Boosting Batch Arguments and RAM Delegation

Yael Tauman Kalai
Microsoft Research and MIT

Alex Lombardi*
Simons Institute and UC Berkeley

Vinod Vaikuntanathan†
MIT

Daniel Wichs
Northeastern University and NTT Research

March 28, 2023

Abstract

We show how to *generically* improve the succinctness of non-interactive publicly verifiable batch argument (BARG) systems. In particular, we show (under a mild additional assumption) how to convert a BARG that generates proofs of length $\mathsf{poly}(m) \cdot k^{1-\epsilon}$, where m is the length of a single instance and k is the number of instances being batched, into one that generates proofs of length $\mathsf{poly}(m, \log k)$, which is the gold standard for succinctness of BARGs. By prior work, such BARGs imply the existence of SNARGs for deterministic time T computation with optimal succinctness $\mathsf{poly}(\log T)$.

Our result reduces the long-standing challenge of building publicly-verifiable delegation schemes to a much easier problem: building a batch argument system that *beats the trivial construction.* It also immediately implies new constructions of BARGs and SNARGs with polylogarithmic succinctness based on either bilinear maps or a combination of the DDH and QR assumptions.

Along the way, we prove an *equivalence* between BARGs and a new notion of SNARGs for (deterministic) RAM computations that we call "*flexible* RAM SNARG*s with partial input soundness*." This is the first demonstration that SNARGs for deterministic computation (of any kind) imply BARGs. Our RAM SNARG notion is of independent interest and has already been used in a recent work on constructing rate-1 BARGs (Devadas et. al. FOCS 2022).

*This research was conducted in part while the author was at MIT, where he was supported by a Charles M. Vest fellowship and the grants of the third author.

†This research was supported in part by DARPA under Agreement No. HR00112020023, a grant from the MIT-IBM Watson AI, a grant from Analog Devices, a Microsoft Trustworthy AI grant, and a Thornton Family Faculty Research Innovation Fellowship from MIT. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

Contents

1 Introduction

Efficient verification of computation is one of the most fundamental problems in theoretical computer science. Recently, with the increasing popularity of blockchain technologies and cloud services, efficient verification schemes are increasingly deployed in practice. This reality motivates the study of *succinct non-interactive arguments* (SNARGs) [Mic94], which are short, easy to verify, and computationally sound proofs that a statement x belongs to a potentially complex language $\mathcal{L}$. We would ideally like to construct SNARGs, given a (short, efficient-to-generate) common reference string, for any language $\mathcal{L}$ decidable in non-deterministic time $T(|x|)$, where the SNARG has proof size $\mathsf{poly}(\lambda, \log T)$ and verification time $\mathsf{poly}(\lambda, \log T) + \tilde{O}(|x|)$ given security parameter λ. In the random oracle model, such SNARGs were already constructed in a seminal work of Micali [Mic94]; however, constructing SNARGs in the "plain model" under falsifiable and preferably standard cryptographic assumptions remains a grand challenge, and will require overcoming some serious barriers [GW11].

In this work, we study two different forms of SNARGs for *restricted* computations: SNARGs for *deterministic* time-T computations and SNARGs for *batch*-NP computations (BARGs).

SNARGs for deterministic time-T computations. There are a number of recent[1] constructions of this form of SNARG based on falsifiable and standard assumptions:

- Kalai, Paneth and Yang [KPY19] constructed SNARGs with succinctness $\mathsf{poly}(\lambda, T^{\epsilon})$ for any constant $\epsilon > 0$ under a falsifiable assumption on groups with bilinear maps.
- Jawale, Kalai, Khurana and Zhang [JKKZ21] constructed SNARGs for any size S and depth D (log-space uniform) computation with succinctness $D \cdot \mathsf{poly}(\lambda, \log S)$ under LWE.
- Very recently, Choudhuri, Jain and Jin [CJJ21a] constructed SNARGs with succinctness $\mathsf{poly}(\lambda, \log T)$ from LWE.

SNARGs for batch NP computations (BARGs). In a BARG scheme, the prover wants to prove k NP statements $x_1, \ldots, x_k$ (given witnesses $w_1, \ldots, w_k$) with communication complexity (and verification time) significantly smaller than $k \cdot m$, where each witness w_i is at most m bits long. The key parameter of interest here is k, the batch size, and we want protocols with a sub-linear (and ideally, poly-logarithmic) dependence on k.

There have also been a number of recent constructions of BARGs from standard assumptions:

- Choudhuri, Jain and Jin [CJJ21b] constructed a BARG with succinctness $\mathsf{poly}(\lambda, m) \cdot \sqrt{k}$ under a combination of the Quadratic Residuosity (QR) and Decisional Diffie-Hellman DDH assumptions (or QR and LWE, but this is subsumed below).
- The work of [CJJ21a] constructed a BARG with succinctness $\mathsf{poly}(\lambda, m, \log k)$ under LWE. Indeed, their BARG was the key building block in their construction of a SNARG for deterministic polynomial-time computations.

[1] We use SNARGs to refer to publicly verifiable SNARGs, which can be verified given the crs alone. We mention that there is a line of work, starting with [KRR13, KRR14], that constructed various privately verifiable SNARGs under standard assumptions. Since our focus is on publicly verifiable SNARGs, we do not elaborate on these works here.

- Building upon [CJJ21b], Hulett, Jawale, Khurana and Srinivasan [HJKS22] constructed a BARG with succinctness $\mathsf{poly}(\lambda, m) \cdot k^{\epsilon}$ for any constant $\epsilon > 0$, under QR and DDH.
- More recently, Waters and Wu [WW22] constructed a BARG with succinctness $\mathsf{poly}(\lambda, m) \cdot k^{\epsilon}$ under the DLIN assumption (or relaxations of it[2]) on bilinear maps.

BARGs imply SNARGs for P. The recent constructions of BARGs and SNARGs for P are quite closely tied together: two recent works [CJJ21a, KVZ21] showed that a BARG for batch NP with L-parameterized[3] succinctness $\mathsf{poly}(m) \cdot L(k, \lambda)$ *implies* a SNARG for any time-T computation with succinctness $\mathsf{poly}(\lambda) \cdot L(T, \lambda)$ assuming the existence of somewhere extractable succinct commitments with local opening. (We show in Section 4 that such commitments can be constructed from any rate-1 string OT, which in turn can be constructed based on any of the following assumptions: LWE, DDH, $O(1)$-LIN, QR, or DCR [DGI+19].)

The [CJJ21a, KVZ21] transformation requires that the underlying BARG satisfy one key property regarding *verification time* that we will also assume throughout this work.[4] Namely, the BARG must be compatible with *succinct implicit inputs*: if a time $\ell = \ell(n)$ Turing machine generates x_i given i as input (for all i), then verifying the BARG for $(x_1, \ldots, x_k)$ can be done in time polynomial in ℓ and (as before) sublinear or better in $k \cdot m$. Note that this means the verifier does not necessarily have to *read* the k statements[5] (separately), but only their implicit description. Throughout this introduction, when we refer to a BARG, we assume it has this verifier efficiency guarantee.

These two works changed the focus of the community from constructing SNARGs to constructing BARGs. Indeed, the recent BARG constructions of [CJJ21a, HJKS22, WW22] all imply constructions of similarly efficient SNARGs for P.

1.1 This Work

Both primitives discussed above – SNARGs for P and BARGs for NP – have a "gold standard" for succinctness and verifier efficiency:

- SNARGs for time T (deterministic) computation with communication complexity $\mathsf{poly}(\lambda, \log T)$.
- BARGs with communication complexity $\mathsf{poly}(\lambda, m, \log k)$.
- For both BARGs and SNARGs, verification can require an additional quasi-linear time to process explicit inputs but should otherwise match the communication complexity bounds.

However, of all of the recent constructions discussed so far, only [JKKZ21] (for bounded-depth deterministic computation) and [CJJ21a] actually match this efficiency. The others [KPY19, CJJ21b, HJKS22, WW22] achieve *sublinear* (and sometimes sub-polynomial) but not polylogarithmic dependence on T and/or k. In this work, our main question centers on achieving optimal succinctness for BARGs and SNARGs:

[2] They can rely on the "k-LIN assumption" for arbitrary constant $k \geq 1$, which we refrain from writing due to notation collision with the batch size k. We will sometimes refer to this as the $O(1)$-LIN assumption.

[3] These works only considered the case $L(k, \lambda) = \mathsf{poly}(\lambda, \log(k))$, but their results readily extend to any L.

[4] In [CJJ21a], a BARG with this efficiency guarantee was referred to as a BARG for index languages. In this work, we actually only assume the existence of an object that is somewhat weaker than an index BARG; see Remark 3.5.

[5] Since a verifier (in general) must read its entire input in order to decide whether to accept a proof, a naive BARG for k NP statements would require the verifier to run for at least $k \cdot n$ time. This is unsatisfactory if the goal is to have efficiency sublinear in k.

Question 1.1. *When is it possible to build* BARG*s and* SNARG*s with* polylogarithmic *(w.r.t. k or T) succinctness and verifier efficiency?*

A second question we ask is whether there is any (partial) converse to the [CJJ21a, KVZ21] result that BARGs imply SNARGs for P:

Question 1.2. *Does some kind of* SNARG *for deterministic computation imply* BARG*s for* NP*?*

A positive answer would establish a (loose) *equivalence* between these two kinds of argument systems.

Our Results. We answer Question 1.1 by giving a generic procedure for *boosting* the efficiency of BARGs. Specifically, we show how to convert any BARG with succinctness $\mathsf{poly}(m) \cdot k/p(\lambda)$, for some (sufficiently large) polynomial $p(\lambda)$, into one with succinctness $\mathsf{poly}(\lambda, m, \log k)$.

Theorem 1.3 (Informal, see Theorem 8.1)**.** *There is a polynomial p such that if there exists*

- *A* BARG *for* NP *with succinctness* $\mathsf{poly}(m) \cdot k/p(\lambda)$ *for all sufficiently large* $k \geq \mathsf{poly}(\lambda)$ *(and efficient verifier), and*
- *A rate-1 (2-message) string OT, which can be constructed based on* LWE*,* DDH*,* $O(1)$-LIN*,* QR*, or* DCR [DGI+19]*.*

then there exists a BARG *for* NP *with succinctness* $\mathsf{poly}(\lambda, m, \log k)$ *(and efficient verifier).*

We briefly give some remarks on Theorem 1.3:

- A BARG with succinctness matching the hypothesis of Theorem 1.3 follows from the existence of a BARG with succinctness $\mathsf{poly}(\lambda, m) \cdot k^{1-\delta}$ for any constant $\delta > 0$.
  - If we make a subexponential security assumption, it is even possible to start with any BARG with succinctness $\mathsf{poly}(\lambda, m) \cdot \frac{k}{(\log k)^{\omega(1)}}$, by setting the security parameter $\lambda = \mathsf{poly} \log(k\lambda')$. However, the resulting fully succinct BARG will only be secure against adversaries that run in time quasi-polynomial in $m\lambda'$, which is meaningful but not ideal.
- Theorem 1.3 can start with BARGs with a *long* ($\mathsf{poly}(k, m, \lambda)$) common reference string, as long as the verifier efficiency remains $\mathsf{poly}(m) \cdot k/p(\lambda)$ (taking as input only a designated part of the crs of appropriate size). This is because a simple transformation can be used to first reduce the size of the crs while preserving the above sublinear succinctness bound: to prove k statements, pick a small constant $\epsilon > 0$ and execute $k^{1-\epsilon}$ copies of the initial BARG with batch size k^{ϵ} (re-using a single crs).
- The rate-1 string OT is used (solely) to construct a somewhere extractable hash (SEH) family with local opening (Section 4). This matches[6] what is required in the generic transformations of [CJJ21a, KVZ21]. We state Theorem 1.3 using string OT simply to highlight the variety of instantiations of the building block, many of which were not previously known (Lemma 4.5).

[6] For simplicity of the write-up, our SEH definition differs slightly from what was used in [CJJ21a]. We make use of a form of *deterministic* succinct commitment schemes, while [CJJ21a] allows randomization. However, such schemes can always be derandomized with a (public seed) PRF, so the primitives are equivalent.

- We emphasize that our results (and proofs) are *entirely in the setting of non-interactive arguments*. Unlike prior work such as [CCH+19, JKKZ21, CJJ21a, CJJ21b, HJKS22], we do *not* make explicit use of interactive proofs or the Fiat-Shamir transform, but instead generically convert a weakly succinct (non-interactive) BARG into a strongly succinct BARG.

Theorem 1.3, combined with the works of [CJJ21a, KVZ21], reduces the problem of constructing ideal SNARGs for time-T computations to constructing *any non-trivial* BARG, one that is slightly more succinct that simply sending all the NP witnesses in the clear.

As corollaries to Theorem 1.3, we obtain multiple new constructions of BARGs for NP and SNARGs for P:

- Together with the result of [WW22], Theorem 1.3 gives a BARG for NP with proof size $\mathsf{poly}(\lambda, m, \log k)$ and a SNARG for time-T computations of size $\mathsf{poly}(\lambda, \log T)$ (as opposed to size $\mathsf{poly}(\lambda, m, k^\epsilon)$ and $\mathsf{poly}(\lambda, T^\epsilon)$) from DLIN (or $O(1)$-LIN) on bilinear maps. Moreover, this BARG can be obtained from the "base" scheme of [WW22] without their "bootstrapping" step.
- Together with the result of [CJJ21b], Theorem 1.3 gives BARGs and SNARGs with the above efficiency from QR and DDH, as opposed to having a $\sqrt{k}$ dependence in [CJJ21b] or a k^ϵ (or T^ϵ) dependence in [HJKS22].

Perhaps more importantly, we believe Theorem 1.3 is an important foundation that will lead to new constructions of BARGs and SNARGs, since it reduces this goal to a significantly easier problem.

In order to prove Theorem 1.3, we also obtain an answer to Question 1.2 by considering the setting of RAM delegation [KP16, BHK17]. We define (and construct) a new notion of RAM SNARG, which we call a *flexible* RAM SNARG with *partial input soundness* (Definition 6.2). We then prove:

Theorem 1.4 (informal, see Theorem 8.3)**.** *Assuming the existence of rate-1 string OT,* BARG*s for* NP *are existentially* equivalent *to flexible* RAM SNARG*s with partial input soundness.*

We prove both directions of this equivalence (assuming rate-1 string OT):

1. BARGs for NP imply flexible RAM SNARGs in an efficiency-preserving manner. This is a strengthening of the [CJJ21a, KVZ21] result, which only constructs a weaker form of RAM delegation from BARGs.
2. Flexible RAM SNARGs, *even with barely non-trivial succinctness*, imply BARGs with succinctness $\mathsf{poly}(\lambda, m, \log k)$.

Sequentially combining these two transformations yields Theorem 1.3. Combining them the opposite order also implies that the succinctness of flexible RAM SNARGs can be boosted.

In addition to facilitating Theorem 1.3, we believe that our notion of flexible RAM SNARGs is of independent interest; indeed, it has already been used to obtain a simplified rate-1 BARG in [DGKV22].

2 Our Techniques

Somewhere extractable BARGs (seBARGs). Before diving into our techniques, we first simplify our problem by replacing BARGs with a slightly stronger primitive *without loss of generality*. Namely,

we consider *somewhere extractable* BARGs, hereafter referred to as seBARGs. A BARG is defined to be somewhere extractable if for some (hidden) choice of i, given an appropriate trapdoor td for the crs, it is possible to *extract* a witness w_i for x_i given a valid BARG proof. This is essentially an argument of knowledge property for BARGs.

Conveniently, assuming the existence of somewhere extractable hash functions with local opening, BARGs can easily be modified to be somewhere extractable with the standard "commit-and-prove" approach (for example, this was used implicitly in [CJJ21a, KVZ21]). From now on we work directly with seBARGs instead of BARGs, since seBARGs are an easier-to-manipulate primitive.

Organization. We now give an overview of our proofs of both directions of Theorem 1.4; these together also imply Theorem 1.3. We begin by recalling RAM delegation and give intuition for why it should be useful for constructing BARGs. We then discuss our new notion of flexible RAM SNARGs with partial input soundness and sketch our main proofs.

RAM delegation. A RAM SNARG, originally defined in [KP16], is similar to a SNARG for deterministic computations but tailored to the RAM computational model. Concretely, we consider the simplified setting of read-only RAM computation. A (read-only) RAM algorithm is given query-access to a large input x (often referred to as its "memory") and returns some output y. Queries to memory are considered unit-cost operations.

In a RAM SNARG, the prover wants to convince the verifier that $M(x) = y$ for some RAM machine M, input x, and output y. In addition to wanting verification that is efficient compared to the *runtime* of $M(x)$, it is also desired that verification is sublinear (preferably polylogarithmic) in the length of the input x. However, the verifier must have some "handle" on x in order for verification to be possible; to do so, the verifier is given a *digest* $\mathsf{d} = \mathsf{Digest}(\mathsf{crs}, x)$, which can roughly be thought of as a Merkle tree commitment to x.

Given this syntax, we arrive at an important question: what does it mean for a RAM SNARG to be *sound*? Observe that the map $x \mapsto \mathsf{Digest}(\mathsf{crs}, x)$ is many-to-one, so the input x is not information-theoretically defined from the point of view of the verifier. Thus, $\mathsf{Digest}(\mathsf{crs}, \cdot)$ is always required to be collision-resistant, capturing the intuition that the prover should be committed to some particular input x when it sends d.

In prior work [KP16, BHK17, KPY19, CJJ21a], soundness was formulated in two different ways:

- In [KP16, BHK17], a RAM SNARG is defined to be sound if it is computationally hard to prove two *contradictory* statements; namely, to prove that both $M(x) = 0$ and $M(x) = 1$ with the same machine M and digest d (note that a specific input x does not necessarily exist in this security notion; the prover may not actually know one).
- In [KPY19, CJJ21a], a *weaker* security property was used: a RAM SNARG was defined to be sound if it is computationally hard to *simultaneously* (1) make the verifier accept with machine M, digest d, and output y and (2) produce an input x such that $M(x) \neq y$ and $\mathsf{Digest}(\mathsf{crs}, x) = \mathsf{d}$. Note that the [KP16, BHK17] security definition implies this one.

Previous constructions of RAM SNARGs from standard assumptions [KPY19, CJJ21a] (and those that follow by combining [CJJ21a] with [CJJ21b, HJKS22, WW22]) were only shown to satisfy the weaker of the above two definitions. We emphasize that this soundness definition is quite weak: soundness is guaranteed only against an adversary that "knows" the entire memory x corresponding

to a digest d. In this work, we revisit the notion of RAM SNARGs and provide a new, stronger definition of soundness that overcomes this weakness and facilitates Theorems 1.3 and 1.4.

How to use SNARGs for RAM to build BARGs Before getting to our new definition, let us sketch why SNARGs for RAM are useful for constructing BARGs; the connection is surprisingly simple in hindsight. At a high level, the idea is for the prover P to treat its k NP witnesses $w_1, \ldots, w_k$ as the *memory* of a RAM machine. Thus, the prover will compute $\mathsf{d} = \mathsf{Digest}(\mathsf{crs}, w_1, \ldots, w_k)$ (where crs is associated with some SNARG for RAM) and send d to the verifier.

