converted into one that is extractable on m indices $i_1, \dots, i_m$ by simply running all the algorithms in parallel m times, where the Gen algorithm is run each time with a different index i_j , resulting with hk_j , and the final hk is $(\text{hk}_1, \dots, \text{hk}_m)$. The rest of the algorithms are run m times, each time with a different hk_j , and they output the concatenation of all the outputs. Under this transformation, the new family will have a hash value and openings of size $m \cdot \text{poly}(\lambda, \log N)$.

Thus, more generally, we think of Gen as taking as input $(1^\lambda, N, I)$ where $I \subseteq [N]$, in which case $\text{Extract}(\text{td}, \text{rt})$ outputs $|I|$ bits $(b_i)_{i \in I}$. We sometimes refer to this as an m -SEH hash family, and sometimes we omit m , and simply refer to it as an SEH hash family.

Next, we recall the definition of a fully local somewhere-extractable hash family from [\[KLW23\]](#).

Definition 3.6 (fSEH). A fully local SEH (fSEH) hash family consists of algorithms

$$(\text{Gen}, \text{Hash}, \text{Digest}, \text{Open}, \text{Verify})$$

with the following syntax:

$\text{Gen}(1^\lambda, N, I) \rightarrow (\text{hk}, \text{vk}, \text{td})$. This is a probabilistic poly-time setup algorithm that takes as input the security parameter 1^λ (in unary), an input length N (in binary), and a set of indices $I \subseteq [N]$. It outputs a hk of length $|I| \cdot \text{poly}(\lambda)$, a (short) vk , and a trapdoor td .

$\text{Hash}(\text{hk}, x) \rightarrow \text{h}$. This is a deterministic poly-time hash algorithm that takes as input hk and message $x \in \{0, 1\}^N$, and outputs hash value h of length $|I| \cdot \text{poly}(\lambda)$.

$\text{Digest}(\text{vk}, \text{h}) \rightarrow \text{rt}$. This is a deterministic poly-time algorithm that takes as input vk and a hash value h , and outputs a (short) digest rt of h of length $\text{poly}(\lambda)$.

$\text{Open}(\text{hk}, x, i) \rightarrow (b, \rho)$. This is a deterministic poly-time opening algorithm that takes as input hk , message $x \in \{0, 1\}^N$, and index $i \in [N]$ and outputs a bit $b \in \{0, 1\}$ and a local opening ρ .

$\text{Verify}(\text{vk}, \text{rt}, i, b, \rho) \rightarrow 0/1$. This is a deterministic poly-time verification algorithm that takes as input a (short) vk , a (short) rt , an index $i \in [N]$, a bit $b \in \{0, 1\}$, and local opening ρ , and outputs 1 (accept) or 0 (reject).

$\text{Extract}(\text{td}, \text{h}) \rightarrow (b_1, \dots, b_{|I|})$. This is a deterministic poly-time extraction algorithm that takes as input a hash value h and a trapdoor td , and outputs a string of bits $(b_1, \dots, b_{|I|})$.

Remark 3.4. We often abuse notation and let

$$\text{Hash}(\text{hk}, \text{vk}, x) = (\text{h}, \text{rt})$$

where $\text{h} = \text{Hash}(\text{hk}, x)$ and $\text{rt} = \text{Digest}(\text{vk}, \text{h})$.

A fISEH hash family is required to satisfy the following properties:

Efficiency. The running time of $\text{Verify}(\text{vk}, \text{rt}, i, b, \rho)$ is $\text{poly}(\lambda, \log N)$. The running time of Gen and the length of a hash value h is $|I| \cdot \text{poly}(\lambda, \log N)$.

Opening completeness. For any polynomial $N = N(\lambda)$, there exists a negligible function $\text{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}$, any set of indices $I \subseteq [N]$, any index $i \in [N]$, and any $x \in \{0, 1\}^N$,

$$\Pr \left[\begin{array}{l} b = x_i \\ \wedge \text{Verify}(\text{vk}, \text{rt}, i, b, \rho) = 1 \end{array} : \begin{array}{l} (\text{hk}, \text{vk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I), \\ (\text{h}, \text{rt}) = \text{Hash}(\text{hk}, \text{vk}, x), \\ (b, \rho) = \text{Open}(\text{hk}, x, i), \end{array} \right] = 1 - \text{negl}(\lambda).$$

Opening soundness w.r.t. digest. For any poly-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} \text{Verify}(\text{vk}, \text{rt}, i, 0, \rho_0) = 1 \\ \wedge \text{Verify}(\text{vk}, \text{rt}, i, 1, \rho_1) = 1 \end{array} : \begin{array}{l} (1^N, I, \text{st}) \leftarrow \mathcal{A}_1(1^\lambda) \\ (\text{hk}, \text{vk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I), \\ (\text{rt}, i, \rho_0, \rho_1) \leftarrow \mathcal{A}_2(\text{st}, \text{hk}, \text{vk}) \end{array} \right] \leq \text{negl}(\lambda).$$

Opening soundness w.r.t. hash value. For any poly-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} i \in I \\ \wedge \text{Verify}(\text{vk}, \text{rt}, i^*, 1 - x_{i^*}, \rho^*) = 1 \end{array} : \begin{array}{l} (1^N, I, \text{st}) \leftarrow \mathcal{A}_1(1^\lambda) \\ (\text{hk}, \text{vk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I), \\ (\text{h}, i^*, \rho^*) \leftarrow \mathcal{A}_2(\text{hk}, \text{vk}, \text{st}). \\ \text{rt} = \text{Digest}(\text{vk}, \text{h}), \\ (x_i)_{i \in I} = \text{Extract}(\text{td}, \text{h}) \end{array} \right] \leq \text{negl}(\lambda).$$

Index hiding. For any poly-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} |I_0| = |I_1| \\ \wedge \mathcal{A}_2(\text{st}, \text{hk}, \text{vk}) = b \end{array} : \begin{array}{l} (1^N, I_0, I_1, \text{st}) \leftarrow \mathcal{A}_1(1^\lambda), \\ b \leftarrow \{0, 1\}, \\ (\text{hk}, \text{vk}, \text{td}) \leftarrow \text{Gen}(1^\lambda, N, I_b) \end{array} \right] \leq \frac{1}{2} + \text{negl}(\lambda).$$

Theorem 3.7 ([KLW23]). *Assuming LWE, $O(1)$ -LIN over bilinear maps, QR, or DDH, there exists a fISEH family.*

3.4 Batch Arguments

In this section, we recall the definition of index batch arguments (BARGs) [KVZ21, CJJ21].

Let `batchCSAT` be the following language:

$$\text{batchCSAT} = \{C : \exists w_1, \dots, w_k \in \{0, 1\}^m \text{ s.t. } \forall i \in [k], C(i, w_i) = 1\}.$$

where $C : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$ is a Boolean circuit.

Definition 3.8 (seBARG for batchCSAT). An index somewhere-extractable batch argument (seBARG) scheme $(\text{Gen}, \mathcal{P}, \mathcal{V}, \text{Extract})$ for `batchCSAT` consists of the following algorithms:

$\text{Gen}(1^\lambda, k, 1^s, i^*) \rightarrow (\text{crs}, \text{td})$. The Gen algorithm takes as input the security parameter 1^λ , the number of instances k , the circuit size 1^s , and an index i^* and outputs a CRS crs and a trapdoor td .

$\mathcal{P}(\text{crs}, C, w_1, \dots, w_k) \rightarrow \pi$. This is a poly-time prover algorithm that takes as input a crs , a circuit $C : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$, k witnesses $w_1, \dots, w_k \in \{0, 1\}^m$, and outputs a proof π .

$\mathcal{V}(\text{crs}, C, \pi) \rightarrow 0/1$. This is a poly-time verification algorithm that takes as input a crs , a circuit $C : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$, and a proof π . It outputs a bit to indicate whether the proof is valid or not.

$\text{Extract}(\text{td}, C, \pi) \rightarrow w^*$. This is a poly-time extraction algorithm that takes as input a trapdoor td , a circuit $C : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$, and a proof π , and it outputs a witness w^* .

A seBARG is required to satisfy the following properties:

Succinctness. The crs and proof π are of length at most $\text{poly}(\log k, \lambda, s)$, and the running time of Gen is at most $\text{poly}(\log k, \lambda, s)$.

Verification Efficiency. The verifier runs in time $\text{poly}(\log k, \lambda, s)$.

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

$$\Pr \left[\begin{array}{l} i_0, i_1 \in [k] \\ \wedge \mathcal{A}_2(\text{st}, \text{crs}) = b \end{array} : \begin{array}{l} (i_0, i_1, \text{st}) \leftarrow \mathcal{A}_1(1^\lambda) \\ b \leftarrow \{0, 1\} \\ (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, k, 1^s, i_b) \end{array} \right] \leq \frac{1}{2} + \text{negl}(\lambda).$$

Completeness. For any polynomials $k = k(\lambda)$ and $s = s(\lambda)$, there exists a negligible function $\text{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}$, any circuit $C : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}$ of size at most s , and corresponding witnesses $w_1, \dots, w_k \in \{0, 1\}^m$,

$$\Pr \left[\mathcal{V}(\text{crs}, C, \pi) = 1 : \begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda, k, 1^s, i^*), \\ \Pi \leftarrow \mathcal{P}(\text{crs}, C, w_1, \dots, w_k) \end{array} \right] = 1 - \text{negl}(\lambda).$$

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

$$\Pr \left[\begin{array}{l} \mathcal{V}(\text{crs}, C, \pi) = 1 \\ \wedge C(i^*, w^*) \neq 1 \end{array} : \begin{array}{l} (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, k, 1^s, i^*) \\ (C, \pi) \leftarrow \mathcal{A}(\text{crs}) \\ w^* \leftarrow \text{Extract}(\text{td}, C, \pi) \end{array} \right] \leq \text{negl}(\lambda).$$

Theorem 3.9 ([CJJ21, WW22, KLVW23, HJKS22]). *Assuming LWE , $O(1)$ -LIN over bilinear maps, or QR and subexponential DDH , there exists an index seBARG for batchCSAT.*

3.5 Flexible RAM SNARGs

In this section, we recall the definition of flexible RAM SNARG with partial input soundness from [KLVW23], as well as an incomparable soundness notion from [KPY19]. (Looking ahead, in Section 4 we construct a flexible RAM SNARG which achieves a generalization of both soundness notions.)

Definition 3.10 (Flexible RAM SNARG [KLVW23]). Let $\mathcal{R}$ be a RAM machine. A flexible RAM SNARG for $\mathcal{R}$ is associated with a hash family with local opening

$$HT = (HT.Gen, HT.Hash, HT.Open, HT.Verify),$$

and consists of the following algorithms (relative to HT):

$Gen(1^\lambda, T) \rightarrow \text{crs}$. This is a randomized poly-time setup algorithm that takes as input a security parameter 1^λ and a time bound T , and outputs a common reference string crs .

$\mathcal{P}(\text{crs}, \text{hk}, x) \rightarrow \pi$. This is a deterministic polynomial-time prover that takes as input a crs , a HT hash key hk , and an input x , and outputs a proof π .

$\mathcal{V}(\text{crs}, \text{vk}, \text{rt}, b, \pi) \rightarrow \{0, 1\}$. This is a deterministic polynomial-time verifier that takes as input a crs , a HT verification key vk , a digest rt of the input, a bit b , and a proof π , and outputs 1 (accept) or 0 (reject).

A RAM SNARG $(Gen, \mathcal{P}, \mathcal{V})$ for a RAM computation $\mathcal{R}$ with local state of size $S \geq \log|x|$, satisfies the following properties:

Efficiency. The running time of Gen , the length of a proof π , and the running time of $\mathcal{V}$ is at most $\text{poly}(\lambda, \log T, S) \cdot \text{time}(HT.Verify)$.

