SNARGs for Monotone Policy Batch NP

Zvika Brakerski
Weizmann Institute of Science

Maya Farber Brodsky
Tel Aviv University

Yael Tauman Kalai
Microsoft Research and MIT

Alex Lombardi
Simons Institute and UC Berkeley

Omer Paneth
Tel Aviv University

July 5, 2023

Abstract

We construct a succinct non-interactive argument (SNARG) for the class of monotone policy batch NP languages under the Learning with Errors (LWE) assumption. This class is a subclass of NP that is associated with a monotone function $f : \{0,1\}^k \to \{0,1\}$ and an NP language $\mathcal{L}$, and contains instances $(x_1, \ldots, x_k)$ such that $f(b_1, \ldots, b_k) = 1$ where $b_j = 1$ if and only if $x_j \in \mathcal{L}$. Our SNARGs are arguments of knowledge in the non-adaptive setting, and satisfy a new notion of somewhere extractability against adaptive adversaries.

This is the first SNARG under standard hardness assumptions for a sub-class of NP that is not known to have a (computational) non-signaling PCP with small locality. Indeed, our approach necessarily departs from the known framework of constructing SNARGs dating back to [Kalai-Raz-Rothblum, STOC '13].

Our construction combines existing quasi-arguments for NP (based on batch arguments for NP) with a novel ingredient which we call a *predicate-extractable hash* (PEHash) family. This notion generalizes the notion of a somewhere extractable hash. Whereas a somewhere extractable hash allows to extract a single input coordinate, our PEHash extracts a *global* property of the input. We view this primitive to be of independent interest, and believe that it will find other applications.

Contents

1 Introduction

Succinct non-interactive arguments (SNARGs) are a powerful cryptographic primitive whose feasibility is still poorly understood. Informally, a SNARG for an NP language $\mathcal{L}$ is a computationally sound non-interactive argument system for $\mathcal{L}$ whose proofs are short (much shorter than the length of an NP witness) and easy to verify.

In the random oracle model, it is known that there are SNARGs for *every* NP language [Kil92, Mic94]. However, constructing SNARGs for NP in the "plain model" under falsifiable and preferably standard cryptographic assumptions remains a grand challenge, and will require overcoming some serious barriers [GW11]. As such, the main focus of this work is making progress on the following question.

Which NP *languages have* SNARG*s in the standard model?*

Prior Positive Results. Constructing SNARGs in the standard model has received extensive attention over the last 15 years, both in the privately verifiable [KR09, KRR14, KP16, BHK17, BKK+18] and publicly verifiable [SW14, PR17, CCH+19, KPY19, JKKZ21, CJJ21b, CJJ21a, WW22] settings. At this point in time, the achieved results in the privately and publicly verifiable settings are similar,[1] so we focus on the latter. There are two main results to summarize:

1. [KPY19, CJJ21a, KVZ21] construct SNARGs for a *restricted* class of NP languages: those that have a (computational) non-signaling PCP with low locality,[2] where the size of the SNARG grows with the locality. This class of NP languages is known to include all languages in P [KRR14, BHK17] and all languages with low (read-once) non-deterministic space complexity [BKK+18].

2. [SW14] constructs a variant of SNARG for any NP language, with the following caveats:

   - The scheme requires a (non-succinct) common reference string that is as large as the NP verification circuit, and in particular as long as the instance and witness.
   - The scheme is only non-adaptively sound; this means that an adversarial prover is only unable to prove false statements that are fixed in advance (before the crs is sampled).
   - The scheme is secure assuming (in addition to one-way functions) the existence of indistinguishability obfuscation (iO) [BGI+01, GGH+13], which is not a falsifiable assumption [GGSW13].[3] This issue was partially circumvented by [JJ22] for some languages in NP ∩ coNP.
   - The scheme does not provide any knowledge extraction (or *argument of knowledge*) guarantee. That is, it is not possible to argue that a convincing prover in their argument system *knows* an NP witness for the statement x.

[1] There are some differences in the computational assumptions required for SNARGs in the two settings.

[2] Loosely speaking, a computational non-signaling PCP with locality ℓ consists of a distribution of answers for every set of ℓ PCP queries $q_1, \ldots, q_\ell$, with the guarantee that for any two sets of queries Q and Q', each of size ℓ, the two distributions of answers, when restricted to the queries $Q \cap Q'$ are computationally indistinguishable.

[3] There are constructions of iO under standard cryptographic hardness assumptions [JLS21] but these are assumed to be hard for 2^n-size adversaries (and thus can be falsified in time 2^n), where n is the instance size of the SNARG.

The Non-Signaling Barrier. Given result (1), it is natural to ask if we are already done – does this give us SNARGs for NP? Probably not. It is unlikely that all languages in NP have (computational) non-signaling PCPs with small locality. Indeed, it is known that if a language $\mathcal{L}$ has a non-signaling PCP with locality ℓ then $\mathcal{L}$ is contained in DTIME(2^ℓ) [DLN+04, KRR14]. Thus, under widely believed complexity assumptions, there is no non-signaling PCP for all of NP with non-trivial locality.

To explain why all previous SNARG constructions that are not based on iO are limited to languages with (computational) non-signaling PCPs, we recognize that all these constructions follow the same blueprint:

- Use cryptography to build a "quasi-argument system", which is an argument system for 3SAT with a weak soundness guarantee: any successful cheating prover P^* can be used to produce a distribution of "locally satisfying assignments" for each set of at most ℓ variables, such that the distributions are non-signaling.[4] The argument size grows with the locality ℓ.
- Use an information-theoretic encoding of an NP instance x into a 3SAT formula ϕ_x such that any locally satisfiable assignment distribution to ϕ_x gives an NP witness for x.

The power of this approach is inherently limited since deciding if a given formula ϕ has a locally satisfying assignment distribution with locality ℓ can be done in time $2^{O(\ell)}$. Therefore, it is unlikely that all languages in NP can be reduced to deciding local satisfiability. This limits the scope of results in this framework to a strict subclass of NP.

In this work, we show how to modify the above approach and circumvent the known complexity-theoretic barrier by **making additional use of cryptography**. More specifically, we replace the information-theoretic encoding ϕ_x above with a *cryptographic encoding*. Our encoding ensures that if ϕ_x has a locally satisfying assignment distribution that can be *generated efficiently* then we can reconstruct from it a global satisfying assignment for ϕ_x and recover an NP witness for x. In particular, even if we can can find a locally satisfying assignment for ϕ_x inefficiently, in time $2^{O(\ell)}$, we can no longer deduce that $x \in \mathcal{L}$.

Prior work beyond local satisfiability. The work of [KRR14] gives an information-theoretic encoding of any (unbounded space) deterministic computation into a formula such that local satisfiability implies global satisfiability. We remark that some follow-up works do make use of cryptographic encodings, but *not* in order to achieve SNARGs for new languages:

- [KP16, BHK17, CJJ21a] use a cryptographic encoding based on collision-resistant hash functions. This enables a reduced proof generation time from RAM computations, and allow fast verification of computations over "delegated memory".
- Encoding based on more sophisticated hash functions were used in [DGKV22, PP22, KLVW23] to improve the efficiency of "batch argument systems" (BARGs), which led to other applications such as a stronger notions of RAM delegation and constructions of incrementally verifiable computation.

To reiterate, **a non-signaling barrier remains:** we have yet to obtain SNARGs for languages that are not known to have corresponding non-signaling PCPs. In this work, we make use of new cryptographic encodings to do precisely this.

[4] The distribution are non-signaling if for any (local) sets of variables I and J, the corresponding assignments x_I and x_J are computationally indistinguishable on the variables $I \cap J$.

1.1 Our Results

In this work, we go beyond the information-theoretic non-signaling approach and construct SNARGs for new languages. More specifically, we construct a SNARG for the class of "monotone policy BatchNP" languages, which is a subclass of NP defined as follows. Fix any NP language $\mathcal{L}$ with witness relation $\mathcal{R}$ and any monotone function f. Let

$$\mathcal{L}_f^{(k)} = \Big\{(x_1, \ldots, x_k) : \exists (w_1, \ldots, w_k) \text{ s.t. } f(b_1, \ldots, b_k) = 1, \text{ where } b_i = \mathcal{R}(x_i, w_i)\Big\}.$$

That is, a statement $(x_1, \ldots, x_k)$ is in $\mathcal{L}_f^{(k)}$ if *enough* of the statements $x_1, \ldots, x_k$ are true to provide a satisfying input assignment to f. SNARGs for monotone policy BatchNP generalize the notion of batch arguments (BARGs) captured by the special case where f is the conjunction $b_1 \wedge b_2 \wedge \ldots \wedge b_k$. Our main result is a construction of SNARGs for $\mathcal{L}_f^{(k)}$ for every f that has a polynomial-size monotone circuit. We first state a version of our construction satisfying non-adaptive soundness.

Theorem 1.1 (informal, see Theorem 6.1)**.** *Assuming the polynomial hardness of learning with errors (*LWE*), there exist non-adaptively sound* SNARG*s for monotone policy* BatchNP *for all polynomial-size monotone circuit policies. Our* SNARG *has the following quantitative succinctness:*

- *The length of the* SNARG *proof is $m \cdot \mathsf{poly}(\lambda)$ – growing with the length of a single* NP *witness $|w_i| = m$ rather than k of them.*
- *The common reference string has length $(m + k) \cdot \mathsf{poly}(\lambda)$.*

In particular, the proof length of our SNARG matches the efficiency of BARGs, and achieving sublinear dependence on m would imply SNARGs for NP (by setting $k = 1$).

Our SNARG additionally has the following properties:

1. **Short CRS for low-width C.** The crs length in Theorem 1.1 can actually be reduced to $(m + \min\{k, \mathrm{width}(C)\}) \cdot \mathsf{poly}(\lambda)$, where $\mathrm{width}(C)$ is the width of the monotone circuit C.[5] In our circuit model (see Section 4), we allow wires in the $(i+1)$th layer to be computed from wires in the ith layer along with the input layer. This allows for sublinear circuit width which corresponds roughly to the space complexity of the evaluation of C.

   As a result, for circuits of width $\mathsf{poly}(\lambda, \log k)$, we obtain a fully succinct crs, as in BARGs.

2. **Argument of Knowledge.** Our SNARG is an argument of knowledge. Namely, for any prover that convinces the verifier to accept some non-adaptively chosen statement $(x_1, \ldots, x_k)$ there exists a PPT extractor that extracts valid witnesses for a subset $J \subseteq [k]$ of the instances, such that $f(b_1, \ldots, b_k) = 1$, where $b_j = 1$ if and only if $j \in J$.

3. **Efficient Verification.** In our protocol, the verifier runs in time polynomial in the proof length plus time linear in the length of the input $(C, x_1, \ldots, x_k)$. When the input has a succinct representation (or if the verifier has a hash of $(C, x_1, \ldots, x_k)$), verification is even faster.

[5] In fact, the crs length can be compressed down to a more delicate complexity measure of the circuit C related to "necessary subsets."

Importantly, we remark that Theorem 1.1 appears to be beyond the reach of the framework used by previous SNARG constructions, dating back to [KRR13]. Instead, we introduce a new technique for building SNARGs that we believe is of independent interest and likely to be used to obtain SNARGs for other NP languages.

Our new approach. To go beyond the non-signaling barrier, we use cryptography to ensure that local satisfiability holds and that a specific *global* predicate is satisfied. Unlike prior work (such as [KRR14]), our global predicate depends on the instance x. In our scheme, we encrypt a description of a predicate P in the crs, which allows us to change it in the analysis to a predicate P_x that depends on x. In fact, we make use of P_x that is not even efficiently computable given x; as long as P_x has a short description, we can hard-wire it in the analysis.

Somewhere Argument of Knowledge. Theorem 1.1 above achieves non-adaptive soundness (and argument of knowledge). We strengthen our result to achieve a variant of soundness against provers that can choose the statements $x_1, \ldots, x_k$ *adaptively* based on the common reference string. We consider a relaxation of adaptive soundness called somewhere extractability, generalizing the notion of somewhere extractability for batch arguments.

Theorem 1.2 (informal, see Theorem 7.1)**.** *Assuming the polynomial hardness of learning with errors, there exist* somewhere extractable SNARG*s for monotone policy* BatchNP *where the common reference string is of size* $(m+k) \cdot \mathsf{poly}(\lambda)$.

We define somewhere extractable SNARGs for $\mathcal{L}_f^{(k)}$ with respect to "necessary subsets" for the policy f. We say that a subset $J \subset [k]$ is necessary for f if every input $b_1, \ldots, b_k$ satisfying $f(b_1, \ldots, b_k) = 1$ has the property that $b_j = 1$ for some $j \in J$. For example, if f is the conjunction $b_1 \wedge b_2 \wedge \ldots \wedge b_k$, then every non-empty subset of $[k]$ is necessary.

Our somewhere extractability property requires that for every necessary subset $J \subset [k]$, it is possible to sample a computationally indistinguishable common reference string crs_J that is extractable on J in the following sense: if the prover adaptively chooses statements $x_1, \ldots, x_k$ and provides a proof π for $(x_1, \ldots, x_k)$, then it is possible to extract a valid NP witness w_j for *some* $j \in J$. In general, even an honest prover may not have a witness for multiple indices in J, so we cannot hope to extract more than one such witness. Relatedly, we remark that the common reference string in Theorem 1.2 *must*, in general, grow with k. This is because the common reference string crs_J must encode the set J, and the number of necessary subsets for a language $\mathcal{L}_f^{(k)}$ may be exponential in k.

Our somewhere extractability property immediately implies that if the crs is sampled to be extractable on J, an adversarial prover P^* cannot produce statements $x_1, \ldots, x_k$ and an accepting proof π such that $x_j \notin \mathcal{L}$ for all $j \in J$. Since this property is testable in time 2^m, under a subexponential security assumption the same must hold when the crs is sampled honestly. *However*, this is still weaker than a full adaptive soundness property: in principle, an adaptive adversary may still be able to prove a false statement $(x_1, \ldots, x_k) \notin \mathcal{L}_f^{(k)}$ such that the "violated" necessary subset J depends adaptively on the crs.

Comparison with [SW14]. Compared to the general NP SNARG result of [SW14], our results above (1) achieve significantly shorter common reference string for all monotone batch NP languages (as [SW14] requires a crs of size $k \cdot m \gg k + m$) and sometimes achieve a fully compact crs (for

width$(C) \leq \mathsf{poly}(\lambda, \log k)$), (2) achieve forms of somewhere extractability (with a crs of size $k + m$), and (3) rely on the (polynomial) hardness of LWE instead of indistinguishability obfuscation.

Low-depth Monotone Policies. Theorem 1.1 and Theorem 1.2 give SNARGs for $\mathsf{BatchNP}$ with policy implemented by any (polynomial-size) monotone circuit. We remark that the need to go beyond non-signaling comes from handling circuits C that have *high depth*. Specifically, we give a complementary, more direct construction of SNARGs for policies given by a *low-depth* monotone circuit.[6]

Theorem 1.3 (informal)**.** *Assuming 2^d-secure (somewhere extractable) BARGs and collision-resistant hash functions, there exist SNARGs for monotone policy $\mathsf{BatchNP}$ for all depth-d polynomial-size monotone circuit policies. The common reference string and the proof in this SNARG scheme have size $m \cdot \mathsf{poly}(d, \lambda)$.*

In particular, based on existing constructions of BARGs [CJJ21b, CJJ21a, WW22, KLVW23, CGJ+22], we get SNARGs for policies given by a *monotone formula* with CRS and proof of size $m \cdot \mathsf{poly}(\lambda)$ under polynomial LWE, DLIN or sub-exponential DDH. Unlike our other results, we prove Theorem 1.3 by (implicitly) constructing a suitable non-signaling PCP, which does not seem to be possible for high depth C.

A New Tool: Predicate Extractable Hash Functions. In order to prove Theorems 1.1 and 1.2, we introduce (and use) a new primitive called a *predicate extractable hash* (PEHash) family. This primitive generalizes the notion of a somewhere extractable hash family [HW15, OPWW15]. While the latter enforces binding to (and extractability of) a single input coordinate, our notion enforces binding to (and extractability of) a potentially *global* property of the input string x.

Specifically, a PEHash family is a hash family with local opening, such that a hash key hk encodes a secret predicate P. Given a trapdoor td_P for hk one can extract the predicate evaluation $P(x)$ from the hash value $\mathsf{v} = h(x)$. Somewhere extractable hash families have exactly this syntax when P is restricted to be an *index function* $x \mapsto x_i$.

In this work, we construct and use PEHash families for *bit-fixing predicates*, where each such predicate $P_{J,y}$ is associated with a subset $J \subseteq [|x|]$ and a string $y \in \{0,1\}^J$. The predicate is defined as

$$P_{J,y}(x) = \bigwedge_{j \in J} \mathbb{1}(x_j = y_j).$$

In other words, the predicate $P_{J,y}$ checks that the input string x matches y on all indices in J simultaneously.

Theorem 1.4 (informal)**.** *Assuming the polynomial hardness of LWE, there exist PEHash families for bit-fixing predicates. The size of the hash key is $\ell \cdot \mathsf{poly}(\lambda)$, where ℓ is a bound on the maximum size of a set J supported by the family.*

Defining security of PEH families is quite subtle. We defer a detailed discussion of this to the technical overview (Section 2), but we roughly require that it is computationally infeasible for an adversary to produce a hash value v along with a local opening at an index j that contradicts the value of the predicate P. A formal definition of this primitive can be found in Section 5.

We believe that the notion of PEH is of independent interest and will be useful in future SNARG constructions and elsewhere in cryptography.

[6] This is based on an unpublished work of Brakerski and Kalai [BK18] which is merged with this work.

2 Our Techniques

We first give an overview of our proof of Theorem 1.1, focusing on our new technique involving *predicate-extractable hash functions.* We mainly discuss the problem of obtaining short proof length; at the end, we briefly list additional ideas for minimizing the crs length and for obtaining somewhere extractable soundness.

Monotone Policy $\mathsf{BatchNP}$. Fix any NP language $\mathcal{L}$ with witness relation $\mathcal{R}$ and fix any function f computable by a monotone circuit family C. Recall that we wish to construct a SNARG for the NP language $\mathcal{L}_f^{(k)}$ defined as follows:

$$\mathcal{L}_f^{(k)} = \Big\{(x_1, \ldots, x_k) : \exists (w_1, \ldots, w_k) \text{ s.t. } f(b_1, \ldots, b_k) = 1, \text{ where } b_i = \mathcal{R}(x_i, w_i)\Big\}.$$

Our goal is to construct a SNARG for $\mathcal{L}_f^{(k)}$ with proof length $m \cdot \mathsf{poly}(\lambda)$, where m denotes the length of a single witness w_i.

Why don't BARGs just work? Before discussing our techniques, we briefly mention a naive idea that is fundamentally flawed but highlights a key difficulty of the problem. For any instance $(x_1, \ldots, x_k) \in \mathcal{L}_f^{(k)}$, there is a set of witnesses $\{w_i\}_{i \in S}$ for a subset $S \subset [k]$ such that $f(\chi_S) = 1$, where χ_S denotes the indicator vector for S. Therefore, one can prove that $(x_1, \ldots, x_k) \in \mathcal{L}_f^{(k)}$ using a BARG to prove the claim that "$x_i \in L$ for all $i \in S$."

This idea results in a sound argument system; however, *the verifier does not know the set S.* As a result, the prover would at least have to communicate S, which may require sending an additional $O(k)$ bits, ruining succinctness.

As a result, one key technical challenge in this work is finding a way to argue about these implicitly defined sets S of size $O(k)$ even though the SNARG proof cannot contain this much information. For a concrete example, one can think about the case where f is the *majority* function, where the sets S in question have size $\lceil \frac{k}{2} \rceil$.

2.1 The Canonical Protocol

We first describe a simple candidate SNARG for monotone policy $\mathsf{BatchNP}$. While we cannot show its soundness, our results are based on variants of this canonical protocol. Roughly following [CJJ21a, KVZ21], a candidate argument system for an *arbitrary* NP language can be built from a (somewhere extractable) *batch argument system* (BARG), which enables proving that a large number of NP statements are *all true*, at the communication cost of roughly one NP witness. Somewhere extractable BARGs are now known from a wide variety of assumptions, including LWE [CJJ21a].

Tailored to our setting, the candidate argument system for $\mathcal{L}_f^{(k)}$ is constructed as follows:

- Given an instance $(x_1, \ldots, x_k) \in \mathcal{L}_f^{(k)}$ together with a witness $(w_1, \ldots, w_k)$ compute the bits $b_j = \mathcal{R}_{\mathcal{L}}(x_j, w_j)$ for $1 \le j \le k$ and evaluate the circuit $C(b_1, \ldots, b_k)$.
- Compute a succinct commitment (i.e., tree hash) v of the values of all the wires in the evaluation of C.

- Use the batch argument system to prove the conjunction of the following $|C|+1$ statements:
  - *Input wires:* For every $j \in [k]$ there exists a witness w_j and a local opening of v in location j to a value b_j such that $b_j = \mathcal{R}_{\mathcal{L}}(x_j, w_j)$.
  - *Internal wires:* For every gate of C with wires $j, j_1, j_2 \in [|C|+1]$ there exist local openings of v in locations j, j_1, j_2 to values b_j, b_{j_1}, b_{j_2} that are consistent with the gate.
  - *Output wire:* There exists a local opening of v in location $|C|$ (corresponding to the output wire) to the value 1 (indicating that C accepts).

Since the witness to each claim proven in the BARG is of length at most $m + \mathsf{poly}(\lambda)$, the overall proof length is $m \cdot \mathsf{poly}(\lambda)$. This style of argument system enforces the *local consistency* of a claimed execution of the NP verifier for $\mathcal{L}_f^{(k)}$. Unfortunately, **we do not know whether, in general, this argument system is sound**!

Prior works can be thought of as using a variant of the canonical protocol with ℓ independent BARG executions (with respect to the same commitment), where ℓ is a locality parameter. In this case, the canonical protocol turns out to be a "quasi-argument" [KPY19] with locality ℓ.

Low-depth Circuits. Towards proving Theorem 1.1, we next describe how to obtain Theorem 1.3 using the canonical protocol (with locality $\ell = 2$). We focus on the case where C is a logarithmic depth monotone circuit. More generally, we can prove soundness for depth d circuits with proof length $m \cdot \mathsf{poly}(\lambda, d)$, under a 2^d-time security assumption.

As stated above, we consider a variant of the canonical protocol that includes two independent BARG executions with respect to the same commitment v. To prove soundness, we rely on the fact that the BARG is *somewhere extractable*. This means that for every $j \in [|C|+1]$, we can sample a computationally indistinguishable CRS_j together with a corresponding trapdoor td_j, so that given any accepting proof under CRS_j we can efficiently extract a witness for the jth claim from the proof:

$$\omega_j \leftarrow \mathsf{Extract}(\mathsf{td}_j, \pi).$$

Since we use BARGs with two independent CRSs, we are actually able to extract witnesses $(\omega_{j_1}, \omega_{j_2})$ for two of the claims at once. Moreover, we can extract from one of the BARG proofs while the index on which the other BARG is extractable remains hidden.

To argue non-adaptive soundness, fix a false statement $(x_1, \ldots, x_k) \notin \mathcal{L}_f^{(k)}$. For $j \in [k]$, let $b_j^* = 1$ if $x_j \in \mathcal{L}$ and $b_j^* = 0$ otherwise. Let $(b_j^*)_{1 \le j \le |C|}$ denote the values of all the wires in the evaluation of $C(b_1^*, \ldots, b_k^*) = 0$. The idea is to argue inductively that if an efficient adversary P^* breaks the argument system, then for every layer, there exists a wire j in the layer such that if the BARGs are extractable on the statement involving the jth wire value, then the committed value b_j in the extracted witness is *greater than* b_j^* (i.e., $b_j = 1$ but $b_j^* = 0$) with non-negligible probability.