Now, the most naive approach would be for the prover to send a SNARG that $w_1, \ldots, w_k$ are all valid witnesses for $x_1, \ldots, x_k$, but it is completely unclear how to argue soundness of the resulting BARG relying on any soundness property of the SNARG.[7] The problem is that fundamentally, there is no way to guarantee that an adversarial prover P^* who makes the verifier accept (with digest d) actually *knows* the contents of a memory $(w_1, \ldots, w_k)$ that corresponds to d.

Instead, we will have the prover produce k *different* SNARGs $\pi_1, \ldots, \pi_k$ on memory $(w_1, \ldots, w_k)$ with respect to k different RAM computations. Specifically, we define the ith RAM computation M_i to consider only the ith "chunk" of memory and verify that w_i is a valid witness for x_i. An initial candidate BARG can then be the digest d along with $\pi_1, \ldots, \pi_k$; the verifier simply checks each π_i separately (with respect to d).

Although we have not argued soundness, this is already a non-trivial candidate BARG! As long as each π_i is significantly shorter than m (the length of w_i), the communication complexity of this protocol will be significantly shorter than the trivial bound of $k \cdot m$. This establishes an intuitive connection between RAM SNARGs and BARGs.

Challenges in Arguing Soundness. Despite having a simple candidate BARG with non-trivial efficiency, soundness of this candidate is not obvious and in fact does not seem to follow from previous security definitions for RAM SNARGs. In fact, the problem seems similar to the "naive" case: soundness of a RAM SNARG is only guaranteed against adversaries that "know" the entire memory, which is $w_1, \ldots, w_k$, but there is no way to argue that an adversary P^* must know such a long string (since the BARG itself is short).

However, there is a key difference from before: each RAM computation M_i only operates on a *small fraction* of its memory, namely, w_i (ignoring all w_j for $j \neq i$). Moreover, if Digest is *somewhere extractable* [HW15] on m locations (henceforth called a m-SEH), it *is* possible to argue that an adversary P^* (at least inside a security reduction) knows the fraction of RAM memory that is *relevant* to any particular M_i. This opens up the possibility for the following kind of security proof:

- Suppose that P^* is a convincing prover for the BARG, and let i be an index such that the statement x_i is false.
- Switch to a hybrid experiment in which the crs is *statistically binding* (and extractable) on w_i. In this hybrid, it is possible to produce both a valid BARG proof (where x_i is false[8]) and obtain the unique w_i consistent with d.

[7] We remark that in the privately verifiable setting, the batch argument system of [BHK17] is somewhat similar to the "naive" construction above, but they do *not* rely on a RAM SNARG. Instead, they rely on what later became known as a *quasi-argument* for NP [KPY19], which is a much more powerful building block.

[8] In this overview, we assume for simplicity that the statements $x_1, \ldots, x_k$ are fixed in advance. The situation is more subtle if the x_i are chosen adaptively, but non-trivial security properties can be argued (see Definition 3.4).

- Argue that the proof π_i produced by P^* contradicts the soundness of the RAM SNARG.

Unfortunately, previous RAM SNARG security definitions [KP16, BHK17, KPY19, CJJ21a] are not compatible with this security reduction. As a result, we next revisit and revise the foundations of RAM delegation.

Flexible SNARGs for RAM. There are two significant issues with previous notions of RAM SNARGs if we want to use them. First of all, we want a RAM SNARG with the property that the Digest algorithm is *somewhere extractable* on m locations. This begs the question: do such RAM SNARGs exist? More generally, one can ask: which additional properties can the Digest algorithm of a RAM SNARG potentially have?

We address these questions by defining *flexible* RAM SNARGs (Definition 6.2), which are a generic RAM SNARG *template* making use of an arbitrary hash family with local opening.[9] Specifically, a flexible RAM SNARG is a scheme defined relative to a generic hash family, which plays the role of the Digest algorithm. A flexible RAM SNARG has the property that for *any* secure hash family with local opening, the resulting RAM SNARG is sound. In other words, flexible RAM SNARGs imply that *any hash family with local opening* can be used as the Digest algorithm for a RAM SNARG. This tells us that we can plug in a Digest algorithm that is somewhere extractable, provided that it *also* has local openings.

Partial-Input Soundness. The second major problem with RAM SNARGs is that, as stated earlier, they provide no security guarantees against adversaries who produce a digest d^* *without knowledge* of a *full opening* of d^* to an input x (representing the full contents of a machine's memory). This is problematic in scenarios where a SNARG is used to prove statements about RAM machines that only access a small fraction of their memory.

This motivates defining soundness (or argument of knowledge) properties for RAM delegation schemes in situations where the adversary does *not* know an entire input x. Formulating such security notions can be quite subtle. In this work, for simplicity, we focus on the following setting:

- The Digest hash function is *extractable* on some set of locations S. This means that given an arbitrary d^*, it is possible to extract an assignment x_S so that an opening of d^* to any location $i \in S$ *must* reveal the bit x_i.
- The RAM machine M, when run on any memory consistent with x_S, only reads locations in S. This is equivalent to the assertion that this holds when M is run on the specific input x^* such that $x^*_i = x_i$ for $i \in S$ and $x^*_i = 0$ otherwise.

In this situation, we say that a RAM SNARG satisfies *partial-input soundness* if it is computationally hard to produce a machine M, digest d^*, output y, and proof π such that

- The verifier accepts $(M, \mathsf{d}^*, y, \pi)$, and

[9] A hash family with local opening (Definition 3.1) is a deterministic, computationally binding succinct commitment to a long string x along with a procedure for producing a short ($\mathsf{poly}(\lambda)$-size) opening to any bit x_i. This commitment need not hide information about x. In this paper, we relax the standard definition to allow for the hash key and hash output to each have two parts: (potentially long) sender components $(\mathsf{hk}, \mathsf{v})$ and (short) receiver components $(\mathsf{vk}, \mathsf{rt})$ (which are used for opening verification).

- $M(x^*) \neq y$, where x^* is obtained from d^* as above, and $M(x^*)$ only reads locations in S.

We emphasize that this definition does not require that the adversary possesses an opening[10] of d^* to x_S; nevertheless, the string x_S is well-defined (and efficiently accessible) in the security game.

Armed with this definition, we return to our main results relating BARGs and RAM SNARGs: assuming the existence of rate-1 string OT, the following two claims hold.

Claim 2.1. seBARG*s imply flexible* RAM SNARG*s with partial-input soundness.*

Claim 2.2. *Flexible* RAM SNARG*s with partial-input soundness imply* seBARG*s. This transformation* boosts *succinctness from "non-trivial" to* $\mathsf{poly}(\lambda, m, \log k)$.

So far, we sketched a weak variant of Claim 2.2 that does *not* boost succinctness. We conclude by discussing how to prove Claim 2.2 (in full) and Claim 2.1.

Boosting Succinctness via Recursion. Recall our candidate non-trivial seBARG making use of a (flexible) RAM SNARG:

- The prover sends a digest $\mathsf{d} = \mathsf{Digest}(w_1, \ldots, w_k)$, and
- The prover sends k SNARGs $\pi_1, \ldots, \pi_k$ associated with d, where π_i is a proof that w_i is a valid witness for x_i.

We indeed show that this construction is sound assuming that Digest is somewhere extractable and the RAM SNARG satisfies partial-input soundness. However, this argument system is only *somewhat* succinct: the size of the proof is $|\mathsf{d}| + \sum |\pi_i|$, which grows linearly with k. Can we do better?

The answer is that we can by adapting an insight from [CJJ21a] to our setting.[11] Namely, we observe that the proof string $(\mathsf{d}, \pi_1, \ldots, \pi_k)$ has *reduced* the problem of verifying $w_1, \ldots, w_k$ to the easier problem of verifying $\pi_1, \ldots, \pi_k$. Provided that the time to verify each π_i is at most half the time required to verify each w_i, we can *pair* adjacent proofs (π_{2i-1}, π_{2i}) together and obtain a batch NP verification problem with $k/2$ witnesses of complexity no larger than that of the original w_i. Then, instead of sending these witnesses (the π_i) in the clear, we can have the prover recursively run our protocol: send $\mathsf{Digest}(\pi_1, \ldots, \pi_k)$ and compute proofs $\pi'_1, \ldots, \pi'_{k/2}$ certifying that all pairs (π_{2i-1}, π_{2i}) would be accepted by the RAM SNARG verifier. This recursion can be executed $\log k$ times in total, resulting in a seBARG in which the prover sends $\log k$ digests $\mathsf{d}_0, \ldots, \mathsf{d}_{\log k - 1}$, where d_i is a digest of $k/2^i$ strings $\pi_1^{(i)}, \ldots, \pi_{k/2^i}^{(i)}$ that are RAM SNARG proofs computed with respect to d_{i-1}. At the end of the recursion, there will be a single RAM SNARG proof $\pi^{(\log k)}$ that the verifier can receive and check on its own.

[10] A natural alternative soundness definition would simply require that it is computationally hard for P^* to produce accepting (M, y, d, π) and local openings to a substring x_S such that $M(x^*)$ only reads locations in S and $M(x^*) \neq y$. However, for technical reasons, this turns out to be an *insufficient* definition to support our transformations, since we cannot always guarantee (in our soundness reductions) that P^* knows how to open x_S.

[11] One can view [CJJ21a] as implementing the following strategy: (1) construct a weakly succinct interactive batch argument scheme, (2) extend this particular scheme to a fully succinct interactive batch argument scheme, and (3) compile it into a BARG using the Fiat-Shamir transform [CCH+19]. (2) is accomplished via an interactive recursion. Theorem 1.3 suggests an alternative approach: (1′) build a weakly succinct interactive scheme, (2′) *apply the Fiat-Shamir transform right away* to get a weakly succinct BARG, and (3′) invoke Theorem 1.3 to boost the succinctness generically.

Crucially, we observe that as long as the RAM SNARG is *non-trivially succinct* – meaning that the computational cost of verifying a pair (π_1, π_2) is lower than the cost of verifying a single NP witness w – then the resulting seBARG will have ideal succinctness $\mathsf{poly}(\lambda, m, \log k)$. This is what enables our generic boosting results; see Section 7 for more details.

Fully Local Hashing. So far, we have sketched how to construct seBARGs given a flexible RAM SNARG with partial-input soundness, when the Digest algorithm for the RAM SNARG is somewhere extractable on m locations. Next, we address a technical issue with this approach.

The problem is that flexible SNARGs for RAM are only as as efficient as the underlying Digest algorithm plugged into them. Specifically, the *verification time* of the SNARG grows with the size of a local opening for Digest. However, a hash family that is somewhere extractable on m locations must necessarily have an output of length $\geq m$ [HW15], so opening verification would seem to require at least m time (even to read the hash value). This would result in a RAM SNARG whose verification time grows with m, which for our candidate BARG above would be useless: the BARG's size would be larger than $k \cdot m$.

This incompatibility is resolved with the recently introduced notion of a "fully local (somewhere extractable) hash family" [DGKV22]. In such a hash family $\mathcal{H}$, a hash evaluation can be divided into two parts: a long (length $\geq m$) component v and a short (length $\mathsf{poly}(\lambda)$) component rt. (Similarly, the hash key can be divided into a long component hk and a short component vk.) The hash family is then required to be *extractable* given v and have *local openings* of rt of size $\mathsf{poly}(\lambda)$ to individual input bits. Finally, consistency between v and rt is enforced; $\mathsf{rt} = \mathcal{H}.\mathsf{Digest}(\mathsf{crs}, \mathsf{v})$ is a fixed function of v.

A fully local SEH hash family $\mathcal{H}$ resolves our technical issue and enables a provable construction of seBARGs from flexible RAM SNARGs. But how is this building block instantiated? [DGKV22] constructed such hash families from the LWE assumption. Their construction was complicated and required powerful tools (including rate-1 FHE and seBARGs themselves!), but they were aiming for a *rate*-1 fully local hash family.

In this work (see Theorem 5.2), we give a simple construction of a (low rate) flSEH family from any SEH family with local opening. That is, we show that SEH families that are binding on a single index (or on m indices with openings that grow linearly with m) can be generically made fully local.

At a high level, our flSEH family is constructed as follows. Suppose that we want to hash inputs $x \in \{0,1\}^n$ in a way that is extractable on m indices $i_1, \ldots, i_m \in [n]$. Since a m-SEH requires openings of size $\geq m$ (as discussed above), we instead *separately* hash x m *different times* using SEH functions $h_1, \ldots, h_m$. Each $h_j \leftarrow \mathcal{H}$ is set up to be extractable on $\mathsf{poly}(\lambda)$ locations and have openings of size $\mathsf{poly}(\lambda)$. The resulting m hash values $\mathsf{v}_1, \ldots, \mathsf{v}_m$ are defined to be the extractable hash output v; v is then *digested* using a hash tree into a root rt of size $\mathsf{poly}(\lambda)$. To open a bit x_i with respect to the root rt, a hash function index j is selected *pseudorandomly*[12] (as a function of i), v_j is opened (w.r.t. rt), and then x_i is opened (w.r.t. v_j). This strategy enables us to make the overall hash family extractable on index i by making the hash function h_j extractable on i, and thus allows for m-location extractability with $\mathsf{poly}(\lambda)$-size openings. See Section 5.2 for details.

Having resolved the local opening subtlety, this completes our overview of Claim 2.2.

[12]The notion of pseudorandomness required is that of a *load-balancing* hash function: the n indices $\{1, \ldots, n\}$ should be mapped pseudorandomly into m buckets so that for any m-tuple $(i_1, \ldots, i_m)$ no bucket has more than $\mathsf{poly}(\lambda)$ of those indices in it.

Constructing our RAM SNARGs from seBARGs. Finally, we turn to Claim 2.1: constructing flexible RAM SNARGs with partial input soundness from seBARGs. This construction additionally uses a SEH family with local opening and closely follows the transformations of [CJJ21a, KVZ21]. To give a succinct proof that $M(x) = y$ with respect to a digest d (computed with respect to an *arbitrary* Digest algorithm with local opening), compute a somewhere extractable hash $h = \mathsf{SEH.Hash}(\mathsf{st}_1, \ldots, \mathsf{st}_T)$, where $\mathsf{st}_1, \ldots, \mathsf{st}_T$ denotes the sequence of memory configurations for the execution of $M(x)$. Then, generate and send a seBARG that, roughly speaking, $\mathsf{st}_i \to \mathsf{st}_{i+1}$ for all i. More formally, the seBARG is executed on a batch NP statement whose witnesses are openings to pairs $(\mathsf{st}_i, \mathsf{st}_{i+1})$ along with openings of d to the bit x_j that st_i asks to read; the NP relation checks that the correct bit is read and that the state transformation $\mathsf{st}_i \to \mathsf{st}_{i+1}$ is executed correctly. Since this batch of NP statements has a succinct representation (given by d along with h and the hash keys), the resulting RAM SNARG will be as succinct (up to $\mathsf{poly}(\lambda, |\mathsf{st}|)$ factors) as the BARG.

The main difference from the [CJJ21a, KVZ21] setting is that we wish to prove *partial-input soundness*, which states that a (malicious) prover cannot produce a digest d and accepting SNARG proof (M, y, π) such that $M(x^*) \neq y$, where x^* is constructed by extracting a substring x_S from d and setting all other x_i to 0. This follows from a hybrid argument combining the (extractable) binding property of Digest with the (extractable) soundness property of the seBARG. Essentially, it is possible to argue sequentially that for every time-step t, if the seBARG crs is set to be extractable on the tth NP statement, then the extracted state st_t must match the state of $M(x^*)$ at time t.

We refer the reader to Section 6 for more details. We also remark that as a side result, we prove in Section 6 that our construction satisfies the original [KP16, BHK17] definition of soundness, which is incomparable to partial-input soundness.

This completes our sketch of Claim 2.1. Combining Claim 2.1 and Claim 2.2 appropriately, we obtain Theorem 1.3 and Theorem 1.4.

2.1 Relation to [DGKV22]

A recent work of Devadas, Goyal, Kalai, and Vaikuntanathan [DGKV22] (concurrently with a work of Paneth and Pass [PP22]) constructs seBARGs that have *rate 1* with respect to the size of an NP witness. The notion of "rate-1 fully local hash" was introduced in an initial version of [DGKV22] for their construction; we then used a relaxation of their notion (a fully local hash family that has *low rate*) in this work. Subsequently, an updated version of [DGKV22] gives a significantly simplified construction of rate-1 seBARGs that leverages our notion of flexible RAM SNARGs (Definition 6.2) adapted to their rate-1 setting.

3 Preliminaries

Notation. We use PPT to denote probabilistic polynomial-time, and denote the set of all positive integers up to n as $[n] := \{1, \ldots, n\}$. 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 Hash Family with Local Opening

In this section we recall the definition of a hash family with local opening [Mer88].[13] We *generalize* the original definition to allow for a *non-succinct* hash key $(\mathsf{hk}, \mathsf{vk})$ that is divided into two components: a (potentially long) sender component and a (short) receiver component that is actually used for opening verification.

Syntax. A hash family (HT) with $\ell(N, \lambda)$-succinct local opening consists of the following algorithms:

$\mathsf{Gen}(N, 1^\lambda) \to (\mathsf{hk}, \mathsf{vk})$**.** This is a probabilistic (not necessarily poly-time) algorithm that takes as input the input length 1^N and security parameter 1^λ in unary and outputs a key pair $(\mathsf{hk}, \mathsf{vk}) \in \{0,1\}^{\mathsf{poly}(N,\lambda)} \times \{0,1\}^{\mathsf{poly}(\lambda)}$. hk is referred to as the hash key, while vk is referred to as the verification key.

$\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{v}$**.** 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 $\mathsf{v} \in \{0,1\}^{\ell(N,\lambda)}$.

$\mathsf{Open}(\mathsf{hk}, x, j) \to (b, \rho)$**.** 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 $\rho \in \{0,1\}^{\ell(N,\lambda)}$.

$\mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, b, \rho)$**.** This is a deterministic poly-time algorithm that takes as input a verification key vk, a hash value v, an index $j \in [N]$, a bit $b \in \{0,1\}$ and an opening $\rho \in \{0,1\}^{\ell(N,\lambda)}$, and outputs 1 (accept) or 0 (reject).

Definition 3.1. *(Properties of* HT*) A* HT *family* $(\mathsf{Gen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ *is required to satisfy the following properties.*

ℓ-Succinctness. *The runtime of* $\mathsf{Gen}(N, 1^\lambda)$ *is bounded by* $\ell(N, \lambda)$*, and the runtime of* Verify *(and hence the size of* v *and* (b, ρ)*) is at most* $\ell(N, \lambda)$*.*

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

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

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

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

We say that a HT family has *succinct* local openings if

[13] In 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.

- Gen runs in time $\mathsf{poly}(\lambda)$ and always outputs $\mathsf{hk} = \mathsf{vk}$, and
- $\ell(N, \lambda) = \mathsf{poly}(\lambda, \log N)$.

This variant is the standard notion from [Mer88].

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

If a HT family has succinct local openings, we drop the vk notation and sometimes refer to the hash value v as a *root* rt.

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,

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

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

Remark 3.2 (Offline/Online Opening Verification)**.** In the interest of making opening verification as efficient as possible, we optionally allow for a hash family with local opening HT to have *offline/online opening verification*, which means that opening verification operates as follows:

- In the offline phase, only the hash value v is available. During this phase, an algorithm $\mathsf{Digest}(\mathsf{vk}, \mathsf{v})$ is executed, outputting a (short) advice string rt.
- In the online phase, $\mathsf{Verify}(\mathsf{vk}, \mathsf{rt}, j, b, \rho)$ now takes rt as input instead of v.