Completeness. For any polynomial $T = T(\lambda)$, there exists a negligible function $\text{negl}(\cdot)$ such that for any $\lambda, n = n(\lambda) \in \mathbb{N}$ such that $n \leq T$ and any $x \in \{0, 1\}^n$ such that $\mathcal{R}(x)$ outputs b within T time steps, we have that

$$\Pr \left[\mathcal{V}(\text{crs}, \text{vk}, \text{rt}, b, \pi) = 1 \quad : \quad \begin{array}{l} \text{crs} \leftarrow Gen(1^\lambda, T), \\ (\text{hk}, \text{vk}) \leftarrow HT.Gen(1^\lambda), \\ \pi = \mathcal{P}(\text{crs}, \text{hk}, x), \\ \text{rt} = HT.Hash(\text{hk}, x) \end{array} \right] = 1 - \text{negl}(\lambda).$$

Soundness. [KPY19] For any poly-size adversary $\mathcal{A}$ and polynomial $T = T(\lambda)$, there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} \mathcal{V}(\text{crs}, \text{vk}, \text{rt}, 0, \pi_0) = 1 \\ \wedge \mathcal{V}(\text{crs}, \text{vk}, \text{rt}, 1, \pi_1) = 1 \end{array} \quad : \quad \begin{array}{l} \text{crs} \leftarrow Gen(1^\lambda, T), \\ (\text{hk}, \text{vk}) \leftarrow HT.Gen(1^\lambda), \\ (\text{rt}, \pi_0, \pi_1) \leftarrow \mathcal{A}(\text{crs}, \text{hk}, \text{vk}), \end{array} \right] \leq \text{negl}(\lambda).$$

Partial-input soundness. [KLVW23] If the underlying hash family with local opening is somewhere-extractable on m locations, now denoted by

$$\text{SEH} = (\text{SEH.Gen}, \text{SEH.Hash}, \text{SEH.Open}, \text{SEH.Verify}, \text{SEH.Extract})$$

then for any poly-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and any polynomial $T = T(\lambda)$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} |I| \leq m \\ \wedge \mathcal{V}(\text{crs}, \text{vk}, \text{rt}, b, \pi) = 1 \\ \wedge \mathcal{R}(x; T) = 1 - b \text{ and doesn't} \\ \quad \text{read any location in } [N] \setminus I \end{array} : \begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda, T), \\ (1^N, I, \text{st}) \leftarrow \mathcal{A}_1(\text{crs}), \\ (\text{hk}, \text{vk}, \text{td}) \leftarrow \text{SEH.Gen}(1^\lambda, N, I), \\ (\text{rt}, b, \pi) \leftarrow \mathcal{A}_2(\text{st}, \text{crs}, \text{hk}), \\ \text{define } x \in \{0, 1\}^N : \\ \quad (x_i)_{i \in I} = \text{SEH.Extract}(\text{td}, \text{rt}), \\ \quad \forall i \in [N] \setminus I, x_i = 0 \end{array} \right] \leq \text{negl}(\lambda).$$

Remark 3.5. For Construction 5.2, we only need a RAM SNARG which satisfies the older notion of soundness, not partial-input soundness as in [KLVW23]. However, we also require the same notion of *extended completeness* as in [ABC⁺23]. They note that this is satisfied by the construction in [KLVW23].

4 Flexible RAM SNARGs with Extract-or-Open Soundness

In this section we introduce “extract-or-open” soundness for RAM SNARGs, and construct a RAM SNARG which achieves this notion. This also requires augmenting the syntax of the flexible RAM SNARG, to allow using many different keys to hash the memory.

Here, the memory will be hashed using a regular (HT) collision resistant hash key hk^* as well as ℓ fISEH keys, which are sampled to be extractable on ℓ sets $I_1, \dots, I_\ell$. A separate RAM SNARG will be computed with respect to each of the ℓ fISEH keys and the HT hash key. To win the soundness game, the adversary has to produce the following:

- a hash value rt^* with respect to the hash key hk^*
- ℓ hash values $\text{h}_1, \dots, \text{h}_\ell$ with respect to the ℓ hash keys $\text{hk}_1, \dots, \text{hk}_\ell$
- a hash of the states of the RAM machine rt_{st}
- ℓ RAM SNARG proofs $\pi_1, \dots, \pi_\ell$, each computed with respect to the HT hash key hk^* and one of the fISEH keys
- an opening o_j^* with respect to the HT hash key hk^* for every bit x_j of memory which is not in one of the sets $I_1, \dots, I_\ell$

Intuitively, this is an interpolation between the two soundness notions defined above, with added flexibility to use many hash keys.

Remark 4.1. We make a few changes to the syntax of the flexible RAM SNARG. In [KLVW23], the RAM SNARG certificate consists of a hash of the states of the RAM machine rt_{st} and a BARG proof π . Here, for convenience, we separate the two parts: the adversary has to produce different proofs $\pi_1, \dots, \pi_\ell$, since each proof is computed with respect to a different fISEH key, but the adversary only produces one rt_{st} , which all of the proofs $\pi_1, \dots, \pi_\ell$ are verified against.

Definition 4.1. Let $\mathcal{R}$ be a RAM machine. A RAM SNARG with extract-or-open soundness for $\mathcal{R}$ is associated with a hash family with local opening

$$\text{HT}^* = (\text{HT}^*.\text{Gen}, \text{HT}^*.\text{Hash}, \text{HT}^*.\text{Open}, \text{HT}^*.\text{Verify}),$$

and a somewhere extractable hash family

$$\text{SEH}^* = (\text{SEH}^*.\text{Gen}, \text{SEH}^*.\text{Hash}, \text{SEH}^*.\text{Open}, \text{SEH}^*.\text{Verify}, \text{SEH}^*.\text{Extract}),$$

which will be used to digest the memory of the RAM computation. The RAM SNARG consists of the following algorithms (parametrized by HT^* and SEH^*):

$\text{Gen}(1^\lambda, T) \rightarrow \text{crs}$. This is a randomized poly-time setup algorithm that takes as input a security parameter 1^λ and a time bound T , and outputs a common reference string crs .

$\mathcal{P}(\text{crs}, \text{hk}^*, (\text{hk}_k, \text{vk}_k)_{k \in [\ell]}, x) \rightarrow (\text{rt}_{\text{st}}, \pi_1, \dots, \pi_\ell)$. This is a deterministic polynomial-time prover that takes as input a crs , a HT^* hash key hk^* , ℓ SEH^* hash keys $(\text{hk}_k, \text{vk}_k)_{k \in [\ell]}$, and an input x , and outputs a root rt_{st} and ℓ proofs $\pi_1, \dots, \pi_\ell$.

$\mathcal{V}(\text{crs}, \text{hk}^*, \text{rt}^*, \text{rt}_{\text{st}}, (\text{vk}, \text{rt}), \pi) \rightarrow \{0, 1\}$. This is a deterministic polynomial-time verifier that takes as input a crs , a HT^* hash key and corresponding root hk^* , a hash rt_{st} of the states of the RAM machine, a SEH^* verification key and corresponding root (vk, rt) , and a proof π , and outputs 1 (accept) or 0 (reject).

A RAM SNARG $(\text{Gen}, \mathcal{P}, \mathcal{V})$ for a RAM computation $\mathcal{R}$ satisfies the following properties:

Efficiency. The running time of Gen , the length of a proof π , and the running time of $\mathcal{V}$ is at most $\text{poly}(\lambda, \log T) \cdot (\text{time}(\text{HT}^*.\text{Verify}) + \text{time}(\text{SEH}^*.\text{Verify}))$.

Completeness. For any polynomial $T = T(\lambda)$, there exists a negligible function $\text{negl}(\cdot)$ such that for any $\lambda, n = n(\lambda) \in \mathbb{N}$ such that $n \leq T$, any sets $I_1, \dots, I_k \subseteq [n]$, and any $x \in \{0, 1\}^n$ such that $\mathcal{R}(x)$ outputs 1 within T time steps, we have that

$$\Pr [\forall k \in [\ell], \mathcal{V}(\text{crs}, \text{hk}^*, \text{rt}^*, \text{rt}_{\text{st}}, (\text{vk}_k, \text{rt}_k), \pi_k) = 1 : \left. \begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda, T), \\ \text{hk}^* \leftarrow \text{HT}^*.\text{Gen}(1^\lambda), \\ \forall k \in [\ell], (\text{hk}_k, \text{vk}_k, \text{td}_k) \leftarrow \text{SEH}^*.\text{Gen}(1^\lambda, N, I_k), \\ (\text{rt}_{\text{st}}, (\pi_k)_{k \in [\ell]}) \leftarrow \mathcal{P}(\text{crs}, \text{hk}^*, (\text{hk}_k, \text{vk}_k)_{k \in [\ell]}, x) \end{array} \right] = 1 - \text{negl}(\lambda).$$

Extract-or-open soundness. For any poly-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and any polynomial $T = T(\lambda)$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda, \ell \in \mathbb{N}$ and every

subset $K \subseteq [\ell]$,

$$\Pr \left[\begin{array}{l} (I_k)_{k \in K} \text{ are disjoint} \\ \wedge \forall k \in K, \mathcal{V}(\text{crs}, \text{hk}^*, \text{rt}^*, \text{rt}_{\text{st}}, (\text{vk}_k, \text{rt}_k), \pi_k) = 1 \\ \wedge \forall j \in J, \text{HT}^*. \text{Verify}(\text{hk}^*, \text{rt}^*, j, x_j, o_j^*) = 1 \\ \wedge \mathcal{R}(x; T) \neq 1 \end{array} \right] : \left[\begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda, T), \\ (1^N, I_1, \dots, I_\ell) \leftarrow \mathcal{A}_1(\text{crs}), \\ \text{hk}^* \leftarrow \text{HT}. \text{Gen}(1^\lambda), \\ \forall k \in [\ell], (\text{hk}_k, \text{vk}_k, \text{td}_k) \leftarrow \text{SEH}^*. \text{Gen}(1^\lambda, N, I_k), \\ \left(\text{rt}^*, \text{rt}_{\text{st}}, (\text{h}_k, \pi_k)_{k \in K}, (x_j, o_j^*)_{j \in J} \right) \\ \leftarrow \mathcal{A}_2(\text{crs}, \text{hk}^*, (\text{hk}_k, \text{vk}_k)_{k \in [\ell]}) \end{array} \right] \leq \text{negl}(\lambda),$$

where $J = [N] \setminus \bigcup_{k \in K} I_k$, and for $k \in K$, we define $(x_i)_{i \in I_k} = \text{SEH}^*. \text{Extract}(\text{td}_k, \text{h}_k)$. If SEH^* is a fully-local somewhere extractable hash family as in [Definition 3.6](#), we define $\text{rt}_k = \text{fISEH}. \text{Digest}(\text{vk}_k, \text{h}_k)$ for $k \in K$. Otherwise, let $\text{rt}_k = \text{h}_k$.

Modeling RAM computation. Most work constructing delegation schemes for RAM computations [\[KPY19, CJJ22\]](#) model the computation as having a local configuration (including pointers), a work tape, and supported read and write operations to the memory. This allowed assuming that the local state of the RAM machine was short, since the work tape could be considered part of the memory. These constructions use a hash tree with efficient write operations to digest the memory, allowing them to update the digest over the course of the computation.

The work of [\[KLVW23\]](#) deviated from this modeling and only supported reading from memory. This was because achieving their notion of partial-input soundness required using a somewhere-extractable hash to digest the memory, which may not have efficient write operations. Thus, their proofs grow with the RAM machine state size, which could potentially be large. They sidestep the issue for their central application by using the RAM SNARG for a computation which has a small state.

In this work, we need both that the RAM SNARG proofs do not grow with length of the RAM machine work tape, as in the first case, and a type of partial-input soundness which requires a somewhere extractable hash, as in the second case. To achieve this, we will support both read and write operations to the work tape and only read operations for the memory.

Morally, our construction uses a generalization of the construction of [\[CJJ22\]](#) to handle the work tape, and a generalization of [\[KLVW23\]](#) to handle the memory. We consider the local configuration and a hash of the work tape to together make up the short state of the RAM machine.