For the output layer of the circuit, this is clear since $b_{|C|}^* = 0$, but $b_{|C|} = 1$ by the correctness of the output-wire statement. For the inductive step, assume that $b_j > b_j^*$ with probability p when the BARGs are extractable on the jth claim and let j_1, j_2 be the two *child* wires of the jth wire. Then for some $\alpha \in [2]$, we must have that $b_{j_\alpha} > b_{j_\alpha}^*$ with probability at least $p/2$ because the (j, j_1, j_2) gate is monotone (AND or OR). Furthermore, the same holds when the BARGs are extractable on the j_αth claim by a standard non-signaling argument.[7]

[7] Consider a hybrid experiment where one of our BARG proofs is extractable on the jth claim and the other BARG

Since the circuit is of logarithmic depth, we can apply this argument inductively and show that for one of the input wires $j \in [k]$ the extracted value b_j is greater than b_j^* with noticeable probability in the appropriate hybrid experiment. This is a contradiction because if $R_{\mathcal{L}}(x_j, w_j) = b_j^* = 0$ then by the correctness of the input-wire statement, $b_j = 0$ with all but negligible probability.

We can extend this analysis to consider circuits of any depth d where the loss in the reduction grows exponentially with d. We can still obtain soundness by relying on subexponential hardness assumptions; however, the security parameter and thus also the length of the SNARG proof must grow with d. We prove this formally in Section 8.

Avoiding the Exponential Decay. Constructing SNARGs for monotone policy BatchNP where the circuit C has *arbitrary* depth (without a depth dependence in the proof length) requires a new type of analysis where the success probability does not decay exponentially with the depth. Previous works [KRR13, KRR14] introduce two techniques for avoiding this exponential decay in the context of SNARGs for P. For completeness, we briefly describe these techniques and explain why they are insufficient in our context.

The solution of [KRR13] was restricted to circuits of small width, allowing the proof to grow with the width. In particular, in this setting it is possible to extract from the proof an entire layer of the circuit instead of just a single gate. Accordingly, in their inductive argument, the success probability only decreases by a negligible amount in each layer, instead of by a factor of 2.

To deal with circuits with unbounded width, [KRR14] proposed the following modification: instead of extracting an entire layer of the circuit (which would require the proof to grow), the prover augments each layer with a short Merkle hash of the layer.[8] Now, by extracting only the hash of the layer and comparing it against the hash of the layer values in the correct evaluation of the deterministic circuit we can certify that each value in the layer was computed correctly.

In our context, however, the circuit in question is non-deterministic and may have many possible evaluations. For example, if $x_j \in \mathcal{L}$ and $b_j^* = 1$, an evaluation using an invalid witness will give the jth input wire of C the value 0. Instead, we want to make sure that the value extracted for each wire b_j is not *greater than* the value b_j^* of the wire in the evaluation of $C(b_1^*, \ldots, b_k^*) = 0$ described above. Indeed, it is not at all clear that such global information can be discerned from a short hash of the layer.

2.2 Enforcing Global Properties with Predicate Extractable Hashing

Our solution for general circuits extends the paradigm from [KRR14] to the regime of non-deterministic languages.[9] We modify the "canonical protocol" by defining and using a more powerful cryptographic hash family to commit to the wire values $b_1, \ldots, b_{|C|}$. At a high level of generality, imagine that we want a hash function h that maps a long input x to a short output

proof is extractable on the j_αth claim. By CRS indistinguishability, the probability that the value $b_{j_\alpha}^{(1)}$ extracted from the *first* proof satisfies $b_{j_\alpha} > b_{j_\alpha}^*$ is close to that of the experiment where both proofs are extractable on j. Similarly, the probability that $b_{j_\alpha}^{(2)} > b_{j_\alpha}^*$ matches the experiment where both proofs are extractable on j_α. Finally, since both claims argue about openings of the commitment v to b_{j_α}, the binding property of the commitment $b_{j_\alpha}^{(1)} = b_{j_\alpha}^{(2)}$ with all but negligible probability.

[8] In fact, [KRR14] proposed an information theoretic analog of this idea where the short hash is replaced by a few random locations in the low-degree extension of the layer.

[9] In contrast, the work of [BKK+18] gives an analysis for read-once *bounded space* non-deterministic computations, extending the earlier work of [KRR13]. This approach is not applicable in our setting.

v, such that the evaluation $\mathsf{v} = h(x)$ is binding to a (potentially arbitrary) predicate $P(x)$. We formalize this with a following syntax for a "predicate-extractable hash family" PEHash:

- Given the description of a predicate P, it should be possible to sample a hash key hk_P along with a trapdoor td_P. The key hk_P should computationally hide the choice of predicate P.
- Given a hash key hk and an input x, we can compute a short hash $\mathsf{v} = H(\mathsf{hk}, x)$.
- As in a standard hash tree, we require that it is possible to locally open a hash value v on an index j to the bit x_j. The opening should be short and $\mathsf{poly}(\lambda)$-efficient to verify.
- Finally, given a hash value v and trapdoor td_P, it should be possible to *extract* a bit $\mathsf{Extract}(\mathsf{td}_P, \mathsf{v})$ which is supposed to correspond to the evaluation $P(x)$ (at least on an honestly generated $\mathsf{v} = H(\mathsf{hk}, x)$).

We view this object as a syntactic generalization of somewhere statistically binding (and somewhere extractable) hash families [HW15, CJJ21a], which have proved to be extremely useful in the construction of cryptographic protocols. In our formalization, somewhere extractable hash families correspond to the case where the predicate P is an *index* function $x \mapsto x_i$. In this special case, security is defined as follows: if the hash key hk is sampled to be binding on index i, then it is infeasible for an adversary to produce a hash value v and an opening of v to a bit b on location i such that $\mathsf{Extract}(\mathsf{td}_i, \mathsf{v}) = 1 - b$. In other words, the ith location opening must be consistent with the extracted bit value.

PEHash families for bit-fixing predicates. In this work, we define, construct, and make use of predicate-extractable hash families for "bit-fixing" predicates. That is, for inputs of length N, a predicate P is specified by a subset $J \subseteq [N]$ and a string $y \in \{0,1\}^J$. The predicate $P_{J,y}(x)$ is defined to be 1 if $y_j = x_j$ for all $j \in J$.

Defining security for bit-fixing PEH requires significant care. Specifically, security is asymmetric with respect to whether $P(x) = 1$ or $P(x) = 0$.

- Security in the $P(x) = 1$ case is relatively easy to describe. We require that if the hash key hk is extractable on bit-fixing predicate (J, y), an adversary cannot produce a hash value v such that $\mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 1$ together with an opening of v to a bit b on index $j \in J$ such that $b \neq y_j$. We note that the adversary here can choose j adaptively. This corresponds to the intuition that an evaluation $\mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 1$ should *simultaneously* bind every value x_j to y_j.
- Security in the $P(x) = 0$ case is more subtle. Intuitively, we want to say that if $\mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 0$ then the adversary *cannot* simultaneously open on every index $j \in J$ to y_j. However, this security property is insufficiently "succinct" to be useful in our construction. Instead, we require that, in addition to the predicate value, we can extract a specific index $j^* = \mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v})$ such that the adversary cannot open v to y_{j^*} on the j^*th location.

Constructing PEHash for bit-fixing predicates. The high-level idea for constructing a bit-fixing PEHash family is simple and takes inspiration from [HW15]: we can hash a string x in a way that is binding to $P_{J,y}$, via a tree of homomorphic evaluations on pairs of ciphertexts. In more detail, the hash key will contain an encryption of the predicate (J, y). We hash x as follows:

- First, for every j we homomorphically evaluate $\mathsf{Enc}(\mathbb{1}((x_j = y_j) \vee (j \notin J)))$. This results in a collection of "leaf ciphertexts" that we want to homomorphically AND together.
- To compute a parent ciphertext ct_j corresponding to some prefix $j \in \{0,1\}^{\leq \log k}$ given sibling ciphertexts $\mathsf{ct}_{j||0}, \mathsf{ct}_{j||1}$ encrypting bits $z_{j||0}, z_{j||1}$, we homomorphically evaluate $z_{j||0} \wedge z_{j||1}$.
- To enforce correctness with respect to a malicious committer, we implement the evaluation tree with an FHE bootstrapping mechanism (each layer has its own FHE secret key, and homomorphic operations are always computed on ciphertexts encrypting sk_{i+1} under pk_i, which are included as part of the hash key).
- The construction above is sufficient to satisfy "one-sided" ($P(x) = 1$) security. To obtain two-sided security, we have the FHE evaluation tree also keep track of the lexicographically first index $j^* \in J$ on which $x_j \neq y_j$. This invariant can be maintained throughout the evaluation.

As a result, an honestly evaluated hash of x will be an encryption of $(P_{J,y}(x), j^*)$, where $j^* = \perp$ if $P_{J,y}(x) = 1$ and j^* equals to the lexicographically first index such that $x_j \neq y_j$ if $P_{J,y}(x) = 0$.

We prove in Section 5.3.1 that this construction satisfies both sides of our security definition. However, it is unfortunately not succinct enough: verifying an opening requires recomputing a path along the tree (as is typical for Merkle-style commitments), but each step of our tree evaluation requires doing a computation involving $\mathsf{Enc}(J)$, which is not succinct if J is a large set. Fortunately, this opening verification can be delegated to the opening *generation* using a RAM SNARG, enabling the verifier to check correctness efficiently given a short hash of hk (generated at setup time).

2.3 New SNARG Construction

Having defined and constructed a predicate extractable hash family for bit-fixing predicates, we now show how to use it to build a SNARG for monotone policy BatchNP. Our SNARG construction is similar in structure to the canonical construction, but (for reasons that will become clear later) in place of a the hash tree we make use of *two* instantiations of a PEHash family. Roughly speaking, the SNARG proof is computed as follows.

- Using two independent hash keys $\mathsf{hk}^{(1)}, \mathsf{hk}^{(2)}$, succinctly commit to $b_1, \ldots, b_{|C|}$, which are all the wire values of the circuit C. Let $\mathsf{v}^{(1)}, \mathsf{v}^{(2)}$ denote the two resulting hash values.
- Use the batch argument system to prove the conjunction of the following $|C|+1$ statements:
  - *Input wires:* For every $j \in [k]$ there exists a witness w_j and local openings of $\mathsf{v}^{(1)}, \mathsf{v}^{(2)}$ in location j to a value b_j such that $b_j = \mathcal{R}_{\mathcal{L}}(x_j, w_j)$.
  - *Intermediate wires:* For every gate of C with wires $j, j_1, j_2 \in [|C|]$ there exist values b_j, b_{j_1}, b_{j_2} that are consistent with the gate and local openings of $\mathsf{v}^{(1)}, \mathsf{v}^{(2)}$ in locations j, j_1, j_2 to b_j, b_{j_1}, b_{j_2}.
  - *Output wires:* The final wire value of C is 1 (according to both commitments $\mathsf{v}^{(1)}, \mathsf{v}^{(2)}$).

How are we better equipped to prove soundness of this argument system? The idea is that predicate extractability gives us a means to argue about the consistency of an entire layer of C based on a single bit.

More specifically, to argue non-adaptive soundness, we again fix a false statement $(x_1, \ldots, x_k)$ and define "correct wire values" $b_1^*, \ldots, b_{|C|}^*$ as before. Next, for a fixed layer i, we switch to a computationally indistinguishable world in which the predicate $P^{(1)}$ on which $\mathsf{hk}^{(1)}$ is extractable is the predicate $P_{J_i,0}$ that checks for the "forced zeroes" of the circuit evaluation. That is, $P(b_1, \ldots, b_{|C|})$ is defined to be 1 if and only if $b_j = 0$ for all j in the ith layer of C such that $b_j^* = 0$. We know that in any honest evaluation of the circuit, this predicate $P^{(1)}$ would evaluate to 1.

We first argue that if an adversary cheats and produces an accepting proof for a false statement with $i = d$ then it must be the case that the extracted value of $P^{(1)}(\cdot)$ is 0 (except with negligible probability). This holds because if the proof is accepting then (by the soundness of the BARG) it must be that $b_{|C|} = 1$ and hence $P^{(1)} = 0$.

The key step is then to argue inductively over the layers; this is where we make use of our two independent hash keys $\mathsf{hk}^{(1)}, \mathsf{hk}^{(2)}$. Specifically, we argue that given $\mathsf{hk}^{(1)}$ and $\mathsf{hk}^{(2)}$ encoding predicates $P^{(i)}$ and $P^{(i-1)}$, for any adversary that generates a valid proof $(\mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi)$, it must be the case that $\mathsf{Extract}(\mathsf{td}^{(1)}, \mathsf{v}^{(1)}), \mathsf{Extract}(\mathsf{td}^{(2)}, \mathsf{v}^{(2)})$ *agree* (except with negligible probability). This follows from the binding properties of the PEHash along with the (extractable) soundness of the batch argument system, because:

- If $\mathsf{Extract}(\mathsf{td}^{(1)}, \mathsf{v}^{(1)}) = 0$, there must be some specific index j^* on which $b_j > b_j^*$ (where b_j is defined if the seBARG is extractable on the jth gate).
- Thus, if the jth gate is correct, there exists a child j_α such that $b_{j_\alpha} > b_{j_\alpha}^*$.
- However, this implies that we should have $\mathsf{Extract}(\mathsf{td}^{(2)}, \mathsf{v}^{(2)}) = 0$ except with a negligible error probability.

Combined with non-signaling arguments, this allows us to inductively argue about predicates $P_{j_i,0}$ corresponding to layer $i = d, d-1, \ldots, 1$. However, if the prover produces a proof where the predicate $P_{j_1,0}$ evaluates to 0, this is also a contradiction: this means that some b_j is equal to 1 for an unsatisfiable x_j, contradicting the somewhere extractability of the BARG.

Crucially, the "exponential decay" in success probability from layer to layer has disappeared – the local gate-by-gate analysis has been replaced by a global layer-by-layer analysis that suffers from a linear loss in the depth, as opposed to an exponential loss.

2.4 Achieving Somewhere Extractability

Having sketched our proof of Theorem 1.1, we now turn our attention to Theorem 1.2. In this setting, we no longer have a fixed false statement $\vec{x} = (x_1, \ldots, x_k)$. Instead, we have a subset $J \subseteq [k]$ of instances that is *necessary* for C (meaning that if C evaluates to 1 then $b_j = 1$ for some $j \in J$). We would like to modify our analysis so that given a prover $\mathcal{P}^*$ that produces instances $(\tilde{x}_1, \ldots, \tilde{x}_k)$ and a proof π that makes the verifier accept, we would like to extract a witness w_j (associated to $\tilde{x}_j$) for some $j \in J$.

We begin by following the analysis of the non-adaptive case. However, rather than programming the PEHash on predicates $P_{J_i,0}$ that check for zeroes in layer i that are "forced" by fixed false statements (x_j), we instead set J_i to be the set of wires that are forced to be 0 by the condition that $b_j = 0$ for all $j \in J$. Then, if $\mathcal{P}^*$ produces an accepting proof, we again must have $\mathsf{Extract}(\mathsf{td}^{(1)}, \mathsf{v}^{(1)}) = 0$ when using the predicate $P_{J_d,0}$. By a similar layer-by-layer analysis to the non-adaptive case, we can conclude that $\mathsf{Extract}(\mathsf{td}^{(1)}, \mathsf{v}^{(1)}) = 0$ for the predicate $P_{J_0,0}$, where $J_0 = J$.

Thus, if we program the crs such that the PEHash is extractable on $P_{J,0}$, we can use the trapdoor $\mathsf{td}^{(1)}$ to extract from the proof an index $j^* = \mathsf{ExtractIndex}(\mathsf{td}^{(1)}, \mathsf{v}^{(1)}) \in J$ where, intuitively, the predicate must be violated, so we should have that $x_{j^*} \in \mathcal{L}$. However, we do not yet have a mechanism for extracting the witness w_{j^*}.

One may hope that we would be able to extract this witness from the seBARG. Unfortunately, it is not clear how to do this because the index j^* *depends on the adversary's behavior*, so a seBARG trapdoor on j^* cannot be programmed in advance (unless you set the seBARG to be extractable on all of J, which would ruin succinctness).

Predicate extractable hash families with tags. To overcome this issue, we extend our notion of PEHash to one which would enable us to extract the witness from $\mathsf{v}^{(1)}$. Specifically, we define and construct an extended notion of PEHash families for bit-fixing predicates, which we call PEHash families with tags. This object supports hashing the input $x \in \{0,1\}^N$ together with tags $t_1, \ldots, t_N$, attaching a tag to each bit of the input. Similarly, the opening and verification algorithms also receive tags.

We correspondingly strengthen our security definition in the $P(x) = 0$ setting. Previously, we required that if $\mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 0$ then the adversary cannot open the index $j^* = \mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v})$ to the bit y_{j^*}. Now, we further require that the adversary is bound to a *specific tag* $t = \mathsf{ExtractTag}(\mathsf{td}, \mathsf{v})$ on the index j^*. This extended notion will enable us to extract a witness w_{j^*} for x_{j^*} at the end of the security analysis above.

We construct a PEHash family with tags (for bit-fixing predicates) by making a simple modification to our original construction: in the FHE evaluation tree, in addition to (b, j), we also keep track of the tag t attached to the index j. An honestly evaluated hash is an encryption of $(P_{J,y}(x), j^*, t)$ where t is the tag attached to index j^*, which guarantees that the adversary is bound to t when opening j^*.

2.5 Shortening the CRS

Finally, we briefly describe an optimization that allows us to shorten the prover common reference string $\mathsf{crs}_{\mathcal{P}}$:

- In our non-adaptive SNARG, we can reduce the length of $\mathsf{crs}_{\mathcal{P}}$ from $(m + \mathrm{width}(C)) \cdot \mathsf{poly}(\lambda)$ to $(m + \min(\mathrm{width}(C), k)) \cdot \mathsf{poly}(\lambda)$.
- In our somewhere extractable SNARG, we can reduce the length of $\mathsf{crs}_{\mathcal{P}}$ from $(m + k + \mathrm{width}(C)) \cdot \mathsf{poly}(\lambda)$ to $(m + k)\mathsf{poly}(\lambda)$.

This is especially useful in our somewhere extractable SNARG, where the size of $\mathsf{crs}_{\mathcal{P}}$ must be at least k since it is binding on a subset $J \subseteq [k]$ of input wires, so the optimization allows us to achieve a length $(m + k) \cdot \mathsf{poly}(\lambda)$ which is optimal.

To understand the optimization, we recall that the length of $\mathsf{crs}_{\mathcal{P}}$ grows with $\mathrm{width}(C) \cdot \mathsf{poly}(\lambda)$ due to the PEHash hash key, which is as long as a description of the predicate $P_{J_i,y}$. In our SNARG, we use the predicates $P_{J_i,0}$ where J_i is a set of wires in a single layer i of the circuit C, so its size grows with the largest layer of C, i.e. its width.

However, we observe that the sets J_i are not arbitrary sets of wires in a layer; they can be described as the set of wires in a layer that are 0 in a computation of C on some input $(b_1, \ldots, b_k)$.

Therefore, one can alternatively encode the predicate $P_{J_i,0}$ using an FHE encryption of some string in $\{0,1\}^k$.

3 Preliminaries

Notations. 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 $x \in \{0,1\}^n$ and any subset $J \subset [n]$ we denote by $x_J = (x_j)_{j\in J}$. 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 the 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].[10]

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

$\mathsf{Gen}(1^\lambda) \to \mathsf{hk}$. This is a PPT algorithm that takes as input the security parameter 1^λ in unary and outputs a hash key hk.

$\mathsf{Hash}(\mathsf{hk}, x) \to \mathsf{rt}$. This is a deterministic poly-time algorithm that takes as input a hash key hk and an input $x \in \{0,1\}^N$ for $N \le 2^\lambda$, and outputs a hash value rt.

$\mathsf{Open}(\mathsf{hk}, x, j) \to \rho$. This is a deterministic poly-time algorithm that takes as input a hash key hk, an input $x \in \{0,1\}^N$ for $N \le 2^\lambda$, and an index $j \in [N]$, and outputs an opening ρ.

$\mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, j, b, \rho) \to 0/1$. This is a deterministic poly-time algorithm that takes as input a hash key hk, a hash value rt, an index $j \in [N]$, a bit $b \in \{0,1\}$ and an opening ρ. It 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.*

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

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

Succinctness. *In the completeness experiment above, we have that $|\mathsf{hk}|+|\mathsf{rt}|+|\rho| = \mathsf{poly}(\lambda)$.*

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

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

[10] 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.

Remark 3.1. We say that a hash family with local opening is T-secure, for $T = T(\lambda)$, if the collision resistance w.r.t. opening property holds against any $\mathsf{poly}(T)$-size adversary (as opposed to $\mathsf{poly}(\lambda)$-size) and the probability that the adversary finds a collision is $\mathsf{negl}(T)$ (as opposed to $\mathsf{negl}(\lambda)$. We refer to this property as T-collision-resistance w.r.t. opening.

Remark 3.2. One can naturally extend the definition of a hash family with local opening to allow the Open algorithm to take as input (hk, x, J) where $J \subseteq [N]$ consists of a set of indices, as opposed to a single index. $\mathsf{Open}(\mathsf{hk}, x, J)$ will simply run $\mathsf{Open}(\mathsf{hk}, x, j)$ for every $j \in J$. Verify can be extended in a similar way to take as input $(\mathsf{hk}, \mathsf{rt}, J, b_J, \rho_J)$, and accept if and only if $\mathsf{Verify}(\mathsf{hk}, \mathsf{rt}, j, b_j, \rho_j) = 1$ for every $j \in J$.

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

3.2 Fully Homomorphic Encryption

In this section we define fully homomorphic encryption.

Syntax. A fully homomorphic encryption scheme consists of a fixed ciphertext size $\ell_{ctxt} = \ell_{ctxt}(\lambda)$ and the following polynomial time algorithms:

$\mathsf{FHE.Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$. This is a probabilistic algorithm that takes as input a security parameter 1^λ. It outputs a public key pk and a secret key sk.

$\mathsf{FHE.Enc}_{\mathsf{pk}}(b) \to c$. This is a probabilistic algorithm that takes as input a public key pk and a bit $b \in \{0,1\}$. It outputs a ciphertext $c \in \{0,1\}^{\ell_{ctxt}}$.

$\mathsf{FHE.Dec}_{\mathsf{sk}}(c) \to b$. This is a deterministic algorithm that takes as input a secret key sk and a ciphertext $c \in \{0,1\}^{\ell_{ctxt}}$. It outputs a bit $b \in \{0,1\}$.

$\mathsf{FHE.Eval}_{\mathsf{pk}}(f, c_1, \ldots, c_n) \to c^*$. This is a deterministic algorithm that takes as input a public key pk, a circuit representing a function $f\colon \{0,1\}^n \to \{0,1\}$ and n ciphertexts $c_1, \ldots, c_n \in \{0,1\}^{\ell_{ctxt}}$. It outputs a ciphertext $c^* \in \{0,1\}^{\ell_{ctxt}}$.

Definition 3.3 (FHE)**.** *A fully homomorphic encryption scheme* $\mathsf{FHE} = (\mathsf{FHE.Setup}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$ *is required to satisfy the following properties:*

- **Encryption Correctness.** *For any choice of* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$*, any* $b \in \{0,1\}$ *and any* $c \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}}(b)$ *we have* $\mathsf{FHE.Dec}_{\mathsf{sk}}(c) = b$*.*
- **Evaluation Correctness.** *For any choice of* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$*, any ciphertexts* $c_1, \ldots, c_n \in \{0,1\}^{\ell_{ctxt}}$ *such that* $\mathsf{FHE.Dec}_{\mathsf{sk}}(c_i) = b_i \in \{0,1\}$ *and any circuit* $f\colon \{0,1\}^n \to \{0,1\}$*, if we set* $c = \mathsf{FHE.Eval}(f, c_1, \ldots, c_n)$ *then* $\mathsf{FHE.Dec}_{\mathsf{sk}}(c) = f(b_1, \ldots, b_n)$*.*
- **Security.** *The encryption scheme is semantically secure.*

Remark 3.3. Given a FHE scheme, one can extend the definition of the encryption algorithm $\mathsf{FHE.Enc}$ to take as input a longer message $m \in \{0,1\}^n$, as opposed to a single bit. $\mathsf{FHE.Enc}_{\mathsf{pk}}(m)$ will simply run $c_i = \mathsf{FHE.Enc}_{\mathsf{pk}}(m_i)$ for every $i \in [n]$, and set $c = (c_1, \ldots, c_n) \in \{0,1\}^{n \cdot \ell_{ctxt}}$. Similarly, we extend $\mathsf{FHE.Eval}$ to take as input a function $f\colon \{0,1\}^n \to \{0,1\}^u$ with multi-bit output.