In such schemes, there are *two* efficiency metrics: (1) the size of the hash value v (and runtime of Digest), and (2) the runtime of $\mathsf{Verify}(\mathsf{vk}, \mathsf{rt}, j, b, \rho)$ (which can be smaller). In this situation, we will use ℓ-succinctness to refer to (2).

3.2 Somewhere Extractable Hash Families

Next, we recall the definition of a somewhere extractable (SEH) hash family based on prior works [HW15, OPWW15].

Syntax. A SEH hash family consists of algorithms

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

where $\mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}$ have the same syntax as those of a hash family with local opening, and Gen and $\mathsf{Extract}$ have the following syntax:

$\mathsf{Gen}(1^\lambda, N, i) \to (\mathsf{hk}, \mathsf{vk}, \mathsf{td})$. 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 $(\mathsf{hk}, \mathsf{vk})$ along with trapdoor td.

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

Definition 3.3 (SEH). *A ℓ-succinct* SEH *hash family* $(\mathsf{Gen}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}, \mathsf{Extract})$ *is required to satisfy the following properties:*

ℓ-succinctness *as in Definition 3.1.*

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

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

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

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

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 statistical (resp. computational) extractability w.r.t. opening. *For any all-powerful (resp. polynomial-time) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,*

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

Remark 3.3. 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 3.4 (m-SEH)**.** Note that any SEH hash family (as defined in Definition 3.3) can be converted into one that is extractable on m indices $i_1, \ldots, i_m$ 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 $(\mathsf{hk}_1, \ldots, \mathsf{hk}_m)$. 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, if the original SEH family had ℓ-local openings, the new family will have $\ell \cdot m$-local openings.

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

3.3 Batch Arguments

Let CSAT be the following language

$$\mathsf{CSAT} = \{(C, x) : \exists w \in \{0,1\}^m \text{ s.t. } C(x, w) = 1\}$$

where $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ is a Boolean circuit and $x \in \{0,1\}^n$ is an instance.

Let BatchCSAT be the following language

$$\mathsf{BatchCSAT} = \{(C, x_1, \ldots, x_k) : \exists w_1, \ldots, w_k \in \{0,1\}^m \text{ s.t. } \forall i \in [k], C(x_i, w_i) = 1\}$$

Syntax. A publicly verifiable non-interactive batch argument (BARG) system for the language BatchCSAT consists of the following algorithms:

$\mathsf{Gen}(1^\lambda, k, 1^s) \to \mathsf{crs}$. This is a randomized (not necessarily poly-time)[14] algorithm that takes as input a security parameter 1^λ, number of instances k, and a circuit size 1^s. It outputs a common reference string crs.

$\mathcal{P}(\mathsf{crs}, C, x_1, \ldots, x_k, w_1, \ldots, w_k) \to \pi$. This is a poly-time prover algorithm that takes as input a crs, a circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, k instances $x_1, \ldots, x_k \in \{0,1\}^n$ and corresponding witnesses $w_1, \ldots, w_k \in \{0,1\}^m$, and outputs a proof π.

$\mathcal{V}(\mathsf{crs}, C, x_1, \ldots, x_k, \pi) \to 0/1$. This is a poly-time verification algorithm that takes as input a crs, a circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, k instances $x_1, \ldots, x_k \in \{0,1\}^n$, and a proof π. It outputs a bit to indicate whether the proof is valid or not.

Definition 3.4 (BARG for BatchCSAT)**.** *An $L(\cdot,\cdot)$-succinct* BARG *scheme $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ for* BatchCSAT *is required to satisfy the following properties:*

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

L-Verification Efficiency. *The verifier runs in time $L(k,\lambda) \cdot \mathsf{poly}(s) + k \cdot \mathsf{poly}(n,\lambda)$.*[15]

Completeness. *For any $\lambda \in \mathbb{N}$, any $k = k(\lambda)$ and $s = s(\lambda)$ of size at most 2^λ, any circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ of size at most s, any k instances $x_1, \ldots, x_k \in \{0,1\}^n$ and their corresponding witnesses $w_1, \ldots, w_k \in \{0,1\}^m$,*

$$\Pr\left[\mathcal{V}(\mathsf{crs}, C, x_1, \ldots, x_k, \pi) = 1 : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s), \\ \Pi \leftarrow \mathcal{P}(\mathsf{crs}, C, x_1, \ldots, x_k, w_1, \ldots, w_k) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

Semi-adaptive soundness. *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 $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, C, x_1, \ldots, x_k, \pi) = 1 \\ \wedge\ (C, x_{i^*}) \notin \mathsf{CSAT} \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s) \\ (C, x_1, \ldots, x_k, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition 3.5 (Index BARG)**.** *An* index BARG *scheme is a special case of* BARG *for* BatchCSAT *where the instances $x_1, \ldots, x_k$ are restricted to be $1, 2, \ldots, k$, and thus we omit them from the prover algorithm $\mathcal{P}$, the verifier algorithm $\mathcal{V}$, and the extraction algorithm* Extract*. However, for such a scheme, we require that $\mathcal{V}$ runs in time $L(k,\lambda) \cdot \mathsf{poly}(s)$ (with no additive $\mathsf{poly}(k,n)$ term), since the inputs $x_1, \ldots, x_k$ no longer need to be read.*

[14] This algorithm is always required to run in time at most $\mathsf{poly}(k, s, \lambda)$ though it is desirable that its runtime be sublinear in k. See the succinctness property in Definition 3.4 below.

[15] We choose this efficiency bound so that it is always smaller than the trivial $k \cdot s$ bound, although one could conceivably consider BARGs with larger verification time. We do not know how to make use of such BARGs. [CJJ21a] instead require a "split verification" property that in particular immediately implies the notion of Index BARG below.

Remark 3.5. Both a BARG for BatchCSAT and an index BARG can be thought of as BARGs that offer an efficiency gain in the case where the NP statements $(C, x_1), \ldots, (C, x_k)$ have an efficient representation. For Definition 3.4, this efficient representation lies in the fact that C is reused across instances, while for Definition 3.5, C alone describes the batch of instances.

For our purposes, the two notions are essentially equivalent:

- Given a BARG for BatchCSAT, setting the input length $n = \lambda$ implies an index BARG with verification time $L(k, \lambda) \cdot \mathsf{poly}(s) + k \cdot \lambda$, which can be made sublinear in k for any sublinear (in k) function L by *merging* small (e.g. size k^δ or $(\log k)^{\omega(1)}$) groups of statements together (increasing s by a small factor) and executing the proof system with a sublinear batch size.
- If there is a circuit family C' that on each input i outputs x_i, one can obtain a BARG for BatchCSAT with respect to $(C, x_1, \ldots, x_k)$ from an index BARG with respect to the circuit $\widetilde{C}(i, w) = C(C'(i), w)$.

One of the main goals of this paper is to generically *boost* the succinctness of index BARGs. However, we will mainly work with a strengthening of index BARGs called (index) *somewhere extractable* BARGs (seBARGs).

Definition 3.6 (seBARG for BatchCSAT)**.** *An $L(\cdot,\cdot)$-succinct* seBARG *scheme* $(\mathsf{Gen}, \mathcal{P}, \mathcal{V}, \mathsf{Extract})$ *for* BatchCSAT *is a $L(\cdot,\cdot)$-succinct* BARG *(Definition 3.4) with the following augmented syntax.*

$\mathsf{Gen}(1^\lambda, k, 1^s, i^*) \to (\mathsf{crs}, \mathsf{td})$. *The* Gen *algorithm now takes as additional input an index i^* and outputs a trapdoor* td *in addition to* crs*.*

$\mathsf{Extract}(\mathsf{td}, C, x_1, \ldots, x_k, \pi) \to w^*$**.** *This is a poly-time extraction algorithm that takes as input a trapdoor* td*, a circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, k instances $x_1 \ldots, x_k \in \{0,1\}^n$, and a proof π, and it outputs a witness w^*.*

An seBARG *is then required to satisfy the following additional properties to that of a* BARG*:*

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

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

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* $\mathsf{negl}(\cdot)$ *such that for every $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, C, x_1, \ldots, x_k, \pi) = 1 \\ \wedge\ C(x_{i^*}, w^*) \neq 1 \end{array} : \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i^*) \\ (C, x_1, \ldots, x_k, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ w^* \leftarrow \mathsf{Extract}(\mathsf{td}, C, x_1, \ldots, x_k, \pi) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

The following two theorems are both used to obtain our main two corollaries.

Theorem 3.7 ([WW22]). *Assuming $O(1)$-LIN, for every constant $\epsilon > 0$ there exists an $L(k, \lambda)$-succinct index* seBARG*, for $L(k, \lambda) = \mathsf{poly}(\lambda) \cdot k^{\epsilon}$.*

Theorem 3.8 ([CJJ21b]). *Assuming* QR *and sub-exponential* DDH*, there exists an $L(k, \lambda)$-succinct index* seBARG*, for $L(k) = \mathsf{poly}(\lambda)\sqrt{k}$.*

Implicit in the results of [CJJ21a, KVZ21] is the following lemma:

Lemma 3.9. *Assume the existence of*

- *An L-succinct index* BARG *system for* BatchCSAT*, and*
- *A* SEH *family with local opening (Definition 3.3) satisfying statistical extractability.*

Then, there exists an L-succinct index seBARG *system for* BatchCSAT*. Moreover, there exists an L-succinct* seBARG *system for* BatchCSAT*.*

Lemma 3.9 is shown via the *commit-and-prove* paradigm: to build an index seBARG, make the prover *commit* (using a SEH that is extractable on m locations, which has openings of size $\mathsf{poly}(\lambda) \cdot m$) to $w_1, \ldots, w_k$, and then send a BARG that for all $i \in [k]$, there exists an opening to a valid witness w_i. If the SEH is extractable on index i^*, a witness w_{i^*} can be extracted from any hash value rt output by the adversary such that no string other than w_{i^*} can be opened on those locations. This implies the somewhere argument of knowledge property. Moreover, this scheme can be extended to a (non-index) seBARG by having the prover additionally commit to $x_1, \ldots, x_k$ and require opening to the x_i as part of its BARG invocation.

4 SEH families from Rate-1 String OT.

In this section, we explain how we instantiate the SEH families required for Theorems 1.3 and 1.4. We show that there is a generic construction of an SEH from any rate-1 String OT protocol satisfying a "verifiable correctness" property (a relaxation of perfect correctness), defined below.

Definition 4.1 (Rate-1 (2-message) String OT)**.** *A rate-1 string OT protocol is executed between a sender S, who has as input two strings $x_0, x_1 \in \{0,1\}^N$, and receiver R, who has as input a single bit b. Such a protocol has the following syntax:*

- $\mathsf{OT.Setup}(1^\lambda, 1^N)$ *is a randomized algorithm that takes as input the security parameter λ and string length N, and outputs a common reference string* crs*.*
- $\mathsf{OT.Com}(\mathsf{crs}, b; r)$ *is a randomized algorithm that takes as input the* crs *and receiver choice bit b; it outputs a string m_R.*
- $\mathsf{OT.Send}(\mathsf{crs}, m_R, x_0, x_1)$ *takes as input the* crs*, receiver message m_R, and sender inputs x_0, x_1; it outputs a string m_S.*
- $\mathsf{OT.Receive}(\mathsf{crs}, r, m_S)$ *takes as input the* crs*, receiver randomness r, and sender message m_S; it outputs a string x.*

We require the following properties to hold of a scheme with this syntax:

- ***Correctness:*** *For any x_0, x_1, b, if the sender and receiver execute their algorithms honestly (and the* crs *is set up honestly), then the receiver's output x is equal to x_b with $1 - \mathsf{negl}$ probability.*
- ***Receiver Privacy:*** $(\mathsf{crs}, \mathsf{OT.Com}(\mathsf{crs}, 0))$ *is computationally indistinguishable from* $(\mathsf{crs}, \mathsf{OT.Com}(\mathsf{crs}, 1))$.
- ***Asymptotic Rate*** 1***:*** *The common reference string has length* $\mathsf{poly}(\lambda, \log N)$ *and an honestly computed sender message m_S has length $N + \mathsf{poly}(\lambda, \log N)$.*

Our construction below works easily if correctness of the OT scheme holds with probability 1, but this does not hold for all instantiations of the primitive (most notably, it does not hold for the DDH-based construction of [DGI+19]). We require the following *verifiable correctness* property.

Definition 4.2. *A string* OT *scheme satisfies* verifiable correctness *if there is a public, efficiently computable predicate* $\mathsf{Valid}(\mathsf{crs}, m_R, x_0, x_1)$ *with the following properties:*

- *If* $\mathsf{Valid}(\mathsf{crs}, m_R, x_0, x_1) = 1$*, then correctness w.r.t.* $(\mathsf{crs}, m_R, x_0, x_1)$ *holds (with probability 1).*
- *For every b, x_0, x_1, the probability that* $\mathsf{Valid}(\mathsf{crs}, \mathsf{OT.Com}(\mathsf{crs}, b; r), x_0, x_1) = 1$ *is* $1 - \mathsf{negl}(\lambda)$.

Rate-1 String OT has a wide variety of known instantiations, as stated by the following lemma.

Lemma 4.3 ([DGI+19])**.** *Rate-1 string OT schemes exist under any of the following cryptographic hardness assumptions: (1) learning with errors; (2) quadratic residuosity; (3) decisional composite residuosity; and (4) decisional Diffie-Hellman.*

Moreover, it can be verified by inspection that the schemes described in [DGI+19] all satisfy verifiable correctness:

- The QR and DCR-based protocols have perfect correctness.
- The DDH-based protocol with $1/\lambda$ error incurs a correctness error only if the sender computes a group element e such that $\mathsf{e} \cdot g^{-1}$ and e do not lie in the same component of a public pseudorandom partition of the group, which is efficiently checkable (see Appendix A for more details). This error is then reduced via repetition and an error-correcting code, and thus correctness is verifiable by checking if the number of "base" errors is small.
- The LWE-based protocol is verifiably correct by a similar argument to the DDH-based protocol.

In addition, a straightforward modification of the techniques in [DGI+19] imply the following additional instantiation. We present a full construction and proof in Appendix A for completeness.

Lemma 4.4. *Rate-1 string OT schemes (with verifiable correctness) exist under the* $O(1)$-LIN *assumption.*

We combine Lemmas 4.3 and 4.4 with Lemma 4.5 below to give a wide variety of constructions of SEH families with succinct local opening.

Lemma 4.5. *Assuming any rate-1 string OT with verifiable correctness, there is a somewhere* statistically *extractable hash function family with succinct local opening.*

Proof. We assume without loss of generality that $\mathsf{OT.Send}$ is a *deterministic* algorithm. If OT satisfies perfect correctness, this is immediate (set the randomness to 0), while if OT only satisfies $1 - \mathsf{negl}$ (verifiable) correctness, this can be done by adding a short PRF seed s to the crs and re-defining $\mathsf{OT.Send}$ to generate its randomness using s.

We follow the binary tree framework of [OPWW15]. That is, we first construct a two-mode hash family $(h, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, b) : \{0,1\}^{2N} \to \{0,1\}^{N+\mathsf{poly}(\lambda, \log N)}$ with the following extractability guarantee: there is an extraction algorithm $\mathsf{Ext}(\mathsf{td}, \cdot)$ such that given $y = h(x_0, x_1)$, $\mathsf{Ext}(\mathsf{td}, y) = x_b$.

This hash family can be constructed as follows: a hash key hk in mode b consists of an OT crs along with a receiver message $m_R \leftarrow \mathsf{OT.Com}(\mathsf{crs}, b)$. To hash an input $(x_0, x_1) \in \{0,1\}^{2N}$ (for sufficiently large $N = \mathsf{poly}(\lambda)$), simply compute a sender message $(\mathsf{crs}, m_R, x_0, x_1) \mapsto m_S$, resulting in an output of length $N + \mathsf{poly}(\lambda, \log N)$. An opening (x_0, x_1) is verified by checking that $\mathsf{Valid}(\mathsf{crs}, m_R, x_0, x_1) = 1$ and that the commitment is equal to $\mathsf{OT.Send}(\mathsf{crs}, m_R, x_0, x_1)$. Extraction is then possible using the algorithm $\mathsf{OT.Receive}$ (setting the extraction trapdoor td to be the OT receiver randomness), and somewhere extractability holds by the verifiable correctness property of OT.

Finally, we use the construction of [OPWW15] Theorem 3.2, which states that any hash family as above can be converted into a somewhere statistical binding hash family with local opening. In this construction, a function in the SSB hash family with local opening will, on input $x \in \{0,1\}^N$ compute and output the root of a Merkle tree, where at level k, pairs of elements of $\{0,1\}^{k \cdot \mathsf{poly}(\lambda)}$ are hashed (using a fresh hash key) to $\{0,1\}^{(k+1) \cdot \mathsf{poly}(\lambda)}$, where the $\mathsf{poly}(\lambda)$ term is a fixed polynomial. Thus, as long as the two-mode hash family is extractable, the resulting SSB hash family with local opening is also extractable: starting from the root, it is possible to recursively extract *one* of each node's two children in the Merkle tree, until one leaf is extracted. The somewhere extractability property of this construction follows from the analogous property of the two-mode family. □

5 Low-Rate Fully-Local Hash (flSEH) Families.

The recent work of Devadas, Goyal, Kalai and Vaikuntanathan [DGKV22] defined the notion of a fully local somewhere extractable hash (flSEH) family. This is an m-SEH, as defined in Definition 3.3 and in Remark 3.4, except that the size of an opening of each index is required to be *smaller than* m, and so is the time to verify the opening.

At first, this notion seems impossible to achieve since the hash value is of size $\geq m$, which in turn follows from the fact that it is statistically binding on m locations. In order to realize this notion, we allow opening verification to work in an offline/online model (Remark 3.2): the hash value v is first *digested* to a fully succinct value rt, and online opening verification takes rt (rather than v) as input.

The work of [DGKV22] constructs a *rate-1* flSEH hash family under LWE, and uses this as a building block in their construction of a rate-1 seBARG. Their construction is quite complicated due to the rate-1 requirement and relies essentially on tools only known from LWE.

In Section 5.1, we recall the definition of flSEH from [DGKV22] and relax their definition by *removing* the rate-1 requirement. In Section 5.2, we give a new construction of a flSEH using *any* SEH; by Section 4, this implies constructions from a wide variety of assumptions as opposed to just LWE. In Section 7, our new flSEH will be used as a key tool for obtaining our main results.

5.1 Defining flSEH Families

Syntax. A fully-local SEH (flSEH) hash family consists of the following algorithms:

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

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

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

$\mathsf{Open}(\mathsf{hk}, x, i) \to (b, \rho)$. 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.

$\mathsf{Verify}(\mathsf{vk}, \mathsf{rt}, i, b, \rho) \to 0/1$. 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).

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

Remark 5.1. We often abuse notation and let

$$\mathsf{Hash}(\mathsf{hk}, \mathsf{vk}, x) = (\mathsf{v}, \mathsf{rt})$$

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

Definition 5.1 (flSEH)**.** *A* flSEH *hash family is required to satisfy the following properties:*

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

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

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

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

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

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* $\mathsf{negl}(\cdot)$ *such that for every $\lambda \in \mathbb{N}$,*

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

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

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

5.2 Construction from any SEH

In this section we construct a flSEH hash family from any SEH hash family with ($\mathsf{poly}(\lambda)$-succinct) local opening.