Construction 4.2 (Flexible RAM SNARG with Extract-or-Open Soundness (following [\[KLVW23\]](#))). We are now ready to construct our flexible RAM SNARG for RAM computations with local state of size S . Our construction is associated with a hash family with local opening

$$\text{HT} = (\text{HT}. \text{Gen}, \text{HT}. \text{Hash}, \text{HT}. \text{Open}, \text{HT}. \text{Verify})$$

and a fully local somewhere-extractable hash family

$$\text{fISEH} = (\text{fISEH}. \text{Gen}, \text{fISEH}. \text{Hash}, \text{fISEH}. \text{Digest}, \text{fISEH}. \text{Open}, \text{fISEH}. \text{Verify}, \text{fISEH}. \text{Extract})$$

extractable on m locations which are both used to hash the memory. Our construction requires the following ingredients:

- A hash family with local opening and efficient write operations

$$\text{HTW} = (\text{HTW.Gen}, \text{HTW.Hash}, \text{HTW.Read}, \text{HTW.Write}, \text{HTW.VerRead}, \text{HTW.VerWrite}).$$

- A somewhere-extractable BARG scheme

$$\text{BARG} = (\text{BARG.Gen}, \text{BARG.}\mathcal{P}, \text{BARG.}\mathcal{V}, \text{BARG.Extract}).$$

- A somewhere-extractable hash family (for hashing the states of the RAM machine)

$$\text{SEH} = (\text{SEH.Gen}, \text{SEH.Hash}, \text{SEH.Open}, \text{SEH.Verify}, \text{SEH.Extract}).$$

$\text{Gen}(1^\lambda, T) \rightarrow \text{crs}$.

1. Sample $\text{HTW.hk} \leftarrow \text{HTW.Gen}(1^\lambda)$.
2. Fix (arbitrarily) $i^* = 1$.
3. Let $(\text{BARG.crs}, \text{BARG.td}) \leftarrow \text{BARG.Gen}(1^\lambda, T, 1^s, i^*)$, where $s = \text{poly}(\lambda, \log T) \cdot \text{time}(\text{flSEH.Verify})$ is specified below.
4. Sample $(\text{SEH.hk}, \text{SEH.td}) \leftarrow \text{SEH.Gen}(1^\lambda, T \cdot S, I_{i^*})$, for extractable index set $I_{i^*} = \{(i^* - 1) \cdot S + 1, \dots, i^* \cdot S\}$ where $S = \text{poly}(\lambda)$ is the local state size of the RAM machine.
5. Output $\text{crs} = (\text{HTW.hk}, \text{BARG.crs}, \text{SEH.hk})$.

$\mathcal{P}(\text{crs}, \text{hk}^*, (\text{hk}_k, \text{vk}_k)_{k \in [\ell]}, x) \rightarrow (\text{rt}_{\text{st}}, \pi_1, \dots, \pi_\ell)$.

0. We assume $\text{hk}^* \leftarrow \text{HT.Gen}(1^\lambda)$ and $(\text{hk}_k, \text{vk}_k) \leftarrow \text{flSEH.Gen}(1^\lambda, |x|, I_k)$ for some $I_k \subseteq [|x|]$ with $|I_k| \leq m$.
1. Parse $\text{crs} = (\text{HTW.hk}, \text{BARG.crs}, \text{SEH.hk})$.
2. Compute $\text{rt}^* = \text{HT.Hash}(\text{hk}^*, x)$.
3. For $k \in [\ell]$, compute $\text{h}_k = \text{flSEH.Hash}(\text{hk}_k, x)$ and $\text{rt}_k = \text{flSEH.Digest}(\text{vk}_k, \text{h}_k)$.
4. Let cf_0 be the initial configuration (including registers, head positions, etc) of $\mathcal{R}$ and let $W_0 = \emptyset$ be the initial work tape.
5. Compute $(\text{tree}_0, \text{rt}W_0) = \text{HTW.Hash}(\text{HTW.hk}, W_0)$.
6. Compute all the local states $(\text{st}_1 = (\text{cf}_1, \text{rt}W_1), \dots, \text{st}_T = (\text{cf}_T, \text{rt}W_T))$ of $\mathcal{R}(x)$.
7. Compute
$$\text{rt}_{\text{st}} = \text{SEH.Hash}(\text{SEH.hk}, (\text{st}_1, \dots, \text{st}_T)).$$
8. For every $i \in [T]$:

- Let $(\text{st}_i = (\text{cf}_i, \text{rtW}_i), \rho_i) = \text{SEH.Open}(\text{SEH.hk}, (\text{st}_1, \dots, \text{st}_T), I_i)$, where $I_i = \{(i-1) \cdot S + 1, \dots, i \cdot S\}$.
 - Let α_i be the location of the bit in the work tape that $\mathcal{R}$ reads given local configuration cf_{i-1} .
 - Let $(\beta_i, \text{HTW}.o_i) = \text{HTW.Read}(\text{HTW.hk}, \text{tree}_{i-1}, \alpha_i)$.
 - Let γ_i, δ_i be the location and the bit in the work tape that $\mathcal{R}$ writes given local configuration cf_{i-1} and reading bit β_i from the work tape.
 - Let $(\text{tree}_i, \text{rtW}_i, \text{HTW}.o'_i) = \text{HTW.Write}(\text{HTW.hk}, \text{tree}_{i-1}, \gamma_i, \delta_i)$.
 - Let j_i be the location of the bit in memory that $\mathcal{R}$ reads given local configuration cf_{i-1} and reading β_i from the work tape.
 - Let $(b_i, o_i^*) = \text{HT.Open}(\text{hk}^*, x, j_i)$.
 - For $k \in [\ell]$, let $(b_i, o_{i,k}) = \text{flSEH.Open}(\text{hk}_k, x, j_i)$.
 - For $k \in [\ell]$, let $w_{i,k} = (\text{st}_{i-1}, \text{st}_i, \rho_{i-1}, \rho_i, \beta_i, \text{HTW}.o_i, \text{HTW}.o'_i, b_i, o_i^*, o_{i,k})$.
9. For $k \in [\ell]$, let $C_k = C[\text{hk}^*, \text{rt}^*, (\text{vk}_k, \text{rt}_k), \text{HTW.hk}, \text{SEH.hk}, \text{rt}_{\text{st}}]$ be the circuit that on input (i, w_i) outputs 1 if and only if the following conditions hold:
- Parse $w_i = (\text{st}_{i-1}, \text{st}_i, \rho_{i-1}, \rho_i, \beta_i, \text{HTW}.o_i, \text{HTW}.o'_i, b_i, o_i^*, o_i)$.
 - Parse $\text{st}_{i-1} = (\text{cf}_{i-1}, \text{rtW}_{i-1})$, $\text{st}_i = (\text{cf}_i, \text{rtW}_i)$.
 - Check that
$$\text{SEH.Verify}(\text{SEH.hk}, \text{rt}_{\text{st}}, I_i, \text{st}_i, \rho_i) = 1,$$
where $I_i = \{(i-1) \cdot S + 1, \dots, (i) \cdot S\}$.
 - If $i > 1$, check that
$$\text{SEH.Verify}(\text{SEH.hk}, \text{rt}_{\text{st}}, I_{i-1}, \text{st}_{i-1}, \rho_{i-1}) = 1,$$
where $I_{i-1} = \{(i-2) \cdot S + 1, \dots, (i-1) \cdot S\}$.
 - Let α_i be the location of the bit in the work tape that $\mathcal{R}$ reads given local configuration cf_{i-1} .
 - Check that $\text{HTW.VerRead}(\text{HTW.hk}, \text{rtW}_{i-1}, \alpha_i, \beta_i, \text{HTW}.o_i) = 1$.
 - Let γ_i, δ_i be the location and the bit in the work tape that $\mathcal{R}$ writes given local state cf_{i-1} and reading bit β_i from the work tape.
 - Check that $\text{HTW.VerWrite}(\text{HTW.hk}, \text{rtW}_{i-1}, \gamma_i, \delta_i, \text{rtW}_i, \text{HTW}.o'_i) = 1$.
 - Let j_i be the location of the bit in memory that $\mathcal{R}$ reads given local configuration cf_{i-1} and reading bit β_i from the work tape.
 - Check that $\text{HT.Verify}(\text{hk}^*, \text{rt}^*, j_i, b_i, o_i^*) = 1$.
 - Check that $\text{flSEH.Verify}(\text{vk}_k, \text{rt}_k, j_i, b_i, o_i) = 1$.
 - Check that if $\mathcal{R}$ given local state cf_{i-1} reads bit β_i from the α_i 'th location on the work tape and b_i from the j_i 'th memory location, then it updates its local configuration to cf_i .
 - If $i = 1$ check that st_0 is the initial state.

- If $i = T$ then check that st_T is an accepting state.

10. Let $s = |C_1|$.

11. For $k \in [\ell]$, let $\pi_k = \text{BARG.P}(\text{BARG.crs}, C_k, w_{1,k}, \dots, w_{T,k})$.

12. Output $(\text{rt}_{\text{st}}, \pi_1, \dots, \pi_\ell)$.

$\mathcal{V}(\text{crs}, \text{hk}^*, \text{rt}^*, \text{rt}_{\text{st}}, (\text{vk}, \text{rt}), \pi) \rightarrow 0/1$.

1. Parse $\text{crs} = (\text{HTW.hk}, \text{BARG.crs}, \text{SEH.hk})$.

2. Output 1 if and only if

$$\text{BARG.V}(\text{BARG.crs}, C[\text{hk}^*, \text{rt}^*, (\text{vk}, \text{rt}), \text{HTW.hk}, \text{SEH.hk}, \text{rt}_{\text{st}}], \pi) = 1.$$

Theorem 4.3. *Assuming existence of a hash family with local opening and efficient write operations (as in Definition 3.3), a somewhere-extractable BARG scheme (as in Definition 3.8) and a somewhere extractable hash family (as in Definition 3.5), Construction 4.2 is a flexible RAM SNARG with extract-or-open soundness with respect to HT and fISEH, as per Definition 4.1.*

Proof. Suppose for the sake of contradiction that there exists a poly-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, a polynomial $T = T(\lambda)$, and a non-negligible function $\epsilon = \epsilon(\lambda)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[\begin{array}{l} (I_k)_{k \in K} \text{ are disjoint} \\ \wedge \forall k \in K, \mathcal{V}(\text{crs}, \text{hk}^*, \text{rt}^*, \text{rt}_{\text{st}}, (\text{vk}_k, \text{rt}_k), \pi_k) = 1 \\ \wedge \forall j \in J, \text{HT}^*. \text{Verify}(\text{hk}^*, \text{rt}^*, j, x_j, o_j^*) = 1 \\ \wedge \mathcal{R}(x; T) \neq 1 \end{array} \right] : \left[\begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda, T), \\ (1^N, I_1, \dots, I_\ell) \leftarrow \mathcal{A}_1(\text{crs}), \\ \text{hk}^* \leftarrow \text{HT.Gen}(1^\lambda), \\ \forall k \in [\ell], (\text{hk}_k, \text{vk}_k, \text{td}_k) \leftarrow \text{SEH}^*. \text{Gen}(1^\lambda, N, I_k), \\ \left(\text{rt}^*, \text{rt}_{\text{st}}, (\text{h}_k, \pi_k)_{k \in K}, (x_j, o_j^*)_{j \in J} \right) \\ \leftarrow \mathcal{A}_2(\text{crs}, \text{hk}^*, (\text{hk}_k, \text{vk}_k)_{k \in [\ell]}) \end{array} \right]$$

where $J = [N] \setminus \bigcup_{k \in K} I_k$, and for $k \in K$, we define $\text{rt}_k = \text{fISEH.Digest}(\text{vk}_k, \text{h}_k)$ and $(x_i)_{i \in I_k} = \text{fISEH.Extract}(\text{td}_k, \text{h}_k)$.