3.3 Somewhere Extractable Batch Arguments (seBARGs)

A batch argument system BARG for an NP language $\mathcal{L}$ enables proving that k NP statements are true with communication cost that is polylogarithmic in k. There are many BARG variants which are known to be existentially equivalent under mild computational assumptions (see, e.g., [CJJ21a, KVZ21, KLVW23]). In this work, for simplicity in our constructions, we make use of an argument system for what we call "batch index Turing machine SAT" (BatchTMSAT), defined below.

Definition 3.4. *The language* BatchIndexTMSAT *consists of instances of the form* $x = (M, z, k, T)$*, where:*

- M *is the description of a Turing machine.*
- z *is an input string (to* M*)*
- k *is a batch size, and*
- T *is a running time.*

An instance $x = (M, z, k, T)$ *is in* BatchIndexTMSAT *if for all* $i \leq i \leq k$*, there exists a string* w_i *such that* $M(z, i, w_i)$ *accepts within* T *steps.*

We sometimes use the notation $\mathcal{R}(x, i, w_i)$ to denote the relation with instance (x, i) and corresponding witness w_i.

Syntax. A (publicly verifiable and non-interactive) somewhere extractable batch argument system seBARG for BatchIndexTMSAT consists of the following polynomial time algorithms:

$\mathsf{Gen}(1^\lambda, 1^n, 1^m, i^*) \to (\mathsf{crs}, \mathsf{td})$. This is a probabilistic polynomial-time algorithm that takes as input a security parameter 1^λ, input length 1^n, witness length 1^m, and an index $i^* \in [2^\lambda]$. It outputs a common reference string crs along with a trapdoor td.

$\mathcal{P}(\mathsf{crs}, M, z, 1^T, w_1, \ldots, w_k) \to \pi$. This deterministic polynomial-time algorithm takes as input a crs, Turing machine M, input z, runtime 1^T, and k witnesses $w_1, \ldots, w_k$. It outputs a proof π.

$\mathcal{V}(\mathsf{crs}, x, \pi) \to 0/1$. This deterministic polynomial-time algorithm takes as input a crs, instance $x = (M, z, k, T)$, and a proof π. It outputs a bit (1 to accept, 0 to reject).

$\mathsf{Extract}(\mathsf{td}, \pi) \to w^*$. This deterministic polynomial-time algorithm takes as input a trapdoor td and a proof π. It outputs a single witness w^*.

Definition 3.5 (seBARG)**.** *A somewhere-extractable batch argument scheme* $\mathsf{seBARG} = (\mathsf{Gen}, \mathcal{P}, \mathcal{V}, \mathsf{Extract})$ *for* BatchIndexTMSAT *is required to satisfy the following properties:*

Completeness. *For any* $\lambda \in \mathbb{N}$*, any* $k(\lambda), n(\lambda), m(\lambda), T(\lambda) \leq 2^\lambda$*, any instance* $x = (M, z, k, T) \in$ BatchIndexTMSAT *with* $|M|+|z| = n$*, any corresponding witnesses* $w_1, \ldots, w_k \in \{0,1\}^m$ *and any index* $i^* \in [k]$*,*

$$\Pr\left[\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \quad : \quad \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^m, i^*), \\ \pi \leftarrow \mathcal{P}(\mathsf{crs}, M, z, 1^T, w_1, \ldots, w_k) \end{array} \right] = 1.$$

Efficiency. *In the completeness experiment above,* $|\mathsf{crs}|+|\pi| \leq m \cdot \mathsf{poly}(\lambda, \log(knT))$. *The running time of the verifier is at most* $\mathsf{poly}(|\mathsf{crs}|+|\pi|) + \mathsf{poly}(\lambda) \cdot |x|$.

Index hiding. *For any poly-size adversary* $\mathcal{A}$ *and any polynomials* $k(\lambda), n(\lambda), m(\lambda)$, *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$ *and every pair of indices* $i_0, i_i \in [k]$,

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

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

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \\ \wedge\ (x, i^*, w^*) \notin \mathcal{R} \end{array} \quad : \quad \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^m, i^*) \\ (M, z, \pi) = \mathcal{A}(\mathsf{crs}) \\ w^* \leftarrow \mathsf{Extract}\,(\mathsf{td}, \pi) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Remark 3.4. We say that a seBARG scheme is T-secure, for $T = T(\lambda)$, if the index hiding property and the somewhere argument of knowledge property hold w.r.t. a $\mathsf{poly}(T)$-size adversary (as opposed to a $\mathsf{poly}(\lambda)$-size), and the advantage probability is $\mathsf{negl}(T)$ (as opposed to $\mathsf{negl}(\lambda)$). We refer to these properties as T-index-hiding and T-somewhere-argument-of-knowledge, respectively.

Throughout this paper, when we refer to a BARG or seBARG, we will implicitly mean a seBARG for BatchIndexTMSAT.

Remark 3.5. Given an seBARG, one can naturally extend the definition of the key generation algorithm Gen to take as input an index *set* $I \subset [k]$, as opposed to a single index. $\mathsf{Gen}(1^\lambda, 1^n, 1^m, I)$ will simply run $\mathsf{Gen}(1^\lambda, 1^n, 1^m, i)$ for every $i \in I$. The prover algorithm $\mathcal{P}$, given a crs that encodes the $|I|$ indices, will simply generate $|I|$ proofs (one for each crs), and the verifier will check these $|I|$ proofs independently.

Theorem 3.6 ([CJJ21a,WW22,KLVW23,CGJ+22]). *There exists an* seBARG *for* BatchIndexTMSAT *assuming* LWE *or* DLIN *or subexponential* DDH.

Remark 3.6. While seBARGs for BatchIndexTMSAT were not discussed explicitly in the above works, they can be constructed generically from all previous "flavors" of BARG along with a somewhere extractable hash family with local opening. This follows from a similar proof to that of Theorem 12 in [CJJ21a]; an index seBARG for batch circuit SAT can be run to batch verify the executions of a RAM delegation scheme (or memory delegation scheme), which checks the computation of $M(z, i, w_i)$ in time $\mathsf{poly}(\lambda, \log k, \log T)$ given a tree hash of (M, z).

3.4 RAM SNARGs

In this section we define RAM SNARGs.

A RAM machine $\mathcal{R}$ is modeled as a deterministic machine with random access to memory of size 2^ℓ bits and a local state of size S. At every step, the machine reads or writes a single memory bit and updates its state.

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.

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

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

$\mathsf{Digest}(\mathsf{crs}, x_{\mathsf{imp}}) \to \mathsf{d}$**.** This is a deterministic algorithm that takes as input a crs and a string x_{imp}. It 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 algorithm that takes as input a crs and a pair $(x_{\mathsf{imp}}, x_{\mathsf{exp}})$ which consists of a (long) input x_{imp} and a (short) input x_{exp}. It 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 algorithm that takes as input a crs, a digest d of the long input, a short input x_{exp}, a bit $b \in \{0,1\}$ and a proof π. It outputs a bit (1 to accept, 0 to reject).

Definition 3.7 (RAM SNARG)**.** *A* RAM SNARG *for machine $\mathcal{R}$ with local state of size $S \geq |x_{\mathsf{exp}}| + \log |x_{\mathsf{imp}}|$ is required to satisfy the following properties:*

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*

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

Efficiency. *In the completeness experiment above, the length of* crs *and the proof π is at most* $\mathsf{poly}(\lambda, S, \log T)$.

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

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

Soundness. *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 $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{ll} \mathcal{V}(\mathsf{crs}, \mathsf{d}, x_{\mathsf{exp}}, b, \pi) = 1 \wedge & : \quad \mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, T) \\ \mathsf{Digest}(\mathsf{crs}, x_{\mathsf{imp}}) = d \wedge & \quad (\mathsf{d}, x_{\mathsf{imp}}, x_{\mathsf{exp}}, b, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}}) \neq b & \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Theorem 3.8 ([CJJ21a, WW22, KLVW23, CGJ+22])**.** *There exists a* RAM SNARG *assuming* LWE *or* DLIN *or subexponential* DDH.

4 SNARGs for Monotone Policy BatchNP

Monotone Policy BatchNP Model. Fix any NP language $\mathcal{L}$ with witness relation $\mathcal{R}$ and fix any function f computable by a monotone circuit family C.

The complexity of C (and its topology) is described in the following way:

- C has some input length $k = k(n)$ and size $|C| = s = s(n)$. These parameters are functions of n, the length of an instance of $\mathcal{L}$.
- C may have a more efficient uniform description, i.e., polynomial-time Turing machine M that generates C on input aux of length less than s.
- The circuit C is *layered* if the wires of C can be partitioned into "layers" $J_0, J_1, \ldots, J_d \subset [s]$ such that for every gate $(j, j_0, j_1, c \in \{\text{AND}, \text{OR}\}) \in \text{Gates}(C)$, if $j \in J_i$ then $j_0, j_1 \in J_{i-1} \cup J_0$. In other words, wire values in layer i are computed from wires in layer $i-1$ along with input wires.
- We say that a layered circuit C has *depth* d if it has d layers (not including the input layer), and has *width* w if each layer (besides the input layer) has size at most w. We note that this notion allows for C to have width less than k.

In what follows, we construct a SNARG for the NP language $\mathcal{L}_f^{(k)}$ defined as follows:

$$\mathcal{L}_f^{(k)} = \Big\{(x_1, \ldots, x_k) : \exists (w_1, \ldots, w_k) \text{ s.t. } f(b_1, \ldots, b_k) = 1, \text{ where } b_i = \mathcal{R}(x_i, w_i)\Big\}.$$

We first define the notion of a SNARG for $\mathcal{L}_f^{(k)}$.

4.1 Definition

We now define succinct non-interactive arguments (SNARGs) for languages of the form $\mathcal{L}_f^{(k)}$ above. The syntax follows that of SNARGs for general (NP) languages; however, similarly to the case of batch arguments (Definition 3.5), we allow the proof size (and other parameters) to grow with the size of a single NP witness w_i.

Syntax. A SNARG for $\mathcal{L}_f^{(k)}$ consists of the following PPT algorithms:

$\mathsf{Gen}(1^\lambda, k, n) \to (\mathsf{crs}_\mathcal{P}, \mathsf{crs}_\mathcal{V})$**.** This is a probabilistic algorithm that takes as input the security parameter 1^λ as well as a batch size k and instance size n. It outputs a common reference string $(\mathsf{crs}_\mathcal{P}, \mathsf{crs}_\mathcal{V})$, separated into two parts for efficiency reasons.

$\mathcal{P}(\mathsf{crs}_\mathcal{P}, C, x_1, \ldots, x_k, w_1, \ldots, w_k) \to \pi$**.** This is a deterministic polynomial-time algorithm that takes as input the crs $\mathsf{crs}_\mathcal{P}$, a description of the monotone circuit C, the k NP instances $x_1, \ldots, x_k$, and corresponding witnesses $w_1, \ldots, w_k$. It outputs a proof string π.

$\mathcal{V}(\mathsf{crs}_\mathcal{V}, C, x_1, \ldots, x_k, \pi) \to 0/1$**.** This is a deterministic polynomial-time algorithm that takes as input the verification key $\mathsf{crs}_\mathcal{V}$, a description of C, the k NP instances $x_1, \ldots, x_k$, and a proof string π. It outputs a bit (1 to accept, 0 to reject).

We will sometimes drop C from our algorithm notation, as it is fixed once and for all based on the language $\mathcal{L}_f^{(k)}$ and choice of implementation C of f.

We next define the properties of a SNARG scheme. We consider two soundness guarantees: non-adaptive soundness and semi-adaptive soundness (somewhere extractability).

Definition 4.1 (SNARG for Monotone Policy $\mathsf{BatchNP}$)**.** *A* SNARG *scheme* $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ *for* $\mathcal{L}_f^{(k)}$ *is required to satisfy the following properties.*

Completeness. *For any* $\lambda \in \mathbb{N}$*, any* $k = k(\lambda)$*,* $n = n(\lambda)$*, and* $x = (x_1, \ldots, x_k) \in \mathcal{L}_f^{(k)}$*, such that* $|x| \leq 2^\lambda$ *and* $|x_i| = n$ *for every* $i \in [k]$*, and any corresponding witness* $w = (w_1, \ldots, w_k) \in \{0,1\}^{k \cdot m}$*,*

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

Succinctness. *In the completeness experiment above, the size of* $\mathsf{crs}_\mathcal{V}$ *and* π *are at most* $\mathsf{poly}(\lambda, \log k, m)$*.*

Non-Adaptive Soundness. *For any polynomials* $k = k(\lambda)$ *and* $n = n(\lambda)$ *and any polynomial-size* $\mathcal{P}^*$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such for any instance* $x = (x_1, \ldots, x_k) \notin \mathcal{L}_f^{(k)}$ *such that* $|x_i| = n$ *for every* $i \in [k]$*, it holds that for every* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\mathcal{V}(\mathsf{crs}_\mathcal{V}, C, x, \pi) = 1 \quad : \quad \begin{array}{l} (\mathsf{crs}_\mathcal{P}, \mathsf{crs}_\mathcal{V}) \leftarrow \mathsf{Gen}(1^\lambda, k, n), \\ \pi \leftarrow \mathcal{P}^*(\mathsf{crs}_\mathcal{P}, \mathsf{crs}_\mathcal{V}) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

In addition to the standard soundness definition, we introduce a variant of "somewhere extractability" for SNARGs for monotone policy $\mathsf{BatchNP}$. Roughly speaking, our definition says that for any "necessary subset" $J \subset [k]$ for C (meaning that if $C(b_1, \ldots, b_k) = 1$ then $b_j = 1$ for some $j \in J$), the common reference string can be programmed to be "extractable on J," meaning that it is possible to extract (from an efficiently generated proof π on an adaptively chosen statement) a witness w_j for some $j \in J$.

Definition 4.2 (Somewhere Extractable SNARGs for Monotone Policy $\mathsf{BatchNP}$)**.** *A* SNARG *for* $\mathcal{L}_f^{(k)}$ *is* somewhere extractable *if it additionally supports the following syntax:*

$\mathsf{Gen}(1^\lambda, k, n, J) \to (\mathsf{crs}_\mathcal{P}, \mathsf{crs}_\mathcal{V}, \mathsf{td})$*. This is a probabilistic algorithm that takes as input the security parameter* 1^λ*, batch size* k*, input length* n*, and the description of a set* $J \subset [k]$*. It outputs a common reference string* $(\mathsf{crs}_\mathcal{P}, \mathsf{crs}_\mathcal{V})$ *along with a trapdoor* td*.*

$\mathsf{Extract}(\mathsf{td}, \pi) \to (i, w_i)$*. This is a polynomial-time algorithm that takes as input a trapdoor* td *and proof string* π*. It outputs an index* i *and witness* w_i*.*

We additionally require the following two properties (which together imply soundness):

Key Indistinguishability. *For any poly-size adversary* $\mathcal{A}$*, any polynomials* $k = k(\lambda)$ *and* $n = n(\lambda)$*, and any sets* $J_0, J_1 \subseteq [k]$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\mathcal{A}(\mathsf{crs}_\mathcal{P}, \mathsf{crs}_\mathcal{V}) = b \quad : \quad \begin{array}{l} b \leftarrow \{0,1\}, \\ (\mathsf{crs}_\mathcal{P}, \mathsf{crs}_\mathcal{V}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, n, J_b) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Somewhere argument of knowledge. *For any polynomials $k = k(\lambda)$ and $n = n(\lambda)$ and any polynomial-size $\mathcal{P}^*$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any set $J \subseteq [k]$ where the assignment $b_i = \mathbf{1}_{i \notin J}$ satisfies $f(b_1, \ldots, b_k) = 0$, and every $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{ll} \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, x_1, \ldots, x_k, \pi) = 1 \wedge & (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, n, J) \\ i \notin J \ \vee \ \mathcal{R}(x_i, w_i) = 0 \quad : & (x_1, \ldots, x_k, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \\ & (i, w_i) \leftarrow \mathsf{Extract}\,(\mathsf{td}, \pi) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

5 Predicate Extractable Hash Families

In order to construct a SNARG scheme for monotone policy BatchNP languages, we first introduce a new tool: a *predicate extractable hash family* (PEHash). PEHash extends the commonly used notion of a *somewhere extractable hash* (SEHash) in the following way:

- A PEHash is associated with a class of predicates $\mathcal{F}$. A SEHash can be viewed as an instance of PEHash where $\mathcal{F}$ consists of all *index* functions $f_i(x) = x_i$.
- A hash key hk is sampled to be *binding* on a specific predicate f; however, hk itself computationally hides f.
- When hk is binding on f, it should be possible to *extract* $f(x)$ from a hash of x.

Defining security of PEHash for general predicates is somewhat challenging. In the case of SEHash, the key security property (aside from index hiding) is that if the bit b is extracted from a hash value v, and the hash function is binding on index i, then it should be hard (or impossible) to locally open v to $1 - b$ on the ith input location.

In this work, we define and construct PEHash for the class of *bit-fixing* predicates $f_{y,J}$ where $f_{y,J}(x) = 1$ if and only if $y_i = x_i$ for all $i \in J$. However, we view PEHash as a more general object and expect extensions to be useful in the future. Therefore, in what follows we define the syntax and basic properties of PEHash for general predicates. Our security definitions, however, are tailored to bit-fixing predicates.

5.1 Syntax and Basic Properties

A predicate extractable hash family PEHash with respect to a family $\mathcal{F}$ consists of the following PPT algorithms:

$\mathsf{Gen}(1^\lambda, N, f) \to (\mathsf{hk}, \mathsf{vk}, \mathsf{td})$. This is a PPT setup algorithm that takes as input a security parameter 1^λ, a message length N, and a predicate $f : \{0,1\}^N \to \{0,1\}$ in $\mathcal{F}$. It outputs a hash key hk, a verification key vk, and a trapdoor td.

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

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

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

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

Most properties of a PEHash can be stated independently of the predicate class $\mathcal{F}$: we require that a PEHash hash key should hide the predicate, the extraction algorithm should output the predicate $f(x)$ on $\mathsf{Hash}(x)$, and that PEHash should be a secure hash family with local opening.

Definition 5.1 (PEHash Basic Properties)**.** *A predicate extractable hash family*

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

satisfies the following properties:

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

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

Succinctness. *In the completeness experiment above, the size of the verification key vk and the hash value v is $\mathsf{poly}(\lambda)$. The size of the hash key hk is at most $|f| \cdot \mathsf{poly}(\lambda, \log N)$.*

Computational binding. *For any poly-size adversary $\mathcal{A}$ it holds that for any polynomial $N = N(\lambda)$ and any predicate $f \in \mathcal{F}$ there exists a negligible function μ such that for any $\lambda \in \mathbb{N}$,*

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

Predicate hiding. *For any poly-size adversary $\mathcal{A}$, any polynomial $N = N(\lambda)$, and any two predicates $f_0, f_1 \in \mathcal{F}$ such that $|f_0| = |f_1|$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l|ll} \mathcal{A}(\mathsf{hk}, \mathsf{vk}) = b & : & b \leftarrow \{0,1\} \\ & & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, f_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

Extraction correctness. *For any $\lambda \in \mathbb{N}$, any $N \leq 2^\lambda$, any predicate $f \in \mathcal{F}$, and any $x \in \{0,1\}^N$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l|ll} f(x) \neq \mathsf{Extract}(\mathsf{td}, \mathsf{v}) & : & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, f) \\ & & \mathsf{v} = \mathsf{Hash}(\mathsf{hk}, x) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

5.2 Extractable Hash for Bit-Fixing Predicates

Next, we describe what it means for a PEHash to be secure for *bit-fixing predicates*. Recall that for $x \in \{0,1\}^n$, we define $f_{y,J}(x) = 1$ if and only if $y_i = x_i$ for all $i \in J$. We want a security notion that restricts the behavior of an adversarial sender, which given $(\mathsf{hk}, \mathsf{vk})$ produces a hash value v. It turns out that security is *asymmetric* depending on whether $\mathsf{Extract}(\mathsf{td}, \mathsf{v})$ is equal to 0 or 1:

- If $\mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 1$, we want it to be hard for the adversary to open the ith bit of x to $1 - y_i$ for any $i \in J$.
- If $\mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 0$, we want to say that there is *some* index $j \in J$ such that the adversary cannot open the jth bit of x to y_j. To formalize this, we introduce an auxiliary algorithm $\mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v}) \to j$ that "points" to which index the adversary is constrained on.

We give a formal definition below.

Definition 5.2 (Bit-fixing PEHash)**.** *A predicate extractable hash family* PEHash *with respect to the bit-fixing predicate family is a* PEHash *satisfying the basic properties above (Definition 5.1), augmented with the following algorithm:*

$\mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v}) \to j$. *This is a deterministic extraction algorithm that takes as input a trapdoor* td *and a hash value* $\mathsf{v} \in \{0,1\}^{\mathsf{poly}(\lambda)}$*, and outputs an index* $j \in [N]$*.*

The hash family is furthermore required to satisfy the following consistency *properties:*

Index Extraction Correctness. *For any* $\lambda \in \mathbb{N}$*, any* $N \leq 2^\lambda$*, any predicate* $f = (J, y) \in \mathcal{F}$ *such that* $J \neq \emptyset$*, and any hash value* v*,*

$$\Pr\left[\ \mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v}) \in J \quad : \quad (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y)\ \right] = 1\ .$$

Consistency of extraction. *For any poly-size adversary* $\mathcal{A}$ *it holds that for any polynomial* $N = N(\lambda)$ *and any bit-fixing predicate described by a set* $J \subset [N]$ *and string* $y \in \{0,1\}^J$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\begin{array}{l} j \in J \ \wedge\ \mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 1 \ \wedge \\ \mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, 1 - y_j, t, \rho) = 1 \end{array} \ : \ \begin{array}{l} (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y), \\ (\mathsf{v}, j, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

and

$$\Pr\left[\begin{array}{l} \mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 0 \ \wedge \\ \mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v}) = j \ \wedge \\ \mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, y_j, t, \rho) = 1 \end{array} \ : \ \begin{array}{l} (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y), \\ (\mathsf{v}, j, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Theorem 5.3. *Assuming the hardness of* LWE*, there exists a* PEHash *family for bit-fixing predicates.*

We defer the proof of Theorem 5.3 to Section 5.3.1, where we construct a stronger object used in our somewhere extractable SNARG construction.

5.3 Extractable Hash with Tags for Bit-Fixing Predicates

In this section we define an extension of PEHash for bit-fixing predicates, which we call a PEHash family with tags. This object supports hashing the input together with a vector of tags, such that each bit of the input has an attached tag.

We use this extended definition in Section 7, to achieve a somewhere extractable SNARG.

We modify the syntax of the basic PEHash algorithms to support tags, as follows:

- Gen, Extract and ExtractIndex are identical.
- Hash and Open receive a vector of tags $\vec{t} = (t_1, \ldots, t_N)$ as an additional input.
- Verify receives a tag t as an additional input, which should correspond to the tag attached to the jth bit of x.

We additionally introduce an auxiliary algorithm ExtractTag that is defined similarly to ExtractIndex except that it extracts the tag associated with input that the adversary is constrained on instead of its index. We give the full syntax below.

Syntax. A predicate extractable hash family PEHash with tags with respect to the bit-fixing predicate family consists of the following PPT algorithms:

$\mathsf{Gen}(1^\lambda, N, J, y) \to (\mathsf{hk}, \mathsf{vk}, \mathsf{td})$. This is a probabilistic setup algorithm that takes as input a security parameter 1^λ in unary, a message length N, a set of indices $J \subseteq [N]$, and a string $y \in \{0,1\}^J$. It outputs a hash key hk, verification key vk and trapdoor td.

$\mathsf{Hash}(\mathsf{hk}, x, \vec{t}) \to \mathsf{v}$. This is a deterministic algorithm that takes as input a hash key hk, an input $x \in \{0,1\}^N$, and tags $\vec{t} = (t_1, \ldots, t_N) \in (\{0,1\}^T)^N$. It outputs a hash value $\mathsf{v} \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$.

$\mathsf{Open}(\mathsf{hk}, x, \vec{t}, j) \to \rho$. This is a deterministic algorithm that takes as input a hash key hk, an input $x \in \{0,1\}^N$, tags $\vec{t} = (t_1, \ldots, t_N) \in (\{0,1\}^T)^N$ and an index $j \in [N]$. It outputs an opening $\rho \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$.