In particular (invoking Lemmas 4.3 to 4.5), we obtain a fully succinct flSEH hash family from any of the $\{\mathsf{DDH}, O(1)\text{-}\mathsf{LIN}, \mathsf{QR}, \mathsf{DCR}, \mathsf{LWE}\}$ assumptions, whereas previously we only knew how to construct it based on LWE [DGKV22]. In addition, the construction presented here is significantly simpler than the one in [DGKV22] since we do not require the rate-1 property.

Construction. We construct a flSEH hash family using three simple building blocks:

- A SEH family with succinct local opening ($\mathsf{SEH.Gen}, \mathsf{SEH.Hash}, \mathsf{SEH.Open}, \mathsf{SEH.Verify}, \mathsf{SEH.Extract}$),
- A hash family with succinct local opening ($\mathsf{HT.Gen}, \mathsf{HT.Hash}, \mathsf{HT.Open}, \mathsf{HT.Verify}$), and
- A λ-wise independent function family $\{F_s : [N] \to [m]\}$ with seed length and evaluation time $\mathsf{poly}(\lambda, \log N, \log m)$. This object exists unconditionally, and moreover has the property that for every m-size subset $I \subset [N]$, the "maximal load" of F_s on I is at most λ with $1 - \mathsf{negl}(\lambda)$ probability. For simplicity, one can also just pick $\{F_s\}$ to be a pseudorandom function family.

Using these three ingredients, we will build a flSEH hash family that takes inputs in $\{0,1\}^N$ and is somewhere extractable on sets I containing any m indices.

The idea behind the construction is simple: each function F_s describes a *partition* of $[N]$ into m subsets $S_1, \ldots, S_m$. For any fixed set $I \subset [N]$ of size m, the λ-wise independence of $\{F_s\}$ implies that $|I \cap S_\ell| \leq \lambda$ for all ℓ with probability $1 - \mathsf{negl}(\lambda)$. Therefore, our construction will use m hash keys $\mathsf{hk}_1, \ldots, \mathsf{hk}_m$ for a λ-SEH; an input x will be hashed using each hk_ℓ to an output v_ℓ, but the local opening of a bit x_j is defined w.r.t. $\mathsf{hk}_{F_s(j)}$ (ignoring all other hk_ℓ). We then use a standard hash function with local opening, such as a Merkle tree [Mer88], to digest $(\mathsf{v}_1, \ldots, \mathsf{v}_m)$ and let the result rt be the digested output of the flSEH hash.

Our construction is described below.

- $\underline{\mathsf{Gen}(1^\lambda, N, I)}$

- Sample a seed $s \leftarrow \{0,1\}^{\mathsf{poly}(\lambda)}$.
- For all $j \in I$, compute $F_s(j)$. Construct (in time $|I| \cdot \mathsf{poly}(\lambda)$) the sets $T_\ell = I \cap \{j \in [N] \mid F_s(j) = \ell\}$. If $|T_\ell| \geq \lambda$ for any ℓ, abort.
- For $1 \leq \ell \leq m$, generate $(\mathsf{hk}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{SEH.Gen}(1^\lambda, N, T_\ell)$.
- Generate $\mathsf{hk}_{\mathsf{HT}} \leftarrow \mathsf{HT.Gen}(1^\lambda)$ and compute $\mathsf{rt}_{\mathsf{hk}} = \mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{HT}}, (\mathsf{hk}_1, \ldots, \mathsf{hk}_m))$.
- Let $\mathsf{hk} = (s, \mathsf{hk}_1, \ldots, \mathsf{hk}_m, \mathsf{hk}_{\mathsf{HT}})$ and $\mathsf{vk} = (s, \mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}_{\mathsf{hk}})$. Let $\mathsf{td} = (s, I, \mathsf{td}_1, \ldots, \mathsf{td}_m)$.
- Output $(\mathsf{hk}, \mathsf{vk}, \mathsf{td})$.

- $\underline{\mathsf{Hash}(\mathsf{hk}, x)}$
  - Parse $\mathsf{hk} = (s, \mathsf{hk}_1, \ldots, \mathsf{hk}_m, \mathsf{hk}_{\mathsf{HT}})$.
  - For every $\ell \in [m]$ compute $\mathsf{v}_\ell = \mathsf{SEH.Hash}(\mathsf{hk}_\ell, x)$.
  - Output $\mathsf{v} = (\mathsf{v}_1, \ldots, \mathsf{v}_m)$.
- $\underline{\mathsf{Digest}(\mathsf{vk}, \mathsf{v})}$
  - Parse $\mathsf{vk} = (s, \mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}_{\mathsf{hk}})$.
  - Output $\mathsf{rt} = \mathsf{HT.Hash}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{v})$.
- $\underline{\mathsf{Open}(\mathsf{hk}, x, j)}$
  - Parse $\mathsf{hk} = (s, \mathsf{hk}_1, \ldots, \mathsf{hk}_m, \mathsf{hk}_{\mathsf{HT}})$ and let $\ell = F_s(j)$.
  - Compute $\mathsf{v} = (\mathsf{v}_1, \ldots, \mathsf{v}_m)$ as above. Note that $\mathsf{v}_\ell = \mathsf{SEH.Hash}(\mathsf{hk}_\ell, x)$.
  - Compute $(\mathsf{v}_\ell, o) = \mathsf{HT.Open}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{v}, I_\ell)$ where I_ℓ is the interval corresponding to v_ℓ.
  - Compute $(\mathsf{hk}_\ell, \sigma) = \mathsf{HT.Open}(\mathsf{hk}_{\mathsf{HT}}, (\mathsf{hk}_1, \ldots, \mathsf{hk}_m), I'_\ell)$, where I'_ℓ is the interval corresponding to hk_ℓ.
  - Compute $(b, \rho) = \mathsf{SEH.Open}(\mathsf{hk}_\ell, x, j)$.
  - Let $\rho^* = (\mathsf{v}_\ell, \mathsf{hk}_\ell, o, \sigma, \rho)$.
  - Output (b, ρ^*).
- $\underline{\mathsf{Verify}(\mathsf{vk}, \mathsf{rt}, j, b, \rho^*)}$
  - Parse $\mathsf{vk} = (s, \mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}_{\mathsf{hk}})$ and $\rho^* = (\mathsf{v}_\ell, \mathsf{hk}_\ell, o, \sigma, \rho)$, where $\ell = F_s(j)$.
  - Output 1 if and only if the following three conditions hold:
    1. $\mathsf{HT.Verify}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}, I_\ell, \mathsf{v}_\ell, o) = 1$,
    2. $\mathsf{HT.Verify}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}_{\mathsf{hk}}, I'_\ell, \mathsf{hk}_\ell, \sigma) = 1$, and
    3. $\mathsf{SEH.Verify}(\mathsf{hk}_\ell, \mathsf{v}_\ell, j, b, \rho) = 1$.
- $\underline{\mathsf{Extract}(\mathsf{td}, \mathsf{v})}$
  - Parse $\mathsf{v} = (\mathsf{v}_1, \ldots, \mathsf{v}_m)$ and $\mathsf{td} = (s, \mathsf{td}_1, \ldots, \mathsf{td}_m)$.
  - For every $i \in I$, compute $(b_{i,i'})_{i' \in T_{F_s(i)}} = \mathsf{SEH.Extract}(\mathsf{td}_{F_s(i)}, \mathsf{v}_{F_s(i)})$. Observe that $i \in T_{F_s(i)}$.
  - Output $(b_{i,i})_{i \in I}$.

Theorem 5.2. *The construction defined above is a* flSEH *scheme (see Definition 5.1).*

Proof of Theorem 5.2. First, observe that the abort in Gen occurs with only negligible probability by the λ-wise independence of $\{F_s\}$, so we will ignore this event for the rest of the analysis.

Efficiency. Follows directly from the efficiency properties of the underlying SEH hash family and HT hash family.

Opening completeness. The opening completeness follows immediately from the opening completeness of the underlying HT scheme (Definition 3.1) and SEH scheme (Definition 3.3).

Index hiding. Follows from the index hiding property of the underlying SEH hash family.

Opening soundness w.r.t. digest. 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 every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|ll} \mathsf{Verify}(\mathsf{vk},\mathsf{rt},j,0,\rho_0^*)=1 & : & (1^k,I)\leftarrow\mathcal{A}_1(1^\lambda),\\ \wedge\ \mathsf{Verify}(\mathsf{vk},\mathsf{rt},j,1,\rho_1^*)=1 & & (\mathsf{hk},\mathsf{vk},\mathsf{td})\leftarrow\mathsf{Gen}(1^\lambda,N,I),\\ & & (\mathsf{rt},j,\rho_0^*,\rho_1^*)\leftarrow\mathcal{A}_2(\mathsf{hk}) \end{array}\right]\geq\epsilon(\lambda).$$

Parse $\mathsf{hk}=(s,\mathsf{hk}_1,\ldots,\mathsf{hk}_m,\mathsf{hk}_{\mathsf{HT}})$, $\mathsf{vk}=(s,\mathsf{hk}_{\mathsf{HT}},\mathsf{rt}_{\mathsf{hk}})$, let $\ell=F_s(j)$, and parse $\rho_b^*=(\mathsf{v}_b',\mathsf{hk}_b',o_b,\sigma_b,\rho_b)$ for every $b\in\{0,1\}$. First, we observe that the collision-resistance w.r.t. opening property of HT (see Definition 3.1) implies that $\mathsf{hk}_b'=\mathsf{hk}_\ell$ for both $b\in\{0,1\}$ (since $\mathsf{rt}_{\mathsf{hk}}$ was computed as an honest HT-hash of $(\mathsf{hk}_1,\ldots,\mathsf{hk}_m)$ and Verify checks for a valid HT-opening of $\mathsf{rt}_{\mathsf{hk}}$) except with negligible probability.

This implies that for every $\lambda\in\mathbb{N}$,

$$\Pr\left[\begin{array}{l|ll} \forall b\in\{0,1\}: & & (1^k,I)\leftarrow\mathcal{A}_1(1^\lambda),\\ \mathsf{HT.Verify}(\mathsf{hk}_{\mathsf{HT}},\mathsf{rt},I_\ell,\mathsf{v}_b',o_b)=1 & : & (\mathsf{hk},\mathsf{vk},\mathsf{td})\leftarrow\mathsf{Gen}(1^\lambda,N,I),\\ \wedge\ \mathsf{SEH.Verify}(\mathsf{hk}_\ell,\mathsf{v}_b',j,b,\rho_b)=1 & & (\mathsf{rt},j,\rho_0^*,\rho_1^*)\leftarrow\mathcal{A}_2(\mathsf{hk}) \end{array}\right]\geq\epsilon(\lambda)-\mathsf{negl}(\lambda).$$

Then, the collision resistance w.r.t. opening property of HT together implies that for every $\lambda\in\mathbb{N}$,

$$\Pr\left[\begin{array}{l|ll} \mathsf{v}_0'=\mathsf{v}_1'=\mathsf{v}' & & (1^k,I)\leftarrow\mathcal{A}_1(1^\lambda),\\ \wedge\ \forall b\in\{0,1\}: & : & (\mathsf{hk},\mathsf{vk},\mathsf{td})\leftarrow\mathsf{Gen}(1^\lambda,N,I),\\ \mathsf{SEH.Verify}(\mathsf{hk}_\ell,\mathsf{v}',j,b,\rho_b)=1 & & (\mathsf{rt},j,\rho_0^*,\rho_1^*)\leftarrow\mathcal{A}_2(\mathsf{hk}) \end{array}\right]\geq\epsilon(\lambda)-\mathsf{negl}(\lambda).$$

This contradicts the computational binding w.r.t. opening property of the underlying SEH hash family (see Definition 3.3 and Remark 3.3).

Opening soundness w.r.t. hash value. 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 every $\lambda\in\mathbb{N}$,

$$\Pr\left[\begin{array}{l|ll} j\in I & & (1^k,I)\leftarrow\mathcal{A}_1(1^\lambda)\\ \wedge\ \mathsf{Verify}(\mathsf{vk},\mathsf{rt},j,1-x_j,\rho^*)=1 & : & (\mathsf{hk},\mathsf{vk},\mathsf{td})\leftarrow\mathsf{Gen}(1^\lambda,N,I),\\ & & (\mathsf{v},j,\rho^*)\leftarrow\mathcal{A}_2(\mathsf{hk}).\\ & & \mathsf{rt}=\mathsf{Digest}(\mathsf{hk},\mathsf{v}),\\ & & (x_i)_{i\in I}=\mathsf{Extract}(\mathsf{td},\mathsf{v}) \end{array}\right]\geq\epsilon(\lambda).$$

Parse $\mathsf{hk} = (s, \mathsf{hk}_1, \ldots, \mathsf{hk}_m, \mathsf{hk}_{\mathsf{HT}})$, $\mathsf{vk} = (s, \mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}_{\mathsf{hk}})$, let $\ell = F_s(j)$, and parse $\rho^* = (\mathsf{v}', o, \rho)$. $\rho_b^* = (\mathsf{v}', \mathsf{hk}', o, \sigma, \rho)$. First, we observe that the collision-resistance w.r.t. opening property of HT implies that $\mathsf{hk}' = \mathsf{hk}_\ell$ (since $\mathsf{rt}_{\mathsf{hk}}$ was computed as an honest HT-hash of $(\mathsf{hk}_1, \ldots, \mathsf{hk}_m)$ and Verify checks for a valid opening of $\mathsf{rt}_{\mathsf{hk}}$) except with negligible probability.

By the definition of Verify this implies that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} & (1^k, I) \leftarrow \mathcal{A}_1(1^\lambda) \\ j \in I & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, I), \\ \wedge\ \mathsf{HT.Verify}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}, I_\ell, \mathsf{v}', o) = 1 & : \ (\mathsf{v}, j, \rho^*) \leftarrow \mathcal{A}_2(\mathsf{hk}). \\ \wedge\ \mathsf{SEH.Verify}(\mathsf{hk}_\ell, \mathsf{v}', j, 1 - x_j, \rho) = 1 & \mathsf{rt} = \mathsf{Digest}(\mathsf{hk}, \mathsf{v}), \\ & (x_i)_{i \in I} = \mathsf{Extract}(\mathsf{td}, \mathsf{v}) \end{array}\right] \geq \epsilon(\lambda) - \mathsf{negl}(\lambda).$$

Similarly, by the collision resistance w.r.t. opening property of the underlying HT hash family it follows that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} & (1^k, I) \leftarrow \mathcal{A}_1(1^\lambda) \\ j \in I & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, I), \\ \wedge\ \mathsf{v}' = \mathsf{v}_\ell & : \ (\mathsf{v}, j, \rho^*) \leftarrow \mathcal{A}_2(\mathsf{hk}). \\ \wedge\ \mathsf{SEH.Verify}(\mathsf{hk}_\ell, \mathsf{v}_\ell, j, 1 - x_j, \rho) = 1 & \mathsf{rt} = \mathsf{Digest}(\mathsf{hk}, \mathsf{v}), \\ & (x_i)_{i \in I} = \mathsf{Extract}(\mathsf{td}, \mathsf{v}) \end{array}\right] \geq \epsilon(\lambda) - \mathsf{negl}(\lambda).$$

This contradicts the somewhere extractability w.r.t. opening property of the underlying SEH hash family (see Definition 3.3), since $\mathsf{Extract}(\mathsf{td}, \mathsf{v})$ computes x_j by running $\mathsf{SEH.Extract}(\mathsf{td}_\ell, \mathsf{v}_\ell)$. □

6 Flexible RAM SNARGs with Partial Input Soundness

In this section, we define and construct our new notion of flexible RAM SNARGs with partial input soundness. We begin by recalling RAM delegation.

6.1 RAM Delegation

A RAM machine is modeled as a deterministic machine with random access to a memory of size 2^W. In its standard definition, the machine has a local state of length $O(W)$ in addition to its memory, and at each time step, the machine reads or writes to a single memory cell and updates its local state. Often it is assumed that the machine has no input outside of its memory.

In this work, we deviate from this modeling in several ways.

- We allow only read access to the memory, and do not allow writing access. As a result,
- We cannot assume that the local state is of length $O(W)$ and allow it to be of arbitrary length.

  Jumping ahead, these modifications will allow us to construct a RAM SNARG where the digest uses any hash family with local opening, including those that do not have efficient write operations, which increases its "flexibility."
- For convenience, we think of the input to the RAM machine as a pair $x = (x_{\mathsf{imp}}, x_{\mathsf{exp}})$ where x_{imp} is large and is stored in the random access memory, and x_{exp} is a short explicit input.

With the model specified, we now proceed to define RAM SNARGs. In Section 6.2, we will introduce new definitions enabling *flexibility* (with respect to the Digest algorithm) and *partial input soundness*.

Syntax. A RAM SNARG for machine $\mathcal{R}$ consists of the following algorithms:

$\mathsf{Gen}(1^\lambda, T) \to \mathsf{crs}$. This is a randomized (not necessarily poly-time)[16] algorithm that takes as input a security parameter 1^λ and a time bound T, and outputs a common reference string crs.

$\mathsf{Digest}(\mathsf{crs}, x) \to \mathsf{d}$. This is a deterministic polynomial-time algorithm that takes as input the crs and a bit string x and outputs a digest d of size $\mathsf{poly}(\lambda)$.

$\mathcal{P}(\mathsf{crs}, (x_{\mathsf{imp}}, x_{\mathsf{exp}})) \to (b, \pi)$. This is a deterministic polynomial-time prover that takes as input the crs and a pair $(x_{\mathsf{imp}}, x_{\mathsf{exp}})$ which consists of a (long) input x_{imp} and a (short) input x_{exp}, and outputs a bit $b = \mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}}) \in \{0,1\}$ and a proof π.

$\mathcal{V}(\mathsf{crs}, \mathsf{d}, x_{\mathsf{exp}}, b, \pi) \to \{0,1\}$. This is a deterministic polynomial-time verifier that takes as input the crs, a digest d of the long input, a short input x_{exp}, a bit $b \in \{0,1\}$, and a proof π, and outputs 1 (accept) or 0 (reject).