For every $i \in [T]$, let Gen_i be identical to Gen , except that rather than setting $i^* = 1$ it sets $i^* = i$, and it outputs $(\text{BARG.td}, \text{SEH.td})$ along with crs . To simplify notation, we define the following experiment:

EXPT _i
$(\text{HTW.hk}, \text{BARG.crs}, \text{SEH.hk}, \text{BARG.td}, \text{SEH.td}) \leftarrow \text{Gen}_i(1^\lambda, T),$ $(1^N, I_1, \dots, I_\ell) \leftarrow \mathcal{A}_1(\text{crs}),$ $\text{hk}^* \leftarrow \text{HT.Gen}(1^\lambda),$ $\forall k \in [\ell], (\text{hk}_k, \text{vk}_k, \text{td}_k) \leftarrow \text{fISEH.Gen}(1^\lambda, N, I_k),$ $\left(\text{rt}^*, \text{rt}_{\text{st}}, (\text{h}_k, \pi_k)_{k \in K}, (x_j, o_j^*)_{j \in J} \right) \leftarrow \mathcal{A}_2(\text{crs}, \text{hk}^*, (\text{hk}_k, \text{vk}_k)_{k \in [\ell]})$ define $J = [N] \setminus \cup_{k \in K} I_k$ $\forall k \in K, \text{rt}_k = \text{fISEH.Digest}(\text{vk}_k, \text{h}_k)$ and $(x_i)_{i \in I_k} = \text{fISEH.Extract}(\text{td}_k, \text{h}_k)$ $\text{st}_i = \text{SEH.Extract}(\text{SEH.td}, \text{rt}_{\text{st}})$ $(\text{st}'_{i-1}, \text{st}'_i, \rho_{i-1}, \rho_i, \beta'_i, \text{HTW.o}_i, \text{HTW.o}'_i, b_i, o_i^*, o_i) = \text{BARG.Extract}(\text{BARG.td}, \pi_\alpha)$

where α is defined as follows: Suppose that $\mathcal{R}$ given the correct $(i-1)$ 'th local configuration on input x reads the bit in location j_i in memory. (Note that j_i can depend on the bit β_i that $\mathcal{R}$ reads from the work tape, but is still determined by x and i .) If $j_i \in I_k$ for some $k \in K$, set $\alpha = k$. Otherwise, i.e., if $j_i \in J$, set $\alpha = 1$.

We also define the following event:

VALID
$I_1, \dots, I_K$ are disjoint $\wedge \forall k \in K, \text{BARG.V}(\text{BARG.crs}, C[\text{hk}^*, \text{rt}^*, (\text{vk}_k, \text{rt}_k), \text{HTW.hk}, \text{SEH.hk}, \text{rt}_{\text{st}}], \pi_k) = 1$ $\wedge \forall j \in J, \text{HT.Verify}(\text{hk}^*, \text{rt}^*, j, x_j, o_j^*) = 1$

By the index hiding properties of SEH and BARG, there exists a negligible function $\mu(\cdot)$ such that for every $i \in [T]$ and every $\lambda \in \mathbb{N}$,

$$\Pr_{\text{EXPT}_i} [\text{VALID} \wedge \mathcal{R}(x; T) \neq 1] \geq \epsilon(\lambda) - \mu(\lambda). \quad (1)$$

Claim 4.4. *There exists a negligible function $\nu(\cdot)$ such that for every $i \in [T]$,*

$$\Pr_{\text{EXPT}_i} [\text{VALID} \wedge \text{st}_i \text{ is not the correct } i\text{'th local state of } \mathcal{R}(x)] \leq \nu(\lambda).$$

(Here, we use “state” to refer to both the configuration of the RAM machine and the root hash of the work tape.) We note that [Claim 4.4](#) (for $i = T$) and [Equation \(1\)](#) (for $i = T$) imply a contradiction. First note they imply that there exists a negligible function $\xi(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr_{\text{EXPT}_T} \left[\begin{array}{l} \text{VALID} \wedge \mathcal{R}(x; T) \neq 1 \\ \wedge \text{st}_T \text{ is not accepting} \end{array} \right] \geq \epsilon(\lambda) - \xi(\lambda).$$

Now consider extracting $w_T = \text{BARG.Extract}(\text{BARG.td}, \pi_1)$ and parsing $w_T = (\text{st}'_{T-1}, \text{st}'_T, \rho_{T-1}, \rho_T, \beta_T, \text{HTW.o}_T, \text{HTW.o}'_T, b_T, o_T^*, o_T)$. By the somewhere argument of knowledge property of BARG and the somewhere-extractable w.r.t. opening property of SEH, we know that $\text{st}'_T = \text{st}_T$ except with negligible probability. However, the somewhere argument of knowledge property of BARG also implies that st'_T is accepting except with negligible probability, so we have reached a contradiction.

Thus, it remains to prove [Claim 4.4](#).

Proof of Claim 4.4. We prove this by induction on i . In particular, we prove that there exists a negligible function $\mu_i(\cdot)$ such that

$$\begin{aligned} & \Pr_{\text{EXPT}_i} [\text{VALID} \wedge \text{st}_i \text{ is not the correct } i\text{'th local state. of } \mathcal{R}(x)] \\ & \leq \Pr_{\text{EXPT}_{i-1}} [\text{VALID} \wedge \text{st}_{i-1} \text{ is not the correct } (i-1)\text{'th local state of } \mathcal{R}(x)] + \mu_i(\lambda). \end{aligned}$$

Let $w_{i,\alpha} = (\text{st}'_{i-1}, \text{st}'_i, \rho_{i-1}, \rho_i, \beta'_i, \text{HTW}.o_i, \text{HTW}.o'_i, b_i, o_i^*, o_i)$. Recall that $C_\alpha(i, w_{i,\alpha}) = 1$ implies the following (parsing $\text{st}'_{i-1} = (\text{cf}'_{i-1}, \text{rtW}'_{i-1})$ and $\text{st}'_i = (\text{cf}'_i, \text{rtW}'_i)$):

1. $\mathcal{R}$ goes from local state cf'_{i-1} to local state cf'_i after reading bit β'_i from location α_i and bit b_i from memory location j_i .
2. $\text{HTW.VerRead}(\text{HTW.hk}, \text{rtW}'_{i-1}, \alpha_i, \beta'_i, \text{HTW}.o_i) = 1$.
3. $\text{HTW.VerWrite}(\text{HTW.hk}, \text{rtW}'_{i-1}, \gamma_i, \delta_i, \text{rtW}'_i, \text{HTW}.o'_i) = 1$.
4. $\text{SEH.Verify}(\text{SEH.hk}, \text{rt}_{\text{st}}, I_{i-1}, \text{st}'_{i-1}, \rho_{i-1}) = 1$.
5. $\text{SEH.Verify}(\text{SEH.hk}, \text{rt}_{\text{st}}, I_i, \text{st}'_i, \rho_i) = 1$.
6. $\text{HT.Verify}(\text{hk}^*, \text{rt}^*, j_i, b_i, o_i^*) = 1$.
7. $\text{fISEH.Verify}(\text{vk}_\alpha, \text{rt}_\alpha, j_i, b_i, o_i) = 1$.

Using the somewhere argument of knowledge property of BARG, we have that

$$\begin{aligned} & \Pr_{\text{EXPT}_i} [\text{VALID} \wedge \text{st}_i \text{ is not correct}] \\ & \leq \Pr_{\text{EXPT}_i} [\text{VALID} \wedge C_\alpha(i, w_{i,\alpha}) = 1 \wedge \text{st}_i \text{ is incorrect}] + \text{negl}(\lambda). \end{aligned}$$

- Because C_α checks that $\text{SEH.Verify}(\text{SEH.hk}, \text{rt}_{\text{st}}, I_i, \text{st}'_i, \rho_i) = 1$, the somewhere extractability w.r.t. opening property of SEH implies that we can consider $\text{st}'_i = (\text{cf}'_i, \text{rtW}'_i)$ instead of st_i .
- Because C_α checks that $\text{HTW.VerRead}(\text{HTW.hk}, \text{rtW}'_{i-1}, \alpha_i, \beta'_i, \text{HTW}.o_i) = 1$, read completeness and soundness of HTW imply β'_i is correct if rtW'_{i-1} is correct.
- Because C_α checks that $\text{HTW.VerWrite}(\text{HTW.hk}, \text{rtW}'_{i-1}, \gamma_i, \delta_i, \text{rtW}'_i, \text{HTW}.o'_i) = 1$, the write completeness and soundness properties of HTW imply that rtW'_i is correct if rtW'_{i-1} is correct.
- If $j_i \in I_\alpha$, then x_{j_i} was extracted from h_α . Because C_α checks that $\text{fISEH.Verify}(\text{vk}_\alpha, \text{rt}_\alpha, j_i, b_i, o_i) = 1$, the opening soundness w.r.t. hash value property of fISEH implies that $b_i = x_{j_i}$.
- If $j_i \in J$, then VALID means that $\text{HT.Verify}(\text{hk}^*, \text{rt}^*, j_i, x_{j_i}, o_{j_i}^*) = 1$. Because C_α checks that $\text{HT.Verify}(\text{hk}^*, \text{rt}^*, j_i, b_i, o_i^*) = 1$, the collision resistance property of HT implies that $b_i = x_{j_i}$.
- Because C_α checks that $\mathcal{R}$ goes from local state cf'_{i-1} to local state cf'_i after reading bit β'_i from location α_i and bit b_i from memory location j_i , if cf'_{i-1} and β'_i are correct and $b_i = x_{j_i}$, we have that cf'_i is correct.

Thus, we can conclude that

$$\begin{aligned} \Pr_{\text{EXPT}_i} [\text{VALID} \wedge \text{st}_i \text{ is not correct}] \\ \leq \Pr_{\text{EXPT}_i} [\text{VALID} \wedge C_\alpha(i, w_{i,\alpha}) = 1 \wedge \text{st}'_{i-1} \text{ is incorrect}] + \text{negl}(\lambda). \end{aligned}$$

For the base case, it suffices to note that $C_\alpha(1, w_{1,\alpha})$ checks that st'_0 is the initial state. For the inductive step, it remains to argue that

$$\begin{aligned} \Pr_{\text{EXPT}_i} [\text{VALID} \wedge C_\alpha(i, w_{i,\alpha}) = 1 \wedge \text{st}'_{i-1} \text{ is incorrect}] \\ \leq \Pr_{\text{EXPT}_{i-1}} [\text{VALID} \wedge \text{st}_{i-1} \text{ is incorrect}] + \text{negl}(\lambda). \end{aligned}$$

We prove this via two hybrids. By the index hiding property of SEH, we can consider running EXPT_i but sampling SEH.hk as in EXPT_{i-1} . Then the somewhere argument of knowledge property of BARG and the somewhere extractability w.r.t. opening property of SEH implies the state st_{i-1} extracted from rt_{st} matches st'_{i-1} and is thus incorrect. Finally, by the index hiding property of BARG, this holds even if we run EXPT_{i-1} instead.

This completes the proof.

5 Robust Distributed SNARGs

In this section, we define and construct robust distributed SNARGs (rdSNARGs) for distributed languages in $\mathcal{P}$.

In [Section 5.1](#), we state the assumptions about the network and corruptions which are necessary for our constructions, establish notation for corrupted networks, and provide the syntax and necessary properties of an rdSNARG. In [Section 5.2](#), we present a modified version of the construction in [\[ABC⁺23\]](#), where the nodes send each other large messages (that grow with the degree of the network) during verification, and show that it is robust to corrupt verifiers. In [Section 5.3](#), we present a construction where the nodes send each other succinct messages during verification, but the certificate size grows with the threshold of corrupt nodes, and show that it is robust to corrupt verifiers.

5.1 Definition

Network assumptions. We need to make the following assumptions about the network and the corruptions.

- We need that the corruptions do not disconnect the network, i.e., that the honest nodes remain connected. This is the natural augmentation for our setting of the usual assumption in distributed certification (with all honest verifiers) that the network is connected.
- We will also need that every corrupt node has an honest neighbor, so the total honest view has a “grip” on all the nodes in the network.
- Lastly, we will need to assume that all nodes know the size of the network. Otherwise, we cannot prevent the cheating prover from inventing “ghost” nodes who are only connected to the corrupt nodes, since the honest nodes collectively have no idea if these nodes exist or not.

We note that the latter two assumptions may not be inherent, but we leave it as an open question of how to remove them.

Notation. We write V_{hon} to denote the list of honest nodes and V_{cor} to denote the list of corrupt nodes, where $V = V_{\text{hon}} \cup V_{\text{cor}}$. We write $E^* \in \{0, 1, *\}^{\binom{n}{2}}$ for a network of size n to denote an adjacency matrix of edges where some edges have the value $*$, indicating that they are unknown. We write $x^* : V \rightarrow \{0, 1\}^* \cup \{*\}$ to denote an assignment x where some assignments $x(v)$ have the value $*$, indicating that they are unknown. We write $G^* = (V, E^*, x^*)$ to denote the configuration with unknown edges/assignments.