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

$\mathsf{Extract}(\mathsf{td}, \mathsf{v}) \to u$. This is a deterministic extraction algorithm that takes as input a trapdoor td and a hash value $\mathsf{v} \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$, and outputs a bit $u \in \{0,1\}$.

$\mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v}) \to j$. This is a deterministic extraction algorithm that takes as input a trapdoor td and a hash value $\mathsf{v} \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$, and outputs an index $j \in [N]$.

$\mathsf{ExtractTag}(\mathsf{td}, \mathsf{v}) \to t$. This is a deterministic extraction algorithm that takes as input a trapdoor td and a hash value $\mathsf{v} \in \{0,1\}^{T \cdot \mathsf{poly}(\lambda)}$, and outputs a tag $t \in \{0,1\}^T$.

The basic properties of PEHash with tags are as in Definition 5.1, with minor adjustments to account for the modified syntax. For security, we extend the consistency of extraction property to hold also for tags.

Definition 5.4 (Bit-fixing PEHash with Tags)**.** *A predicate extractable hash family* PEHash *with tags with respect to the bit-fixing predicate family is a* PEHash *satisfying the basic properties defined in Definition 5.1 modified to account for tags, and the following consistency properties:*

Index Extraction Correctness. *For any $\lambda \in \mathbb{N}$, any $N \leq 2^\lambda$, any predicate $f = (J, y) \in \mathcal{F}$ such that $J \neq \emptyset$, and any hash value* v*,*

$$\Pr\left[\ \mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v}) \in J \quad : \quad (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y)\ \right] = 1\ .$$

Consistency of extraction. *For any poly-size adversary $\mathcal{A}$ it holds that for any polynomial $N = N(\lambda)$ and any bit-fixing predicate described by a set $J \subseteq [N]$ and string $y \in \{0,1\}^J$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l} j \in J \ \wedge\ \mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 1 \ \wedge \\ \mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, \bar{y}_j, t, \rho) = 1 \end{array} \ : \ \begin{array}{l} (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y), \\ (\mathsf{v}, j, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

and

$$\Pr\left[\begin{array}{l} \mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 0 \ \wedge \\ \mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v}) = j \ \wedge \\ \mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, y_j, t, \rho) = 1 \end{array} \ : \ \begin{array}{l} (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y), \\ (\mathsf{v}, j, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Consistency of tag extraction. *For any poly-size adversary $\mathcal{A}$ it holds that for any polynomial $N = N(\lambda)$ and any bit-fixing predicate described by a set $J \subseteq [N]$ and string $y \in \{0,1\}^J$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any $\lambda \in \mathbb{N}$,*

$$\Pr\left[\begin{array}{l} \mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 0 \ \wedge \\ \mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v}) = j \ \wedge \\ \mathsf{ExtractTag}(\mathsf{td}, \mathsf{v}) \neq t \ \wedge \\ \mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, \bar{y}_j, t, \rho) = 1 \end{array} \ : \ \begin{array}{l} (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y), \\ (\mathsf{v}, j, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 5.1. We note that a bit-fixing PEHash (Definition 5.2) can be derived from any bit-fixing PEHash with tags by fixing all the tags to $\perp$. Therefore, it is sufficient to construct a bit-fixing PEHash with tags.

5.3.1 Construction.

In this section we construct bit-fixing PEHash with tags. We first give a "base construction" where the length of the verification key vk grows with the predicate description (same as the hash key hk), however, the hash values and openings are fully succinct. We analyze the security of the base construction in Section 5.3.2. Then, in Section 5.3.3, we use RAM SNARGs to generically shorten vk to the desired $\mathsf{poly}(\lambda)$ length.

Our base construction uses a FHE scheme (Definition 3.3)

$$\mathsf{FHE} = (\mathsf{FHE.Setup}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec}).$$

$\mathsf{Gen}(1^\lambda, N, f = (J, y))$ is defined as follows:

1. Assume wlog that N is a power of 2, and let $N = 2^n$.
2. Let $(\mathsf{pk}_{\mathsf{leaf}}, \mathsf{sk}_{\mathsf{leaf}}) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$, and for every $0 \le j \le n$ let $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$.
3. Let $c_0 \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_0}(\mathsf{sk}_{\mathsf{leaf}})$, and for every $1 \le j \le n$ let $c_j \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_j}(\mathsf{sk}_{j-1})$.
4. Let $c_f = c_{f,0} \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_0}(J, y)$.
5. Output $\mathsf{hk} = (\mathsf{pk}_{\mathsf{leaf}}, \mathsf{pk}_0, \ldots, \mathsf{pk}_n, c_0, \ldots, c_n, c_f)$, $\mathsf{vk} = \mathsf{hk}$, $\mathsf{td} = (J, \mathsf{sk}_n)$.

$\mathsf{Hash}(\mathsf{hk}, x, \vec{t})$ is as follows:

1. Parse $\mathsf{hk} = (\mathsf{pk}_{\mathsf{leaf}}, \mathsf{pk}_0, \ldots, \mathsf{pk}_n, c_0, \ldots, c_n, c_f)$.
2. We consider a binary tree of height n whose vertices are represented as $v \in \{0,1\}^j$ for all $0 \le j \le n$ (where the root is represented by the empty string ε). Compute a ciphertext for each of the vertices, as follows:
   (a) For every leaf $v \in \{0,1\}^n = [N]$, we compute $\mathsf{leaf}_v = \mathsf{FHE.Enc}_{\mathsf{pk}_{\mathsf{leaf}}}(x_v, t_v)$ using fixed all-zero randomness.
   (b) For $j = 0$, we proceed as follows:
      i. Let $c_{f,0} = c_f$.
      ii. For every vertex $v \in \{0,1\}^n = [N]$, we compute $\mathsf{ct}_v = \mathsf{FHE.Eval}_{\mathsf{pk}_0}(g_{v,\mathsf{leaf}_v}, (c_0, c_{f,0}))$, where $g_{v,\mathsf{leaf}_v}(\mathsf{sk}_{\mathsf{leaf}}, J, y)$ is as follows:
      - Compute $(b, t) = \mathsf{FHE.Dec}_{\mathsf{sk}_{\mathsf{leaf}}}(\mathsf{leaf}_v)$.
      - If $v \in J$ and $b \ne y_v$, output $(0, v, t)$.
      - Otherwise, output $(1, \perp, \perp)$.
   (c) For every $1 \le j \le n$, we proceed as follows:
      i. Compute $c_{f,j} = \mathsf{FHE.Eval}_{\mathsf{pk}_j}(g_{c_{f,j-1}}, c_j)$, where $g_{c_{f,j-1}}(\mathsf{sk}_{j-1})$ outputs $(J, y) = \mathsf{FHE.Dec}_{\mathsf{sk}_{j-1}}(c_{f,j-1})$.
      ii. For every vertex $v \in \{0,1\}^{n-j}$, we compute $\mathsf{ct}_v = \mathsf{FHE.Eval}_{\mathsf{pk}_j}(g_{\mathsf{ct}_{v||0},\mathsf{ct}_{v||1}}, (c_j, c_{f,j}))$, where $g_{\mathsf{ct}_{v||0},\mathsf{ct}_{v||1}}(\mathsf{sk}_{j-1}, J, y)$ is as follows:
      - For every $i \in \{0,1\}$, compute $(b_i, j_i, t_i) = \mathsf{FHE.Dec}_{\mathsf{sk}_{j-1}}(\mathsf{ct}_{v||i})$.
      - If there exists an $i \in \{0,1\}$ such that $b_i = 0, j_i \in J$ and $v||i$ is a prefix of j_i (in other words, j_i is a descendant of $v||i$), take the minimum such i and output $(0, j_i, t_i)$.
      - Otherwise, output $(1, \perp, \perp)$.
3. Output $\mathsf{v} = \mathsf{ct}_\varepsilon$ associated with the root of the tree.

$\mathsf{Open}(\mathsf{hk}, x, \vec{t}, j)$ is as follows:

1. Write $j \in [N]$ as a string of bits $j = (j_1, \ldots, j_n) \in \{0,1\}^n$.
2. For every $0 \le i \le n$, let $v_i = (j_1, \ldots, j_{n-i-1}, j_{n-i})$, and $v'_i = (j_1, \ldots, j_{n-i-1}, 1 - j_{n-i})$. In other words, v_i is the ith node in the path from j to the root ε, and v'_i is v_i's sibling.
3. Compute the tree as in $\mathsf{Hash}(\mathsf{hk}, x, \vec{t})$, and all leaf and ct ciphertexts.
4. Output $\rho = \left(\mathsf{leaf}_j, \left\{\mathsf{ct}_{v_i}, \mathsf{ct}_{v'_i}\right\}_{0 \le i \le n}\right)$.

$\mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, b, t, \rho)$ is as follows:

1. Parse $\mathsf{vk} = (\mathsf{pk}_{\mathsf{leaf}}, \mathsf{pk}_0, \ldots, \mathsf{pk}_n, c_0, \ldots, c_n, c_f)$.
2. Parse $\rho = \left(\mathsf{leaf}^*_j, \left\{\mathsf{ct}^*_{v_i}, \mathsf{ct}^*_{v'_i}\right\}_{0 \le i \le n}\right)$.
3. Accept if all the following checks pass:
   (a) Check that $\mathsf{leaf}^*_j = \mathsf{FHE.Enc}_{\mathsf{pk}_{\mathsf{leaf}}}(b, t)$ with fixed all-zero randomness.
   (b) For $i = 0$, proceed as follows:
      i. Let $c_{f,0} = c_f$.
      ii. Recall that $v_0 = j$ and check that $\mathsf{ct}^*_j = \mathsf{FHE.Eval}_{\mathsf{pk}_0}(g_{j,\mathsf{leaf}^*_j}, (c_0, c_{f,0}))$, where g_{j,leaf^*_j} is defined as in Hash.
   (c) For every $1 \le i \le n$, proceed as follows:
      i. Let $c_{f,i} = \mathsf{FHE.Eval}_{\mathsf{pk}_i}(g_{c_{f,i-1}}, c_i)$, where $g_{c_{f,i-1}}$ is defined as in Hash.
      ii. Check that $\mathsf{ct}^*_{v_i} = \mathsf{FHE.Eval}_{\mathsf{pk}_i}(g_{\mathsf{ct}^*_{v_{i-1}}, \mathsf{ct}^*_{v'_{i-1}}}, (c_i, c_{f,i}))$, where $g_{\mathsf{ct}^*_{v_{i-1}}, \mathsf{ct}^*_{v'_{i-1}}}$ is defined as in Hash.
   (d) Recall that $v_n = \varepsilon$ and check that $\mathsf{v} = \mathsf{ct}^*_\varepsilon$.

$\mathsf{Extract}(\mathsf{td}, \mathsf{v})$ is as follows:

1. Parse $\mathsf{td} = J, \mathsf{sk}_n$.
2. Compute $(b, j, t) = \mathsf{FHE.Dec}_{\mathsf{sk}_n}(\mathsf{v})$.
3. Output b.

$\mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v})$ is as follows:

1. Parse $\mathsf{td} = J, \mathsf{sk}_n$.
2. Compute $(b, j, t) = \mathsf{FHE.Dec}_{\mathsf{sk}_n}(\mathsf{v})$.
3. Output j, or an arbitrary element of J if $j \notin J$.

$\mathsf{ExtractTag}(\mathsf{td}, \mathsf{v})$ is as follows:

1. Parse $\mathsf{td} = J, \mathsf{sk}_n$.
2. Compute $(b, j, t) = \mathsf{FHE.Dec}_{\mathsf{sk}_n}(\mathsf{v})$.
3. Output t.

Remark 5.2 (Shorter hash key for restricted families of predicates)**.** We can modify the above construction to achieve a shorter hash key hk, if we consider restricted families $\mathcal{F}$ of bit-fixing predicates $f = (J, y)$ that have a succinct description d_f (potentially shorter than $|J|$).

Namely, let $\mathcal{F}$ be a family of bit-fixing predicates, such that there exists a poly-time TM $\mathcal{M} = \mathcal{M}_{\mathcal{F}}$ that receives an input $z = z_{\mathcal{F}}$ and a description d_f of a predicate $f \in \mathcal{F}$, and outputs $f = (J, y)$. Then, we can construct a PEHash family for the family $\mathcal{F}$, with the following syntax and efficiency:

- The Gen algorithm receives d_f rather than f, and the rest of the algorithms receive z as an additional input.
- The size of the hash key hk generated by $\mathsf{Gen}(1^\lambda, N, d_f)$ is $|d_f| \cdot \mathsf{poly}(\lambda)$.

The construction is identical to the construction above, with the following changes:

- Gen computes $c_{d_f} \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_0}(d_f)$ rather than $c_f \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_0}(f)$, includes c_{d_f} in $\mathsf{hk} = \mathsf{vk}$, and sets $\mathsf{td} = d_f, \mathsf{sk}_n$.
- $\mathsf{Hash}, \mathsf{Open}, \mathsf{Verify}$ first compute $c_f \leftarrow \mathsf{FHE.Eval}_{\mathsf{pk}_0}(g_z, c_{d_f})$ where $g_z(d_f) = \mathcal{M}(z, d_f)$, then proceed as in the original algorithms.
- $\mathsf{Extract}, \mathsf{ExtractIndex}, \mathsf{ExtractTag}$ parse $\mathsf{td} = d_f, \mathsf{sk}_n$, compute $(J, y) = \mathcal{M}(z, d_f)$, then proceed as in the original algorithms.

This variant of PEHash with shorter hk is used in our SNARG construction in Section 7, to achieve a prover common reference string $\mathsf{crs}_{\mathcal{P}}$ of size $(k+m) \cdot \mathsf{poly}(\lambda)$ rather than $(k+m+\mathrm{width}(C)) \cdot \mathsf{poly}(\lambda)$.

5.3.2 Analysis.

In this section we analyze the above construction, and show that it satisfies Definition 5.4, except for the verification efficiency requirement.

Opening completeness. Follows immediately by construction and the correctness of the FHE scheme.

Predicate hiding. We define a series of hybrid algorithms Gen_i for $0 \le i \le n+1$, as follows:

- For $i = n+1$, we define $\mathsf{Gen}_{n+1} = \mathsf{Gen}$.
- For $1 \le i \le n$, we define Gen_i that is identical to Gen_{i+1}, except that we compute $c_i \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_i}(0)$ rather than $c_i \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_i}(\mathsf{sk}_{i-1})$.
- For $i = 0$, we define Gen_0 that is identical to Gen_1, except that we compute $c_f \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_0}(0)$ rather than $c_f \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_0}(J, y)$.

We observe that $\mathsf{Gen}_{n+1}(1^\lambda, N, J, y) = \mathsf{Gen}(1^\lambda, N, J, y)$, and $\mathsf{Gen}_0(1^\lambda, N, J, y)$ is independent of the predicate J, y. Therefore, by a hybrid argument it suffices to show that $\mathsf{Gen}_i \approx_c \mathsf{Gen}_{i+1}$ for every $0 \le i \le n$. This follows from the semantic security of the underlying FHE scheme (with encryption key pk_i), since the outputs of both Gen_i and Gen_{i+1} can be sampled efficiently given pk_i without sk_i.

Index extraction correctness. Follows immediately by definition of $\mathsf{ExtractIndex}$.

Consistency of extraction. Fix any poly-size adversary $\mathcal{A}$, and polynomial $N = N(\lambda)$, and any bit-fixing predicate described by a set $J \subseteq [N]$ and a string $y \in \{0,1\}^J$.

The proof of the consistency of extraction property is given by the following two claims. Note that the original properties require a negligible error probability, but in this construction it is 0.

Claim 5.5.

$$\Pr\left[\begin{array}{l} j \in J \ \wedge \ \mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 1 \ \wedge \\ \mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, 1-y_j, t, \rho) = 1 \end{array} \ : \ \begin{array}{l} (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y), \\ (\mathsf{v}, j, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{vk}) \end{array}\right] = 0.$$

Proof of Claim 5.5. In the above experiment, we parse:

- $\mathsf{hk} = \mathsf{vk} = (\mathsf{pk}_{\mathsf{leaf}}, \mathsf{pk}_0, \ldots, \mathsf{pk}_n, c_0, \ldots, c_n, c_f)$.
- $\rho = \left(\mathsf{leaf}^*_j, \left\{\mathsf{ct}^*_{v_i}, \mathsf{ct}^*_{v'_i}\right\}_{0 \le i \le n}\right)$.

Assume $j \in J$ and $\mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, 1-y_j, t, \rho) = 1$. We show by induction that for every $0 \le i \le n$ we have $\mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{ct}^*_{v_i}) = (0, j', \cdot)$ for some $j' \in J$ that is a descendant of v_i. This suffices, since we also check that $\mathsf{ct}^*_{v_n} = \mathsf{v}$, so we must have $\mathsf{FHE.Dec}_{\mathsf{sk}_n}(\mathsf{v}) = (0, j', \cdot)$ which implies that $\mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 0$.

For the base case of $i = 0$ and $v_0 = j$, we have that $c_0, c_{f,0}$ are honest encryptions of $\mathsf{sk}_{\mathsf{leaf}}, J, y$, so using the evaluation correctness of the FHE scheme, we have

$$\mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{ct}^*_j) = \mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{FHE.Eval}_{\mathsf{pk}_0}(g_{j,\mathsf{leaf}^*_j}, (c_0, c_{f,0}))) = g_{j,\mathsf{leaf}^*_j}(\mathsf{sk}_{\mathsf{leaf}}, J, y).$$

Now, by definition of g_{j,leaf^*_j}, and since leaf^*_j is $\mathsf{FHE.Enc}_{\mathsf{pk}_{\mathsf{leaf}}}(1-y_j, t)$ and we have $j \in J$ and $1 - y_j \neq y_j$, we get $\mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{ct}^*_j) = (0, j, t)$.

Now, assume the claim holds for $i-1$. Similarly, since $c_i, c_{f,i}$ are honest encryptions of sk_{i-1}, J, y, evaluation correctness of the FHE scheme implies that

$$\mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{ct}^*_{v_i}) = \mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{FHE.Eval}_{\mathsf{pk}_i}(g_{\mathsf{ct}^*_{v_{i-1}}, \mathsf{ct}^*_{v'_{i-1}}}, (c_i, c_{f,i}))) = g_{\mathsf{ct}^*_{v_{i-1}}, \mathsf{ct}^*_{v'_{i-1}}}(\mathsf{sk}_{i-1}, J, y).$$

By the induction hypothesis we have $\mathsf{FHE.Dec}_{\mathsf{sk}_{i-1}}(\mathsf{ct}^*_{v_{i-1}}) = (0, j', \cdot)$ for some $j' \in J$ that is a descendant of v_{i-1}, so by definition of $g_{\mathsf{ct}^*_{v_{i-1}}, \mathsf{ct}^*_{v'_{i-1}}}$ we get that $\mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{ct}^*_{v_i}) = (0, j'', \cdot)$ for some $j'' \in J$ that is a descendant of v_i (either j' or the index j'' in $\mathsf{FHE.Dec}_{\mathsf{sk}_{i-1}}(\mathsf{ct}^*_{v'_{i-1}})$ if it is valid), which shows that the claim holds for i, and finishes the proof of Claim 5.5.

Claim 5.6.

$$\Pr\left[\begin{array}{l|ll} \mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 0 \wedge & & (\mathsf{hk}, \mathsf{vk}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y), \\ \mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v}) = j \wedge & : & (\mathsf{v}, j, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk}, \mathsf{vk}) \\ \mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, y_j, t, \rho) = 1 & & \end{array}\right] = 0.$$

Proof of Claim 5.6. We proceed similarly to the proof of Claim 5.5. Assume $\mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, y_j, t, \rho) = 1$. We show by induction that for every $0 \le i \le n$ we have either $\mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{ct}^*_{v_i}) = (1, \bot, \bot)$ or $\mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{ct}^*_{v_i}) = (0, j', \cdot)$ for some $j' \neq j$. This suffices, since we also check that $\mathsf{ct}^*_{v_n} = \mathsf{v}$, so we must have $\mathsf{Extract}(\mathsf{td}, \mathsf{v}) = 1$ or $\mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v}) = j' \neq j$.

For the base case of $i = 0$ and $v_0 = j$, as in the previous claim, we have $\mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{ct}^*_j) = g_{j,\mathsf{leaf}^*_j}(\mathsf{sk}_{\mathsf{leaf}}, J, y)$. By definition of g_{j,leaf^*_j}, and since leaf^*_j is $\mathsf{FHE.Enc}_{\mathsf{pk}_{\mathsf{leaf}}}(y_j, t)$, we get $\mathsf{FHE.Dec}_{\mathsf{sk}_0}(\mathsf{ct}^*_j) = (1, \bot, \bot)$.

Now, assume the claim holds for $i-1$. Similarly, we have $\mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{ct}^*_{v_i}) = g_{\mathsf{ct}^*_{v_{i-1}}, \mathsf{ct}^*_{v'_{i-1}}}(\mathsf{sk}_{i-1}, J, y)$. By the induction hypothesis we know that $\mathsf{FHE.Dec}_{\mathsf{sk}_{i-1}}(\mathsf{ct}^*_{v_{i-1}}) = (1, \bot, \bot)$ or $(0, j', \cdot)$ for some $j' \neq j$, and since j is not a descendant of v'_{i-1} we get that $\mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{ct}^*_{v_i})$ preserves the same property, which finishes the proof of Claim 5.6.

Consistency of tag extraction. Fix any poly-size adversary $\mathcal{A}$, and polynomial $N = N(\lambda)$, and any bit-fixing predicate described by a set $J \subseteq [N]$ and a string $y \in \{0,1\}^J$. We claim that

$$\Pr\left[\begin{array}{ll} \mathsf{Extract}(\mathsf{td},\mathsf{v}) = 0 \wedge & \\ \mathsf{ExtractIndex}(\mathsf{td},\mathsf{v}) = j \wedge & : \quad (\mathsf{hk},\mathsf{vk},\mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y), \\ \mathsf{ExtractTag}(\mathsf{td},\mathsf{v}) \neq t \wedge & \quad\quad (\mathsf{v}, j, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk},\mathsf{vk}) \\ \mathsf{Verify}(\mathsf{vk},\mathsf{v}, j, 1-y_j, t, \rho) = 1 & \end{array}\right] = 0.$$

Indeed, assume $\mathsf{Verify}(\mathsf{vk},\mathsf{v}, j, 1-y_j, t, \rho) = 1$. Similarly to Claim 5.6, it follows by induction that for every $0 \leq i \leq n$ it holds that $\mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{ct}^*_{v_i}) = (0, j, t)$ or $\mathsf{FHE.Dec}_{\mathsf{sk}_i}(\mathsf{ct}^*_{v_i}) = (0, j', \cdot)$ for some $j' \neq j$, which in the case of $i = n$ implies $\mathsf{ExtractTag}(\mathsf{td},\mathsf{v}) = t$ or $\mathsf{ExtractIndex}(\mathsf{td},\mathsf{v}) = j' \neq j$.

Computational binding. This property follows from the predicate hiding and consistency of extraction properties. Indeed, fix any poly-size adversary $\mathcal{A}$, polynomial $N = N(\lambda)$, and bit-fixing predicate described by a set $J \subseteq [N]$ and a string $y \in \{0,1\}^J$. We define

$$\epsilon(\lambda) \triangleq \Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{vk},\mathsf{v}, j, 0, t_0, \rho_0) = 1 \wedge & : \quad (\mathsf{hk},\mathsf{vk},\mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y), \\ \mathsf{Verify}(\mathsf{vk},\mathsf{v}, j, 1, t_1, \rho_1) = 1 & \quad\quad (\mathsf{v}, j, t_0, t_1, \rho_0, \rho_1) \leftarrow \mathcal{A}(\mathsf{hk},\mathsf{vk}) \end{array}\right],$$

and show that ϵ is negligible. First, there exists an index $j^* \in [N]$ such that

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{vk},\mathsf{v}, j, 0, t_0, \rho_0) = 1 \wedge & \\ \mathsf{Verify}(\mathsf{vk},\mathsf{v}, j, 1, t_1, \rho_1) = 1 \wedge & : \quad (\mathsf{hk},\mathsf{vk},\mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J, y), \\ j = j^* & \quad\quad (\mathsf{v}, j, t_0, t_1, \rho_0, \rho_1) \leftarrow \mathcal{A}(\mathsf{hk},\mathsf{vk}) \end{array}\right] \geq \frac{\epsilon(\lambda)}{N(\lambda)}.$$

We now switch to an experiment where we program the PEHash on the predicate described by the set $J' = \{j^*\}$ and the string $y' = 0$. By the predicate hiding property, there exists a negligible function μ such that

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{vk},\mathsf{v}, j, 0, t_0, \rho_0) = 1 \wedge & \\ \mathsf{Verify}(\mathsf{vk},\mathsf{v}, j, 1, t_1, \rho_1) = 1 \wedge & : \quad (\mathsf{hk},\mathsf{vk},\mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J', y'), \\ j = j^* & \quad\quad (\mathsf{v}, j, t_0, t_1, \rho_0, \rho_1) \leftarrow \mathcal{A}(\mathsf{hk},\mathsf{vk}) \end{array}\right] \geq \frac{\epsilon(\lambda)}{N(\lambda)} - \mu(\lambda).$$

It suffices to show that this probability is negligible to conclude that ϵ is negligible.