Definition 6.1. *An L-succinct* RAM SNARG $(\mathsf{Gen}, \mathsf{Digest}, \mathcal{P}, \mathcal{V})$ *for a* RAM *computation $\mathcal{R}$ with local state of size $S \geq |x_{\mathsf{exp}}| + \log |x_{\mathsf{imp}}|$, satisfies the following properties:*

L-Succinct. *The running time of* Gen *is at most $L(T,\lambda) \cdot \mathsf{poly}(S)$, and the length of a proof π is at most $L(T,\lambda) \cdot \mathsf{poly}(S)$.*

L-Verifier Efficiency. *The running time of $\mathcal{V}$ is at most $L(T,\lambda) \cdot \mathsf{poly}(S)$.*

Completeness. *For any $\lambda, n \in \mathbb{N}$ such that $n \leq T(n) \leq 2^\lambda$ and any $x = (x_{\mathsf{imp}}, x_{\mathsf{exp}}) \in \{0,1\}^n$ such that $\mathcal{R}(x)$ halts within T time steps, we have that*

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, \mathsf{d}_{x_{\mathsf{imp}}}, x_{\mathsf{exp}}, b, \pi) = 1 \\ \wedge\ b = \mathcal{R}(x) \end{array} \;:\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, T), \\ (b, \pi) = \mathcal{P}(\mathsf{crs}, x = (x_{\mathsf{imp}}, x_{\mathsf{exp}})), \\ \mathsf{d}_{x_{\mathsf{imp}}} = \mathsf{Digest}(\mathsf{crs}, x_{\mathsf{imp}}) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

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

$$\Pr\left[\begin{array}{l} \mathsf{Digest}(\mathsf{crs}, x) = \mathsf{Digest}(\mathsf{crs}, x') \\ \wedge\ x \neq x' \end{array} \;:\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, T), \\ (x, x') \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

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

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, \mathsf{d}, x_{\mathsf{exp}}, 0, \pi_0) = 1 \\ \wedge\ \mathcal{V}(\mathsf{crs}, \mathsf{d}, x_{\mathsf{exp}}, 1, \pi_1) = 1 \end{array} \;:\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, T), \\ (\mathsf{d}, x_{\mathsf{exp}}, \pi_0, \pi_1) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 6.1. The above definition of soundness is taken from [KP16, BHK17]. This definition was later weakened in [KPY19, CJJ21a]. These latter works construct a publicly verifiable RAM SNARG under the weaker soundness condition which guarantees soundness only for adversaries who "know"

[16] This algorithm is always required to run in time at most $\mathsf{poly}(T, \lambda)$ though it is desirable that its runtime be sublinear in T. See the succinctness property in Definition 6.1 below.

the memory. Formally, this weaker soundness definition guarantees that for any poly-size adversary $\mathcal{A}$ and polynomial $T = T(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathcal{R}(x) \text{ does not output } b \text{ within } T \text{ time steps} \\ \wedge\ \mathcal{V}(\mathsf{crs}, \mathsf{d}_{x_{\mathsf{imp}}}, x_{\mathsf{exp}}, b, \pi) = 1 \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, T), \\ (x = (x_{\mathsf{imp}}, x_{\mathsf{exp}}), b, \pi) \leftarrow \mathcal{A}(\mathsf{crs}), \\ \mathsf{d}_{x_{\mathsf{imp}}} = \mathsf{Digest}(\mathsf{crs}, x_{\mathsf{imp}}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

All known (publicly verifiable) RAM SNARGs under standard assumptions are w.r.t. this weakened definition. We construct a RAM SNARG and prove soundness under the stronger definition (focusing on the read-only setting). This proof is given only for completeness, since even this stronger soundness guarantee is insufficient for our application of boosting a semi-succinct seBARG into a succinct one.

To be useful for our application, we will require an incomparable strengthening of the weaker soundness property. Intuitively, we require that if the adversary knows only part of the memory then he cannot prove false claims about RAM computations that only touch that part of memory. We give a formal definition of this soundness guarantee in Section 6.2 below.

6.2 Defining Flexible RAM SNARGs with Partial Input Soundness

As discussed in the introduction, our new RAM SNARG crucially achieves a soundness guarantee that is stronger than the previously realized definition [KPY19, CJJ21a], which guaranteed soundness only if the adversary knows the entire memory. We introduce *partial input soundness*, which intuitively guarantees soundness even if the adversary only knows a part of the memory, as long as the RAM computation only depends on that part of the memory.

In what follows we define the syntax of a flexible RAM SNARG. The syntax is almost identical to that of a standard RAM SNARG (defined in Section 6.1), with the only difference being that the hash key used to digest the memory is given explicitly as opposed to being part of the crs.

Syntax. Let $\mathcal{R}$ be a RAM machine. A **flexible** RAM SNARG for $\mathcal{R}$ is associated with a hash family with local opening

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

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

$\mathsf{Gen}(1^\lambda, T) \to \mathsf{crs}$. This is a randomized (not necessarily 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.

$\mathsf{Digest}(\mathsf{hk}, \mathsf{vk}, x) \to \mathsf{rt}$. This is a deterministic polynomial-time algorithm that takes as input a key pair $(\mathsf{hk}, \mathsf{vk})$ generated by $\mathsf{HT.Gen}(1^\lambda)$ and a string x, and outputs the digest $\mathsf{rt} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{vk}, x)$. This algorithm is fixed by the choice of HT.

$\mathcal{P}(\mathsf{crs}, \mathsf{hk}, x_{\mathsf{imp}}, x_{\mathsf{exp}}) \to (b, \pi)$. This is a deterministic polynomial-time prover that takes as input a crs, a hash key hk, and a pair $(x_{\mathsf{imp}}, x_{\mathsf{exp}})$ which consists of a (long) implicit input x_{imp} and a (short) explicit input x_{exp}, and outputs a bit $b = \mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}}) \in \{0, 1\}$ and a proof π.

$\mathcal{V}(\mathsf{crs}, \mathsf{vk}, \mathsf{rt}, x_{\mathsf{exp}}, b, \pi) \to \{0,1\}$. This is a deterministic polynomial-time verifier that takes as input a crs, a HT verification key vk, a digest rt of the (long) implicit input, a (short) explicit input x_{exp}, a bit $b \in \{0,1\}$, and a proof π, and outputs 1 (accept) or 0 (reject).

Definition 6.2. *An L-succinct flexible* RAM SNARG

$$(\mathsf{Gen}, \mathsf{Digest}, \mathcal{P}, \mathcal{V})$$

associated with a hash family with local opening

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

satisfies the properties from Definition 6.1 (where the prover and verifier take as input also hk*, which is implicit in Definition 6.1). Since the succinctness ℓ of local openings of* HT *is unspecified, verifier efficiency of the* RAM SNARG *is required to be $L(T,\lambda) \cdot \mathsf{poly}(S, \ell)$.*

In addition, it satisfies the following partial-input soundness *property:*

Partial-input soundness*: If the underlying hash family with local opening is somewhere extractable on m locations, now denoted by*

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

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 $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} |I| \leq m \\ \wedge \mathcal{V}(\mathsf{crs}, \mathsf{vk}, \mathsf{rt}, x_{\mathsf{exp}}, b^*, \pi) = 1 \\ \wedge\ \mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}}, T) = 1 - b^* \text{ and does} \\ \quad \text{not read any location in } [N] \setminus I \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, T), \\ (1^N, I) = \mathcal{A}_1(\mathsf{crs}), \\ (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{SEH.Gen}(1^\lambda, N, I), \\ \left(\mathsf{rt}, x_{\mathsf{exp}}, b^*, \pi\right) = \mathcal{A}_2(\mathsf{crs}, \mathsf{hk}), \\ \text{define } x_{\mathsf{imp}} \in \{0,1\}^N : \\ (b_i)_{i \in I} = \mathsf{SEH.Extract}(\mathsf{td}, \mathsf{rt}), \\ \forall i \in I, (x_{\mathsf{imp}})_i = b_i \\ \forall i \in [N] \setminus I, (x_{\mathsf{imp}})_i = 0 \end{array}\right] \leq \mathsf{negl}(\lambda)$$

6.3 Construction from seBARGs

Theorem 6.3. *Assume the existence of an $L(k,\lambda)$-succinct index* seBARG *and a somewhere extractable hash family with succinct local opening. Then there exists an $L(T,\lambda) \cdot \mathsf{poly}(\lambda, \log T)$-succinct flexible* RAM SNARG *(for some fixed polynomial* poly *independent of* seBARG*).*

Remark 6.2. In addition, we show that our construction satisfies the soundness guarantee from [KP16, BHK17] (see Definition 6.1), which has not been shown before for a publicly verifiable scheme.

Proof of Theorem 6.3. Fix an L-succinct index seBARG scheme

$$(\mathsf{seBARG.Gen}, \mathsf{seBARG}.\mathcal{P}, \mathsf{seBARG}.\mathcal{V}, \mathsf{seBARG.Extract})$$

(Definition 3.6 and Definition 3.5) and a $\mathsf{poly}(\lambda, S)$-somewhere extractable hash family with $\mathsf{poly}(\lambda, S)$-succinct local opening

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

where we denote by S the size of a local state of $\mathcal{R}$. The L-succinct flexible RAM SNARG for a machine $\mathcal{R}$, corresponding to a hash family with ℓ-local opening

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

is defined below.

- $\underline{\mathsf{Gen}(1^\lambda, T) \to \mathsf{crs}.}$
  1. Fix (arbitrarily) $i = 1$.
  2. Let $(\mathsf{seBARG.crs}, \mathsf{seBARG.td}) \leftarrow \mathsf{seBARG}(1^\lambda, T, 1^s, i)$, where $s = \mathsf{poly}(\lambda, S)$ is specified below.
  3. Sample[17] $(\mathsf{SEH.hk}, \mathsf{SEH.td}) \leftarrow \mathsf{SEH.Gen}(1^\lambda, T \cdot S, I_i)$, where $I_i = \{(i-1) \cdot S + 1, \ldots, i \cdot S\}$.
  4. Output $\mathsf{crs} = (\mathsf{seBARG.crs}, \mathsf{SEH.hk})$.
- $\underline{\mathcal{P}(\mathsf{crs}, \mathsf{hk}, \mathsf{vk}, x_{\mathsf{imp}}, x_{\mathsf{exp}}).}$
  1. Compute $\mathsf{rt} = \mathsf{HT.Hash}(\mathsf{hk}, \mathsf{vk}, x_{\mathsf{imp}})$.
  2. Compute all the local states $(\mathsf{st}_1, \ldots, \mathsf{st}_T)$ of $\mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}}, T)$, and compute
  $$\mathsf{rt}_{\mathsf{st}} = \mathsf{SEH.Hash}(\mathsf{hk}, (\mathsf{st}_1, \ldots, \mathsf{st}_T)).$$
  3. If st_T is an accepting state then set $\mathsf{out} = 1$ and if st_T is a rejecting state then set $\mathsf{out} = 0$.
  4. Parse $\mathsf{crs} = (\mathsf{seBARG.crs}, \mathsf{SEH.hk})$.
  5. Let $C = C_{\mathsf{vk},\mathsf{SEH.hk},\mathsf{rt},\mathsf{rt}_{\mathsf{st}},x_{\mathsf{exp}},\mathsf{out}}$ be the circuit that on input (i, w_i) outputs 1 if and only if the following conditions hold:
     - Parse $w_i = (\mathsf{st}_{i-1}, \mathsf{st}_i, \rho_{i-1}, \rho_i, b_i, o_i)$.
     - For every $b \in \{0, 1\}$ check that if $i - b > 0$ then
     $$\mathsf{SEH.Verify}(\mathsf{hk}, \mathsf{rt}_{\mathsf{st}}, I_{i-b}, \mathsf{st}_{i-b}, \rho_{i-b}) = 1,$$
     where $I_{i-b} = \{(i-b-1) \cdot S + 1, \ldots, (i-b) \cdot S\}$.
     - Suppose that $\mathcal{R}$ given local state st_{i-1} reads bit $j_i \in [|x_{\mathsf{imp}}| + |x_{\mathsf{exp}}|]$ from memory and let $N = |x_{\mathsf{imp}}|$.
     - If $j_i \in [N]$ then check that $\mathsf{HT.Verify}(\mathsf{hk}, \mathsf{rt}, j_i, b_i, o_i) = 1$, and if $j_i > N$ then check that $b_i = (x_{\mathsf{exp}})_{j_i - N}$.
     - Check that if $\mathcal{R}$ given local state st_{i-1} reads bit b_i from the j_i'th memory location, then it updates its local state to st_i.
     - If $i = 1$ check that st_0 is the initial state.
     - If $i = T$ and $\mathsf{out} = 1$ then check that st_T is an accepting state.
     - If $i = T$ and $\mathsf{out} = 0$ then check that st_T is a rejecting state.
  6. Let $s = |C|$. Note that $s = \mathsf{poly}(\lambda, S, \log T) \cdot \mathsf{time}(\mathsf{HT.Verify})$.
  7. For every $i \in [T]$:

[17] Note that for our SEH succinctness parameters, no separate verification key is required.

- Let $(\mathsf{st}_i, \rho_i) \leftarrow \mathsf{SEH.Open}(\mathsf{SEH.hk}, (\mathsf{st}_1, \ldots, \mathsf{st}_T), I_i)$, where $I_i = \{(i-1) \cdot S + 1, \ldots, i \cdot S\}$.
- Let j_i be the bit in memory that $\mathcal{R}$ reads given local state st_{i-1}.
- Let $(b_i, o_i) \leftarrow \mathsf{HT.Open}(\mathsf{hk}, x_{\mathsf{imp}}, j_i)$ if $j_i \in [N]$, and otherwise let $b_i = (x_{\mathsf{exp}})_{j_i - N}$ and $o_i = \bot$.
- Let $w_i = (\mathsf{st}_{i-1}, \mathsf{st}_i, \rho_{i-1}, \rho_i, b_i, o_i)$.

8. Let $\mathsf{seBARG}.\pi = \mathsf{seBARG}.\mathcal{P}(\mathsf{seBARG.crs}, C, w_1, \ldots, w_T)$.
9. Let $\pi = (\mathsf{rt}_{\mathsf{st}}, \mathsf{seBARG}.\pi)$
10. Output (out, π).

- $\mathcal{V}(\mathsf{crs}, \mathsf{vk}, \mathsf{rt}, x_{\mathsf{exp}}, \mathsf{out}, \pi)$.

1. Parse $\pi = (\mathsf{rt}_{\mathsf{st}}, \mathsf{seBARG}.\pi)$ and parse $\mathsf{crs} = (\mathsf{seBARG.crs}, \mathsf{SEH.hk})$.
2. Output 1 if and only if

$$\mathsf{seBARG}.\mathcal{V}(\mathsf{seBARG.crs}, C_{\mathsf{vk},\mathsf{SEH.hk},\mathsf{rt},\mathsf{rt}_{\mathsf{st}},x_{\mathsf{exp}},\mathsf{out}}, \mathsf{seBARG}.\pi) = 1.$$

Remark 6.3 (Modified Construction for Offline/Online Opening Verification)**.** In the event that HT has offline/online opening verification (Remark 3.2), we make a slight modification to the above construction for an efficiency gain. Namely, if Digest outputs an offline-online pair $(\mathsf{v}, \mathsf{rt})$, then the circuit C is defined to only run the *online* verification algorithm (taking rt as input), while consistency between v and rt is checked directly by $\mathcal{V}$ as an additional verification step. In this version, the runtime of the RAM SNARG verifier will only grow with the online opening verification time of HT.

Jumping ahead, when HT is in fact a flSEH family, we will use this variant of the construction.

We now proceed to show that our construction satisfies all of the properties of a flexible RAM SNARG with partial input soundness.

L-Relative Succinctness. The succinctness (and verifier efficiency) of the seBARG used in the construction is $L(T, \lambda) \cdot \widetilde{\mathsf{poly}}(|C|)$ for some unspecified polynomial $\widetilde{\mathsf{poly}}$. Thus, $L(T, \lambda)\mathsf{poly}(\lambda, S, \log T)$-relative succinctness[18] follows immediately from the fact that the underlying seBARG scheme is L-succinct, the succinctness of the underlying SEH hash family, and the size bound $|C| \leq \mathsf{poly}(\lambda, S, \log T) \cdot \mathsf{time}(\mathsf{HT.Verify})$.

Completeness. Follows immediately from the completeness of the underlying seBARG scheme and the completeness of the underlying SEH hash family and the HT hash family.

Collision resistance of RAM digest Follows immediately from the collision resistance w.r.t. opening property of the underlying hash family with local opening HT (Definition 3.1).

[18] To achieve efficiency $L(T, \lambda)\mathsf{poly}(\lambda, S, \log T)$, we plug in the largest security parameter $\lambda' \leq \lambda$ into the construction such that $\widetilde{\mathsf{poly}}(\lambda') \leq \lambda$. We assume that L is monotone so that $L(T, \lambda') \leq L(T, \lambda)$.

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

$$\Pr\left[\begin{array}{l} \forall b \in \{0,1\} \\ \mathcal{V}(\mathsf{crs}, \mathsf{vk}, \mathsf{rt}, x_{\mathsf{exp}}, b, \pi_b) = 1 \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, T), \\ (\mathsf{hk}, \mathsf{vk}) \leftarrow \mathsf{HT.Gen}(1^\lambda), \\ \left(\mathsf{rt}, x_{\mathsf{exp}}, \pi_0, \pi_1\right) = \mathcal{A}(\mathsf{crs}, \mathsf{hk}) \end{array}\right] \geq \epsilon(\lambda).$$

Parse $\mathsf{crs} = (\mathsf{seBARG.crs}, \mathsf{SEH.hk})$, and for every $b \in \{0,1\}$ parse $\pi_b = (\mathsf{rt}_{\mathsf{st},b}, \mathsf{seBARG}.\pi_b)$. By the definition of $\mathcal{V}$ the above equation implies that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \forall b \in \{0,1\} \\ \mathsf{seBARG}.\mathcal{V}(\mathsf{seBARG.crs}, C_b, \mathsf{seBARG}.\pi_b) = 1 \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, T), \\ (\mathsf{hk}, \mathsf{vk}) \leftarrow \mathsf{HT.Gen}(1^\lambda), \\ \left(\mathsf{rt}, x_{\mathsf{exp}}, \pi_0, \pi_1\right) = \mathcal{A}(\mathsf{crs}, \mathsf{hk}) \end{array}\right] \geq \epsilon(\lambda)$$

where $C_b = C_{\mathsf{vk},\mathsf{SEH.hk},\mathsf{rt},\mathsf{rt}_{\mathsf{st},b},x_{\mathsf{exp}},b}$.

For every $j \in [T]$, let Gen_j be identical to Gen, except that rather than setting $i = 1$ it sets $i = j$. By the index hiding property of SEH and the index hiding property of seBARG, the above equation implies that there exists a negligible function $\mu(\cdot)$ such that for every $i \in [T]$ and every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \forall b \in \{0,1\} \\ \mathsf{seBARG}.\mathcal{V}(\mathsf{seBARG.crs}, C_b, \mathsf{seBARG}.\pi_b) = 1 \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}_i(1^\lambda, T), \\ (\mathsf{hk}, \mathsf{vk}) \leftarrow \mathsf{HT.Gen}(1^\lambda), \\ \left(\mathsf{rt}, x_{\mathsf{exp}}, \pi_0, \pi_1\right) = \mathcal{A}(\mathsf{crs}, \mathsf{hk}) \end{array}\right] \geq \epsilon(\lambda) - \mu(\lambda)$$

In the equations below, to avoid lengthy equations, we abuse notation and let td denote both $\mathsf{seBARG.td}$ and $\mathsf{SEH.td}$ (where these trapdoors correspond to $\mathsf{crs} = (\mathsf{seBARG.crs}, \mathsf{SEH.hk})$), and it will be clear from the context which one we are referring to.

By the somewhere argument of knowledge property of the underlying seBARG scheme, the above equation implies that there exists a negligible function $\nu(\cdot)$ such that for every $i \in [T]$ and every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \\ \forall b \in \{0,1\} \\ \mathsf{seBARG}.\mathcal{V}(\mathsf{seBARG.crs}, C_b, \mathsf{seBARG}.\pi_b) = 1 \\ \wedge\ C_b(i, w_b) = 1 \\ \\ \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}_i(1^\lambda, T), \\ (\mathsf{hk}, \mathsf{vk}) \leftarrow \mathsf{HT.Gen}(1^\lambda), \\ \left(\mathsf{rt}, x_{\mathsf{exp}}, \pi_0, \pi_1\right) = \mathcal{A}(\mathsf{crs}, \mathsf{hk}), \\ \forall b \in \{0,1\} \\ w_b = \mathsf{seBARG.Extract}(\mathsf{td}, C_b, \mathsf{seBARG}.\pi_b) \end{array}\right] \geq \epsilon(\lambda) - \nu(\lambda) \tag{1}$$

For every $b \in \{0,1\}$ parse $w_b = (\mathsf{st}_{b,i-1}, \mathsf{st}_{b,i}, \rho_{b,i-1}, \rho_{b,i}, z_{b,i}, o_{b,i})$.