Unlike in the case of all honest verifiers, here, the adversary has control not only over the certificates distributed to the nodes but also over some of the messages sent from node to node during the verification stage. We write $\text{msg}_{\text{cor} \rightarrow \text{hon}}$ to denote a directory of messages sent from corrupt nodes to honest nodes during the verification stage.

For simplicity, here we assume that the messages sent by the verifiers to each other are deterministically computed from their certificates.

We write $\mathcal{V}(\text{crs}; G^*; x^*, \pi, \text{msg}_{\text{cor} \rightarrow \text{hon}})$ to denote the protocol output (one bit per party) when the honest nodes in G^* behave according to the distributed verification algorithm $\mathcal{V}$ on input π and receive the messages in $\text{msg}_{\text{cor} \rightarrow \text{hon}}$, and the corrupt nodes all output 1. We use the notation $\mathcal{V}(\text{crs}; G^*; x^*, \pi, \text{msg}_{\text{cor} \rightarrow \text{hon}}) = 1$ to denote the case that *all* parties output 1.

Definition 5.1. Let $\mathcal{L}$ be a distributed language. A robust distributed SNARG for $\mathcal{L}$ is a tuple of PPT algorithms $\text{rdSNARG} = (\text{Gen}, \mathcal{P}, \mathcal{V})$ with the following syntax:

$\text{Gen}(1^\lambda, 1^n, t) \rightarrow \text{crs}$. A randomized setup algorithm that takes as input a security parameter 1^λ , a graph size 1^n , the number of corrupt nodes t , and outputs a common reference string crs .

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

$\mathcal{V}(\text{crs}; G; x, \pi) \rightarrow \{0, 1\}$. A one-round distributed decision algorithm that takes as common input to the entire network a common reference string crs , executes in the network G , where each node $v \in V(G)$ locally receives an assignment $x(v)$ and a certificate $\pi(v)$, and outputs acceptance bits $b : V(G) \rightarrow \{0, 1\}$.

that satisfy the following properties:

Completeness. There exists a negligible function $\text{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}$, $(G, x) \in \mathcal{L}$, $t \leq |V(G)|$,

$$\Pr \left[\mathcal{V}(\text{crs}; G; x, \pi) = 1 \quad : \quad \begin{array}{l} \text{crs} \leftarrow \text{Gen}(1^\lambda, 1^{|V(G)|}, t) \\ \pi \leftarrow \mathcal{P}(\text{crs}, G, x) \end{array} \right] = 1 - \text{negl}(\lambda).$$

Soundness. For any poly-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\text{negl}(\cdot)$

such that for any $\lambda \in \mathbb{N}$

$$\Pr \left[\begin{array}{l} E^* \text{ is admissible} \\ (G^*, x^*) \text{ not consistent with } (G, x) \in \mathcal{L} : \\ \mathcal{V}(\text{crs}; G^*; x^*, \pi, \text{msg}_{\text{cor} \rightarrow \text{hon}}) = 1 \end{array} \quad \begin{array}{l} (V_{\text{hon}}, V_{\text{cor}}, E^*, x^*, \text{st}) \leftarrow \mathcal{A}_1(1^\lambda) \\ \text{crs} \leftarrow \text{Gen}(1^\lambda, 1^{|V|}, |V_{\text{cor}}|) \\ (\pi, \text{msg}_{\text{cor} \rightarrow \text{hon}}) \leftarrow \mathcal{A}_2(\text{crs}, \text{st}) \\ G^* = (V = V_{\text{hon}} \cup V_{\text{cor}}, E^*) \end{array} \right] \leq \text{negl}(\lambda)$$

where we say that E^* is admissible if all edges incident to honest nodes are defined, all edges between two corrupt nodes are undefined, the honest nodes are connected, and every corrupt node has an honest neighbor.

5.2 Construction with Large Messages

We begin by giving a warmup construction of rdSNARGs, without using all the tools developed in the previous section, in which the parties send possibly large messages.

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

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

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

Let $\mathcal{L} \in \mathbf{P}$ be a distributed language over configurations represented as adjacency lists (as above).

Construction 5.2 (rdSNARG with large messages). Our construction of an rdSNARG for $\mathcal{L}$ uses the following ingredients:

- A hash family with local openings $\text{HT} = (\text{HT.Gen}, \text{HT.Hash}, \text{HT.Open}, \text{HT.Verify})$.
- A RAM SNARG scheme $\text{RAM} = (\text{RAM.Gen}, \text{RAM.P}, \text{RAM.V})$ with respect to HT with regular soundness (as [Definition 3.10](#)) for the RAM machine $\mathcal{M}_{\mathcal{L}}$ which checks that the graph has undirected edges and decides membership in $\mathcal{L}$, which we assume never reads past $\perp$ in its memory.

$\text{Gen}(1^\lambda, 1^n) \rightarrow \text{crs}$.

1. Sample $\text{hk} \leftarrow \text{HT.Gen}(1^\lambda)$.
2. Sample $\text{RAM.crs} \leftarrow \text{RAM.Gen}(1^\lambda, T)$ where $T = \text{poly}(n)$ is a bound on the runtime of $\mathcal{M}_{\mathcal{L}}$.
3. Output $\text{crs} = (n, \text{hk}, \text{RAM.crs})$.

$\mathcal{P}(\text{crs}, G, x) \rightarrow \pi$.

1. Parse $\text{crs} = (n, \text{hk}, \text{RAM.crs})$.

2. For each $i \in [n + 1]$, compute $h_i = \text{HT.Hash}(\text{hk}, L_{G,x}[i])$.
3. Compute $C = \text{HT.Hash}(\text{hk}, (h_i)_{i \in [n+1]})$.
4. For $i \in [n + 1]$, compute $\rho_i = \text{HT.Open}(\text{hk}, (h_i)_{i \in [n+1]}, i)$.
5. Compute $\text{RAM}.\pi = \text{RAM}.\mathcal{P}(\text{RAM.crs}, L_{G,x})$. (Recall $\text{RAM}.\mathcal{P}$ is w.r.t. $\mathcal{M}_{\mathcal{L}}$.)
6. For every $i \in [n]$, set $\pi(v_i) = (C, i, \rho_i, \rho_{n+1}, \text{RAM}.\pi)$.
7. Output $\pi(v_i)$ at each node v_i .

$\mathcal{V}(\text{crs}; G; x, \pi) \rightarrow 0/1$ at node $v \in V(G)$.

1. Parse $\text{crs} = (n, \text{hk}, \text{RAM.crs})$.
2. Parse $\pi(v) = (C, i, \rho_i, \rho_{n+1}, \text{RAM}.\pi)$.
3. Check that $\text{HT.Verify}(\text{hk}, C, n + 1, \text{HT.Hash}(\text{hk}, \perp), \rho_{n+1}) = 1$.
4. Check that $1 \leq i \leq n$. (This is the party's alleged index.)
5. Compute $h_i = \text{HT.Hash}(\text{hk}, (ID(v), x(v), N(v)))$.
6. Check that $\text{HT.Verify}(\text{hk}, C, i, h_i, \rho_i) = 1$.
7. Check that $\text{RAM}.\mathcal{V}(\text{RAM.crs}, \text{hk}, C, 1, \text{RAM}.\pi) = 1$.
8. Send $C, i, \rho_i, x(v), N(v)$ to every neighbor $v_j \in N(v)$.
9. For $v_j \in N(v)$,
 - (a) Receive $C_j, j, \rho_j, x(v_j), N(v_j)$.
 - (b) Check that $C_j = C$.
 - (c) Check that $1 \leq j \leq n$.
 - (d) Construct $L_{G,x}[j] = (ID(v_j), x(v_j), N(v_j))$.
 - (e) Compute $h_j = \text{HT.Hash}(\text{hk}, L_{G,x}[j])$.
 - (f) Check that $\text{HT.Verify}(\text{hk}, C, j, h_j, \rho_j) = 1$.
10. Output 1 if and only if all checks pass.

Theorem 5.3. *Assuming the existence of a collision-resistant hash family with local openings (as in [Definition 3.1](#)) and a RAM SNARG scheme (as in [Definition 3.10](#)) , [Construction 5.2](#) is an rdSNARG on n nodes and up to $t < n$ corruptions, with CRS of size $\text{poly}(\lambda, \log n)$, certificates of size $\text{poly}(\lambda, \log n)$, and a single round of node-to-node messages of size $d \cdot \text{poly}(\lambda, \log n) + m$, where d is the degree of the network and m is a bound on the size of the assignments from x .*

Proof. Completeness and efficiency follow from the completeness and efficiency of HT and RAM.

Succinctness. The CRS consists of the value n , a hash key $\mathbf{hk}$, and a RAM SNARG CRS RAM.crs , all of which are short by succinctness of HT and RAM. The certificates consist of succinct hash values, local openings, and a succinct RAM SNARG certificate, so have size $\text{poly}(\lambda, \log n)$. The node-to-node messages consist of an index between 1 and n , a node's neighborhood, their assignment from x , succinct hash values, and local openings, so have size $d \cdot \text{poly}(\lambda, \log n) + m$ where d is the maximum size of a node's neighborhood and m is the maximum size of a node's assignment $x(v)$.

Soundness. Suppose for the sake of contradiction that there exists a poly-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and a non-negligible function $\epsilon(\cdot)$ such that for any $\lambda \in \mathbb{N}$

$$\Pr \left[\begin{array}{l} E^* \text{ is admissible} \\ (G^*, x^*) \text{ not consistent with } (G, x) \in \mathcal{L} : \\ \mathcal{V}(\text{crs}; G^*; x^*, \pi, \text{msg}_{\text{cor} \rightarrow \text{hon}}) = 1 \end{array} \quad \begin{array}{l} (V_{\text{hon}}, V_{\text{cor}}, \text{st}) \leftarrow \mathcal{A}_1(1^\lambda) \\ \text{crs} \leftarrow \text{Gen}(1^\lambda, 1^{|V|}, |V_{\text{cor}}|) \\ (E^*, x^*, \pi, \text{msg}_{\text{cor} \rightarrow \text{hon}}) \leftarrow \mathcal{A}_2(\text{crs}, \text{st}) \\ G^* = (V = V_{\text{hon}} \cup V_{\text{cor}}, E^*) \end{array} \right] \geq \epsilon(\lambda)$$

where we say that E^* is admissible if all edges incident to honest nodes are defined, all edges between two corrupt nodes are undefined, the honest nodes are connected, and every corrupt node has an honest neighbor.

We now show soundness against t corruptions. First, we reason about the indices received by honest nodes and claimed by corrupt nodes.

- By the fact that honest nodes are connected, and the check in [Item 9b](#), we have that there is a global consistent C .
- By the check in [Item 6](#), collision-resistance of HT, and unique IDs, all honest nodes must have received distinct indices between 1 and n .
- Fix an arbitrary corrupt node v_j . This node must have an honest neighbor, who we denote by v_i .
- Let $n, C, j, x(v_j), N(v_j)$ be what v_i receives from v_j .
- In [Item 9f](#), v_i will verify an opening of C to $ID(v_j), x(v_j), N(v_j)$ at j . Recall we assume that v_i knows the true value of $ID(v_j)$.
- By collision-resistance of HT and unique IDs, it cannot be that any honest node received j as their claimed index, or that any other corrupt node claimed j was their index to their honest neighbor.
- By pigeonhole principle, it must be that the indices received by all honest nodes and the indices claimed by all corrupt nodes must include all values between 1 and n .

Each of these indices corresponds to a verified opening to part of the memory digested into C , and nodes also verify an opening to the marker at the end of the memory. Now we have $n + 1$ verified openings to a complete memory for C . Furthermore, nodes verify a RAM SNARG proof that $\mathcal{M}_{\mathcal{L}}$

outputs 1 on the memory digested into C . Thus we have by extended completeness and soundness of the RAM SNARG that the machine $\mathcal{M}$ must output 1 on the memory, and thus that the memory contains a well-defined graph G with undirected edges and that $(G, x) \in \mathcal{L}$. $\square$

5.3 Construction with Succinct Messages

Given a configuration (G, x) , let $V = \{v_1, \dots, v_n\}$ be the nodes of G in lexicographic order. (The order does not matter, but for concreteness, since the prover will use it, we fix a specific ordering.) Here, we do not assume anything about the UUIDs of the nodes, other than the standard assumption that they can be represented in $O(\log n)$ bits. Let E be an adjacency matrix representation of G : E has length $\binom{n}{2}$, where for each $i, j \in [n]$ with $i < j$, $E[i, j] \in \{0, 1\}$. Representing the network as an adjacency matrix means that we can avoid the need to have the RAM machine check that it is well-defined with undirected edges, as we needed to in the previous section.