This follows from the consistency of extraction property together with the definition of J', y', since it implies that for every $b \in \{0,1\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{hk},\mathsf{v}, j^*, b, t, \rho) = 1 \wedge & : \quad (\mathsf{hk},\mathsf{vk},\mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, N, J', y'), \\ \mathsf{Extract}(\mathsf{td},\mathsf{v}) = b & \quad\quad (\mathsf{v}, t, \rho) \leftarrow \mathcal{A}(\mathsf{hk},\mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

5.3.3 Construction with efficient verification.

In this section, we describe how to obtain a bit-fixing PEHash with tags that satisfies our efficiency requirement of $|\mathsf{vk}| = \mathsf{poly}(\lambda)$, given the base construction in Section 5.3.1 with $\mathsf{vk} = \mathsf{hk}$.

Our construction uses the following building blocks:

- A bit-fixing PEHash family with tags

$$\mathsf{PEHash}' = (\mathsf{Gen}', \mathsf{Hash}', \mathsf{Open}', \mathsf{Verify}', \mathsf{Extract}', \mathsf{ExtractIndex}', \mathsf{ExtractTag}')$$

  such that Gen' outputs $(\mathsf{hk},\mathsf{vk},\mathsf{td})$ with $\mathsf{hk} = \mathsf{vk}$.

- A RAM SNARG (Definition 3.7) for the machine $\mathcal{R}$ defined in the Open algorithm below

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

We describe the PEHash algorithms.

$\mathsf{Gen}(1^\lambda, N, f)$ is as follows:

1. Let $(\mathsf{hk}', \mathsf{vk}' = \mathsf{hk}', \mathsf{td}') \leftarrow \mathsf{Gen}'(1^\lambda, N, f)$.
2. Let $\mathsf{crs}_{\mathsf{RAM}} \leftarrow \mathsf{Gen}_{\mathsf{RAM}}(1^\lambda, T)$, where T is the runtime of Verify'.
3. Let $\mathsf{d} = \mathsf{Digest}_{\mathsf{RAM}}(\mathsf{hk}')$.
4. Output $\mathsf{hk} = (\mathsf{crs}_{\mathsf{RAM}}, \mathsf{hk}')$, $\mathsf{vk} = (\mathsf{crs}_{\mathsf{RAM}}, \mathsf{d})$, $\mathsf{td} = \mathsf{td}'$.

$\mathsf{Hash}(\mathsf{hk}, x, \vec{t})$ is as follows:

1. Parse $\mathsf{hk} = (\mathsf{crs}_{\mathsf{RAM}}, \mathsf{hk}')$.
2. Output $\mathsf{v} = \mathsf{Hash}'(\mathsf{hk}', x, \vec{t})$.

$\mathsf{Open}(\mathsf{hk}, x, \vec{t}, j)$ is as follows:

1. Let $\mathcal{R}(x_{\mathsf{imp}}, x_{\mathsf{exp}})$ be a RAM machine defined as follows:
   (a) Parse $x_{\mathsf{imp}} = \mathsf{hk}'$, and $x_{\mathsf{exp}} = (\mathsf{v}, j, b, t, \rho')$.
   (b) Output $\mathsf{Verify}'(\mathsf{hk}', \mathsf{v}, j, b, t, \rho')$.
2. Parse $\mathsf{hk} = (\mathsf{crs}_{\mathsf{RAM}}, \mathsf{hk}')$.
3. Let $\mathsf{v} = \mathsf{Hash}'(\mathsf{hk}', x, \vec{t})$, and $\rho' = \mathsf{Open}'(\mathsf{hk}', x, \vec{t}, j)$.
4. Let $x_{\mathsf{imp}} = \mathsf{hk}'$ and $x_{\mathsf{exp}} = (\mathsf{v}, j, x_j, t_j, \rho')$.
5. Compute $\pi_{\mathsf{RAM}} = \mathcal{P}_{\mathsf{RAM}}(\mathsf{crs}_{\mathsf{RAM}}, (x_{\mathsf{imp}}, x_{\mathsf{exp}}))$.
6. Output $\rho = (\rho', \pi_{\mathsf{RAM}})$.

$\mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, b, t, \rho)$ is as follows:

1. Parse $\rho = (\rho', \pi_{\mathsf{RAM}})$.
2. Let $x_{\mathsf{exp}} = (\mathsf{v}, j, b, t, \rho')$.
3. Output $\mathcal{V}_{\mathsf{RAM}}(\mathsf{crs}_{\mathsf{RAM}}, \mathsf{d}, x_{\mathsf{exp}}, 1, \pi)$.

$\mathsf{Extract}(\mathsf{td}, \mathsf{v})$ outputs $\mathsf{Extract}'(\mathsf{td}, \mathsf{v})$.

$\mathsf{ExtractIndex}(\mathsf{td}, \mathsf{v})$ outputs $\mathsf{ExtractIndex}'(\mathsf{td}, \mathsf{v})$.

$\mathsf{ExtractTag}(\mathsf{td}, \mathsf{v})$ outputs $\mathsf{ExtractTag}'(\mathsf{td}, \mathsf{v})$.

Efficiency. By construction and the efficiency of the underlying RAM SNARG,

- The size of hk is $|\mathsf{crs}_{\mathsf{RAM}}|+|\mathsf{hk}'|= \mathsf{poly}(\log T, \lambda) + |\mathsf{hk}'|$.
- The size of vk is $|\mathsf{crs}_{\mathsf{RAM}}|+|\mathsf{d}|= \mathsf{poly}(\log T, \lambda)$.
- The size of the hash value and openings is $m \cdot \mathsf{poly}(\log N, \lambda)$, where m is the tag length.

Analysis. The opening completeness and predicate hiding properties are immediate.

For the consistency properties, we observe that RAM SNARG soundness implies that if an adversary $\mathcal{A}(\mathsf{hk}, \mathsf{vk})$ produces $(\mathsf{v}, j, b, t, \rho)$ such that $\mathsf{Verify}(\mathsf{vk}, \mathsf{v}, j, b, t, \rho) = 1$, then if we parse $\mathsf{hk} = (\mathsf{crs}_{\mathsf{RAM}}, \mathsf{hk}')$ and $\rho = (\rho', \pi_{\mathsf{RAM}})$, we have $\mathsf{Verify}'(\mathsf{hk}', \mathsf{v}, j, b, t, \rho') = 1$ except with negligible probability. Therefore, the consistency properties follow from the consistency of PEHash'.

6 Non-Adaptive SNARG Construction

In what follows we construct a SNARG scheme $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ for the language $\mathcal{L}_f^{(k)}$ defined in Section 4 with respect to an underlying NP language $\mathcal{L}$ and a monotone circuit C computing function f.

Our construction uses the following building blocks:

- A PEHash family (Definition 5.2)

$$\mathsf{PEHash} = (\mathsf{Gen}_{\mathsf{PEH}}, \mathsf{Hash}_{\mathsf{PEH}}, \mathsf{Open}_{\mathsf{PEH}}, \mathsf{Verify}_{\mathsf{PEH}}, \mathsf{Extract}_{\mathsf{PEH}}, \mathsf{ExtractIndex}_{\mathsf{PEH}})$$

  with respect to the bit-fixing predicate family $\mathcal{F}$ where each $f \in \mathcal{F}$ is a Boolean function with domain $\{0,1\}^N$, where $N = s := |C|$. Each function $f \in \mathcal{F}$ is associated with a subset $J \subseteq [N]$ of size $\ell \leq \mathrm{width}(C)$ and a string $y \in \{0,1\}^\ell$.

- A seBARG scheme (Definition 3.5)

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

We are now ready to define our SNARG for $\mathcal{L}_f^{(k)}$.

$\mathsf{Gen}(1^\lambda, k, n)$ does the following:

1. Fix two arbitrary predicates $f_1, f_2 \in \mathcal{F}$. Recall that $|f_\beta| = \ell \cdot (\log s + 1)$ for every $\beta \in \{1,2\}$.
2. Generate $(\mathsf{hk}_{\mathsf{PEH}}^{(\beta)}, \mathsf{vk}_{\mathsf{PEH}}^{(\beta)}, \mathsf{td}_{\mathsf{PEH}}^{(\beta)}) \leftarrow \mathsf{Gen}_{\mathsf{PEH}}(1^\lambda, N, f_\beta)$ for each $\beta \in \{1,2\}$.
3. Generate $(\mathsf{crs}_{\mathsf{seBARG}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_{\mathsf{seBARG}}(1^\lambda, 1^{n'}, 1^{m'}, I)$, where

$$n' = O(1) + \log N + 2|\mathsf{vk}_{\mathsf{PEH}}| + 2|\mathsf{v}|,$$

$$m' = m + 3 + 6|\rho|,$$

   $|\mathsf{v}|$ is the length of the output of $\mathsf{Hash}_{\mathsf{PEH}}(\mathsf{hk}_{\mathsf{PEH}}, \cdot)$, $|\rho|$ is the length of the output of $\mathsf{Open}_{\mathsf{PEH}}(\mathsf{hk}_{\mathsf{PEH}}, \cdot)$, and where $I \subset [N]$ is a set of size three initialized to $\{1, 2, N\}$.
4. Let $\mathsf{crs}_{\mathcal{P}} = (\mathsf{hk}_{\mathsf{PEH}}^{(1)}, \mathsf{vk}_{\mathsf{PEH}}^{(1)}, \mathsf{hk}_{\mathsf{PEH}}^{(2)}, \mathsf{vk}_{\mathsf{PEH}}^{(2)}, \mathsf{crs}_{\mathsf{seBARG}})$ and let $\mathsf{crs}_{\mathcal{V}} = (\mathsf{vk}_{\mathsf{PEH}}^{(1)}, \mathsf{vk}_{\mathsf{PEH}}^{(2)}, \mathsf{crs}_{\mathsf{seBARG}})$.
5. Output $(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}})$.

$\mathcal{P}(\mathsf{crs}_{\mathcal{P}}, C, x_1, \ldots, x_k, w_1, \ldots, w_k)$ does the following:

1. For every $1 \leq j \leq k$, compute $b_j = \mathcal{R}_{\mathcal{L}}(x_j, w_j)$. Then, compute the values of all the wires in the circuit evaluation $C(b_1, \ldots, b_k)$. Denote these values by $b_1, \ldots, b_N$ (the first k bits are the input wire values).

2. Parse $\mathsf{crs}_{\mathcal{P}} = (\mathsf{hk}^{(1)}_{\mathsf{PEH}}, \mathsf{vk}^{(1)}_{\mathsf{PEH}}, \mathsf{hk}^{(2)}_{\mathsf{PEH}}, \mathsf{vk}^{(2)}_{\mathsf{PEH}}, \mathsf{crs}_{\mathsf{seBARG}})$.
3. Compute $\mathsf{v}^{(\beta)} = \mathsf{Hash}_{\mathsf{PEH}}(\mathsf{hk}^{(\beta)}_{\mathsf{PEH}}, (b_1, \ldots, b_N))$ for $\beta \in \{1, 2\}$.
4. Define an instance $X = (M, z, N, T)$ of $\mathsf{BatchIndexTMSAT}$. The input z is defined as $z = (C, x_1, \ldots, x_k, (\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)})_{\beta \in \{1,2\}})$. The batch size is set to N. The Turing machine $M(z, j, \omega_j)$ is defined to operate as follows:
   (a) Parse $z = (C, x_1, \ldots, x_k, (\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)})_{\beta \in \{1,2\}})$.
   (b) If $1 \le j \le k$:
      i. Parse $\omega_j = \left(w_j, b_j, \rho^{(1)}, \rho^{(2)}\right)$.
      ii. Check that $\mathsf{Verify}(\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)}, j, b_j, \rho^{(\beta)}) = 1$ for $\beta \in \{1, 2\}$.
      iii. Check that $\mathcal{R}_{\mathcal{L}}(x_j, w_j) = b_j$.
   (c) If $j > k$:
      i. Compute the jth gate of C, $g_j = (j, j_1, j_2, c \in \{\mathrm{AND}, \mathrm{OR}\})$.
      ii. Parse $\omega_j = \left(b_j, b_{j_1}, b_{j_2}, \left(\rho^{(\beta)}_j, \rho^{(\beta)}_{j_1}, \rho^{(\beta)}_{j_2}\right)_{\beta \in \{1,2\}}\right)$.
      iii. Check that $\mathsf{Verify}_{\mathsf{PEH}}(\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)}, j, b_j, \rho^{(\beta)}_j) = 1$ for $\beta \in \{1, 2\}$.
      iv. Check that $\mathsf{Verify}_{\mathsf{PEH}}(\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)}, j_\alpha, b_{j_\alpha}, \rho^{(\beta)}_{j_\alpha}) = 1$ for $\alpha, \beta \in \{1, 2\}$.
      v. Check that $b_j = c(b_{j_1}, b_{j_2})$. (That is, check that the gate is satisfied.)
      vi. If $j = N$ is the output wire then check that $b_j = 1$.

   The description length of M is a constant. Finally, the time bound $T = \mathsf{poly}(N, n, m, k)$ is set so that the pseudocode above terminates.
5. For every $j \in [N]$, construct a witness (j, ω_j) for X, using the $\mathsf{Open}_{\mathsf{PEH}}$ algorithm to produce openings for $(b_1, \ldots, b_N)$ as appropriate.
6. Compute $\pi_{\mathsf{seBARG}} = \mathcal{P}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, M, z, 1^T, \omega_1, \ldots, \omega_N)$.
7. Output $(\mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$.

$\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, x_1, \ldots, x_k, \pi)$ does the following:

1. Parse $\mathsf{crs}_{\mathcal{V}} = (\mathsf{vk}^{(1)}_{\mathsf{PEH}}, \mathsf{vk}^{(2)}_{\mathsf{PEH}}, \mathsf{crs}_{\mathsf{seBARG}})$.
2. Parse $\pi = (\mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$.
3. Define $X = (M, z, N, T)$ as above.
4. Output $\mathcal{V}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, X, \pi_{\mathsf{seBARG}})$.

6.1 Analysis

Theorem 6.1. *The construction given in Section 6 is a* SNARG *for* $\mathcal{L}^{(k)}_f$ *(Definition 4.1) with the following additional efficiency properties:*

- *The runtime of the verifier is* $\mathsf{poly}(|\mathsf{crs}_{\mathcal{V}}| + |\pi|) + (kn + |C|)\mathsf{poly}(\lambda, \log k)$.
- *The (prover) common reference string* $\mathsf{crs}_{\mathcal{P}}$ *has size* $\mathsf{poly}(\lambda, \log k)(m + \mathrm{width}(C))$.

Proof of Theorem 6.1.

Completeness. Follows directly from the correctness properties of the underlying PEHash and seBARG.

Efficiency. To bound $|\mathsf{crs}_{\mathcal{V}}|+|\pi|$, we make use of the following facts:

- By the efficiency of PEHash, $|\mathsf{vk}_{\mathsf{PEH}}^{(\beta)}|+|\mathsf{v}_{\mathsf{PEH}}^{(\beta)}|\leq \mathsf{poly}(\lambda, \log k)$ for $\beta \in \{1,2\}$.
- The length of the BatchIndexTMSAT instance X is $|X|\leq kn + |C|+\mathsf{poly}(\lambda, \log(knm))$. The length of a witness ω_i is at most $\mathsf{poly}(\lambda, \log k) + m$.
- Therefore, by the succinctness of seBARG, we conclude that $|\mathsf{crs}_{\mathsf{seBARG}}|+|\pi_{\mathsf{seBARG}}|\leq m \cdot \mathsf{poly}(\lambda, \log(knm))$.

To bound the running time of the verifier, we again invoke the efficiency of the seBARG to conclude the claimed $\mathsf{poly}(|\mathsf{crs}_{\mathcal{V}}|+|\pi|) + (kn + |C|)\mathsf{poly}(\lambda, \log k)$ bound.

Finally, the size of $|\mathsf{crs}_{\mathcal{P}}|$ is at most $m \cdot \mathsf{poly}(\lambda, \log(knm)) + 2 \cdot |\mathsf{hk}_{\mathsf{PEH}}|\leq m \cdot \mathsf{poly}(\lambda, \log(knm)) + \mathsf{poly}(\lambda, \log k) \cdot \mathrm{width}(C)$ by the efficiency of the PEHash family.

Remark 6.1 (Improved Verifier Efficiency)**.** Suppose that there is a polynomial-time Turing machine M' and input string aux such that:

- $M(\mathsf{aux}, \text{“circuit”})$ outputs C, and
- $M(\mathsf{aux}, \text{“input”})$ outputs $(x_1, \ldots, x_k)$.

Then, the efficiency of the verifier can be improved to $\mathsf{poly}(|\mathsf{crs}_{\mathcal{V}}|+|\pi|) + \mathsf{poly}(\lambda, \log k) \cdot |\mathsf{aux}|$. This is accomplished simply by having the Turing machine M defined in the construction take as input aux (as part of z) rather than $(C, x_1, \ldots, x_k)$ and generate $(C, x_1, \ldots, x_k)$ from aux at the beginning of its execution.

Soundness. Fix any poly-size cheating prover $\mathcal{P}^*$, any two polynomials $k = k(\lambda)$ and $n = n(\lambda)$, and any instance $\mathbf{x} = (x_1, \ldots, x_k) \notin \mathcal{L}_f^{(k)}$ such that $|x_j|= n$ for every $j \in [k]$. By definition, the success probability of $\mathcal{P}^*$ is

$$\epsilon(\lambda) \triangleq \Pr\left[\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \quad : \quad \begin{array}{l} (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}(1^\lambda, k, n), \\ \pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \end{array} \right]. \tag{1}$$

We want to show that $\epsilon(\lambda)$ is a negligible function. To do so, we first define the following information (which depends on $\mathbf{x}$) that may be hard-wired during steps of our (non-adaptive) soundness analysis:

- Define $(b_1^*, \ldots, b_k^*) \in \{0,1\}^k$ where for every $j \in [k]$, $b_j^* = 1$ if and only if $x_j \in \mathcal{L}$.
- Define $(b_1^*, \ldots, b_N^*)$ to be the values of all the wires in C on input $(b_1^*, \ldots, b_k^*)$.
- Let $d = \mathsf{depth}(C)$. For each $i \in [d]$ denote by $J_i \subset [N]$ the wires that correspond to the ith layer of C that have the value 0. Note that $|J_i|$ is at most the width of C, as we do not include the input layer in this definition.

We prove soundness by a hybrid argument based on alternative key generation algorithms $(\mathsf{Gen}_{i,j})_{0\le i,j\le d}$ defined below.

$\mathsf{Gen}_{i,j}(1^\lambda, k, n)$ does the following:

1. Define predicates $f_1 = f_{J_i,0^\ell}$ and $f_2 = f_{J_j,0^\ell}$. If $i = 0$, we define f_1 to be the all zeroes function (and similarly for j).
2. Generate $(\mathsf{hk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{td}^{(\beta)}_{\mathsf{PEH}}) \leftarrow \mathsf{Gen}_{\mathsf{PEH}}(1^\lambda, N, f_\beta)$ for each $\beta \in \{1,2\}$.
3. Proceed otherwise as in Gen.

By the predicate hiding property of the underlying PEH family (Definition 5.1), Equation (1) implies there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $i \in [d]$

$$\Pr\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \quad : \quad \begin{array}{l}(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}_{i,i-1}(1^\lambda, k, n),\\ \pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}})\end{array}\right] \ge \epsilon - \mathsf{negl}(\lambda). \tag{2}$$

Parse $\pi = (\mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$. For every $i \in [d]$ and every $\lambda \in \mathbb{N}$, denote by

$$\mu_i(\lambda) = \mu_i^{(1)}(\lambda) \triangleq \Pr\left[\begin{array}{ll}\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \wedge & \\ \mathsf{Extract}(\mathsf{td}^{(1)}_{\mathsf{PEH}}, \mathsf{v}^{(1)}) = 0 & : \begin{array}{l}(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}_{i,i-1}(1^\lambda, k, n),\\ \pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}})\end{array}\end{array}\right]$$

In other words, μ_i is the probability that the verifier accepts and the evaluation of the ith layer predicate $f_{J_i,0^\ell}$ is inconsistent with the correct evaluation of $C(b_1^*, \ldots, b_k^*)$.

Lemma 6.2. *μ_d is a negligible function.*

Before proving Lemma 6.2, we first argue that the lemma indeed implies soundness. To see this, we make use of the seBARG: by the definition of Gen_d (and Gen), it is extractable on index N. Thus, we consider the following experiment:

- Sample $(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}, \mathsf{td}^{(1)}_{\mathsf{PEH}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_{d,d-1}(1^\lambda, k, n)$.
- Compute $\pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}})$. Parse $\pi = (\mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$.
- Compute $(\omega_1^*, \omega_2^*, \omega_N^*) = \mathsf{Extract}_{\mathsf{seBARG}}(\mathsf{td}_{\mathsf{seBARG}}, \pi_{\mathsf{seBARG}})$.
- Parse $\omega_N^* = \left(b_N, b_{N_1}, b_{N_2}, \left(\rho_N^{(\beta)}, \rho_{N_1}^{(\beta)}, \rho_{N_2}^{(\beta)}\right)_{\beta\in\{1,2\}}\right)$.
- Output $(b_N, \rho_N^{(1)})$.

Let $z = \left(C, x_1, \ldots, x_k, (\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)})_{\beta\in\{1,2\}}\right)$ and let M denote the Turing machine described in the construction. By the somewhere argument of knowledge property of the underlying seBARG, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge \ M(z, N, \omega_N^*) = 0\right] = \mathsf{negl}(\lambda). \tag{3}$$

Equations (2) and (3) imply that there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge \ M(z, N, \omega_N^*) = 1\right] \geq \epsilon - \mathsf{negl}(\lambda),$$

Moreover, $M(z, N, \omega_N) = 1$ implies that $b_N = 1$ and $\rho_N^{(1)}$ is a valid opening of $\mathsf{v}^{(1)}$ to b_N on the final wire of the circuit assignment. Thus, by the consistency of extraction property of the PEHash family (and the fact that $b_N^* = 0$), we know that

$$\Pr\left[M(z, N, \omega_N^*) = 1 \ \wedge \ \mathsf{Extract}(\mathsf{td}_{\mathsf{PEH}}^{(1)}, \mathsf{v}^{(1)}) = 1\right] = \mathsf{negl}(\lambda).$$

Thus, if μ_d is negligible (Lemma 6.2) we conclude that ϵ is negligible, as desired.