We next argue that Equation (1) implies that there exists a negligible function $\xi(\cdot)$ such that for every $i \in [T]$ and every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \forall b \in \{0,1\} \\ \mathsf{seBARG}.\mathcal{V}(\mathsf{seBARG.crs}, C_b, \mathsf{seBARG}.\pi_b) = 1 \\ \wedge\ C_b(i, w_b) = 1 \\ \wedge\ z_{0,i} = z_{1,i} \\ \wedge\ \mathsf{st}_{0,i} = \mathsf{st}_{1,i} \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Gen}_i(1^\lambda, T), \\ (\mathsf{hk}, \mathsf{vk}) \leftarrow \mathsf{HT.Gen}(1^\lambda), \\ \left(\mathsf{rt}, x_{\mathsf{exp}}, \pi_0, \pi_1\right) = \mathcal{A}(\mathsf{crs}, \mathsf{hk}), \\ \forall b \in \{0,1\} \\ w_b = \mathsf{seBARG.Extract}(\mathsf{td}, C_b, \mathsf{seBARG}.\pi_b) \end{array}\right] \geq \epsilon(\lambda) - i \cdot \xi(\lambda) \tag{2}$$

Equation (2) with $i = T$ implies a contradiction, since it implies that with non-negligible probability $\mathsf{st}_{0,T} = \mathsf{st}_{1,T}$, and yet $\mathsf{st}_{0,T}$ is a rejecting state and $\mathsf{st}_{1,T}$ is an accepting state.
Thus, it remains to prove Equation (2), which we do by induction on i.

Base case: $i = 1$**.** Follows immediately from Equation (1) together with the definition of C_b, the collision resistance w.r.t. opening property of the underlying HT hash family, and the fact that there is a unique initial state st_0 (and therefore a unique specified memory access location j_0).

Induction step: Supposing Equation (2) holds for $i-1$, we proceed to prove that it holds for i, as follows: The induction hypothesis implies that when generating $\mathsf{crs} \leftarrow \mathsf{Gen}_{i-1}(1^\lambda, T)$, it holds that $\mathsf{st}_{0,i-1} = \mathsf{st}_{1,i-1}$ with probability $\epsilon - (i-1)\xi$. By the somewhere extractability w.r.t. opening property of the underlying SEH family it holds that there exists a negligible function $\nu_1(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \forall b \in \{0,1\} & \mathsf{crs} \leftarrow \mathsf{Gen}_{i-1}(1^\lambda, T), \\ \mathsf{seBARG}.\mathcal{V}(\mathsf{seBARG.crs}, C_b, \pi_b) = 1 & (\mathsf{hk}, \mathsf{vk}) \leftarrow \mathsf{HT.Gen}(1^\lambda), \\ \wedge\ C_b(i-1, w_b) = 1 & \\ \wedge\ z_{0,i-1} = z_{1,i-1} & \left(\mathsf{rt}, x_{\mathsf{exp}}, \pi_0, \pi_1\right) = \mathcal{A}(\mathsf{crs}, \mathsf{hk}), \\ \wedge\ \mathsf{st}_{0,i-1} = \mathsf{st}_{1,i-1} & \forall b \in \{0,1\} \\ \wedge\ \forall b \in \{0,1\}, & w_b = \mathsf{seBARG.Extract}(\mathsf{td}, C_b, \mathsf{seBARG}.\pi_b) \\ \mathsf{st}_{b,i-1} = \mathsf{SEH.Extract}(\mathsf{td}, \mathsf{rt}_{\mathsf{st},b}) & \end{array}\right] \geq \epsilon(\lambda) - (i-1)\cdot\xi(\lambda) - \nu_1(\lambda)$$

Let Gen'_i be the algorithm that on input $(1^\lambda, T)$ generates $\mathsf{seBARG.crs}$ w.r.t. index i, but generates $\mathsf{SEH.hk}$ w.r.t. I_{i-1} (as opposed to I_i). Namely it generates

$$(\mathsf{seBARG.crs}, \mathsf{seBARG.td}) \leftarrow \mathsf{seBARG}(1^\lambda, T, 1^s, i) \quad \text{and} \quad (\mathsf{SEH.hk}, \mathsf{SEH.td}) \leftarrow \mathsf{SEH.Gen}(1^\lambda, T\cdot S, I_{i-1}),$$

where $I_{i-1} = \{(i-2)\cdot S + 1, \ldots, (i-1)\cdot S\}$.

The equation above, together with the index hiding property of the underlying seBARG scheme, implies that there exists a negligible function $\nu_2(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \forall b \in \{0,1\} & \mathsf{crs} \leftarrow \mathsf{Gen}'_i(1^\lambda, T), \\ \mathsf{seBARG}.\mathcal{V}(\mathsf{seBARG.crs}, C_b, \pi_b) = 1 & (\mathsf{hk}, \mathsf{vk}) \leftarrow \mathsf{HT.Gen}(1^\lambda), \\ \wedge\ \mathsf{SEH.Extract}(\mathsf{td}, \mathsf{rt}_{\mathsf{st},0}) = & \\ \mathsf{SEH.Extract}(\mathsf{td}, \mathsf{rt}_{\mathsf{st},1}) & \left(\mathsf{rt}, x_{\mathsf{exp}}, \pi_0, \pi_1\right) = \mathcal{A}(\mathsf{crs}, \mathsf{hk}), \end{array}\right] \geq \epsilon(\lambda) - (i-1)\cdot\xi(\lambda) - \nu_1(\lambda) - \nu_2(\lambda)$$

By the somewhere argument of knowledge property of the underlying seBARG scheme, the above equation implies that there exists a negligible function $\nu_3(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l|l} \forall b \in \{0,1\} & \mathsf{crs} \leftarrow \mathsf{Gen}'_i(1^\lambda, T), \\ \mathsf{seBARG}.\mathcal{V}(\mathsf{seBARG.crs}, C_b, \pi_b) = 1 & (\mathsf{hk}, \mathsf{vk}) \leftarrow \mathsf{HT.Gen}(1^\lambda), \\ \wedge\ C_b(i, w_b) = 1 & \\ \wedge\ \mathsf{SEH.Extract}(\mathsf{td}, \mathsf{rt}_{\mathsf{st},0}) = & \left(\mathsf{rt}, x_{\mathsf{exp}}, \pi_0, \pi_1\right) = \mathcal{A}(\mathsf{crs}, \mathsf{hk}), \\ \mathsf{SEH.Extract}(\mathsf{td}, \mathsf{rt}_{\mathsf{st},1}) & w_b = \mathsf{seBARG.Extract}(\mathsf{td}, C_b, \mathsf{seBARG}.\pi_b) \end{array}\right] \geq \epsilon(\lambda) - (i-1)\cdot\xi(\lambda) - \nu_1(\lambda) - \nu_2(\lambda) - \nu_3(\lambda)$$

By the somewhere extractability w.r.t. opening property of the underlying SEH family, together

with the definition of C_b, there exists a negligible function $\nu_4(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} \forall b \in \{0,1\} & \mathsf{crs} \leftarrow \mathsf{Gen}'_i(1^\lambda, T), \\ \mathsf{seBARG}.\mathcal{V}(\mathsf{seBARG}.\mathsf{crs}, C_b, \pi_b) = 1 & (\mathsf{hk}, \mathsf{vk}) \leftarrow \mathsf{HT}.\mathsf{Gen}(1^\lambda), \\ \wedge\ C_b(i, w_b) = 1 & \left(\mathsf{rt}, x_{\mathsf{exp}}, \pi_0, \pi_1\right) = \mathcal{A}(\mathsf{crs}, \mathsf{hk}), \\ \wedge\ \mathsf{st}_{0,i-1} = \mathsf{st}_{1,i-1} & w_b = \mathsf{seBARG}.\mathsf{Extract}(\mathsf{td}, C_b, \mathsf{seBARG}.\pi_b) \end{array}\right]$$
$$\geq \epsilon(\lambda) - (i-1)\cdot\xi(\lambda) - \nu_1(\lambda) - \nu_2(\lambda) - \nu_3(\lambda) - \nu_4(\lambda)$$

where as above, $w_b = (\mathsf{st}_{b,i-1}, \mathsf{st}_{b,i}, \rho_{b,i-1}, \rho_{b,i}, z_{b,i}, o_{b,i})$.

By the index hiding property of the underlying SEH hash family, there exists a negligible function $\nu_5(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} \forall b \in \{0,1\} & \mathsf{crs} \leftarrow \mathsf{Gen}_i(1^\lambda, T), \\ \mathsf{seBARG}.\mathcal{V}(\mathsf{seBARG}.\mathsf{crs}, C_b, \pi_b) = 1 & (\mathsf{hk}, \mathsf{vk}) \leftarrow \mathsf{HT}.\mathsf{Gen}(1^\lambda), \\ \wedge\ C_b(i, w_b) = 1 & \left(\mathsf{rt}, x_{\mathsf{exp}}, \pi_0, \pi_1\right) = \mathcal{A}(\mathsf{crs}, \mathsf{hk}), \\ \wedge\ \mathsf{st}_{0,i-1} = \mathsf{st}_{1,i-1} & w_b = \mathsf{seBARG}.\mathsf{Extract}(\mathsf{td}, C_b, \mathsf{seBARG}.\pi_b) \end{array}\right]$$
$$\geq \epsilon(\lambda) - (i-1)\cdot\xi(\lambda) - \nu_1(\lambda) - \nu_2(\lambda) - \nu_3(\lambda) - \nu_4(\lambda) - \nu_5(\lambda)$$

By the collision resistance w.r.t. opening property of the underlying HT hash family, and the definition of C_b, the above equation proves the induction step, by setting $\xi(\lambda) = \sum_{i=1}^{5} \nu(\lambda)$, as desired.

Partial-Input Soundness. Suppose the underlying hash family with local opening HT is somewhere extractable, and denote it by

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

(we use the notation SEH^* to distinguish it from the hash family SEH used in the RAM SNARG construction though of course these hash families may be the same).[19]

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 $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} \mathcal{V}(\mathsf{crs}, \mathsf{vk}, \mathsf{rt}, x_{\mathsf{exp}}, b^*, \pi) = 1 & \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, T), \\ \wedge\ \mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}}, T) = 1 - b^* \text{ and does} & (1^N, I) = \mathcal{A}_1(\mathsf{crs}), \\ \quad \text{not read any location in } [N] \setminus I & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{SEH}^*.\mathsf{Gen}(1^\lambda, N, I), \\ & \left(\mathsf{rt}, x_{\mathsf{exp}}, b^*, \pi\right) = \mathcal{A}_2(\mathsf{crs}, \mathsf{hk}), \\ & \text{define } x_{\mathsf{imp}} \in \{0,1\}^N : \\ & (b_j)_{j \in I} = \mathsf{SEH}^*.\mathsf{Extract}(\mathsf{td}, \mathsf{rt}), \\ & \forall j \in I, (x_{\mathsf{imp}})_j = b_j \\ & \forall j \in [N] \setminus I, (x_{\mathsf{imp}})_j = 0 \end{array}\right] \geq \epsilon(\lambda)$$

As before, for every $j \in [T]$, let Gen_j be identical to Gen, except that rather than setting $i = 1$ it sets $i = j$. By the index hiding property of the somewhere extractable hash family SEH and the

[19] We note that now the RAM SNARG uses two somewhere extractable hash families: SEH^* to digest the implicit memory x_{imp} and SEH to digest all the states $(\mathsf{st}_1, \ldots, \mathsf{st}_T)$ of the RAM machine.

index hiding property of seBARG it holds that there exists a negligible function $\mu(\cdot)$ such that for every $i \in [T]$ and every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs},\mathsf{vk},\mathsf{rt},x_{\mathsf{exp}},b^*,\pi)=1 \\ \wedge\ \mathcal{R}(x_{\mathsf{imp}},x_{\mathsf{exp}},T)=1-b^* \text{ and does} \\ \quad \text{not read any location in } [N]\setminus I \end{array} : \begin{array}{l} \mathsf{crs}\leftarrow \mathsf{Gen}_i(1^\lambda,T), \\ (1^N,I)=\mathcal{A}_1(\mathsf{crs}), \\ (\mathsf{hk},\mathsf{vk},\mathsf{td})\leftarrow \mathsf{SEH}^*.\mathsf{Gen}(1^\lambda,N,I), \\ \left(\mathsf{rt},x_{\mathsf{exp}},b^*,\pi\right)=\mathcal{A}_2(\mathsf{crs},\mathsf{hk}), \\ \text{define } x_{\mathsf{imp}}\in\{0,1\}^N: \\ (b_j)_{j\in I}=\mathsf{SEH}^*.\mathsf{Extract}(\mathsf{td},\mathsf{rt}), \\ \forall j\in S,(x_{\mathsf{imp}})_j=b_j \\ \forall j\in[N]\setminus I,(x_{\mathsf{imp}})_j=0 \end{array}\right] \geq \epsilon(\lambda)-\mu(\lambda) \tag{3}$$

For any $i \in [T]$ let $\mathsf{BAD}_i = \mathsf{BAD}_i(\lambda)$ be the event that for $\mathsf{crs} \leftarrow \mathsf{Gen}_i(1^\lambda, T)$, $(1^N, I) = \mathcal{A}_1(\mathsf{crs})$, $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{SEH}^*.\mathsf{Gen}(1^\lambda, N, I)$, $\left(\mathsf{rt}, x_{\mathsf{exp}}, b^*, (\mathsf{rt}_{\mathsf{st}}, \mathsf{seBARG}.\pi)\right) = \mathcal{A}_2(\mathsf{crs}, \mathsf{hk})$, and $(b_j)_{j\in I} = \mathsf{SEH}^*.\mathsf{Extract}(\mathsf{td}, \mathsf{rt})$, the following holds for $x_{\mathsf{imp}} \in \{0,1\}^N$ defined as above:

1. $\mathcal{V}(\mathsf{crs}, \mathsf{vk}, \mathsf{rt}, x_{\mathsf{exp}}, b^*, \pi) = 1$.
2. $\mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}}, T)$ does not read any location in $[N] \setminus I$.
3. $\mathsf{st}_i = \mathsf{SEH}.\mathsf{Extract}(\mathsf{SEH}.\mathsf{td}, \mathsf{rt}_{\mathsf{st}})$ is not the correct i'th local state of $\mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}}, T)$, where $(\mathsf{SEH}.\mathsf{hk}, \mathsf{SEH}.\mathsf{td}) \leftarrow \mathsf{SEH}.\mathsf{Gen}(1^\lambda, T, I_i)$ is sampled when generating $\mathsf{crs} = (\mathsf{seBARG}.\mathsf{crs}, \mathsf{SEH}.\mathsf{hk})$.

Claim 6.4. *There exists a negligible function $\nu = \nu(\lambda)$ such that for every $i \in [T]$,*

$$\Pr[\mathsf{BAD}_i] \leq \nu.$$

We note that Claim 6.4 (for $i = T$) and Equation (3) (for $i = T$) imply a contradiction, since they imply that there exists a negligible function $\xi(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs},\mathsf{vk},\mathsf{rt},x_{\mathsf{exp}},b^*,\pi)=1 \\ \wedge\ \mathcal{R}(x_{\mathsf{imp}},x_{\mathsf{exp}},T)=1-b^* \text{ and does} \\ \quad \text{not read any location in } [N]\setminus I \\ \wedge\ \mathsf{st}_T \text{ is accepting iff } b^*=0 \end{array} : \begin{array}{l} \mathsf{crs}\leftarrow \mathsf{Gen}_T(1^\lambda,T), \\ (1^N,I)=\mathcal{A}_1(\mathsf{crs}), \\ (\mathsf{hk},\mathsf{vk},\mathsf{td})\leftarrow \mathsf{SEH}^*.\mathsf{Gen}(1^\lambda,N,I), \\ \left(\mathsf{rt},x_{\mathsf{exp}},b^*,\pi=(\mathsf{rt}_{\mathsf{st}},\mathsf{seBARG}.\pi)\right)=\mathcal{A}_2(\mathsf{crs},\mathsf{hk}), \\ \mathsf{st}_T=\mathsf{SEH}.\mathsf{Extract}(\mathsf{rt}_{\mathsf{st}},\mathsf{SEH}.\mathsf{td}), \\ \text{define } x_{\mathsf{imp}}\in\{0,1\}^N: \\ (b_j)_{j\in I}=\mathsf{SEH}^*.\mathsf{Extract}(\mathsf{td},\mathsf{rt}), \\ \forall j\in S,(x_{\mathsf{imp}})_j=b_j \\ \forall j\in[N]\setminus I,(x_{\mathsf{imp}})_j=0 \end{array}\right] \geq \epsilon(\lambda)-\xi(\lambda)$$

which, together with somewhere argument of knowledge property of the underlying seBARG scheme and the somewhere extractable w.r.t. opening property of the underlying SEH scheme, implies that there exists a negligible function $\xi(\cdot)$ such that for every $\lambda \in \mathbb{N}$, and for $\mathsf{crs} = (\mathsf{seBARG}.\mathsf{crs}, \mathsf{SEH}.\mathsf{hk}) \leftarrow$

$\mathsf{Gen}_T(1^\lambda, T)$,

$$\Pr\left[\begin{array}{l} \\ \mathcal{V}(\mathsf{crs},\mathsf{vk},\mathsf{rt},x_{\mathsf{exp}},b^*,\pi)=1 \\ \wedge\ \mathcal{R}(x_{\mathsf{imp}},x_{\mathsf{exp}},T)=1-b^* \text{ and does} \\ \quad \text{not read any location in } [N]\setminus I \\ \wedge\ \mathsf{st}_T \text{ is accepting iff } b^*=0 \\ \\ \end{array} : \begin{array}{l} \mathsf{crs}\leftarrow \mathsf{Gen}_T(1^\lambda,T), \\ (1^N,I)=\mathcal{A}_1(\mathsf{crs}), \\ (\mathsf{hk},\mathsf{vk},\mathsf{td})\leftarrow \mathsf{SEH}^*.\mathsf{Gen}(1^\lambda,N,I), \\ \Big(\mathsf{rt},x_{\mathsf{exp}},b^*,\pi=(\mathsf{rt}_{\mathsf{st}},\mathsf{seBARG}.\pi)\Big)=\mathcal{A}_2(\mathsf{crs},\mathsf{hk}), \\ w_T=\mathsf{seBARG}.\mathsf{Extract}(\mathsf{seBARG}.\mathsf{td},\mathsf{seBARG}.\pi), \\ \text{parse } w_T=(\mathsf{st}_{T-1},\mathsf{st}_T,\rho_{T-1},\rho_T,b'_T,o'_T), \\ \text{define } x_{\mathsf{imp}}\in\{0,1\}^N: \\ (b_j)_{j\in I}=\mathsf{SEH}^*.\mathsf{Extract}(\mathsf{td},\mathsf{rt}), \\ \forall j\in I,(x_{\mathsf{imp}})_j=b_j; \forall j\in[N]\setminus I,(x_{\mathsf{imp}})_j=0, \end{array}\right] \geq \epsilon(\lambda)-\xi(\lambda)$$

This contradicts the somewhere argument of knowledge property of the underlying seBARG scheme.

Thus, it remains to prove Claim 6.4.