Let $\mathcal{L} \in \mathbf{P}$ be a distributed language over networks represented via an adjacency matrix.

In this section, we construct an **rdSNARG** for any distributed language $\mathcal{L}$ in $\mathbf{P}$ which is robust to t corruptions where the certificate at each node is of size $\binom{t}{2}/(n-t) \cdot \text{poly}(\lambda, \log n)$.

For the sake of clarity, we present here a construction in which we do not consider assignments $x : V(G) \rightarrow \{0, 1\}^*$. However, this construction extends to that setting, with corresponding efficiency. We refer to [Remark 5.1](#) for the details.

Construction 5.4 (rdSNARG with succinct messages). Our construction of an **rdSNARG** for $\mathcal{L}$ uses the following ingredients:

- A hash family with local openings $\text{HT} = (\text{HT.Gen}, \text{HT.Hash}, \text{HT.Open}, \text{HT.Verify})$.
- A fully local somewhere-extractable hash family

$$\text{fISEH} = (\text{fISEH.Gen}, \text{fISEH.Hash}, \text{fISEH.Digest}, \text{fISEH.Open}, \text{fISEH.Verify}, \text{fISEH.Extract}).$$

- A RAM SNARG scheme $\text{RAM}^{(V)} = (\text{RAM}^{(V)}.Gen, \text{RAM}^{(V)}.P, \text{RAM}^{(V)}.V)$ with respect to HT with regular soundness and extended completeness (as [Definition 3.10](#)) for the RAM machine $\mathcal{M}$ which takes as input a list of UUIDs V and decides if it is lexicographically ordered. We assume $\mathcal{M}$ never reads past $\perp$ in its memory.
- A RAM SNARG scheme $\text{RAM}^{(E)} = (\text{RAM}^{(E)}.Gen, \text{RAM}^{(E)}.P, \text{RAM}^{(E)}.V)$ with respect to HT and fISEH for the RAM machine $\mathcal{M}_{\mathcal{L}}$ which takes as input an adjacency matrix E and decides membership in $\mathcal{L}$, and has extract-or-open soundness as in [Definition 4.1](#).

$\text{Gen}(1^\lambda, 1^n, t) \rightarrow \text{crs}$.

1. Sample $\text{hk}^* \leftarrow \text{HT.Gen}(1^\lambda)$.
2. Let $L = \min\left(\binom{t}{2}, n-t\right)$.
3. For $i \in [n]$, sample $(\text{hk}_i, \text{vk}_i) \leftarrow \text{fISEH.Gen}(1^\lambda, \binom{n}{2}, \lceil \binom{t}{2}/L \rceil)$.
4. Sample $\text{RAM}^{(V)}.crs \leftarrow \text{RAM}^{(V)}.Gen(1^\lambda, T)$ where $T = \text{poly}(n)$ is a bound on the runtime of $\mathcal{M}$.

5. Sample $\text{RAM}^{(E)}.\text{crs} \leftarrow \text{RAM}^{(E)}.\text{Gen}(1^\lambda, T')$ where $T' = \text{poly}(n)$ is a bound on the runtime of $\mathcal{M}_{\mathcal{L}}$.
6. Output $\text{crs} = (n, \text{hk}^*, (\text{hk}_i, \text{vk}_i)_{i \in [n]}, \text{RAM}^{(V)}.\text{crs}, \text{RAM}^{(E)}.\text{crs})$.

$\mathcal{P}(\text{crs}, G) \rightarrow \pi$.

1. Parse $\text{crs} = (n, \text{hk}^*, (\text{hk}_i, \text{vk}_i)_{i \in [n]}, \text{RAM}^{(V)}.\text{crs}, \text{RAM}^{(E)}.\text{crs})$ and $G = (V, E)$.
2. Parse V as a lexicographically ordered list $(\text{uid}_1, \dots, \text{uid}_n)$.
3. Compute $\text{rt}^{(V)} = \text{HT.Hash}(\text{hk}^*, (\text{uid}_1, \dots, \text{uid}_n, \perp))$.
4. For $i \in [n + 1]$, compute $\rho_i^{(V)} = \text{HT.Open}(\text{hk}^*, (\text{uid}_1, \dots, \text{uid}_n, \perp), i)$.
5. Compute $\pi^{(V)} = \text{RAM}^{(V)}.\mathcal{P}(\text{RAM}^{(V)}.\text{crs}, \text{hk}, (\text{uid}_1, \dots, \text{uid}_n, \perp))$.

$$6. \text{ Parse } E = \begin{bmatrix} e_{1,n} & e_{1,n-1} & e_{1,n-2} & \cdots & e_{1,3} & e_{1,2} \\ e_{2,n} & e_{2,n-1} & e_{2,n-2} & \cdots & e_{2,3} & \\ \vdots & \vdots & \vdots & & & \\ e_{n-2,n} & e_{n-2,n-1} & & & & \\ e_{n-1,n} & & & & & \end{bmatrix} .$$

7. Compute “row-wise” hash and openings.

- For each $i \in \{1, \dots, n-1\}$, compute $\text{rt}_i^{(1)} = \text{HT.Hash}(\text{hk}^*, (e_{i,i+1}, \dots, e_{i,n}))$.
- Compute $\text{rt}^{(1)} = \text{HT.Hash}(\text{hk}^*, (\text{rt}_j^{(1)})_{j \in \{1, \dots, n-1\}})$.
- For each $i \in \{1, \dots, n-1\}$, compute $\rho_i^{(1)} = \text{HT.Open}(\text{hk}^*, (\text{rt}_j^{(1)})_{j \in \{1, \dots, n-1\}}, i)$.
- Set $\rho_n^{(1)} = \emptyset$.

8. Compute “column-wise” hash and openings.

- For each $i \in \{2, \dots, n\}$, compute $\text{rt}_i^{(2)} = \text{HT.Hash}(\text{hk}^*, (e_{1,i}, \dots, e_{i-1,i}))$.
- Compute $\text{rt}^{(2)} = \text{HT.Hash}(\text{hk}^*, (\text{rt}_j^{(2)})_{j \in \{2, \dots, n\}})$.
- For each $i \in \{2, \dots, n\}$, compute $\rho_i^{(2)} = \text{HT.Open}(\text{hk}^*, (\text{rt}_j^{(2)})_{j \in \{2, \dots, n\}}, i)$.
- Set $\rho_1^{(2)} = \emptyset$.

9. For $i \in [n]$, compute somewhere-extractable hash $\text{h}_i = \text{fISEH.Hash}(\text{hk}_i, E)$.

10. Compute $(\text{rt}_{\text{st}}, \pi_1^{(E)}, \dots, \pi_n^{(E)}) = \text{RAM}^{(E)}.\mathcal{P}(\text{RAM}^{(E)}.\text{crs}, \text{hk}^*, (\text{hk}_i, \text{vk}_i)_{i \in [n]}, E)$.

11. For every $i \in [n]$, set $\pi(v_i) = (i, \text{rt}^{(V)}, \rho_i^{(V)}, \rho_{n+1}^{(V)}, \pi^{(V)}, \text{rt}^{(1)}, \text{rt}^{(2)}, \rho_i^{(1)}, \rho_i^{(2)}, \text{h}_i, \text{rt}_{\text{st}}, \pi_i^{(E)})$.

12. Output $\pi(v_i)$ at each node v_i .

$\mathcal{V}(\text{crs}; G; \pi) \rightarrow 0/1$ at node $v \in V$.

1. Parse $\text{crs} = (n, \text{hk}^*, (\text{hk}_i, \text{vk}_i)_{i \in [n]}, \text{RAM}^{(V)}. \text{crs}, \text{RAM}^{(E)}. \text{crs})$.
2. Parse $\pi(v) = (i, \text{rt}^{(V)}, \rho_i^{(V)}, \rho_{n+1}^{(V)}, \pi^{(V)}, \text{rt}^{(1)}, \text{rt}^{(2)}, \rho_i^{(1)}, \rho_i^{(2)}, \text{h}_i, \text{rt}_{\text{st}}, \pi_i^{(E)})$.
3. Check that $1 \leq i \leq n$.
4. Check that $\text{HT.Verify}(\text{hk}^*, \text{rt}^{(V)}, i, \text{uid}(v), \rho_i^{(V)}) = 1$.
5. Check that $\text{HT.Verify}(\text{hk}^*, \text{rt}^{(V)}, n+1, \perp, \rho_{n+1}^{(V)}) = 1$.
6. Send $(i, \text{rt}_{\text{st}}, \text{rt}^{(V)}, \rho_i^{(V)}, \text{rt}^{(1)}, \text{rt}^{(2)})$ to every neighbor $v_j \in N(v)$.
7. For $v_j \in N(v)$,
 - (a) Receive $(j, \text{rt}_{\text{st},j}, \text{rt}_j^{(V)}, \rho_j^{(V)}, \text{rt}_j^{(1)}, \text{rt}_j^{(2)})$.
 - (b) Check that $\text{rt}_{\text{st}} = \text{rt}_{\text{st},j}$.
 - (c) Check that $\text{rt}_j^{(V)} = \text{rt}^{(V)}$, $\text{rt}_j^{(1)} = \text{rt}^{(1)}$, and $\text{rt}_j^{(2)} = \text{rt}^{(2)}$.
 - (d) Check that $1 \leq j \leq n$.
 - (e) Check that $\text{HT.Verify}(\text{hk}^*, \text{rt}^{(V)}, j, \text{uid}(v_j), \rho_j^{(V)}) = 1$.
8. Check that $\text{RAM}^{(V)}. \mathcal{V}(\text{RAM}^{(V)}. \text{crs}, \text{hk}^*, \text{rt}^{(V)}, 1, \pi^{(V)}) = 1$.
9. If $i \neq n$, check “row-wise” opening:
 - Compute $\text{rt}_i^{(1)} = \text{HT.Hash}(\text{hk}^*, (e_{i,i+1}, \dots, e_{i,n}))$.
 - Check that $\text{HT.Verify}(\text{hk}^*, \text{rt}^{(1)}, i, \text{rt}_i^{(1)}, \rho_i^{(1)})$.
10. If $i \neq 1$, check “column-wise” opening:
 - Compute $\text{rt}_i^{(2)} = \text{HT.Hash}(\text{hk}^*, (e_{1,i}, \dots, e_{i-1,i}))$.
 - Check that $\text{HT.Verify}(\text{hk}^*, \text{rt}^{(2)}, i, \text{rt}_i^{(2)}, \rho_i^{(2)})$.
11. Compute $\text{rt}_i = \text{flSEH.Digest}(\text{vk}_i, \text{h}_i)$.
12. Check that $\text{RAM}^{(E)}. \mathcal{V}(\text{RAM}^{(E)}. \text{crs}, (\text{hk}^*, \text{vk}_i), (\text{rt}^{(1)}, \text{rt}^{(2)}, \text{rt}_i), \text{rt}_{\text{st}}, \pi^{(E)}) = 1$.
13. Output 1 if and only if all checks pass.

For clarity, we write that $\text{RAM}^{(E)}$ is associated with HT. However, the collision-resistant hash family which $\text{RAM}^{(E)}. \mathcal{P}$ uses is actually a “joint” hash family, where the hash value consists of both a row-wise hash value and a column-wise hash value, and a hash opening consists of openings to both hash values.

Theorem 5.5. *Assuming the existence of a fully local somewhere-extractable hash family (as in Definition 3.6) and a flexible RAM SNARG scheme with extract-or-open soundness (as in Definition 4.1), then Construction 5.4 is an rdSNARG for $\mathcal{L}$ on n nodes and up to t corruptions, with CRS of size $n \cdot \lceil \binom{t}{2} / (n-t) \rceil \cdot \text{poly}(\lambda, \log n)$, certificates of size $\lceil \binom{t}{2} / (n-t) \rceil \cdot \text{poly}(\lambda, \log n)$, and a single round of node-to-node messages of size $\text{poly}(\lambda, \log n)$.*

Proof. Completeness and efficiency follow from the completeness, succinctness, and efficiency of the underlying primitives.