Proof of Lemma 6.2. We prove that there exists a negligible function $\mathsf{negl}(\lambda)$ such that for every $i \in [d]$,

$$\mu_i^{(1)} \leq \mu_{i-1}^{(1)} + \mathsf{negl}(\lambda),$$

where μ_0 is defined to be zero. To prove this, we define the intermediate quantity

$$\mu_{i-1}^{(2)}(\lambda) \triangleq \Pr\left[\begin{array}{ll} \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge & : \ (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}_{i,i-1}(1^\lambda, k, n), \\ \mathsf{Extract}(\mathsf{td}_{\mathsf{PEH}}^{(2)}, \mathsf{v}^{(2)}) = 0 & \ \ \pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \end{array}\right]$$

for $i > 1$, and define $\mu_0^{(2)} = 0$. We will prove the following two claims (which together imply Lemma 6.2):

Claim 6.3. *For all $i \geq 1$, $\mu_i^{(1)} \leq \mu_{i-1}^{(2)} + \mathsf{negl}(\lambda)$.*

Claim 6.4. *For all $i > 1$, $\mu_{i-1}^{(2)} \leq \mu_{i-1}^{(1)} + \mathsf{negl}(\lambda)$.*

Proof of Claim 6.3. It suffices to show that

$$\delta_{i,i-1} \triangleq \Pr\left[\begin{array}{ll} \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge & \\ \mathsf{Extract}(\mathsf{td}_{\mathsf{PEH}}^{(1)}, \mathsf{v}^{(1)}) = 0 \ \wedge & : \ (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}_{i,i-1}(1^\lambda, k, n), \\ \mathsf{Extract}(\mathsf{td}_{\mathsf{PEH}}^{(2)}, \mathsf{v}^{(2)}) = 1 & \ \ \pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \end{array}\right] = \mathsf{negl}(\lambda).$$

In the above experiment (and the following experiments), let $\mathsf{BAD} = \mathsf{BAD}_{i,i-1}$ denote the event that $\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1$, $\mathsf{Extract}(\mathsf{td}_{\mathsf{PEH}}^{(1)}, \mathsf{v}^{(1)}) = 0$, and $\mathsf{Extract}(\mathsf{td}_{\mathsf{PEH}}^{(2)}, \mathsf{v}^{(2)}) = 1$ simultaneously.

We upper bound the probability of BAD in the above experiment making use of a hybrid setup algorithm $\widetilde{\mathsf{Gen}}_{i,i-1}(1^\lambda, k, n)$, which is identical to $\mathsf{Gen}_{i,i-1}(1^\lambda, k, n)$ except for how the seBARG setup procedure is executed:

- Sample a uniformly random index $j = j_0$ from the ith level of C.
- Let the jth gate of C be described as $(j_1, j_2, c_j \in \{\mathrm{AND}, \mathrm{OR}\})$.
- Sample $(\mathsf{crs}_{\mathsf{seBARG}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_{\mathsf{seBARG}}(1^\lambda, 1^{n'}, 1^{m'}, \{j, j_1, j_2\})$.

Let $\Pr[\ \cdot\ ], \widetilde{\Pr}[\ \cdot\ ]$ denote probabilities under $\mathsf{Gen}_{i,i-1}$ and $\widetilde{\mathsf{Gen}}_{i,i-1}$ setup, respectively. We now proceed to bound $\delta_{i,i-1}$.

Step 1: Switch to $\widetilde{\mathsf{Gen}}$. This is accomplished in two steps. First, we note that

$$\Pr\Big[\mathsf{BAD} \ \wedge \ j = \mathsf{ExtractIndex}(\mathsf{td}^{(1)}_{\mathsf{PEH},i}, \mathsf{v}^{(1)})\Big] = \delta_{i,i-1} \cdot \frac{1}{\mathrm{width}(C)} \geq \frac{\delta_{i,i-1}}{\mathsf{poly}(\lambda)},$$

since j is independent of the $\mathsf{Gen}_{i,i-1}$ experiment. Then, we also have that

$$\widetilde{\Pr}\Big[\mathsf{BAD} \ \wedge \ j = \mathsf{ExtractIndex}(\mathsf{td}^{(1)}_{\mathsf{PEH},i}, \mathsf{v}^{(1)})\Big] \geq \frac{\delta_{i,i-1}}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda)$$

by the index-hiding of seBARG.

Step 2: Analyze the output of $\mathsf{Extract}_{\mathsf{seBARG}}$. Consider the experiment of running $\mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}})$ on a common reference string generated using $\mathsf{Gen}'_{i,i-1}(1^\lambda, k, n)$. The prover outputs a string π, which we parse as $(\mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$. Then, running the extractor $\mathsf{Extract}_{\mathsf{seBARG}}(\mathsf{td}_{\mathsf{seBARG}}, \pi_{\mathsf{seBARG}})$ produces potential witnesses $\omega^*_j, \omega^*_{j_1}, \omega^*_{j_2}$.

Let $\mathsf{Correct}_M$ denote the event that $M(z, j_\alpha, \omega^*_{j_\alpha}) = 1$ for all $\alpha \in \{0,1,2\}$. By the extractability property of seBARG, we know that

$$\widetilde{\Pr}\Big[\mathsf{BAD} \ \wedge \ j = \mathsf{ExtractIndex}(\mathsf{td}^{(1)}_{\mathsf{PEH},i}, \mathsf{v}^{(1)}) \ \wedge \ \mathsf{Correct}_M\Big] \geq \frac{\delta_{i,i-1}}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda).$$

Parse the witness ω^*_j as

$$\omega^*_j = \left(b_j, b_{j_1}, b_{j_2}, \left(\rho^{(\beta)}_N, \rho^{(\beta)}_{j_1}, \rho^{(\beta)}_{j_2}\right)_{\beta \in \{1,2\}}\right).$$

The structure of $\omega^*_{j_1}, \omega^*_{j_2}$ depends on whether j_1 and j_2 correspond to input wires, but $\omega^*_{j_1}$ (respectively, $\omega^*_{j_2}$) will always be parsed to contain an opening of each $\mathsf{v}^{(\beta)}$ to the j_1th wire value (respectively, the j_2th wire value). The fact that each $\omega^*_{j_\alpha}$ is a valid witness and the computational binding of PEHash imply that the j_1, j_2 openings are to bits b_{j_1}, b_{j_2} (except with negligible probability). From now on, we will assume that the bits b_{j_1}, b_{j_2} are well-defined and contained in both ω^*_j and $\omega^*_{j_1}/\omega^*_{j_2}$.

Finally, since BAD implies $\mathsf{Extract}(\mathsf{td}^{(1)}_{\mathsf{PEH}}, \mathsf{v}^{(1)}) = 0$, consistency of $\mathsf{PEH}^{(1)}$ extraction implies that

$$\widetilde{\Pr}\Big[\mathsf{BAD} \ \wedge \ j = \mathsf{ExtractIndex}(\mathsf{td}^{(1)}_{\mathsf{PEH},i}, \mathsf{v}^{(1)}) \ \wedge \ \mathsf{Correct}_M \ \wedge \ b_j = 1\Big] \geq \frac{\delta_{i,i-1}}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda).$$

Step 3: Analyze the jth Gate. At this point in the hybrid argument, we are analyzing an event in which $b_j = 1$; however, we also know that $b^*_j = 0$, since $j = \mathsf{ExtractIndex}(\mathsf{td}^{(1)}_{\mathsf{PEH}}, \mathsf{v}^{(1)})$ is an extracted index with respect to $f_{J_i,0^\ell}$. We now claim that for a randomly sampled $\alpha \in \{1,2\}$,

$$\widetilde{\Pr}\Big[\mathsf{BAD} \ \wedge \ j = \mathsf{ExtractIndex}(\mathsf{td}^{(1)}_{\mathsf{PEH},i}, \mathsf{v}^{(1)}) \ \wedge \ \mathsf{Correct}_M \ \wedge \ b_{j_\alpha} > b^*_{j_\alpha}\Big] \geq \frac{\delta_{i,i-1}}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda).$$

Specifically, the probability compared to above is cut in half (at most). This holds because $M(z, j, \omega^*_j) = 1$ implies that $b_j = c_j(b_{j_1}, b_{j_2})$. Moreover, we know that $b_j = 1$, $b^*_j = 0$, and $b^*_j = c_j(b^*_{j_1}, b^*_{j_2})$. These conditions together imply that $b_{j_\alpha} > b^*_{j_\alpha}$ for some $\alpha \in \{1,2\}$.

Step 4: Analyze the "incorrect child" j_α. We split the Step 3 event into two cases.

- When $j_\alpha \le k$, the event happens with probability zero. This is because $\mathsf{Correct}_M$ implies that $M(z, j_\alpha, \omega^*_\alpha) = 1$, which implies that ω^*_α contains a string w_{j_α} such that $\mathcal{R}_{\mathcal{L}}(x_{j_\alpha}, w_{j_\alpha}) = b_{j_\alpha}$. But since $b_{j_\alpha} > b^*_{j_\alpha}$, this is impossible ($b^*_{j_\alpha} = 0$ means that $x_j \notin \mathcal{L}$).
- Thus, we can append "$j_\alpha > k$" to the event without loss of generality. If $i = 1$ we are done (we have reached an empty event). Otherwise, we conclude that $\delta_{i,i-1} = \mathsf{negl}(\lambda)$ by the extraction consistency of $\mathsf{PEH}^{(2)}$. This is because the (valid) witness ω^*_j contains an opening $\rho^{(2)}_{j_\alpha}$ of $\mathsf{v}^{(2)}$ to $b_{j_\alpha} > b^*_{j_\alpha}$ on location $j_\alpha \in J_{i-1}$. Moreover, BAD implies that $\mathsf{Extract}(\mathsf{td}^{(2)}, \mathsf{v}^{(2)}) = 1$, so extraction consistency gives the desired bound.

This completes the proof of Claim 6.3.

Proof of Claim 6.4 We observe that by PEH key indistinguishability,

$$\mu^{(2)}_{i-1} \le \Pr\left[\begin{array}{ll} \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \wedge & : \quad (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}_{i-1,i-1}(1^\lambda, k, n), \\ \mathsf{Extract}(\mathsf{td}^{(2)}_{\mathsf{PEH}}, \mathsf{v}^{(2)}) = 0 & \quad \pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \end{array}\right] + \mathsf{negl}(\lambda)$$

and

$$\Pr\left[\begin{array}{ll} \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \wedge & : \quad (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}_{i-1,i-1}(1^\lambda, k, n), \\ \mathsf{Extract}(\mathsf{td}^{(1)}_{\mathsf{PEH}}, \mathsf{v}^{(1)}) = 0 & \quad \pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \end{array}\right] \le \mu^{(1)}_{i-1} + \mathsf{negl}(\lambda).$$

Thus, it suffices to show that

$$\delta_{i-1} \triangleq \Pr\left[\begin{array}{ll} \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \wedge & \\ \mathsf{Extract}(\mathsf{td}^{(2)}_{\mathsf{PEH}}, \mathsf{v}^{(2)}) = 0 \wedge & : \quad (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}_{i-1,i-1}(1^\lambda, k, n), \\ \mathsf{Extract}(\mathsf{td}^{(1)}_{\mathsf{PEH}}, \mathsf{v}^{(1)}) = 1 & \quad \pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \end{array}\right] = \mathsf{negl}(\lambda).$$

To prove this, we make use of a further hybrid setup $\widetilde{\mathsf{Gen}}_{i-1,i-1}(1^\lambda, k, n)$ where the key generation for the seBARG is modified as follows:

- Sample a uniformly random index j from the $i-1$th level of C.
- Sample $(\mathsf{crs}_{\mathsf{seBARG}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_{\mathsf{seBARG}}(1^\lambda, 1^{n'}, 1^{m'}, \{1, 2, j\})$.

The proof that $\delta_{i-1} = \mathsf{negl}(\lambda)$ now proceeds analogously to the proof that $\delta_{i,i-1} = \mathsf{negl}(\lambda)$ above (with fewer steps, since we are only concerned with a single index j rather than a gate).
This completes the proofs of Claim 6.4, Lemma 6.2, and Theorem 6.1.

6.2 Argument of Knowledge

In this section, we show that our non-adaptive SNARG is an argument of knowledge.

Theorem 6.5. *The* SNARG *for* $\mathcal{L}^{(k)}_f$ *given in Section 6 satisfies the following argument of knowledge property: for every constant* $c \in \mathbb{N}$*, there exists a* PPT *oracle machine* $\mathcal{E}_c$ *such that for every*

polynomials $k = k(\lambda)$ and $n = n(\lambda)$ and every poly-size $\mathcal{P}^*$, if for infinitely many $\lambda \in \mathbb{N}$ there exist instances $\mathbf{x} = (x_1, \ldots, x_k)$ with $|x_i| = n$ such that

$$\Pr\left[\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, \mathbf{x}, \pi) = 1 \quad : \quad \begin{array}{l} (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}(1^\lambda, k, n), \\ \pi \leftarrow \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \end{array} \right] \geq \frac{1}{\lambda^c},$$

then for these λs,

$$\Pr\left[\begin{array}{l} f(b_1, \ldots, b_k) = 1 \\ \text{for } b_i = \mathcal{R}(x_i, w_i) \end{array} \quad : \quad (w_1, \ldots, w_k) \leftarrow \mathcal{E}_c^{\mathcal{P}^*}(1^\lambda, C, \mathbf{x}) \right] = 1 - \mathsf{negl}(\lambda).$$

Proof of Theorem 6.5. Let $c \in \mathbb{N}$. We define the PPT oracle machine $\mathcal{E}_c$ that is given oracle access to a cheating prover $\mathcal{P}^*$, and does the following given $1^\lambda, C, \mathbf{x} = (x_1, \ldots, x_k)$ as input:

1. Let $t = t(\lambda)$ be some polynomial chosen later.
2. For each $i \in [k]$ and $j \in [t]$,
   (a) Generate $(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_i(1^\lambda, k, n)$, where Gen_i is identical to Gen, except that it generates $(\mathsf{crs}_{\mathsf{seBARG}}, \mathsf{td}_{\mathsf{seBARG}})$ programmed on the set $I = \{1, 2, i\}$, and additionally outputs $\mathsf{td}_{\mathsf{seBARG}}$.
   (b) Query $\mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}})$ and obtain a proof $\pi = (\mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$.
   (c) Extract $\omega_{i,j} = \mathsf{Extract}_{\mathsf{seBARG}}(\mathsf{td}^{(3)}_{\mathsf{seBARG}}, \pi^{(3)}_{\mathsf{seBARG}})$, and parse $\omega_{i,j} = (w_{i,j}, b_{i,j}, \rho^{(1)}, \rho^{(2)})$.
3. For each $i \in [k]$, let $w_i = w_{i,j}$ for some $j \in [t]$ such that $\mathcal{R}(x_i, w_{i,j}) = 1$, or $w_i = \perp$ if there does not exist such a j. Output $\mathbf{w} = (w_1, \ldots, w_k)$.

Fix polynomials $k = k(\lambda), n = n(\lambda)$ and a poly-size cheating prover $\mathcal{P}^*$ such that for infinitely many $\lambda \in \mathbb{N}$ there exist instances $\mathbf{x} = (x_1, \ldots, x_k)$ with $|x_i| = n$ such that

$$\Pr\left[\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, \mathbf{x}, \pi) = 1 \quad : \quad \begin{array}{l} (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}(1^\lambda, k, n), \\ \pi \leftarrow \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \end{array} \right] \geq \frac{1}{\lambda^c}.$$

For $i \in [k]$, let EXP_i be the following experiment:

- Generate $(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_i(1^\lambda, k, n)$, where Gen_i is defined as in $\mathcal{E}_c$.
- Query $\mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}})$ and obtain a proof $\pi = (\mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$.
- Extract $\omega_i^* = \mathsf{Extract}_{\mathsf{seBARG}}(\mathsf{td}^{(3)}_{\mathsf{seBARG}}, \pi^{(3)}_{\mathsf{seBARG}})$, and parse $\omega_i^* = (w_i, b_i, \rho^{(1)}, \rho^{(2)})$.
- Output $(\mathsf{crs}_{\mathcal{V}}, \pi, w_i)$.

For $c' \in \mathbb{N}$, define the set $J^{c'} \subseteq [k]$ such that $i \in J^{c'}$ if

$$\Pr_{\mathsf{EXP}_i}\left[\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, \mathbf{x}, \pi) = 1 \ \wedge \ \mathcal{R}(x_i, w_i) = 1 \right] \leq \frac{1}{\lambda^{c'}}.$$

Lemma 6.6. *There exists $c' \in \mathbb{N}$ such that for $J = J^{c'}$ it holds that $f(b_1, \ldots, b_k) = 1$, where $(b_1, \ldots, b_k)$ are defined by $b_i = \mathbf{1}_{i \notin J}$.*

Before proving Lemma 6.6, we first argue that it implies the correctness of $\mathcal{E}_c$, and thus Theorem 6.5. Let c' given by the lemma. We define $t(\lambda) = \lambda^{c'+1}$, and argue that for this choice of t for $\mathcal{E}_c$, we have

$$\Pr\left[\begin{array}{l} f(b_1,\ldots,b_k)=1 \\ \text{for } b_i = \mathcal{R}(x_i,w_i) \end{array} \;:\; (w_1,\ldots,w_k) \leftarrow \mathcal{E}_c^{\mathcal{P}^*}(1^\lambda, C, \mathbf{x}) \right] = 1 - \mathsf{negl}(\lambda).$$

By the lemma, it suffices to show that in the above experiment, for every $i \in [k]$ such that $i \notin J$, we have $b_i = 1$ except with negligible probability. Indeed, observe that $b_i = 0$ occurs if and only if $w_i = \bot$, which means that $\mathcal{R}(x_i, w_{i,j}) = 0$ for all $j \in [t]$. These are independent experiments, and by definition of J occur with probability $> \frac{1}{\lambda^{c'}}$, so we have

$$\Pr\left[\; b_i = 0 \;\right] < \left(1 - \frac{1}{\lambda^{c'}}\right)^t = \left(1 - \frac{1}{\lambda^{c'}}\right)^{\lambda^{c'}\cdot\lambda} \leq e^{-\lambda} = \mathsf{negl}(\lambda)\ .$$

Thus, if Lemma 6.6 holds, we conclude that $\mathcal{E}_c$ is correct.

Proof of Lemma 6.6. We proceed similarly to the proof of Theorem 6.1, with a few key differences due to the new definition of $J^{c'}$, which is no longer the set of false instances.

We first define the sets $J_1^{c'}, \ldots, J_d^{c'} \subseteq [N]$: let $(b_1^*, \ldots, b_k^*)$ defined by $b_j^* = \mathbf{1}_{j \notin J^{c'}}$, and let $(b_1^*, \ldots, b_N^*)$ be the values of all the wires in C on input $(b_1^*, \ldots, b_k^*)$. We define $J_i^{c'} \subseteq [N]$, the set of wires that correspond to the ith layer of C that have the value 0.

We recall the alternative key generation algorithms $\mathsf{Gen}_{i,j}$, and the functions $\mu_i = \mu_i^{(1)}, \mu_i^{(2)}$, defined by $J_1^{c'}, \ldots, J_d^{c'}$ as in the proof of Theorem 6.1. We prove the following claim:

Claim 6.7. *There exists a polynomial* poly *such that for all* $c' \in \mathbb{N}$ *and* $i \geq 1$, $\mu_i \leq \mu_{i-1} + \frac{\mathsf{poly}(\lambda)}{\lambda^{c'}} + \mathsf{negl}(\lambda)$.

We now observe that Claim 6.7 implies Lemma 6.6. Iteratively applying the claim and using the fact that $\mu_0 = 0$, we get that $\mu_d \leq \frac{\mathsf{poly}(\lambda)}{\lambda^{c'}}$ for some polynomial poly. Defining c' such that $\frac{\mathsf{poly}(\lambda)}{\lambda^{c'}} \leq \frac{1}{2\lambda^c}$ for large enough λ, we get

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \;\wedge \\ \mathsf{Extract}(\mathsf{td}_{\mathsf{PEH}}^{(1)}, \mathsf{v}^{(1)}) = 1 \end{array} \;:\; \begin{array}{l} (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}_{d,d-1}(1^\lambda, k, n), \\ \pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \end{array} \right] \geq \frac{1}{\lambda^c} - \mu_d(\lambda) \geq \frac{1}{2\lambda^c}\ .$$

Since the seBARG in $\mathsf{Gen}_{d,d-1}$ is extractable on index N, extracting and parsing $\omega_N^* = \left(b_N, b_{N_1}, b_{N_2}, \left(\rho_N^{(\beta)}, \rho_{N_1}^{(\beta)}, \rho_{N_2}^{(\beta)}\right)_{\beta\in\{1,2\}}\right)$ from the seBARG proof in the above experiment, we have that

$$\Pr\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \;\wedge \\ \mathsf{Extract}(\mathsf{td}_{\mathsf{PEH}}^{(1)}, \mathsf{v}^{(1)}) = 1 \;\wedge \\ M(z, N, \omega_N^*) = 1 \end{array} \;:\; \begin{array}{l} (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}_{d,d-1}(1^\lambda, k, n), \\ \pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \end{array} \right] \geq \frac{1}{2\lambda^c} - \mathsf{negl}(\lambda)\ .$$

Observe that $M(z, N, \omega_N^*) = 1$ implies that $b_N = 1$ and $\rho_N^{(1)}$ is a valid opening of $\mathsf{v}^{(1)}$ to b_N. Assume towards contradiction that $f(b_1^*, \ldots, b_k^*) = 0$, then $b_N^* = 0$ so $J_d^{c'}$ contains wire N, but then since $\mathsf{Extract}(\mathsf{td}_{\mathsf{PEH}}^{(1)}, \mathsf{v}^{(1)}) = 1$ by PEHash extraction correctness the above can only happen with negligible probability. So, once we prove Claim 6.7, we can conclude that $f(b_1^*, \ldots, b_k^*) = 1$.

Proof of Claim 6.7. We first show that for all $i \geq 1$, $\mu_i^{(1)} \leq \mu_{i-1}^{(2)} + \frac{\mathsf{poly}(\lambda)}{\lambda^{c'}} + \mathsf{negl}(\lambda)$. The proof follows similarly to Claim 6.3. Steps 1,2,3 in the proof are identical, since they do not depend on the set J. In the notation of the original proof, we have

$$\widetilde{\Pr}\left[\mathsf{BAD} \ \wedge \ j = \mathsf{ExtractIndex}(\mathsf{td}_{\mathsf{PEH},i}^{(1)}, \mathsf{v}^{(1)}) \ \wedge \ \mathsf{Correct}_M \ \wedge \ b_{j_\alpha} > b_{j_\alpha}^*\right] \geq \frac{\delta_{i,i-1}}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda).$$

In step 4, we claimed that the above event happens with probability zero when $j_\alpha \leq k$. This is no longer the case, however we can claim that it happens with probability $\leq \frac{1}{\lambda^{c'}} + \mathsf{negl}(\lambda)$. Indeed, by definition, in case the above event happens we have $b_{j_\alpha} = 1$, and $b_{j_\alpha}^* = 0$. The latter implies that $j_\alpha \in J^{c'}$, and we know that in this case (by definition of $J^{c'}$) the former happens with probability $\leq \frac{1}{\lambda^{c'}}$ (since $b_{j_\alpha} = 1$ and $\mathsf{Correct}_M$ imply that $\mathcal{R}(x_{j_\alpha}, w_{j_\alpha}) = 1$), with negligible error to account for BARG and PEHash key indistinguishability errors in switching to the experiment in $\widetilde{\Pr}$.

The case $j_\alpha > k$ is identical, and happens with negligible probability. Therefore, we get

$$\frac{1}{\lambda^{c'}} + \mathsf{negl}(\lambda) \geq \frac{\delta_{i,i-1}}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda).$$

which implies the desired bound, since

$$\mu_i^{(1)} \leq \mu_{i-1}^{(2)} + \delta_{i,i-1} \leq \mu_{i-1}^{(2)} + \frac{\mathsf{poly}(\lambda)}{\lambda^{c'}} + \mathsf{negl}(\lambda).$$

Now, we observe that Claim 6.4 still holds (the proof does not depend on J), so we have $\mu_{i-1}^{(2)} \leq \mu_{i-1}^{(1)} + \mathsf{negl}(\lambda)$, which together with the above bound finishes the proof of Claim 6.7.

7 Somewhere Extractable SNARG Construction

In this section, we construct and analyze a somewhere extractable SNARG scheme $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ for the language $\mathcal{L}_f^{(k)}$ defined in Section 4 with respect to an underlying NP language $\mathcal{L}$ and a monotone circuit C computing function f. We give the construction in Section 7.1 and analyze its security and efficiency in Section 7.2.