Proof of Claim 6.4. It suffices to prove that for every $i\in[T]$ there exists a negligible function $\mu_i=\mu_i(\lambda)$ such that $\Pr[\mathsf{BAD}_i]\leq\mu_i$. We prove this by induction on i.

To this end, for every $i\in[T]$ let G_i be the event that the following conditions hold for $\mathsf{crs}=(\mathsf{seBARG}.\mathsf{crs},\mathsf{SEH}.\mathsf{hk})\leftarrow\mathsf{Gen}_i(1^\lambda,T)$, $(1^N,I)=\mathcal{A}_1(\mathsf{crs})$, $(\mathsf{hk},\mathsf{vk},\mathsf{td})\leftarrow\mathsf{SEH}^*(1^\lambda,N,I)$, and

$$\Big(\mathsf{rt},x_{\mathsf{exp}},b^*,\pi=(\mathsf{rt}_{\mathsf{st}},\mathsf{seBARG}.\pi)\Big)=\mathcal{A}_2(\mathsf{crs},\mathsf{hk})$$

where $C=C_{\mathsf{vk},\mathsf{SEH}.\mathsf{hk},\mathsf{rt},\mathsf{rt}_{\mathsf{st}},x_{\mathsf{exp}},b^*}$:

1. $\mathsf{seBARG}.\mathcal{V}(\mathsf{seBARG}.\mathsf{crs},C,\mathsf{seBARG}.\pi)=1$.
2. $\mathcal{R}(x_{\mathsf{imp}},x_{\mathsf{exp}},T)$ does not read any location in $[N]\setminus I$.

Base case: $i=1$. Let

$$w_1=(\mathsf{st}_0,\mathsf{st}_1,\rho_1,b'_1,o'_1)=\mathsf{seBARG}.\mathsf{Extract}(\mathsf{seBARG}.\mathsf{td},\mathsf{seBARG}.\pi).$$

Recall that $C(1,w_1)=1$ implies the following:

1. st_0 is the correct initial state.
2. $\mathcal{R}(x_{\mathsf{imp}},x_{\mathsf{exp}},T)$ goes from the initial state st_0 to local state st_1 after reading bit b'_1 from memory location j_1 in memory.
3. If $j_1\in[N]$ then $\mathsf{SEH}^*.\mathsf{Verify}(\mathsf{hk},\mathsf{rt},j_1,b'_1,o'_1)=1$, and if $j_1>N$ then $b'_1=(x_{\mathsf{exp}})_{j_1-N}$.
4. $\mathsf{SEH}.\mathsf{Verify}(\mathsf{SEH}.\mathsf{hk},\mathsf{rt}_{\mathsf{st}},I_1,\mathsf{st}_1,\rho_1)=1$.

There exist negligible functions ν_1 and ν_2 such that

$$\begin{aligned}
&\Pr[\ G_1\ \wedge\ \mathsf{st}_1 \text{ is incorrect state }]\leq \\
&\Pr[\ G_1\ \wedge\ C(1,w_1)=0\ ]+\Pr[\ G_1\ \wedge\ C(1,w_1)=1\ \wedge\ \mathsf{st}_1 \text{ is incorrect state }]\leq \\
&\nu_1+\Pr[\ G_1\ \wedge\ C(1,w_1)=1\ \wedge\ \mathsf{st}_1 \text{ is incorrect state }]\leq \\
&\nu_1+\Pr[G_1\ \wedge\ C(1,w_1)=1\ \wedge\ (b'_1\neq b_{j_1})\ ]\leq \\
&\nu_1+\nu_2,
\end{aligned}$$

where the first equation follows from basic probability; the second equation follows from the somewhere argument of knowledge property of the underlying seBARG scheme (Definition 3.6); the third equation follows from the fact that $C(1, w_1) = 1 \ \wedge \ (b'_1 = b_{j_1})$ implies that st_1 is the correct local state; and the fourth equation follows from the somewhere extractability w.r.t. opening property of the underlying hash family SEH (where b_{j_1} is the bit extracted from rt if $j_1 \in I$ and $b_{j_1} = (x_{\mathsf{exp}})_{j_1 - N}$ otherwise).

Inductive step: We prove that there exists a negligible function μ_i such that

$$\Pr[\mathsf{BAD}_i] \leq \Pr[\mathsf{BAD}_{i-1}] + \mu_i. \tag{4}$$

Let

$$w_i = (\mathsf{st}_{i-1}, \mathsf{st}_i, \rho_{i-1}, \rho_i, b'_i, o'_i) = \mathsf{seBARG.Extract}(\mathsf{seBARG.td}, \mathsf{seBARG}.\pi),$$

where $(\mathsf{rt}_{\mathsf{st}}, \mathsf{seBARG}.\pi)$ is the proof generated by $\mathcal{A}_2(\mathsf{crs}, \mathsf{hk})$, for $\mathsf{crs} \leftarrow \mathsf{Gen}_i(1^\lambda, T)$, $(1^N, I) = \mathcal{A}_1(\mathsf{crs})$, $(\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{SEH}^*.\mathsf{Gen}(1^\lambda, N, I)$. Recall that $C(i, w_i) = 1$ implies the following:

1. $\mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}}, T)$ goes from local state st_{i-1} to local state st_i after reading bit b'_i from memory.
2. If $j_i \in [N]$ then $\mathsf{SEH}^*.\mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, j_i, b'_i, o'_i) = 1$, and if $j_i > N$ then $b'_j = (x_{\mathsf{exp}})_{j_i - N}$.
3. For every $b \in \{0, 1\}$, $\mathsf{SEH.Verify}(\mathsf{SEH.hk}_i, \mathsf{v}, I_{i-b}, \mathsf{st}_{i-b}, \rho_{i-b}) = 1$.

There exist negligible functions ν_1, ν_2, ν_3 such that

$$\begin{aligned}
&\Pr[\ G_i \ \wedge \ \mathsf{st}_i \text{ is incorrect state }] \leq \\
&\Pr[\ G_i \ \wedge \ C(i, w_i) = 0\ ] + \Pr[\ G_i \ \wedge \ C(i, w_i) = 1 \ \wedge \ \mathsf{st}_i \text{ is incorrect state }] \leq \\
&\nu_1 + \Pr[\ G_i \ \wedge \ C(i, w_i) = 1 \ \wedge \ \mathsf{st}_i \text{ is incorrect state }] \leq \\
&\nu_1 + \Pr[\ G_i \ \wedge \ C(i, w_i) = 1 \ \wedge \ \mathsf{st}_{i-1} \text{ is incorrect state }] + \\
&\Pr[\ G_i \ \wedge \ C(i, w_i) = 1 \ \wedge \ \mathsf{st}_{i-1} \text{ is correct state } \wedge \ \mathsf{st}_i \text{ is incorrect state }] \leq \\
&\nu_1 + \ \Pr[\mathsf{BAD}_{i-1}] + \nu_2 + \Pr[\ G_i \ \wedge \ C(i, w_i) = 1 \ \wedge \ \mathsf{st}_{i-1} \text{ is correct state } \wedge \ \mathsf{st}_i \text{ is incorrect state }] \leq \\
&\nu_1 + \Pr[\mathsf{BAD}_{i-1}] + \nu_2 + \Pr[G_i \ \wedge \ C(i, w_i) = 1 \ \wedge \ (b'_i \neq b_{j_i})\ ] \leq \\
&\nu_1 + \Pr[\mathsf{BAD}_{i-1}] + \nu_2 + \nu_3,
\end{aligned}$$

where the first equation follows from basic probability; the second equation follows from the somewhere argument of knowledge property of the underlying seBARG scheme; the third equation follows from basic probability; the fourth equation is explained below; the fifth equation follows from the fact that

$$C(i, w_i) = 1 \ \wedge \ \mathsf{st}_{i-1} \text{ is correct state } \wedge \ (b'_i = b_{j_i})$$

implies that st_i is the correct local state; and the sixth equation follows from the somewhere extractability w.r.t. opening property of the underlying hash family SEH (where b_{j_i} is the bit extracted from rt if $j_i \in I$ and $b_{j_i} = (x_{\mathsf{exp}})_{j_i - N}$ otherwise).

It remains to argue that there exists a negligible function ν such that

$$\Pr[\ G_i \ \wedge \ C(i, w_i) = 1 \ \wedge \ \mathsf{st}_{i-1} \text{ is incorrect state }] \leq \Pr[\mathsf{BAD}_{i-1}] + \nu. \tag{5}$$

We prove Equation (5) via two hybrids. First, by the index hiding property of the underlying SEH hash family (Definition 3.3), it suffices to prove Equation (5) when $\mathsf{SEH.hk}$ is replaced by $\mathsf{SEH.hk}_{i-1}$ generated by

$$(\mathsf{SEH.hk}_{i-1}, \mathsf{SEH.td}_{i-1}) \leftarrow \mathsf{SEH.Gen}\left(1^\lambda, T \cdots, \{(i-2)\cdot S+1, \ldots, (i-1)\cdot S\}\right).$$

Second, by the somewhere extractable w.r.t. opening property of the underlying SEH hash family (Definition 3.3), it suffices to prove Equation (5) where $\mathsf{st}_{i-1} = \mathsf{SEH.Extract}(\mathsf{SEH.td}_{i-1}, \mathsf{v})$. With these two changes the probability in Equation (5) is at most the probability of BAD_{i-1}, as desired. □

This completes the proof of Theorem 6.3. □

7 From Weak RAM SNARGs to Strong BARGs

In this section we construct a $\mathsf{poly}(\lambda, m, \log k)$-succinct index seBARG for $\mathsf{BatchCSAT}$ assuming the existence of the following primitives:

- A somewhere extractable hash family with succinct local opening (Definition 3.3). Rather than using a SEH family directly, we invoke Theorem 5.2 and instead use as a building block a flSEH hash family

  $$(\mathsf{flSEH.Gen}, \mathsf{flSEH.Hash}, \mathsf{flSEH.Digest}, \mathsf{flSEH.Open}, \mathsf{flSEH.Verify})$$

  as in Definition 5.1.

- An L-succinct flexible $\mathsf{RAM\ SNARG}$ scheme with partial input soundness

  $$(\mathsf{RAM.Gen}, \mathsf{RAM.Digest}, \mathsf{RAM}.\mathcal{P}, \mathsf{RAM}.\mathcal{V}, \mathsf{RAM.Extract})$$

  as in Section 6 (see Theorem 6.3), where for all sufficiently large $T \geq T_0(\lambda) = \mathsf{poly}(\lambda)$, we have that $L(T, \lambda) \leq \frac{T}{c \log T}$ (note that $\frac{T}{c \log T} \leq \frac{T}{c\lambda}$ since $T \leq 2^\lambda$ by definition) for a specific constant c.

Combining this construction with Theorem 6.3, we will establish Theorems 1.3 and 1.4.

7.1 Construction

Let $\mathcal{R}$ be the RAM machine that takes as implicit input $(C, w_1, \sigma_1, \ldots, w_k, \sigma_k)$ (and explicit input (k, i)) and outputs 1 if and only if $C(i, w_i) = 1$ and σ_i consists of the values of all intermediate wires of C in the computation of $C(i, w_i)$. Observe that a natural instantiation of $\mathcal{R}$ has $S = O(\log(ks))$, where $s = |C|$. We let $\langle C \rangle$ denote the description length of C, which is $O(s \log s)$.

Our construction uses an L-succinct flexible $\mathsf{RAM\ SNARG}$ with partial input soundness (see Definition 6.2) w.r.t. the underlying flSEH hash family. We will treat the entirety of $(C, i, w_1, \ldots, w_\ell)$ as the implicit input (which will be digested).

In what follows, we assume for simplicity (and without loss of generality) that k is a power of 2. For every $\ell \in [\log k]$ we denote by $k_\ell = k/2^{\ell-1}$.

- $\underline{\mathsf{Gen}(1^\lambda, k, 1^s, i)}$

  1. Let $T = \frac{c}{2} \cdot s \log s$, which is an upper bound on the runtime of $\mathcal{R}$ for a circuit C of size at most s. If $T < T_0(\lambda)$ re-define (increase) s so that $T = T_0(\lambda)$.
  2. For every $\ell \in [\log k]$ sample
  $$(\mathsf{hk}_\ell, \mathsf{vk}_\ell, \mathsf{td}_\ell) \leftarrow \mathsf{flSEH.Gen}(1^\lambda, (k_\ell + 1)s, I_\ell)$$
  where $I_\ell = \{1, \ldots, \langle s \rangle\} \cup \{\langle s \rangle + (\lceil \frac{i}{2^{\ell-1}} \rceil - 1)s + 1, \ldots, \langle s \rangle + \lceil \frac{i}{2^{\ell-1}} \rceil s\}$, where $\langle s \rangle \leq T$ denotes the description length of a size-s circuit, and let
  $$\mathsf{hk} = (\mathsf{hk}_\ell)_{\ell \in [\log k]}, \mathsf{vk} = (\mathsf{vk}_\ell)_{\ell \in [\log k]}$$
  3. For every $\ell \in [\log k]$ generate $\mathsf{RAM.crs}_\ell \leftarrow \mathsf{RAM.Gen}(1^\lambda, T)$.
  4. Output
  $$\mathsf{crs} = (\mathsf{hk}, \mathsf{vk}, \mathsf{RAM.crs}_1, \ldots, \mathsf{RAM.crs}_{\log k})$$
  and
  $$\mathsf{td} = (\mathsf{td}_\ell)_{\ell \in [\log k]}.$$

- $\underline{\mathcal{P}(\mathsf{crs}, C, w_1, \ldots, w_k)}$

  1. Parse $\mathsf{crs} = (\mathsf{hk}, \mathsf{vk}, \mathsf{RAM.crs}_1, \ldots, \mathsf{RAM.crs}_{\log k})$, $\mathsf{hk} = (\mathsf{hk}_\ell)_{\ell \in [\log k]}$, and $\mathsf{vk} = (\mathsf{vk}_\ell)_{\ell \in [\log k]}$.
  2. Let $C^{(1)} = C$, and let $w_i^{(1)} = w_i$ for every $i \in [k]$.
  3. Set $\ell = 1$.
  4. For every $i \in [k_\ell]$, let $\sigma_i^{(\ell)}$ denote the wire assignment for $(C^{(\ell)}, i, w_i))$.
  5. Compute $(\mathsf{v}_\ell, \mathsf{rt}_\ell) = \mathsf{flSEH.Hash}\left(\mathsf{hk}_\ell, \left(C^{(\ell)}, w_1^{(\ell)}, \sigma_1^{(\ell)}, \ldots, w_{k_\ell}^{(\ell)}, \sigma_{k_\ell}^{(\ell)}\right)\right)$.
  6. Compute an opening ρ_ℓ of rt_ℓ to $C^{(\ell)}$ on locations $\{1, \ldots, \langle C \rangle\}$.
  7. For every $i \in [k_\ell]$, compute
  $$\mathsf{RAM}.\pi_i^{(\ell)} = \mathsf{RAM}.\mathcal{P}\left(\mathsf{RAM.crs}_\ell, \mathsf{hk}_\ell, \left(\left(C^{(\ell)}, w_1^{(\ell)}, \sigma_1^{(\ell)}, \ldots, w_{k_\ell}^{(\ell)}, \sigma_{k_\ell}^{(\ell)}\right), (k_\ell, i)\right)\right).$$
  8. For every $i \in [k_{\ell+1}]$, let
  $$w_i^{(\ell+1)} = (\mathsf{RAM}.\pi_{2i-1}^{(\ell)}, \mathsf{RAM}.\pi_{2i}^{(\ell)}).$$
  Let $C^{(\ell+1)} = C^{(\ell+1)}_{\mathsf{crs}_\ell, \mathsf{vk}_\ell, \mathsf{rt}_\ell}$ be a circuit such that for every $i \in [k_{\ell+1}]$,
  $$C^{(\ell+1)}\left(i, w_i^{(\ell+1)}\right) = 1$$
  if and only if for every $b \in \{0, 1\}$,
  $$\mathsf{RAM}.\mathcal{V}\left(\mathsf{RAM.crs}_\ell, \mathsf{vk}_\ell, \mathsf{rt}_\ell, (k_\ell, 2i - b), \mathsf{RAM}.\pi_{2i-b}^{(\ell)}\right) = 1.$$
  We note that by the efficiency property of $\mathsf{RAM}.\mathcal{V}$, we know that
  $$|C^{(\ell+1)}| \leq 2 \cdot L(T, \lambda).$$
  Since $T \geq T_0$, we know that $2 \cdot L(T, \lambda) \leq 2 \cdot \frac{T}{c \log T} = s$. This maintains an invariant that $|C^{(\ell)}| \leq s$ for all ℓ.

9. If $\ell = \log k$ then output $\left(\mathsf{v}^{(1)}, \rho_1, \ldots, \mathsf{v}^{(\log k)}, \rho_{\log k}, w_1^{(\log k+1)}\right)$.
10. Else, go back to Item 4 with $\ell = \ell + 1$.

- $\underline{\mathcal{V}(\mathsf{crs}, C, \pi)}$
  1. Parse $\pi = \left(\mathsf{v}_1, \rho_1, \ldots, \mathsf{v}_{\log k}, \rho_{\log k}, w_1^{(\log k+1)}\right)$.
  2. Compute $\mathsf{rt}_\ell = \mathsf{fISEH.Digest}(\mathsf{vk}_\ell, \mathsf{v}_\ell)$ for all ℓ.
  3. For every ℓ, construct the circuit $C^{(\ell)}$ and verify that ρ_ℓ is an opening of rt_ℓ to $C^{(\ell)}$ on the first $\langle s \rangle$ input locations.
  4. Output 1 if and only if $C^{(\log k+1)}_{\mathsf{rt}_{\log k}}\left(1, w_1^{(\log k+1)}\right) = 1$.
- $\underline{\mathsf{Extract}(\mathsf{td}, C, \pi)}$
  1. Parse $\mathsf{td} = (\mathsf{td}_\ell)_{\ell \in [\log k]}$ and $\pi = \left(\mathsf{v}_1, \rho_1, \ldots, \mathsf{v}_{\log k}, \rho_{\log k}, w_1^{(\log k+1)}\right)$.
  2. Output $w = \mathsf{fISEH.Extract}\left(\mathsf{td}_1, \mathsf{v}_1\right)$.

7.2 Analysis

Theorem 7.1. *The above construction is a* polylog*-succinct index* seBARG *for* BatchCSAT *assuming there exists a polynomial $T_0(\lambda)$ such that $L(T, \lambda) \leq T/c \log T$ for all $T \geq T_0(\lambda)$.*

Proof of Theorem 7.1.

Completeness. Follows from the completeness of the underlying RAM SNARG and opening correctness of fISEH. We note that the ℓth invocation of the RAM SNARG uses a statement that is independent of crs_ℓ, and the ℓth invocation of the fISEH is used on an input that is independent of hk_ℓ, so this holds even in the case of $1 - \mathsf{negl}(\lambda)$ non-adaptive completeness.

polylog**-Efficiency.** This follows from the invariant that $|C^{(\ell)}| \leq \max(s, T_0(\lambda))$ for all ℓ (see Item 8) and the efficiency properties of fISEH and RAM.$\mathcal{V}$. Specifically, the efficiency of fISEH.Digest (with our choice of extractability parameter) is at most $T \cdot \mathsf{poly}(\lambda) \leq s \log(s) \cdot \mathsf{poly}(\lambda)$ and the efficiency of RAM.$\mathcal{V}$ is at most $L(T, \lambda) \leq \max(s, \mathsf{poly}(\lambda))$.