Succinctness. The messages sent between nodes consist of an index between 1 and n , succinct hash values, and a succinct opening, so have total size $\text{poly}(\lambda, \log n)$.

The size of the certificate at each node is dominated by h_i , which has size $\binom{t}{2}/L \cdot \text{poly}(\lambda, \log n)$, where $L = \min\left(\binom{t}{2}, n - t\right)$.

Soundness. Suppose for the sake of contradiction that there exists a poly-size adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and a non-negligible function $\epsilon(\cdot)$ such that for any $\lambda \in \mathbb{N}$

$$\Pr \left[\begin{array}{l} E^* \text{ is admissible} \\ G^* \text{ not consistent with } (G, x) \in \mathcal{L} \\ \mathcal{V}(\text{crs}; G^*; \pi, \text{msg}_{\text{cor} \rightarrow \text{hon}}) = 1 \end{array} : \begin{array}{l} (V_{\text{hon}}, V_{\text{cor}}, E^*, \text{st}) \leftarrow \mathcal{A}_1(1^\lambda) \\ \text{crs} \leftarrow \text{Gen}(1^\lambda, 1^{|V|}, |V_{\text{cor}}|) \\ (\pi, \text{msg}_{\text{cor} \rightarrow \text{hon}}) \leftarrow \mathcal{A}_2(\text{crs}, \text{st}) \\ G^* = (V = V_{\text{hon}} \cup V_{\text{cor}}, E^*) \end{array} \right] \geq \epsilon(\lambda)$$

where we say that E^* is admissible if all edges incident to honest nodes are defined, all edges between two corrupt nodes are undefined, the honest nodes are connected, and every corrupt node has an honest neighbor.

Define $\text{Gen}'(1^\lambda, 1^n, t, V_{\text{hon}}, V_{\text{cor}})$ which samples hk^* , $\text{RAM}^{(E)}. \text{crs}$, and $\text{RAM}^{(V)}. \text{crs}$ in the same way as Gen , but samples the fISEH keys as follows.

- Let $\{\alpha_1, \dots, \alpha_{n-t}\} \subseteq [n]$ be the indices between 1 and n corresponding to the UUIDs of the honest nodes in the lexicographically ordered list of vertices $V = V_{\text{hon}} \cup V_{\text{cor}}$.
- Let $I \subseteq \binom{[n]}{2}$ be the set of $\binom{t}{2}$ indices in the adjacency matrix for V which correspond to the edges between corrupt nodes.
- Construct disjoint sets $I_1, \dots, I_n$ of size at most $\binom{t}{2}/L$ (where $L = \min\left(\binom{t}{2}, n - t\right)$) such that $\cup_{i \in [n-t]} I_{\alpha_i} = I$ (the sets corresponding to corrupt nodes can be chosen arbitrarily).
- Sample $(\text{hk}_i, \text{vk}_i, \text{td}_i) \leftarrow \text{fISEH.Gen}(1^\lambda, \binom{|V|}{2}, I_i)$.

By index hiding of fISEH , there exists a negligible function μ such that

$$\Pr \left[\begin{array}{l} \forall v \in V_{\text{hon}}, u \in V, E^*[v, u] \in \{0, 1\} \\ V_{\text{hon}} \text{ are connected} \\ \forall v \in V_{\text{cor}}, \exists u \in V_{\text{hon}} \text{ s.t. } E^*[v, u] = 1 \\ G^* \text{ not consistent with any } G \in \mathcal{L} \\ \mathcal{V}(\text{crs}; G^*; \pi, \text{msg}_{\text{cor} \rightarrow \text{hon}}) = 1 \end{array} : \begin{array}{l} (V_{\text{hon}}, V_{\text{cor}}, E^*, \text{st}) \leftarrow \mathcal{A}(1^\lambda) \\ (\text{crs}, \text{td}) \leftarrow \text{Gen}'(1^\lambda, 1^{|V|}, |V_{\text{cor}}|) \\ (\pi, \text{msg}_{\text{cor} \rightarrow \text{hon}}) \leftarrow \mathcal{A}(\text{crs}, \text{st}) \\ G^* = (V = V_{\text{hon}} \cup V_{\text{cor}}, E^*) \end{array} \right] \geq \epsilon(\lambda) - \mu(\lambda).$$

Parse $\pi(v) = (i, \text{rt}^{(V)}, \rho_i^{(V)}, \rho_{n+1}^{(V)}, \pi^{(V)}, \text{rt}^{(1)}, \text{rt}^{(2)}, h_i, \rho_i^{(1)}, \rho_i^{(2)}, \pi_i^{(E)})$. First, we reason about the indices received by honest nodes and claimed by corrupt nodes in $\text{msg}_{\text{cor} \rightarrow \text{hon}}$.

- By the fact that honest nodes are connected and the check in [Item 7c](#), we have that there are global consistent values of $\text{rt}^{(V)}$, $\text{rt}^{(1)}$, and $\text{rt}^{(2)}$.
- By the check in [Item 4](#), collision resistance of HT, and unique IDs, all honest nodes must have received distinct indices between 1 and n .

- Fix an arbitrary corrupt node v_j . This node must have an honest neighbor, who we denote by v_i .
- Let $(j, \text{rt}_j^{(V)}, \rho_j^{(V)}, \text{rt}_j^{(1)}, \text{rt}_j^{(2)})$ be what v_i receives from v_j .
- In [Item 7e](#), v_i will verify an opening of $\text{rt}^{(V)}$ to $\text{uid}(v_j)$ (which v_i knows) at j .
- By collision-resistance of HT and unique IDs, it cannot be that any honest node received j as their claimed index, or that any other corrupt node claimed j was their index to their honest neighbor.
- By pigeonhole principle, it must be that the indices received by all honest nodes and the indices claimed by all corrupt nodes must include all values between 1 and n .

From now on, we will refer to both honest and corrupt nodes by these indices (i.e., v_i is the node who received or claimed index i).

Consider the memory $(\text{uid}(v_1), \dots, \text{uid}(v_n), \perp)$. We have openings w.r.t. $\text{rt}^{(V)}$ for the entire memory. By extended completeness and soundness of RAM SNARG, it must be that $(\text{uid}(v_1), \dots, \text{uid}(v_n))$ are lexicographically ordered except with negligible probability. We can conclude that the honest nodes receive the indices $\alpha_1, \dots, \alpha_{n-t}$ from the prover as were assigned to them during Gen' .

We know that the honest nodes are verifying openings w.r.t. $\text{rt}^{(1)}$ or $\text{rt}^{(2)}$ for all the bits of E incident to honest nodes, i.e., all the locations of E which are not in I . Since a hash opening for the hash family used by $\text{RAM}^{(E)}.\mathcal{P}$ consists of openings to both hash values, verifying an opening to one is still enough to use collision resistance. By RAM SNARG soundness, we have that if we define G by filling in the unknown edges of G^* with what can be extracted from each $\mathbf{h}_i$ using td_i , then $\mathcal{M}_{\mathcal{L}}(G) = 1 \implies G \in \mathcal{L}$. $\square$

Remark 5.1. [Construction 5.4](#) can be extended to handle proving properties of networks including assignments $x : V(G) \rightarrow \{0, 1\}^*$ as follows. Let $m = m(n)$ be the size of each local assignment $x(v)$ (for simplicity we assume each local assignment has the same size; this can be achieved w.l.o.g. by padding).

- We define the RAM machine $\mathcal{M}_{\mathcal{L}}$ to take as input both the adjacency matrix E and the assignment x to decide membership in $\mathcal{L}$.
- The adversary produces a partial assignment x^* for the honest nodes in the security game, which is also taken as input to the distributed verification algorithm.
- The hash key hk^* is also used to hash the assignments x , and the prover includes this hash value in each node's certificate, as well as a local opening to their assignment $x(v)$. Nodes exchange this hash value to ensure consistency and check these openings during verification (in addition to all the other checks they perform in [Construction 5.4](#)).
- L is set to be the minimum of $\binom{t}{2} + m \cdot t$ and $n - t$. The somewhere-extractable hash keys $(\text{hk}_i, \text{vk}_i)_{i \in [n]}$ are sampled to hash strings of length $\binom{n}{2} + |x|$ and to be extractable on $(\binom{t}{2} + m \cdot t) / L$ bits. They are used to hash both the adjacency matrix E and the assignments x .

The soundness analysis is identical to the analysis above, except that in Gen' , the set I is constructed to include the indices corresponding to the assignments for the corrupt nodes, i.e., $x(v)$ for $v \in V_{\text{cor}}$. Then, by RAM SNARG soundness, we have that if we define the configuration (G, x) by filling in the unknown values of (G^*, x^*) with what can be extracted from each h_i using td_i , we have that $\mathcal{M}_{\mathcal{L}}(G, x) = 1 \implies (G, x) \in \mathcal{L}$.

Remark 5.2. If instantiated with a *rate-1* fSEH family, the above construction achieves certificates of size $\binom{t}{2}/(n-t) + \text{poly}(\lambda)$ rather than size $\binom{t}{2}/(n-t) \cdot \text{poly}(\lambda)$. [DGKV22] constructs such a hash family from LWE .

6 Communication Lower Bound

In this section, we show that an rdSNARG which is asymptotically more succinct than our construction in fact implies a SNARG for NP .

We first recall the notion of a succinct non-interactive argument (SNARG).

Definition 6.1 (SNARG). A succinct non-interactive argument (SNARG) for an NP language $\mathcal{L}$ is a tuple of PPT algorithms $\text{SNARG} = (\text{SNARG.Gen}, \text{SNARG.P}, \text{SNARG.V})$ where

$\text{SNARG.Gen}(1^\lambda, 1^n, 1^m) \rightarrow \text{crs}$. This is a randomized setup algorithm that takes as input the security parameter λ , an input length n , a witness length m , and outputs a common reference string crs .

$\text{SNARG.P}(\text{crs}, x, w) \rightarrow \pi$. This is a prover algorithm that takes as input the common reference string crs , an input $x \in \{0, 1\}^n$, and a witness $w \in \{0, 1\}^m$, and outputs a proof π .

$\text{SNARG.V}(\text{crs}, x, \pi) \rightarrow 0/1$. This is a verifier algorithm that takes as input the common reference string crs , an input $x \in \{0, 1\}^n$, and a proof π , and outputs a value in $\{0, 1\}$.

The scheme is required to satisfy the following properties:

Efficiency. The proof has size $|\pi| = o(m)$. Moreover, the verifier runtime is $\text{poly}(\lambda, n)$.⁷

Completeness. For all $\lambda \in \mathbb{N}$, all polynomials $n = n(\lambda)$, $m = m(\lambda)$, and all $w \in \{0, 1\}^m$ such that $C_{\mathcal{L}}(x, w) = 1$, where $C_{\mathcal{L}}$ is the NP verification circuit for $\mathcal{L}$,

$$\Pr \left[\text{SNARG.V}(\text{crs}, x, \pi) = 1 \quad : \quad \begin{array}{l} \text{crs} \leftarrow \text{SNARG.Gen}(1^\lambda, 1^n, 1^m), \\ \pi \leftarrow \text{SNARG.P}(\text{crs}, x, w) \end{array} \right] = 1.$$

Soundness. For all sufficiently large λ and all poly-sized adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\Pr \left[\begin{array}{l} x \notin \mathcal{L} \\ \wedge \text{V}(\text{crs}, x^*, \pi^*) = 1 \end{array} \quad : \quad \begin{array}{l} (x, 1^m, \text{st}) \leftarrow \mathcal{A}_1(1^\lambda) \\ (\text{crs}) \leftarrow \text{SNARG.Gen}(1^\lambda, 1^n, 1^m), \\ (x^*, \pi^*) \leftarrow \mathcal{A}_2(\text{crs}, \text{st}) \end{array} \right] \leq \text{negl}(\lambda).$$

We now construct a SNARG for an NP language $\mathcal{L}$ with instances of length N and witnesses of length M . First, we pick n, t so that

⁷Note that this verifier efficiency can be generically obtained via RAM delegation techniques [CJJ22, KLVW23]. See Remark 2.7 of [WW24].