7.1 Construction

Our construction is similar to the construction in Section 6, with the following differences:

- To obtain somewhere extractability, we add an additional PEHash family with tags, that is used to hash the input layer to the monotone circuit, such that each bit of the input is attached with the corresponding witness as its tag.
- We obtain a $\mathsf{crs}_{\mathcal{P}}$ of length $(m+k) \cdot \mathsf{poly}(\lambda, \log k)$, rather than $(m + \mathrm{width}(C)) \cdot \mathsf{poly}(\lambda, \log k)$. This is achieved by using a PEH family with shorter hash key (discussed in Remark 5.2) to hash the monotone circuit wires.
- We assume for simplicity that the layers of the circuit are partitioned such that wire values in layer i are computed only from wires in layer $i-1$, and only layer 1 is computed from input wires in layer 0. We no longer have circuits with width less than k, but this has no effect on efficiency, since $\mathsf{crs}_{\mathcal{P}}$ grows with k (unlike the SNARG in Section 6).

We now describe the construction, which uses the following building blocks:

- A PEHash family with tags (Definition 5.4)

  $$(\mathsf{Gen}_{\mathsf{PEHT}}, \mathsf{Hash}_{\mathsf{PEHT}}, \mathsf{Open}_{\mathsf{PEHT}}, \mathsf{Verify}_{\mathsf{PEHT}}, \mathsf{Extract}_{\mathsf{PEHT}}, \mathsf{ExtractIndex}_{\mathsf{PEHT}}, \mathsf{ExtractTag}_{\mathsf{PEHT}})$$

  with respect to the bit-fixing predicate family.

- A PEHash family (Definition 5.2 and Remark 5.2)

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

  for a restricted set of bit-fixing predicates $\mathcal{F}$ with succinct description. We consider the set of predicates defined by the TM $\mathcal{M} = \mathcal{M}_{\mathcal{F}}$, that does the following given input $z = z_{\mathcal{F}} = C$ and predicate description d_f:

  1. Parse $d_f = (x, i)$, where $x \in \{0,1\}^k$ is an input to the circuit and $i \in [d]$ is a layer in the circuit.
  2. Let $(b_1^*, \ldots, b_N^*)$ be the values of all the wires in C on input x.
  3. Let $J \subseteq [N]$ be the set of wires in the ith layer of C whose values b^* are 0, and let $y \in \{0,1\}^J$ be the all-zero vector.
  4. Output J, y.

  Recall that the PEHash algorithms (except $\mathsf{Gen}_{\mathsf{PEH}}$) are modified to receive $z = C$ as an additional input.

- A seBARG scheme (Definition 3.5)

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

We are now ready to define our SNARG for $\mathcal{L}_f^{(k)}$.

$\mathsf{Gen}(1^\lambda, k, n, J)$ does the following:

1. Generate $(\mathsf{hk}_{\mathsf{PEHT}}, \mathsf{vk}_{\mathsf{PEHT}}, \mathsf{td}_{\mathsf{PEHT}}) \leftarrow \mathsf{Gen}_{\mathsf{PEHT}}(1^\lambda, N, J, y)$, where $y = 0^J$.
2. For each $\beta \in \{1,2\}$, generate $(\mathsf{hk}_{\mathsf{PEH}}^{(\beta)}, \mathsf{vk}_{\mathsf{PEH}}^{(\beta)}, \mathsf{td}_{\mathsf{PEH}}^{(\beta)}) \leftarrow \mathsf{Gen}_{\mathsf{PEH}}(1^\lambda, N, d_{f_\beta})$, where d_{f_β} is a description of an arbitrary predicate $f_\beta \in \mathcal{F}$.
3. Generate $(\mathsf{crs}_{\mathsf{seBARG}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_{\mathsf{seBARG}}(1^\lambda, 1^{n'}, 1^{m'}, I)$ where

   $$n' = O(1) + |C| + kn + |\mathsf{vk}_{\mathsf{PEHT}}| + |\mathsf{v}_{\mathsf{PEHT}}| + 2|\mathsf{vk}_{\mathsf{PEH}}| + 2|\mathsf{v}_{\mathsf{PEH}}|,$$
   $$m' = m + 3 + 6|\rho|,$$

   and where $I \subseteq [N]$ is initialized to $\{1, 2, N\}$.
4. Let $\mathsf{crs}_{\mathcal{P}} = (\mathsf{hk}_{\mathsf{PEHT}}, \mathsf{vk}_{\mathsf{PEHT}}, \mathsf{hk}_{\mathsf{PEH}}^{(1)}, \mathsf{vk}_{\mathsf{PEH}}^{(1)}, \mathsf{hk}_{\mathsf{PEH}}^{(2)}, \mathsf{vk}_{\mathsf{PEH}}^{(2)}, \mathsf{crs}_{\mathsf{seBARG}})$.
5. Let $\mathsf{crs}_{\mathcal{V}} = (\mathsf{vk}_{\mathsf{PEHT}}, \mathsf{vk}_{\mathsf{PEH}}^{(1)}, \mathsf{vk}_{\mathsf{PEH}}^{(2)}, \mathsf{crs}_{\mathsf{seBARG}})$.
6. Output $(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}, \mathsf{td} = \mathsf{td}_{\mathsf{PEHT}})$.

$\mathcal{P}(\mathsf{crs}_{\mathcal{P}}, C, x_1, \ldots, x_k, w_1, \ldots, w_k)$ does the following:

1. Compute the values of all the wires in the circuit C. Denote these values by $(b_1, \ldots, b_N)$.
2. Parse $\mathsf{crs}_{\mathcal{P}} = (\mathsf{hk}_{\mathsf{PEHT}}, \mathsf{vk}_{\mathsf{PEHT}}, \mathsf{hk}^{(1)}_{\mathsf{PEH}}, \mathsf{vk}^{(1)}_{\mathsf{PEH}}, \mathsf{hk}^{(2)}_{\mathsf{PEH}}, \mathsf{vk}^{(2)}_{\mathsf{PEH}}, \mathsf{crs}_{\mathsf{seBARG}})$.
3. Compute $\mathsf{v}_{\mathsf{PEHT}} = \mathsf{Hash}_{\mathsf{PEHT}}(\mathsf{hk}_{\mathsf{PEHT}}, (b_1, \ldots, b_k), (w_1, \ldots, w_k))$.
4. Compute $\mathsf{v}^{(\beta)} = \mathsf{Hash}_{\mathsf{PEH}}(\mathsf{hk}^{(\beta)}_{\mathsf{PEH}}, (b_1, \ldots, b_N), C)$ for $\beta \in \{1, 2\}$.
5. Define an instance $X = (M, z, N, T)$ of $\mathsf{BatchIndexTMSAT}$. The input z is defined as $z = (C, x_1, \ldots, x_k, \mathsf{vk}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}, (\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)})_{\beta \in \{1,2\}})$. The batch size is set to N. The Turing machine $M(z, j, \omega_j)$ is defined to operate as follows:
   (a) Parse $z = (C, x_1, \ldots, x_k, \mathsf{vk}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}, (\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)})_{\beta \in \{1,2\}})$.
   (b) If $1 \le j \le k$:
      i. Parse $\omega_j = (w_j, b_j, \rho_{\mathsf{PEHT}}, \rho^{(1)}, \rho^{(2)})$.
      ii. Check that $\mathsf{Verify}_{\mathsf{PEHT}}(\mathsf{vk}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}, j, b_j, w_j, \rho_{\mathsf{PEHT}}) = 1$.
      iii. Check that $\mathsf{Verify}_{\mathsf{PEH}}(\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)}, j, b_j, \rho^{(\beta)}, C) = 1$ for $\beta \in \{1, 2\}$.
      iv. Check that $\mathcal{R}_{\mathcal{L}}(x_j, w_j) = b_j$.
   (c) If $j > k$:
      i. Compute the jth gate of C, $g_j = (j, j_1, j_2, c \in \{\mathrm{AND}, \mathrm{OR}\})$.
      ii. Parse $\omega_j = \left(b_j, b_{j_1}, b_{j_2}, \left(\rho^{(\beta)}_j, \rho^{(\beta)}_{j_1}, \rho^{(\beta)}_{j_2}\right)_{\beta \in \{1,2\}}\right)$.
      iii. Check that $\mathsf{Verify}_{\mathsf{PEH}}(\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)}, j, b_j, \rho^{(\beta)}_j, C) = 1$ for $\beta \in \{1, 2\}$.
      iv. Check that $\mathsf{Verify}_{\mathsf{PEH}}(\mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{v}^{(\beta)}, j_\alpha, b_{j_\alpha}, \rho^{(\beta)}_{j_\alpha}, C) = 1$ for $\alpha, \beta \in \{1, 2\}$.
      v. Check that $b_j = c(b_{j_1}, b_{j_2})$. (That is, check that the gate is satisfied.)
      vi. If $j = N$ is the output wire then check that $b_j = 1$.

   The description length of M is a constant. Finally, the time bound $T = \mathsf{poly}(N, n, m, k)$ is set so that the pseudocode above terminates.
6. For every $j \in [N]$, construct a witness (j, ω_j) for X, using the $\mathsf{Open}_{\mathsf{PEH}}, \mathsf{Open}_{\mathsf{PEHT}}$ algorithms to produce openings for $(b_1, \ldots, b_N)$ and $(b_1, \ldots, b_k)$ with tags $(w_1, \ldots, w_k)$ as appropriate.
7. Compute $\pi_{\mathsf{seBARG}} = \mathcal{P}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, M, z, 1^T, \omega_1, \ldots, \omega_N)$.
8. Output $\pi = (\mathsf{v}_{\mathsf{PEHT}}, \mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$.

$\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, x_1, \ldots, x_k, \pi)$ does the following:

1. Parse $\mathsf{crs}_{\mathcal{V}} = (\mathsf{vk}_{\mathsf{PEHT}}, \mathsf{vk}^{(1)}_{\mathsf{PEH}}, \mathsf{vk}^{(2)}_{\mathsf{PEH}}, \mathsf{crs}_{\mathsf{seBARG}})$.
2. Parse $\pi = (\mathsf{v}_{\mathsf{PEHT}}, \mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$.
3. Define $X = (M, z, N, T)$ as above.
4. Output $\mathcal{V}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, X, \pi_{\mathsf{seBARG}})$.

$\mathsf{Extract}(\mathsf{td}, \pi)$ does the following:

1. Parse $\pi = (\mathsf{v}_{\mathsf{PEHT}}, \mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$.
2. Compute $j = \mathsf{ExtractIndex}_{\mathsf{PEHT}}(\mathsf{td}, \mathsf{v}_{\mathsf{PEHT}})$, and $w_j = \mathsf{ExtractTag}(\mathsf{td}, \mathsf{v}_{\mathsf{PEHT}})$.
3. Output (j, w_j).

7.2 Analysis

Theorem 7.1. *The construction given in Section 7 is a somewhere extractable* SNARG *for* $\mathcal{L}_f^{(k)}$ *(Definition 4.1) with the following additional efficiency properties:*

- *The runtime of the verifier is* $\mathsf{poly}(|\mathsf{crs}_{\mathcal{V}}|+|\pi|) + (kn + |C|)\mathsf{poly}(\lambda, \log k)$.
- *The (prover) common reference string* $\mathsf{crs}_{\mathcal{P}}$ *has size* $\mathsf{poly}(\lambda, \log k)(m + k)$.

Proof of Theorem 7.1.

Completeness. Follows immediately from the completeness of the underlying seBARG and the opening completeness of the underlying PEHash families.

Efficiency. To bound $|\mathsf{crs}_{\mathcal{V}}|+|\pi|$, we make use of the following facts:

- By the efficiency of PEH, $|\mathsf{vk}_{\mathsf{PEH}}^{(\beta)}|+|\mathsf{v}_{\mathsf{PEH}}^{(\beta)}| \leq \mathsf{poly}(\lambda, \log k)$ for $\beta \in \{1, 2\}$.
- By the efficiency of PEHT, $|\mathsf{vk}_{\mathsf{PEHT}}|+|\mathsf{v}_{\mathsf{PEHT}}| \leq m \cdot \mathsf{poly}(\lambda, \log k)$.
- The length of the BatchIndexTMSAT instance X is $|X| \leq |C|+kn + \mathsf{poly}(\lambda, \log(knm))$. The length of a witness ω_i is at most $m + \mathsf{poly}(\lambda, \log k)$.
- Therefore, by the succinctness of seBARG, we conclude that $|\mathsf{crs}_{\mathsf{seBARG}}|+|\pi_{\mathsf{seBARG}}| \leq m \cdot \mathsf{poly}(\lambda, \log(knm))$.

The bound the running time of the verifier, we again invoke the efficiency of the seBARG to conclude the claimed $\mathsf{poly}(|\mathsf{crs}_{\mathcal{V}}|+|\pi|) + (kn + |C|)\mathsf{poly}(\lambda, \log k)$ bound.

Finally, the size of $|\mathsf{crs}_{\mathcal{P}}|$ is at most $m \cdot \mathsf{poly}(\lambda, \log(knm)) + 2 \cdot |\mathsf{hk}_{\mathsf{PEH}}|+|\mathsf{hk}_{\mathsf{PEHT}}|$ which is at most $m \cdot \mathsf{poly}(\lambda, \log(knm)) + \mathsf{poly}(\lambda, \log k) \cdot k$ by the efficiency of the PEHash family with succinct description.

Key Indistinguishability. Follows immediately from the predicate hiding property of the underlying PEHash families.

Somewhere argument of knowledge. Fix any poly-size cheating prover $\mathcal{P}^*$, any two polynomials $k = k(\lambda)$ and $n = n(\lambda)$, and any set $J \subseteq [k]$ such that the assignment $b \in \{0,1\}^k$ defined by $b_i = \mathbf{1}_{i \notin J}$ satisfies $C(b_1, \ldots, b_k) = 0$.

Let EXP be the experiment of the somewhere argument of knowledge requirement:

$$\left\{ \begin{array}{l} (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, k, n, J) \\ (x_1, \ldots, x_k, \pi) \leftarrow \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \\ (j, w_j) \leftarrow \mathsf{Extract}(\mathsf{td}, \pi) \end{array} \right\}.$$

By definition, the success probability of $\mathcal{P}^*$ is

$$\epsilon(\lambda) \triangleq \Pr_{\mathsf{EXP}} \left[\begin{array}{l} \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, x_1, \ldots, x_k, \pi) = 1 \wedge \\ j \notin J \ \vee \ \mathcal{R}(x_j, w_j) = 0 \end{array} \right]. \tag{4}$$

We want to show that $\epsilon(\lambda)$ is a negligible function.

We prove soundness by a hybrid argument based on alternative experiments $(\mathsf{EXP}_{i,j})_{0\le i,j\le d}$. The experiment $\mathsf{EXP}_{i,j}$ is identical to EXP, except for an alternative key generation algorithm $\mathsf{Gen}_{i,j}$ defined below.

$\mathsf{Gen}_{i,j}(1^\lambda, k, n, J)$ does the following:

1. Define predicates $d_{f_1} = (b, i)$ and $d_{f_2} = (b, j)$, where $b \in \{0,1\}^k$ is defined by $b_i = \mathbf{1}_{i \notin J}$.
2. Generate $(\mathsf{hk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{vk}^{(\beta)}_{\mathsf{PEH}}, \mathsf{td}^{(\beta)}_{\mathsf{PEH}}) \leftarrow \mathsf{Gen}_{\mathsf{PEH}}(1^\lambda, N, d_{f_\beta})$ for $\beta \in \{1,2\}$.
3. Proceed otherwise as in Gen.

By the predicate hiding property of the underlying PEH family (Definition 5.1), Eq. (4) implies there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $i \in [d]$

$$\Pr_{\mathsf{EXP}_{i,i-1}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, x_1, \ldots, x_k, \pi) = 1 \ \wedge \\ j \notin J \ \vee \ \mathcal{R}(x_j, w_j) = 0\end{array}\right] \ge \epsilon(\lambda) - \mathsf{negl}(\lambda).$$

Parse $\pi = (\mathsf{v}_{\mathsf{PEHT}}, \mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$. For every $i \in [d]$ and every $\lambda \in \mathbb{N}$, denote by

$$\mu_i(\lambda) = \mu_i^{(1)}(\lambda) \triangleq \Pr_{\mathsf{EXP}_{i,i-1}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, x_1, \ldots, x_k, \pi) = 1 \ \wedge \\ j \notin J \ \vee \ \mathcal{R}(x_j, w_j) = 0 \ \wedge \\ \mathsf{Extract}(\mathsf{td}^{(1)}_{\mathsf{PEH}}, \mathsf{v}^{(1)}, C) = 0\end{array}\right],$$

$$\mu^{(2)}_{i-1}(\lambda) \triangleq \Pr_{\mathsf{EXP}_{i,i-1}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, x_1, \ldots, x_k, \pi) = 1 \ \wedge \\ j \notin J \ \vee \ \mathcal{R}(x_j, w_j) = 0 \ \wedge \\ \mathsf{Extract}(\mathsf{td}^{(2)}_{\mathsf{PEH}}, \mathsf{v}^{(2)}, C) = 0\end{array}\right].$$

Additionally, denote

$$\mu_{\mathsf{PEHT}}(\lambda) \triangleq \Pr_{\mathsf{EXP}_{0,0}}\left[\begin{array}{l}\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, x_1, \ldots, x_k, \pi) = 1 \ \wedge \\ j \notin J \ \vee \ \mathcal{R}(x_j, w_j) = 0 \ \wedge \\ \mathsf{Extract}(\mathsf{td}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}) = 0\end{array}\right].$$

Lemma 7.2. *μ_d is a negligible function.*

First, we argue that Lemma 7.2 implies that ϵ is negligible. This follows as in the analysis in Section 6, using the fact that the predicate described by $d_{f_d} = (b, d)$ requires that the output wire is 0 (since we assumed $C(b_1, \ldots, b_k) = 0$) and the somewhere argument of knowledge and consistency of extraction properties of the underlying seBARG and PEHash.

Proof of Lemma 7.2. We prove the following three claims, which together imply Lemma 7.2.

Claim 7.3. *There exists a negligible function negl such that for every $i \in [d]$,*

$$\mu_i \le \mu_{i-1} + \mathsf{negl}(\lambda),$$

where we define $\mu_0(\lambda) = \mu_0^{(2)}(\lambda)$.

Claim 7.4. $\mu_0 \le \mu_{\mathsf{PEHT}} + \mathsf{negl}(\lambda)$.

Claim 7.5. $\mu_{\mathsf{PEHT}} \le \mathsf{negl}(\lambda)$.

Proof of Claim 7.3. Recall that in Section 6, we proved Claim 6.3 and Claim 6.4, which state that $\mu_i^{(1)} \leq \mu_{i-1}^{(2)} + \mathsf{negl}(\lambda)$ for all $i \geq 1$, and $\mu_{i-1}^{(2)} \leq \mu_{i-1}^{(1)} + \mathsf{negl}(\lambda)$ for all $i > 1$.

Observe that these claims still hold for our construction, with the following minor differences in the proof:

- The quantities $\mu_i^{(1)}, \mu_i^{(2)}$ in Section 6, unlike our definition, do not have the condition that $j \notin J \ \vee \ \mathcal{R}(x_j, w_j) = 0$. However, Claim 6.3 and Claim 6.4 respectively bound the difference between them by showing that quantities $\delta_{i,i-1}$ and δ_{i-1} are negligible, and adding the condition $j \notin J \ \vee \ \mathcal{R}(x_j, w_j) = 0$ to $\delta_{i,i-1}, \delta_{i-1}$ only reduces them, so it suffices to bound the original $\delta_{i,i-1}, \delta_{i-1}$ as in Section 6.
- The seBARG instances for $j > k$ in our construction are identical to the construction in Section 6, except for slightly modified syntax since we use a PEHash family for predicates with succinct description.
- In Step 4 in the proof of Claim 6.3, it is no longer true that the event with $j_\alpha \leq k$ happens with probability zero (this uses the fact that the instances $x_1, \ldots, x_k$ are fixed). However, since we assumed that input wires can only be used as inputs to layer 1, this only changes the claim in the case of $i = 1$, but there we can use the same argument as in $j_\alpha > k$ to claim that the event occurs with negligible probability, and get that $\mu_1 \leq \mu_0 + \mathsf{negl}(\lambda)$ (where observe we have $\mu_0 = \mu_0^{(2)}$, and not $\mu_0 = 0$).

This completes the proof of Claim 7.3, since we get that for every $i \in [d]$,

$$\mu_i \leq \mu_{i-1}^{(2)} + \mathsf{negl}(\lambda) \leq \mu_{i-1} + \mathsf{negl}(\lambda).$$

Proof of Claim 7.4. The proof follows similarly to the proof of Claim 6.4. First, by the predicate hiding property of the underlying PEH,

$$\mu_0 = \mu_0^{(2)} \leq \Pr_{\mathsf{EXP}_{0,0}}\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, x_1, \ldots, x_k, \pi) = 1 \ \wedge \\ j \notin J \ \vee \ \mathcal{R}(x_j, w_j) = 0 \ \wedge \\ \mathsf{Extract}(\mathsf{td}_{\mathsf{PEH}}^{(2)}, \mathsf{v}^{(2)}, C) = 0 \end{array}\right] + \mathsf{negl}(\lambda).$$

Therefore, it suffices to show that

$$\delta_{0,\mathsf{PEHT}} \triangleq \Pr_{\mathsf{EXP}_{0,0}}\left[\begin{array}{l} \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, x_1, \ldots, x_k, \pi) = 1 \ \wedge \\ j \notin J \ \vee \ \mathcal{R}(x_j, w_j) = 0 \ \wedge \\ \mathsf{Extract}(\mathsf{td}_{\mathsf{PEH}}^{(2)}, \mathsf{v}^{(2)}, C) = 0 \ \wedge \\ \mathsf{Extract}(\mathsf{td}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}) = 1 \end{array}\right] = \mathsf{negl}(\lambda).$$

Observe that by definition $\mathsf{v}^{(2)}, \mathsf{v}_{\mathsf{PEHT}}$ are both programmed on the predicate $J, 0^J$, and the seBARG instances for $1 \leq j \leq k$ verify openings wrt both $\mathsf{v}^{(2)}$ and $\mathsf{v}_{\mathsf{PEHT}}$, so this follows exactly as in Claim 6.4, which completes the proof of Claim 7.4.

Proof of Claim 7.5. We define a hybrid experiment $\widetilde{\mathsf{EXP}}_{0,0}$, that is identical to $\mathsf{EXP}_{0,0}$, except for the following changes:

- Sample a uniformly random index $1 \leq j^* \leq k$.

- Sample $(\mathsf{crs}_{\mathsf{seBARG}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_{\mathsf{seBARG}}(1^\lambda, 1^{n'}, 1^{m'}, \{1, 2, j^*\})$.
- Parse $\pi = (\mathsf{v}_{\mathsf{PEHT}}, \mathsf{v}^{(1)}, \mathsf{v}^{(2)}, \pi_{\mathsf{seBARG}})$, where π is the proof outputted by $\mathcal{P}$ in $\mathsf{EXP}_{0,0}$.
- Extract $\omega_1, \omega_2, \omega_{j^*} \leftarrow \mathsf{Extract}_{\mathsf{seBARG}}(\mathsf{td}_{\mathsf{seBARG}}, \pi_{\mathsf{seBARG}})$.

We proceed to bound μ_{PEHT}. Denote by BAD the event defined in μ_{PEHT}.

Step 1: Switch to $\widetilde{\mathsf{EXP}}_{0,0}$. This is accomplished in two steps. First, we note that

$$\Pr_{\mathsf{EXP}_{0,0}}\left[\ \mathsf{BAD} \ \wedge\ j^* = \mathsf{ExtractIndex}(\mathsf{td}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}})\ \right] = \frac{\mu_{\mathsf{PEHT}}}{k} = \frac{\mu_{\mathsf{PEHT}}}{\mathsf{poly}(\lambda)},$$

since j is independent of the $\mathsf{EXP}_{0,0}$ experiment. Then, we also have that

$$\Pr_{\widetilde{\mathsf{EXP}}_{0,0}}\left[\ \mathsf{BAD} \ \wedge\ j^* = \mathsf{ExtractIndex}(\mathsf{td}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}})\ \right] \geq \frac{\mu_{\mathsf{PEHT}}}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda),$$

by the index-hiding of seBARG.