Index hiding. Follows directly from the index hiding property of the underlying fISEH hash family.

Somewhere argument of knowledge. Suppose for the sake of contradiction that there exists a poly-size adversary $\mathcal{A}$, polynomials $k = k(\lambda)$ and $s = s(\lambda)$, an index $i^* = i^*(\lambda) \in [k(\lambda)]$, and a non-negligible function $\epsilon(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{ll} \mathcal{V}(\mathsf{crs}, C, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, 1^s, i^*) \\ \wedge\, C(i^*, w^*) = 0 \;:\; & (C, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & w^* \leftarrow \mathsf{Extract}(\mathsf{td}, C, \pi) \end{array}\right] > \epsilon(\lambda). \tag{6}$$

Parse

$$\mathsf{crs} = (\mathsf{hk}, \mathsf{vk}, \mathsf{RAM.crs}_1, \ldots, \mathsf{RAM.crs}_{\log k})$$

where $\mathsf{hk} = (\mathsf{hk}_\ell)_{\ell \in [\log k]}$ and $\mathsf{vk} = (\mathsf{vk}_\ell)_{\ell \in [\log k]}$, parse $\mathsf{td} = (\mathsf{td}_\ell)_{\ell \in [\log k]}$, and parse

$$\pi = \left(\mathsf{v}_1, \rho_1, \ldots, \mathsf{v}_{\log k}, \rho_{\log k}, w_1^{(\log k+1)}\right).$$

For every $\ell \in [\log k]$ let

$$\left(\widetilde{C}^{(\ell)}, w^{(\ell)}_{\lceil i^*/2^{\ell-1} \rceil}\right) = \mathsf{fISEH.Extract}\left(\mathsf{td}_\ell, \mathsf{v}_\ell\right). \tag{7}$$

Equation (6) implies that

$$\Pr\left[\mathcal{V}(\mathsf{crs}, C, \pi) = 1 \ \wedge \ C\left(i^*, w_{i^*}^{(1)}\right) = 0\right] > \epsilon(\lambda).$$

Moreover, the somewhere extractability property of fISEH implies that

$$\Pr\left[\mathcal{V}(\mathsf{crs}, C, \pi) = 1 \ \wedge \ C\left(i^*, w_{i^*}^{(1)}\right) = 0 \ \wedge \ \forall \ell, \widetilde{C}^{(\ell)} = C^{(\ell)}\right] > \epsilon(\lambda) - \mathsf{negl}(\lambda).$$

By definition, $\mathcal{V}(\mathsf{crs}, C, \pi) = 1$ implies that $C^{(\log k+1)}\left(1, w_1^{(\log k+1)}\right) = 1$. Thus, the equation above implies that

$$\Pr\left[C^{(\log k+1)}\left(1, w_1^{(\log k+1)}\right) = 1 \ \wedge \ C^{(1)}\left(i^*, w_{i^*}^{(1)}\right) = 0 \ \wedge \ \forall \ell, \widetilde{C}^{(\ell)} = C^{(\ell)}\right] > \epsilon(\lambda) - \mathsf{negl}(\lambda).$$

By a standard hybrid argument, this implies that that there exists $\ell \in [\log k]$ and a non-negligible function $\delta = \delta(\lambda)$ such that

$$\Pr\left[C^{(\ell+1)}\left(\lceil \frac{i^*}{2^\ell} \rceil, w^{(\ell+1)}_{\lceil i^*/2^\ell \rceil}\right) = 1 \quad \wedge \quad C^{(\ell)}\left(\lceil \frac{i^*}{2^{\ell-1}} \rceil, w^{(\ell)}_{\lceil i^*/2^{\ell-1} \rceil}\right) = 0 \ \wedge \ \widetilde{C}^{(\ell)} = C^{(\ell)}\right] > \delta(\lambda). \tag{8}$$

Parse

$$w^{(\ell+1)}_{\lceil i^*/2^\ell \rceil} = (\mathsf{RAM}.\pi_0, \mathsf{RAM}.\pi_1).$$

The fact that $C^{(\ell+1)}\left(\lceil \frac{i^*}{2^\ell} \rceil, w^{(\ell+1)}_{\lceil i^*/2^\ell \rceil}\right) = 1$ implies that for every $b \in \{0,1\}$,

$$\mathsf{RAM}.\mathcal{V}\left(\mathsf{RAM.crs}_\ell, \mathsf{vk}_\ell, \mathsf{rt}_\ell, \left(k_\ell, 2\lceil i^*/2^\ell \rceil - b\right), \mathsf{RAM}.\pi_b\right) = 1.$$

Fix $b^* \in \{0,1\}$ such that $\lceil \frac{i^*}{2^{\ell-1}} \rceil = 2\lceil \frac{i^*}{2^\ell} \rceil - b^*$. Then, the above equation for $b = b^*$ can equivalently be written as

$$\mathsf{RAM}.\mathcal{V}\left(\mathsf{RAM.crs}_\ell, \mathsf{vk}_\ell, \mathsf{rt}_\ell, \left(k_\ell, \lceil \frac{i^*}{2^{\ell-1}} \rceil\right), \mathsf{RAM}.\pi_{b^*}\right) = 1. \tag{9}$$

Equation (8) implies that with non-negligible probability $\delta = \delta(\lambda)$ the following three conditions hold:

1. Equation (9) holds.
2. $\mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}}) = 0$ where $x_{\mathsf{exp}} = \left(k_\ell, \lceil \frac{i^*}{2^{\ell-1}} \rceil\right)$ and $x_{\mathsf{imp}} \in \{0,1\}^{\langle s \rangle + k_\ell s}$ is zero everywhere except on the set of indices $I \subseteq [\langle s \rangle + k_\ell s]$ of size $\langle s \rangle + s$ corresponding to the circuit $\widetilde{C}^{(\ell)} = C^{(\ell)}$ and the $\lceil \frac{i^*}{2^{\ell-1}} \rceil$'th wire assignment which is set to contain the invalid witness $w^* = w^{(\ell)}_{\lceil i^*/2^{\ell-1} \rceil}$.
3. $\mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}})$ does not read from memory any location in $[|x_{\mathsf{imp}}|] \setminus I$.

This contradicts the partial input soundness property of the underlying flexible RAM SNARG scheme. □

8 Obtaining our Main Results

In this section, we formally state our results (Theorems 1.3 and 1.4) and show how they follow from the results proved in Sections 3 to 7.

Theorem 8.1. *Assume the existence of rate-1 String OT with verifiable correctness (Definitions 4.1 and 4.2), or more generally a* SEH *family with succinct local opening (Definition 3.3).*

Then, there exists an explicit polynomial $p(\lambda)$ such that the following holds.

Let $L(k,\lambda)$ denote any function such that $L(k,\lambda) \leq k/p(\lambda)$ for all sufficiently large $k \geq \mathsf{poly}(\lambda)$. Assuming the existence of a $L(k,\lambda)$-succinct index BARG *for* BatchCSAT*, there exists a $\mathsf{poly}(\lambda, \log k)$-succinct index* BARG *for* BatchCSAT*. Moreover, there exists a $\mathsf{poly}(\lambda, \log T)$-succinct* SNARG *for* P *(and for* RAM *computation).*

Remark 8.1. As discussed in the introduction,

- Index BARGs with efficiency $\mathsf{poly}(m,\lambda)k^{1-\delta}$ for any constant $\delta > 0$ suffice for the $L(k,\lambda)$ hypothesis, and thus imply fully succinct BARGs (assuming a SEH).
- Index BARGs with efficiency $\mathsf{poly}(m,\lambda)\frac{k}{(\log k)^{\omega(1)}}$ with sub-exponential security also suffice by setting $\lambda = \mathsf{poly}\log(k \cdot \lambda')$ for a new security parameter λ'. The resulting fully succinct BARG will only be secure against adversaries that run in time quasi-polynomial in $m \cdot \lambda'$, as the proof of Theorem 8.1 calls the weak BARG with batch size $\mathsf{poly}(m)$. This is a significant drawback but still a meaningful BARG.

By Remark 3.5, the use of index BARGs in these two instantiations could be replaced with the use of (non-index) BARGs for BatchCSAT. However, this remains a stronger assumption than the existence BARGs for L^k for some NP-complete language L.

Proof of Theorem 8.1. First of all, Lemma 4.5 tells us that rate-1 String OT satisfying verifiable correctness implies an SEH family with succinct local opening. In turn, Theorem 5.2 implies that an SEH family with succinct local opening implies the existence of a flSEH family (Definition 5.1).

We proceed to prove Theorem 8.1 by a composition of several transformations.

- By Lemma 3.9, $L(k,\lambda)$-succinct index BARGs for BatchCSAT (along with a SEH family with local opening) imply $L^{(2)}(k,\lambda) = L(k,\lambda) \cdot \mathsf{poly}(\lambda)$-succinct index seBARGs for BatchCSAT.
- By Theorem 6.3, $L^{(2)}(k,\lambda)$-succinct index BARGs for BatchCSAT (along with a SEH family with local opening) imply $L^{(3)}(T,\lambda) = L(T,\lambda) \cdot \mathsf{poly}(\lambda, \log T)$-succinct flexible SNARGs for RAM with partial-input soundness.
- By Theorem 7.1, $L^{(3)}(T,\lambda)$-succinct flexible SNARGs for RAM with partial-input soundness imply polylog-succinct index seBARGs provided that $L^{(3)}(T,\lambda) \leq T/\lambda$ for sufficiently large $T \geq T_0(\lambda)$.
- Finally, by [CJJ21a, KVZ21] we already know that polylog-succinct index seBARGs imply SNARGs for P and for RAM. By Theorem 6.3, they in fact even imply flexible RAM SNARGs with partial-input soundness.

Since each of the transformations can be implemented in a way that incurs a *fixed* $\mathsf{poly}(\lambda)$ overhead, the theorem follows. □

Corollary 8.2. *There exist* $\mathsf{poly}(\lambda, \log k)$*-succinct index* BARG*s for* BatchCSAT *and* $\mathsf{poly}(\lambda, \log T)$*-succinct* SNARG*s for* P *under either*

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

Proof. By Lemmas 4.3 and 4.4, we know that under any of the DDH, QR, and $O(1)$-LIN assumptions, there exists a rate-1 string OT scheme to fulfill the hypothesis of Theorem 8.1.

Moreover, [WW22] constructed an index-BARG scheme for BatchCSAT with sublinear succinctness under $O(1)$-LIN on bilinear maps. They first construct a scheme with (polylogarithmic online communication and) a large crs of size $\mathsf{poly}(k, m, \lambda)$; instead of reducing the crs size by using Section 5 of [WW22], we can simply execute $k^{1-\delta}$ copies of the scheme with batch size k^{δ} (re-using the same short crs) to immediately obtain sublinear overall succinctness (choosing small enough $\delta < 1/2$).

Additionally, [CJJ21b] constructed[20] an index-BARG scheme for BatchCSAT with sublinear succinctness under sub-exponential DDH and QR

Given these building blocks, the claimed results follow by Theorem 8.1. □

Theorem 8.3. *Assume the existence of rate-*1 *String OT with verifiable correctness (Definitions 4.1 and 4.2), or more generally a* SEH *family with succinct local opening (Definition 3.3).*

Then, $\mathsf{poly}(\lambda, \log k)$*-succinct index* BARG*s for* BatchCSAT *exist if and only if* $\mathsf{poly}(\lambda, \log T, \log N)$*-succinct flexible* RAM SNARG*s with partial-input soundness (Definition 6.2) exist.*

Proof. Lemma 4.5 tells us that the String OT building block implies an SEH family with succinct local opening. In turn, Theorem 5.2 implies that an SEH family with succinct local opening implies the existence of a flSEH family.

The equivalence can then be established as follows:

- By Lemma 3.9, succinct index BARGs for BatchCSAT (along with a SEH family with local opening) imply succinct index seBARGs for BatchCSAT.
- By Theorem 6.3, succinct seBARGs for BatchCSAT (along with a SEH family with local opening) imply flexible RAM SNARGs with partial-input soundness.
- By Theorem 7.1, flexible RAM SNARGs with partial-input soundness (along with a flSEH family) imply succinct BARGs for BatchCSAT. □

References

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, August 2004. 43

[20] Index BARGs were not defined in [CJJ21b], but it is easily seen that their construction satisfies the required efficiency property. [CJJ21b] Corollary 1 and Corollary 2 establish $(|C|+k)\mathsf{poly}(\lambda)$ efficiency for C-index languages; by combining groups of $\sqrt{k}$ statements together, we obtain sublinear succinctness. Similarly, the notion of semi-adaptive soundness was not defined in [CJJ21b], but their unmodified construction satisfies it.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539. Springer, Heidelberg, August 2016. 43

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 474–482. ACM, 2017. 4, 5, 6, 7, 10, 24, 26

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019. 4, 8

[CJJ21a] Arka Rai Choudhuri, Abhihsek Jain, and Zhengzhong Jin. Snargs for $\mathcal{P}$ from LWE. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 68–79. IEEE, 2021. 1, 2, 3, 4, 5, 7, 8, 10, 14, 16, 24, 25, 39

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*, pages 394–423, 2021. 1, 2, 4, 5, 16, 40

[DGI+19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019. 2, 3, 17, 43

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-np and applications. In *Proceedings of FOCS 2022*, 2022. 2, 4, 9, 10, 18, 20

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. 1

[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, Heidelberg, May / June 2022. 2, 4, 5

[HK07] Dennis Hofheinz and Eike Kiltz. Secure hybrid encryption from weakened key encapsulation. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 553–571. Springer, Heidelberg, August 2007. 43

[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. 6, 9, 12

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. Snargs for bounded depth computations and PPAD hardness from sub-exponential LWE. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 708–721. ACM, 2021. 1, 2, 4

[KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 91–118, 2016. 4, 5, 7, 10, 24, 26

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1115–1124, 2019. 1, 2, 5, 6, 7, 24, 25

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 565–574, 2013. 1

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 485–494, 2014. 1

[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, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part I*, volume 13042 of *Lecture Notes in Computer Science*, pages 330–368. Springer, 2021. 2, 3, 4, 5, 10, 16, 39

[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, Heidelberg, August 1988. 11, 12, 20

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994. 1

[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, Heidelberg, November / December 2015. 12, 18

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *63rd FOCS*, pages 1045–1056. IEEE Computer Society Press, October / November 2022. 10

[Sha07] Hovav Shacham. A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants. *Cryptology ePrint Archive*, 2007. 43

[WW22] Brent Waters and David J Wu. Batch arguments for np and more from standard bilinear group assumptions. *Cryptology ePrint Archive*, 2022. 2, 4, 5, 16, 40

A Rate-1 String-OT from k-LIN

Lemma A.1 (Lemma 4.4, restated)**.** *Rate-1 string OT schemes (with verifiable correctness) exist under the k-LIN assumption [BBS04, HK07, Sha07] for any constant $k \geq 1$.*

Proof sketch. We show a direct construction of rate-1 string OT from k-LIN. Recall that the k-LIN assumption postulates that it is computationally hard to distinguish between the distribution of $(g, g^{\mathbf{M}})$ where $\mathbf{M} \in \mathbb{Z}_p^{(k+1)\times n}$ has rank k versus when it is uniformly random. By a hybrid argument, this also implies the hardness of distinguishing between $(g, g^{\mathbf{M}})$ where $\mathbf{M} \in \mathbb{Z}_p^{(k+n)\times n}$ has rank k versus when it is uniformly random. It is this version that we will use in our protocol.

We construct a rate-1 batch-OT scheme from which a construction of rate-1 string-OT scheme follows immediately. The receiver of the OT has a choice-vector $\mathbf{x} \in \{0,1\}^n$. She picks a uniformly random matrix $\mathbf{B} \in \mathbb{Z}_p^{(k+n)\times n}$ of rank k together with vectors $\mathbf{u}_i = (-\mathbf{v}_i, \mathbf{e}_i) \in \mathbb{Z}_p^{k+n}$ in its left-kernel, where $\mathbf{e}_i$ is the i^{th} unit vector in $\{0,1\}^n$. Let $\mathbf{X} = \mathsf{Diag}(\mathbf{x}) \in \mathbb{Z}_p^{n\times n}$ be a matrix with x_i as the $(i,i)^{th}$ entry and zeroes elsewhere; and let $\mathbf{Y} \in \mathbb{Z}_p^{(n+k)\times n}$ be a matrix with $\mathbf{X}$ as its bottom $n \times n$ block and zeroes elsewhere. The receiver sends $g^{\mathbf{B}+\mathbf{Y}}$ to the sender. Notice already that the k-LIN assumption immediately tells us that the receiver's message hides $\mathbf{x}$.

The sender has a pair of vectors $\mathbf{a}, \mathbf{b} \in \{0,1\}^n$ and wishes to transmit $(\mathbf{a}-\mathbf{b}) \odot \mathbf{x} + \mathbf{b} \in \{0,1\}^n$ to the receiver, where $\odot$ denotes componentwise product. This step makes additional use of a public seed sd for a PRF mapping $\mathbb{Z}_p$ to $\{0,1\}^t$ for large enough $t = O(\log n)$.

The sender computes

$$\mathbf{h} = g^{(\mathbf{B}+\mathbf{Y})(\mathbf{a}-\mathbf{b})} \odot g^{0^k||\mathbf{b}}$$

Let $\mathbf{h} = (\mathbf{h}^\top, \mathbf{h}^\perp) \in \mathbb{G}^{k+n}$. Moreover, let $\mathbf{z} = (z_i = \mathsf{Dist}_{\mathsf{sd}}(h_i^\perp) \pmod 2)_{i=1}^n$, where $\mathsf{Dist}_{\mathsf{sd}}(h)$ is the smallest integer z such that $\mathsf{PRF}_{\mathsf{sd}}(h \cdot g^z) = 0^t$ [BGI16]. The sender sends $(\mathbf{h}^\top, \mathbf{z})$ to the receiver.

Given these k group elements $\mathbf{h}^\top$ and n bits $\mathbf{z}$, the receiver computes and outputs (for every i)

$$\mathsf{Dist}_{\mathsf{sd}}((\mathbf{v}_i\mathbf{h}^\top)^{-1}) \oplus z_i,$$

where $\mathbf{v}_i\mathbf{h}^\top$ is interpreted to mean applying the linear map $\mathbf{v}_i$ in the exponent of the group.

Note that by construction, for every i we have

$$\mathbf{v}_i\mathbf{h}^\top \cdot h_i^\perp = \mathbf{u}_i \cdot \mathbf{h} = g^{a_i x_i + b_i(1-x_i)}.$$

Therefore, if $\mathsf{Dist}_{\mathsf{sd}}(h_i^\perp \cdot g^{-1}) = \mathsf{Dist}_{\mathsf{sd}}(h_i^\perp) \oplus 1$, the receiver is guaranteed to decode the output correctly (because $\left(\mathbf{v}_i\mathbf{h}^\top\right)^{-1}$ is guaranteed to be either $h_i^\perp$ or $h_i^\perp \cdot g^{-1}$). Finally, by choosing t appropriately, this condition will hold with probability $1 - 1/n\lambda$, resulting in overall correctness error $1 - 1/\lambda$.

This scheme is then bootstrapped to $1 - \mathsf{negl}(\lambda)$ (verifiable) correctness by use of repetition and an error-correcting code with efficient erasure decoding as in [DGI+19]. □