- $\binom{t}{2} = \Theta(M)$
- $n - t < o\left(\binom{t}{2}\right)$
- $\binom{n-t-1}{2} > N$.

Note that this restricts us to languages where $M = \omega(\sqrt{N})$, i.e., the witness length is asymptotically larger than the square root of the instance length.

Define the distributed language $\mathcal{L}'$ as follows: $G \in \mathcal{L}'$ if the instance parsed from the edges between the first $n - t - 1$ nodes and the witness parsed from the edges between the last t nodes are a satisfying instance-witness pair for the NP language $\mathcal{L}$.

Construction 6.2. We define a SNARG for $\mathcal{L}$ as follows, using an rdSNARG scheme $\text{rdSNARG} = (\text{rdSNARG.Setup}, \text{rdSNARG.P}, \text{rdSNARG.V})$ for $\mathcal{L}'$.

$\text{Gen}(1^\lambda, 1^N, 1^M) \rightarrow \text{crs}$.

1. Sample $\text{crs} \leftarrow \text{rdSNARG.Setup}(1^\lambda, 1^n, t)$, for n, t chosen as above.

$\mathcal{P}(\text{crs}, x, w) \rightarrow \Pi$.

1. Generate G as follows: encode x in the edges between the first $n - t - 1$ nodes and w in the edges between the last t nodes, and connect the $(n - t)$ th node to every other node (do not add any other edges).
2. Compute $(\pi_1, \dots, \pi_n) = \text{rdSNARG.P}(\text{crs}, G)$.
3. Compute the messages $\text{msg}_{\text{cor} \rightarrow \text{hon}}$ that the t corrupt nodes would send to the $(n - t)$ th node based on crs, G and the certificates $\pi_{n-t+1}, \dots, \pi_n$.
4. Output $\Pi = (\pi_1, \dots, \pi_{n-t}, \text{msg}_{\text{cor} \rightarrow \text{hon}})$.

$\mathcal{V}(\text{crs}, x, \Pi) \rightarrow 0/1$.

1. Generate G^* , i.e., the part of G incident to the honest nodes, as above.
2. Output 1 iff all honest nodes would accept their certificates $\pi_1, \dots, \pi_{n-t}$ when interacting with each other and receiving the messages in $\text{msg}_{\text{cor} \rightarrow \text{hon}}$.

Theorem 6.3. Assume an rdSNARG for distributed languages in $\mathbf{P}$ on n nodes with t corruptions, with certificates of size $\ell = o(\binom{t}{2}/(n - t))$ and node-to-node messages of size $o(t)$. Then for any $\mathcal{L} \in \mathbf{NP}$ for which the witness length $|w|$ is least $|w| \in \omega(\sqrt{|x|})$ of its input length $|x|$, it holds that [Construction 6.2](#) is a SNARG for $\mathcal{L}$.

Proof. Completeness follows from completeness of rdSNARG.

Succinctness. This follows from the fact that Π consists of $n - t$ certificates of size ℓ and t messages of size $o(t)$, so has size $\ell \cdot (n - t) + o(t) \cdot t$. Thus if $\ell = o(\binom{t}{2}/(n - t))$, then Π has size $o(\binom{t}{2}) = o(M)$.

Soundness. This follows from soundness of rdSNARG, as we now argue. Suppose towards contradiction there exists a cheating adversary $\mathcal{A}$ which breaks soundness of this SNARG. We build an adversary $\mathcal{B}$ which breaks soundness of rdSNARG using $\mathcal{A}$ as follows.

- $\mathcal{B}$ forwards 1^λ to $\mathcal{A}$ and receives false $x \notin \mathcal{L}$ and witness length 1^M .
- $\mathcal{B}$ selects n, t based on the input and witness size N, M of the language $\mathcal{L}$, as described above. $\mathcal{B}$ sets $V = [n]$ and chooses to corrupt the last t nodes, $V_{\text{cor}} = \{n - t + 1, \dots, n\}$.
- $\mathcal{B}$ generates E_{hon} from x as described above (and connects the $(n - t)$ th node to everyone), and sends

$$V_{\text{hon}}, V_{\text{cor}}, E_{\text{hon}}$$

to the rdSNARG challenger.

- After receiving crs from the rdSNARG challenger, $\mathcal{B}$ sends it to $\mathcal{A}$ and gets back accepting SNARG proof $\Pi = (\pi_1, \dots, \pi_{n-t}, \text{msg}_{\text{cor} \rightarrow \text{hon}})$ with non-negligible probability.
- $\mathcal{B}$ sends $(\pi_1, \dots, \pi_{n-t}, \text{msg}_{\text{cor} \rightarrow \text{hon}})$ to the rdSNARG challenger.

Since Π is accepting w.r.t. x , we have that all the honest verifiers accept. But since x is false, and thus there exists no valid witness, G_{hon} is not consistent with any $G \in \mathcal{L}'$. Thus $\mathcal{B}$ breaks soundness of rdSNARG. $\square$

Remark 6.1. In the above transformation, we construct a SNARG for $\mathcal{L}$. However, we note that if we were to require a stronger notion of *extractability* from the rdSNARG, which our construction in fact achieves, the transformation would yield a SNARK for $\mathcal{L}$. In particular, suppose we require that a network which is consistent with the honest view and satisfies the property can be extracted from any successful prover. Note that [Construction 5.4](#) satisfies this notion. An rdSNARG which simultaneously achieves this notion of extractability and the succinctness described above would yield a SNARK for NP after applying the above transformation.

References

- [ABC⁺23] Eden Aldema Tshuva, Elette Boyle, Ran Cohen, Tal Moran, and Rotem Oshman. Locally verifiable distributed SNARGs. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part I*, volume 14369 of *LNCS*, pages 65–90. Springer, Cham, November / December 2023. [2](#), [3](#), [4](#), [7](#), [8](#), [11](#), [12](#), [20](#), [28](#)
- [AO24] Eden Aldema Tshuva and Rotem Oshman. Fully Local Succinct Distributed Arguments. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:24, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. [3](#), [7](#)
- [AO26] Eden Aldema Tshuva and Rotem Oshman. Model-Generic Incrementally Verifiable Computation from Updatable BARGs. In Shubhangi Saraf, editor, *17th Innovations in Theoretical Computer Science Conference (ITCS 2026)*, volume 362 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:22, Dagstuhl, Germany, 2026. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. [7](#)

- [APSV91] Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction (extended abstract). In *32nd FOCS*, pages 268–277. IEEE Computer Society Press, October 1991. [1](#)
- [BBC⁺19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 67–97. Springer, Cham, August 2019. [7](#)
- [BDFO17] Alkida Balliu, Gianlorenzo D’Angelo, Pierre Fraigniaud, and Dennis Olivetti. What can be verified locally? In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, pages 8–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. [2](#)
- [BKO22] Aviv Bick, Gillat Kol, and Rotem Oshman. Distributed zero-knowledge proofs over networks. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2426–2458. SIAM, 2022. [2](#)
- [BSFO22] Yoav Ben Shimon, Orr Fischer, and Rotem Oshman. Proof labeling schemes for reachability-related problems in directed graphs. In *International Colloquium on Structural Information and Communication Complexity*, pages 21–41. Springer, 2022. [2](#)
- [CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 394–423, Virtual Event, August 2021. Springer, Cham. [18](#), [19](#)
- [CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd FOCS*, pages 68–79. IEEE Computer Society Press, February 2022. [3](#), [5](#), [22](#), [38](#)
- [CJW25] Liyan Chen, Zhengzhong Jin, and Daniel Wichs. Succinct non-interactive arguments of proximity. In *57th ACM STOC*, pages 955–964. ACM Press, June 2025. [5](#), [7](#)
- [DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *63rd FOCS*, pages 1057–1068. IEEE Computer Society Press, October / November 2022. [15](#), [38](#)
- [DHK⁺11] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 363–372. ACM Press, June 2011. [2](#)
- [FBP22] Laurent Feuilloley, Nicolas Bousquet, and Théo Pierron. What can be certified compactly? compact local certification of mso properties in tree-like graphs. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 131–140, 2022. [2](#)
- [Feu21] Laurent Feuilloley. Introduction to local certification. *Discrete Mathematics & Theoretical Computer Science*, 23(Distributed Computing and Networking), 2021. [2](#), [6](#)

- [FF22] Laurent Feuilloley and Pierre Fraigniaud. Error-sensitive proof-labeling schemes. *Journal of Parallel and Distributed Computing*, 166:149–165, 2022. [2](#), [6](#)
- [FFH⁺21] Laurent Feuilloley, Pierre Fraigniaud, Juho Hirvonen, Ami Paz, and Mor Perry. Redundancy in distributed proofs: L. feuilloley et al. *Distributed Computing*, 34(2):113–132, 2021. [2](#)
- [FGKS13] Pierre Fraigniaud, Mika Göös, Amos Korman, and Jukka Suomela. What can be decided locally without identifiers? In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 157–165, 2013. [2](#)
- [FHK12] Pierre Fraigniaud, Magnús M Halldórsson, and Amos Korman. On the impact of identifiers on local decision. In *International Conference On Principles Of Distributed Systems*, pages 224–238. Springer, 2012. [2](#)
- [FMO⁺19] Pierre Fraigniaud, Pedro Montealegre, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. On distributed merlin-arthur decision protocols. In *International Colloquium on Structural Information and Communication Complexity*, pages 230–245. Springer, 2019. [2](#)
- [FMRT24] Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. A meta-theorem for distributed certification. *Algorithmica*, 86(2):585–612, 2024. [2](#)
- [FPSP19] Pierre Fraigniaud, Boaz Patt-Shamir, and Mor Perry. Randomized proof-labeling schemes: P. fraigniaud et al. *Distributed Computing*, 32(3):217–234, 2019. [2](#)
- [GJKP26] Paweł Garncarek, Tomasz Jurdzinski, Dariusz Kowalski, and Subhajit Pramanick. Error-resilient distributed local verification. *arXiv preprint arXiv:2603.20831*, 2026. [2](#)
- [GS16] Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12(1):1–33, 2016. [2](#)
- [HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 520–549. Springer, Cham, May / June 2022. [19](#)
- [HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015. [9](#), [15](#)
- [KKP05] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 9–18, 2005. [1](#), [2](#), [6](#), [7](#)
- [KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 1545–1552. ACM Press, June 2023. [3](#), [5](#), [9](#), [10](#), [15](#), [16](#), [17](#), [19](#), [20](#), [22](#), [38](#)

- [KOS18] Gillat Kol, Rotem Oshman, and Raghuvansh R Saxena. Interactive distributed proofs. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 255–264, 2018. [2](#)
- [KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1115–1124. ACM Press, June 2019. [3](#), [5](#), [19](#), [22](#)
- [KR15] Yael Tauman Kalai and Ron D. Rothblum. Arguments of proximity - [extended abstract]. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 422–442. Springer, Berlin, Heidelberg, August 2015. [5](#), [7](#)
- [KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 330–368. Springer, Cham, November 2021. [18](#)
- [Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO’87*, volume 293 of *LNCS*, pages 369–378. Springer, Berlin, Heidelberg, August 1988. [13](#), [15](#)
- [NPY20] Moni Naor, Merav Parter, and Eylon Yogev. The power of distributed verifiers in interactive proofs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1096–1115. SIAM, 2020. [2](#)
- [OPR17] Rafail Ostrovsky, Mor Perry, and Will Rosenbaum. Space-time tradeoffs for distributed verification. In *International Colloquium on Structural Information and Communication Complexity*, pages 53–70. Springer, 2017. [2](#)
- [OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 121–145. Springer, Berlin, Heidelberg, November / December 2015. [9](#), [15](#)
- [Pel00] David Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, 2000. [2](#), [11](#), [12](#)
- [PSP17] Boaz Patt-Shamir and Mor Perry. Proof-labeling schemes: Broadcast, unicast and in between. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 1–17. Springer, 2017. [2](#)
- [WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 433–463. Springer, Cham, August 2022. [19](#)
- [WW24] Brent Waters and David J. Wu. Adaptively-sound succinct arguments for NP from indistinguishability obfuscation. In Bojan Mohar, Igor Shinkar, and Ryan O’Donnell, editors, *56th ACM STOC*, pages 387–398. ACM Press, June 2024. [38](#)