Step 2: Analyze the output of $\mathsf{Extract}_{\mathsf{seBARG}}$. By the somewhere argument of knowledge property of seBARG, we know that

$$\Pr_{\widetilde{\mathsf{EXP}}_{0,0}}\left[\ \mathsf{BAD} \ \wedge\ j^* = \mathsf{ExtractIndex}(\mathsf{td}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}) \ \wedge\ M(z, j^*, \omega_{j^*}) = 1\ \right] \geq \frac{\mu_{\mathsf{PEHT}}}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda).$$

Parse the witness ω_{j^*} as $\omega_{j^*} = (w, b, \rho_{\mathsf{PEHT}}, \rho^{(1)}, \rho^{(2)})$. Recall that $M(z, j^*, \omega_{j^*}) = 1$ implies that $\mathsf{Verify}_{\mathsf{PEHT}}(\mathsf{vk}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}, j, b, w, \rho_{\mathsf{PEHT}}) = 1$ and $\mathcal{R}_{\mathcal{L}}(x_j, w) = b$.

Recall that BAD implies $j \notin J \vee \mathcal{R}_{\mathcal{L}}(x_j, w_j) = 0$, where $(j, w_j) \leftarrow \mathsf{Extract}(\mathsf{td}, \pi)$ are defined by $j \leftarrow \mathsf{ExtractIndex}(\mathsf{td}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}})$ and $w_j \leftarrow \mathsf{ExtractTag}(\mathsf{td}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}})$. But we have $j \in J$ by definition.

Step 3: Use PEHT consistency. We get

$$\Pr_{\widetilde{\mathsf{EXP}}_{0,0}}\left[\begin{array}{l}\mathsf{Extract}(\mathsf{td}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}) = 0 \ \wedge \\ \mathsf{ExtractIndex}(\mathsf{td}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}) = j \ \wedge \\ \mathsf{ExtractTag}(\mathsf{td}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}) = w_j \ \wedge \\ \mathsf{Verify}_{\mathsf{PEHT}}(\mathsf{vk}_{\mathsf{PEHT}}, \mathsf{v}_{\mathsf{PEHT}}, j, b, w, \rho_{\mathsf{PEHT}}) = 1 \ \wedge \\ \mathcal{R}_{\mathcal{L}}(x_j, w) = b \ \wedge\ \mathcal{R}_{\mathcal{L}}(x_j, w_j) = 0\end{array}\right] \geq \frac{\mu_{\mathsf{PEHT}}}{\mathsf{poly}(\lambda)} - \mathsf{negl}(\lambda).$$

Now, by the PEHT consistency of extraction property, and since $y = 0^J$ and $j \in J$, we have that except with negligible probability, $b = 1$. By the consistency of tag extraction property, we get that $w = w_j$ except with negligible probability.

This implies that the above probability is negligible, since we get $1 = b = \mathcal{R}_{\mathcal{L}}(x_j, w) = \mathcal{R}_{\mathcal{L}}(x_j, w_j) = 0$. So we get that μ_{PEHT} is negligible, which finishes the proof of Claim 7.5, Lemma 7.2, and Theorem 7.1.

8 SNARGs for Low-Depth Monotone BatchNP Circuits

In this section, we construct and analyze a SNARG scheme $(\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ for the language $\mathcal{L}_f^{(k)}$ defined in Section 4 with respect to an underlying NP language $\mathcal{L}$ and a monotone circuit C computing function f.

We give the construction in Section 8.1 and analyze its security and efficiency in Section 8.2.

8.1 Construction

Our construction is similar to the construction in Section 6, with the following differences:

- Rather than using a bit-fixing PEHash family to hash the monotone circuit wire values (with two separate keys), we use a standard hash family with local opening.
- Due to the analysis, both the hash family and seBARG are required to be subexponentially secure rather than polynomially secure.

We now describe the construction, which uses the following building blocks:

- A subexponentially-secure hash family with local opening (Definition 3.1)

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

- A subexponentially-secure seBARG scheme (Definition 3.5)

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

We are now ready to define our SNARG for $\mathcal{L}_f^{(k)}$.

$\mathsf{Gen}(1^\lambda, k, n)$ does the following:

1. Set $\lambda' = \mathsf{poly}(\lambda, d(\lambda))$, where $d = d(\lambda)$ is the depth of the monotone circuit C.
2. Generate $\mathsf{hk}_{\mathsf{HT}} \leftarrow \mathsf{Gen}_{\mathsf{HT}}(1^{\lambda'})$.
3. Generate $(\mathsf{crs}_{\mathsf{seBARG}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_{\mathsf{seBARG}}(1^{\lambda'}, 1^{n'}, 1^{m'}, I)$ where
$$n' = O(1) + |C| + kn + \mathsf{poly}(\lambda'),$$
$$m' = m + 3 + 3 \cdot \mathsf{poly}(\lambda'),$$
and where $I \subseteq [N]$ is initialized to $\{1, N\}$.
4. Output $\mathsf{crs}_{\mathcal{P}} = \mathsf{crs}_{\mathcal{V}} = (\mathsf{hk}_{\mathsf{HT}}, \mathsf{crs}_{\mathsf{seBARG}})$.

$\mathcal{P}(\mathsf{crs}_{\mathcal{P}}, C, x_1, \ldots, x_k, w_1, \ldots, w_k)$ does the following:

1. Compute the values of all the wires in the circuit C. Denote these values by $(b_1, \ldots, b_N)$.
2. Parse $\mathsf{crs}_{\mathcal{P}} = (\mathsf{hk}_{\mathsf{HT}}, \mathsf{crs}_{\mathsf{seBARG}})$.
3. Compute $\mathsf{rt} = \mathsf{Hash}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}}, (b_1, \ldots, b_N))$.

4. Define an instance $X = (M, z, N, T)$ of $\mathsf{BatchIndexTMSAT}$. The input z is defined as $z = (C, x_1, \ldots, x_k, \mathsf{hk}_{\mathsf{HT}}, \mathsf{rt})$. The batch size is set to N. The Turing machine $M(z, j, \omega_j)$ is defined to operate as follows:
   (a) Parse $z = (C, x_1, \ldots, x_k, \mathsf{hk}_{\mathsf{HT}}, \mathsf{rt})$.
   (b) If $1 \le j \le k$:
      i. Parse $\omega_j = (w_j, b_j, \rho)$.
      ii. Check that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}, j, b_j, \rho) = 1$.
      iii. Check that $\mathcal{R}_{\mathcal{L}}(x_j, w_j) = b_j$.
   (c) If $j > k$:
      i. Compute the jth gate of C, $g_j = (j, j_1, j_2, c \in \{\mathrm{AND}, \mathrm{OR}\})$.
      ii. Parse $\omega_j = (b_j, b_{j_1}, b_{j_2}, \rho_j, \rho_{j_1}, \rho_{j_2})$.
      iii. Check that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}, j, b_j, \rho_j, C) = 1$.
      iv. Check that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}}, \mathsf{rt}, j_\alpha, b_{j_\alpha}, \rho_j, C) = 1$ for $\alpha \in \{1, 2\}$.
      v. Check that $b_j = c(b_{j_1}, b_{j_2})$. (That is, check that the gate is satisfied.)
      vi. If $j = N$ is the output wire then check that $b_j = 1$.

   The description length of M is a constant. Finally, the time bound $T = \mathsf{poly}(N, n, m, k)$ is set so that the pseudocode above terminates.
5. For every $j \in [N]$, construct a witness (j, ω_j) for X, using the $\mathsf{Open}_{\mathsf{HT}}$ algorithm to produce openings for $(b_1, \ldots, b_N)$ as appropriate.
6. Compute $\pi_{\mathsf{seBARG}} = \mathcal{P}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, M, z, 1^T, \omega_1, \ldots, \omega_N)$.
7. Output $\pi = (\mathsf{rt}, \pi_{\mathsf{seBARG}})$.

$\mathcal{V}(\mathsf{crs}_{\mathcal{V}}, C, x_1, \ldots, x_k, \pi)$ does the following:

1. Parse $\mathsf{crs}_{\mathcal{V}} = (\mathsf{hk}_{\mathsf{HT}}, \mathsf{crs}_{\mathsf{seBARG}})$.
2. Parse $\pi = (\mathsf{rt}, \pi_{\mathsf{seBARG}})$.
3. Define $X = (M, z, N, T)$ as above.
4. Output $\mathcal{V}_{\mathsf{seBARG}}(\mathsf{crs}_{\mathsf{seBARG}}, X, \pi_{\mathsf{seBARG}})$.

8.2 Analysis

Theorem 8.1. *The construction given in Section 8 is a subexponentially-secure* SNARG *for* $\mathcal{L}_f^{(k)}$ *(Definition 4.1) with the following efficiency properties:*

- *The common reference string* $\mathsf{crs}_{\mathcal{P}} = \mathsf{crs}_{\mathcal{V}}$ *and the proof* π *have size* $m \cdot \mathsf{poly}(\lambda, d(\lambda), \log k)$.
- *The runtime of the verifier is* $\mathsf{poly}(|\mathsf{crs}_{\mathcal{V}}| + |\pi|) + (kn + |C|)\mathsf{poly}(\lambda, d(\lambda), \log k)$.

Completeness. Follows immediately from the completeness of the underlying seBARG and the opening completeness of the underlying HT family.

Efficiency. To bound $|\mathsf{crs}_{\mathcal{P}}|, |\mathsf{crs}_{\mathcal{V}}|, |\pi|$, we make use of the following facts:

- By definition of λ', $|\mathsf{hk}_{\mathsf{HT}}| \leq \mathsf{poly}(\lambda') = \mathsf{poly}(\lambda, d(\lambda))$.
- The length of the $\mathsf{BatchIndexTMSAT}$ instance X is $|X| \leq |C| + kn + \mathsf{poly}(\lambda, d(\lambda))$. The length of a witness ω_i is at most $m + \mathsf{poly}(d(\lambda))$.
- Therefore, by the succinctness of seBARG, we conclude that $|\mathsf{crs}_{\mathsf{seBARG}}| + |\pi_{\mathsf{seBARG}}| \leq m \cdot \mathsf{poly}(\lambda, d(\lambda), \log(knm))$.

To bound the running time of the verifier, we again invoke the efficiency of the seBARG to conclude the claimed $\mathsf{poly}(|\mathsf{crs}_{\mathcal{V}}| + |\pi|) + (kn + |C|)\mathsf{poly}(\lambda, d(\lambda), \log k)$ bound.

Soundness. Let $T = T(\lambda)$ be a subexponential function. Fix any size T cheating prover $\mathcal{P}^*$, any two polynomials $k = k(\lambda)$ and $n = n(\lambda)$, and any instance $\mathbf{x} = (x_1, \ldots, x_k) \notin \mathcal{L}_f^{(k)}$ such that $|x_j| = n$ for every $j \in [k]$.

Let EXP be the experiment of the soundness requirement:

$$\left\{ \begin{array}{l} (\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \leftarrow \mathsf{Gen}(1^\lambda, k, n) \\ \pi = \mathcal{P}^*(\mathsf{crs}_{\mathcal{P}}, \mathsf{crs}_{\mathcal{V}}) \end{array} \right\}.$$

Define

$$\epsilon(T(\lambda)) \triangleq \Pr_{\mathsf{EXP}} \left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \right],$$

and assume towards contradiction that ϵ is non-negligible.

We prove soundness using hybrid experiments $(\mathsf{EXP}_{j_1,j_2})_{1 \leq j_1, j_2 \leq N}$. The experiment EXP_{j_1,j_2} is identical to EXP, except for the following changes:

- In Gen, sample $(\mathsf{crs}_{\mathsf{seBARG}}, \mathsf{td}_{\mathsf{seBARG}}) \leftarrow \mathsf{Gen}_{\mathsf{seBARG}}(1^{\lambda'}, 1^{n'}, 1^{m'}, \{j_1, j_2\})$.
- Extract $\omega_{j_1}^{(1)}, \omega_{j_2}^{(2)} \leftarrow \mathsf{Extract}_{\mathsf{seBARG}}(\mathsf{td}_{\mathsf{seBARG}}, \pi_{\mathsf{seBARG}})$, where we parse $\pi = (\mathsf{rt}, \pi_{\mathsf{seBARG}})$.
- For $\alpha \in \{1, 2\}$, parse:
  1. If $1 \leq j \leq k$, parse $\omega_{j_\alpha}^{(\alpha)} = (w_{j_\alpha}^{(\alpha)}, b_{j_\alpha}^{(\alpha)}, \rho_{j_\alpha}^{(\alpha)})$.
  2. If $j > k$, parse $\omega_{j_\alpha}^{(\alpha)} = (b_{j_\alpha}^{(\alpha)}, b_{j_{\alpha,1}}^{(\alpha)}, b_{j_{\alpha,2}}^{(\alpha)}, \rho_{j_\alpha}^{(\alpha)}, \rho_{j_{\alpha,1}}^{(\alpha)}, \rho_{j_{\alpha,2}}^{(\alpha)})$.

For ease of notation, if $j_1 = j_2 = j$, we use the notation EXP_j instead of $\mathsf{EXP}_{j,j}$, and skip the superscript (1) when referring to $\omega_j^{(1)}$ and the values parsed from it.

We define $(b_1^*, \ldots, b_k^*) \in \{0,1\}^k$ where for every $j \in [k]$ we have $b_j^* = 1$ if and only if $x_j \in \mathcal{L}$, and extend them to $(b_1^*, \ldots, b_N^*)$, the values of all the wires in C on input $(b_1^*, \ldots, b_k^*)$.

Lemma 8.2. *For every $0 \leq i \leq d(\lambda)$, there exists a wire j in the ith layer of C such that for every $\lambda \in \mathbb{N}$,*

$$\Pr_{\mathsf{EXP}_j} \left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge \ b_j > b_j^* \ \right] \geq \frac{\epsilon(T(\lambda))}{3^{d(\lambda)-i+1}}.$$

Before proving Lemma 8.2, we first argue that the lemma indeed implies soundness. For $i = 0$, the lemma implies there exists an input wire $j \in [k]$ such that

$$\Pr_{\mathsf{EXP}_j}\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge\ b_j > b_j^*\ \right] \geq \frac{\epsilon(T(\lambda))}{3^{d(\lambda)+1}}.$$

Let $z = (C, x_1, \ldots, x_k, \mathsf{hk}_{\mathsf{HT}}, \mathsf{rt})$ and let M defined as in $\mathcal{P}$. By the somewhere argument of knowledge property of the underlying seBARG, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$,

$$\Pr_{\mathsf{EXP}_j}\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge\ b_j > b_j^* \ \wedge\ M(z, j, \omega_j) = 1\ \right] \geq \frac{\epsilon(T(\lambda))}{3^{d(\lambda)+1}} - \mathsf{negl}\left(T(\lambda), 2^{d(\lambda)}\right),$$

which is non-zero since ϵ is non-negligible. This is a contradiction, since $M(z, j, \omega_j) = 1$ implies that $\mathcal{R}_{\mathcal{L}}(x_j, w_j) = b_j = 1$, but $b_j^* = 0$ implies $x_j \notin \mathcal{L}$.

Proof of Lemma 8.2. We first prove the base case of $i = d$. Let $j = N$, by the index hiding and somewhere argument of knowledge properties of the underlying seBARG we have

$$\Pr_{\mathsf{EXP}_N}\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge\ M(z, N, \omega_N) = 1\ \right] \geq \epsilon(T(\lambda)) - \mathsf{negl}\left(T(\lambda), 2^{d(\lambda)}\right) \geq \frac{\epsilon(T(\lambda))}{3}\ .$$

This implies the lemma for $i = d$, since $M(z, N, \omega_N) = 1$ implies by definition $b_N = 1$, and we have $b_N^* = 0$ since $\vec{x} \notin \mathcal{L}_f^{(k)}$.

Assume by induction that the lemma holds for $i + 1$, and let j in the $i + 1$th layer of C given by the lemma. By the somewhere argument of knowledge property of the underlying seBARG and the induction hypothesis, we have

$$\Pr_{\mathsf{EXP}_j}\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge\ b_j > b_j^* \ \wedge\ M(z, j, \omega_j) = 1\ \right] \geq \frac{\epsilon(T(\lambda))}{3^{d(\lambda)-i}} - \mathsf{negl}\left(T(\lambda), 2^{d(\lambda)}\right).$$

We now analyze the jth gate. We know that $M(z, j, \omega_j) = 1$ implies $b_j = c_j(b_{j_1}, b_{j_2})$. Moreover, if we also have $b_j > b_j^*$, we get $b_{j_\alpha} > b_{j_\alpha}^*$ for some $\alpha \in \{1, 2\}$. Therefore, there exists an $\alpha \in \{1, 2\}$ for which we have

$$\Pr_{\mathsf{EXP}_j}\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge\ b_{j_\alpha} > b_{j_\alpha}^*\ \right] \geq \frac{\epsilon(T(\lambda))}{2 \cdot 3^{d(\lambda)-i}} - \mathsf{negl}\left(T(\lambda), 2^{d(\lambda)}\right). \tag{5}$$

It remains to prove the following claim, which shows that the lemma holds for j_α, and thus finishes the inductive step.

Claim 8.3.

$$\Pr_{\mathsf{EXP}_{j_\alpha}}\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge\ b_{j_\alpha} > b_{j_\alpha}^*\ \right] \geq \frac{\epsilon(T(\lambda))}{3^{d(\lambda)-i+1}}\ .$$

Proof of Claim 8.3. Recall that EXP_j is a shorthand for $\mathsf{EXP}_{j,j}$, and rewrite Eq. (5) as follows:

$$\Pr_{\mathsf{EXP}_{j,j}}\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}}, \mathbf{x}, \pi) = 1 \ \wedge\ b_{j_\alpha}^{(1)} > b_{j_\alpha}^*\ \right] \geq \frac{\epsilon(T(\lambda))}{2 \cdot 3^{d(\lambda)-i}} - \mathsf{negl}\left(T(\lambda), 2^{d(\lambda)}\right).$$

We first switch to the hybrid experiment $\mathsf{EXP}_{j,j_\alpha}$. By the index hiding of seBARG, we have

$$\Pr_{\mathsf{EXP}_{j,j_\alpha}}\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}},\mathbf{x},\pi)=1\ \wedge\ b^{(1)}_{j_\alpha}>b^*_{j_\alpha}\ \right]\geq\frac{\epsilon(T(\lambda))}{2\cdot 3^{d(\lambda)-i}}-\mathsf{negl}\left(T(\lambda),2^{d(\lambda)}\right).$$

Let $\mathsf{Correct}_M$ denote the event that $M(z,j,\omega^{(1)}_j)=1$ and $M(z,j_\alpha,\omega^{(2)}_{j_\alpha})=1$. By the somewhere argument of knowledge property of the seBARG, we have that in the above experiment, $\mathsf{Correct}_M$ holds except with negligible probability.

Now, observe that $M(z,j,\omega^{(1)}_j)=1$ implies that $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}},\mathsf{rt},j_\alpha,b^{(1)}_{j_\alpha},\rho^{(1)}_{j_\alpha})=1$, whereas $M(z,j_\alpha,\omega^{(2)}_{j_\alpha})=1$ implies $\mathsf{Verify}_{\mathsf{HT}}(\mathsf{hk}_{\mathsf{HT}},\mathsf{rt},j_\alpha,b^{(2)}_{j_\alpha},\rho^{(2)}_{j_\alpha})=1$. By the collision resistance wrt opening property of the HT family, we get that except with negligible probability, $b^{(1)}_{j_\alpha}=b^{(2)}_{j_\alpha}$. Therefore,

$$\Pr_{\mathsf{EXP}_{j,j_\alpha}}\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}},\mathbf{x},\pi)=1\ \wedge\ b^{(2)}_{j_\alpha}>b^*_{j_\alpha}\ \right]\geq\frac{\epsilon(T(\lambda))}{2\cdot 3^{d(\lambda)-i}}-\mathsf{negl}\left(T(\lambda),2^{d(\lambda)}\right).$$

Finally, we switch to the experiment $\mathsf{EXP}_{j_\alpha,j_\alpha}$. By the index hiding of seBARG,

$$\Pr_{\mathsf{EXP}_{j_\alpha,j_\alpha}}\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}},\mathbf{x},\pi)=1\ \wedge\ b^{(2)}_{j_\alpha}>b^*_{j_\alpha}\ \right]\geq\frac{\epsilon(T(\lambda))}{2\cdot 3^{d(\lambda)-i}}-\mathsf{negl}\left(T(\lambda),2^{d(\lambda)}\right).$$

Applying the same argument using somewhere argument of knowledge and collision resistance wrt opening, we get that in the above experiment we also have $b^{(1)}_{j_\alpha}>b^*_{j_\alpha}$ except with negligible probability, which implies

$$\Pr_{\mathsf{EXP}_{j_\alpha}}\left[\ \mathcal{V}(\mathsf{crs}_{\mathcal{V}},\mathbf{x},\pi)=1\ \wedge\ b_{j_\alpha}>b^*_{j_\alpha}\ \right]\geq\frac{\epsilon(T(\lambda))}{2\cdot 3^{d(\lambda)-i}}-\mathsf{negl}\left(T(\lambda),2^{d(\lambda)}\right)\geq\frac{\epsilon(T(\lambda))}{3^{d(\lambda)-i+1}}\ .$$

This concludes the proof of Claim 8.3, Lemma 8.2 and Theorem 8.1.

Acknowledgements. Zvika Brakerski is supported by the Israel Science Foundation (Grant No. 3426/21), and by the European Union Horizon 2020 Research and Innovation Program via ERC Project REACT (Grant 756482).

Maya Farber Brodsky is supported by an ISF grant 1789/19.

Yael Tauman Kalai is supported by DARPA under Agreement No. HR00112020023. 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.

Alex Lombardi was supported in part by a Simons-Berkeley postdoctoral fellowship, and in part by DARPA under Agreement No. HR00112020023.

Omer Paneth is a member of the Checkpoint Institute of Information Security and is supported by an Azrieli Faculty Fellowship, and ISF grant 1789/19.

References

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001. 1

[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, *49th ACM STOC*, pages 474–482. ACM Press, June 2017. 1, 2

[BK18] Zvika Brakerski and Yael Tauman Kalai. Monotone batch np-delegation with applications to access control. *IACR Cryptol. ePrint Arch.*, 2018:375, 2018. May 2018 version: https://eprint.iacr.org/archive/2018/375/20180513:062615. 5

[BKK+18] Saikrishna Badrinarayanan, Yael Tauman Kalai, Dakshita Khurana, Amit Sahai, and Daniel Wichs. Succinct delegation for low-space non-deterministic computation. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th ACM STOC*, pages 709–721. ACM Press, June 2018. 1, 8

[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. 1

[CGJ+22] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and snargs from sub-exponential ddh. *Cryptology ePrint Archive*, 2022. 5, 16, 17

[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, 5, 6, 9, 15, 16, 17

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 394–423, Virtual Event, August 2021. Springer, Heidelberg. 1, 5

[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

[DLN+04] Cynthia Dwork, Michael Langberg, Moni Naor, Kobbi Nissim, and Omer Reingold. Succinct proofs for np and spooky interactions. *Unpublished manuscript, available at http://www. cs. bgu. ac. il/˜ kobbi/papers/spooky_ sub_crypto. pdf*, 2004. 2

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013. 1

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013. 1

[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

[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. 5, 9

[JJ22] Abhishek Jain and Zhengzhong Jin. Indistinguishability obfuscation via mathematical proofs of equivalence. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1023–1034, 2022. 1

[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

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. 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 60–73. ACM, 2021. 1

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992. 1

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1545–1552. ACM, 2023. 2, 5, 15, 16, 17

[KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 91–118. Springer, Heidelberg, October / November 2016. 1, 2

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1115–1124. ACM Press, June 2019. 1, 7

[KR09] Yael Tauman Kalai and Ran Raz. Probabilistically checkable arguments. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 143–159. Springer, Heidelberg, August 2009. 1

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 565–574. ACM Press, June 2013. 4, 8

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In David B. Shmoys, editor, *46th ACM STOC*, pages 485–494. ACM Press, May / June 2014. 1, 2, 4, 8

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 330–368. Springer, Heidelberg, November 2021. 1, 6, 15

[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. 13, 14

[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. 5

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *Proceedings of FOCS 2022*, 2022. 2

[PR17] Omer Paneth and Guy N. Rothblum. On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 283–315. Springer, Heidelberg, November 2017. 1

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014. 1, 4

[WW22] Brent Waters and David J. Wu. Batch arguments for sfNP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 433–463. Springer, Heidelberg, August 2022. 1, 5, 16, 17