Boosting SNARKs and Rate-1 Barrier in Arguments of Knowledge

Jiaqi Cheng
UW-Madison*

Rishab Goyal
UW-Madison†

Abstract

We design a generic compiler to boost any non-trivial succinct non-interactive argument of knowledge (SNARK) to full succinctness. Our results come in two flavors:

1. For any constant $\epsilon > 0$, any SNARK with proof size $|\pi| < \frac{|\omega|}{\lambda^\epsilon} + \mathsf{poly}(\lambda, |x|)$ can be upgraded to a fully succinct SNARK, where all system parameters (such as proof/CRS sizes and setup/verifier run-times) grow as fixed polynomials in λ, independent of witness size.
2. Under an additional assumption that the underlying SNARK has as an *efficient* knowledge extractor, we further improve our result to upgrade any non-trivial SNARK. For example, we show how to design fully succinct SNARKs from SNARKs with proofs of length $|\omega| - \Omega(\lambda)$, or $\frac{|\omega|}{1+\epsilon} + \mathsf{poly}(\lambda, |x|)$, any constant $\epsilon > 0$.

Our result reduces the long-standing challenge of designing fully succinct SNARKs to designing *arguments of knowledge that beat the trivial construction*. It also establishes optimality of rate-1 arguments of knowledge (such as NIZKs [Gentry-Groth-Ishai-Peikert-Sahai-Smith; JoC'15] and BARGs [Devadas-Goyal-Kalai-Vaikuntanathan, Paneth-Pass; FOCS'22]), and suggests any further improvement is tantamount to designing fully succinct SNARKs, thus requires bypassing established black-box barriers [Gentry-Wichs; STOC'11].

1 Introduction

Succinct non-interactive arguments, commonly called SNARGs, are a central object in both theory and practice of numerous cryptographic systems today. Informally speaking, a SNARG for an **NP** language $\mathcal{L}$ is an argument[1] system that creates a *short* membership proof π for any instance $x \in \mathcal{L}$, given a corresponding **NP** witness ω. In the study of SNARGs, a highly desirable goal is to achieve *full succinctness* for non-trivial **NP** languages. That is, design membership proofs that are of fixed polynomial size and can be verified in fixed polynomial time, independent of witness size, $|\omega|$, and the time taken to verify $x \in \mathcal{L}$ given ω. Applications of SNARGs are well spread across the literature [WB15] and even extend to popular systems such as blockchains [BCG+14].

Since early 90s, starting with seminal works of Kilian [Kil92] and Micali [Mic00], we have known that SNARGs are indeed realizable for **NP** although in the idealized random oracle model [BR93]. Over the last decade, many more succinct SNARG schemes have been designed [BCCT12, BCCT13,

*Email: jiaqicheng@cs.wisc.edu.
†Email: rishab@cs.wisc.edu. Support for this research was provided by OVCRGE at UW-Madison with funding from the Wisconsin Alumni Research Foundation.
[1]In an argument system [BCC88], soundness is only guaranteed against polynomial-time cheating provers.

Gro10, SW14, Gro16, BCI+13, BCC+17], however these rely on non-falsifiable assumptions [Nao03][2]. A longstanding open problem in this area has been to design SNARGs for **NP** in the standard model while reducing security to standard and falsifiable cryptographic assumptions. As it currently stands, only a handful constructions [BHK17, KPY19, BKK+18, JKKZ21, CJJ22, KLVW23, BBK+23, NWW23, JKLV24] for SNARGs have been designed from falsifiable assumptions in the standard model, despite over thirty years of research. All these constructions support proving membership only for a subclass of **NP**.

The reason for such a huge disparity between provable SNARG constructions from non-falsifiable and standard cryptographic assumptions was studied in the seminal work of Gentry-Wichs [GW11]. The Gentry-Wichs result states that we cannot design SNARGs for all of **NP**, while reducing their security to any falsifiable assumption via a polynomial-time "black-box reduction".

In a black-box reduction, the reduction algorithm is only allowed to make "oracle" queries to a successful attacker, and it must break the underlying assumption without any additional description about the attacker. That is, a black-box reduction can not access the description of an attacker's code, but only gets to see input/output behavior. In a non-black-box reduction, however, the reduction gets full access to the attacker, and can use the attacker's code arbitrarily. Given most security proofs in modern cryptography typically only make black-box use of an attacker, thus this explains why SNARGs for **NP** have been such an elusive target.

Gentry-Wichs also proved that designing *slightly-succinct*[3] SNARGs also suffers from the same black-box reduction barriers. In this work, we study the following question:

Is bypassing this barrier for *slightly-succinct* SNARGs enough to bypass the barrier for *fully-succinct* SNARGs, the 'gold-standard' succinctness?

In other words, we do not know whether designing fully-succinct SNARGs is much harder than designing slightly-succinct SNARGs. It is conceivable that there is a hierarchy of black-box barriers between SNARGs with differing levels of succinctness.

This work. We answer the above question affirmatively. We design a new bootstrapping compiler to upgrade any slightly-succinct SNARG into a fully-succinct SNARG, as long as the SNARG satisfies knowledge soundness (i.e., it is a SNARK).

Throughout the sequel, when we write the SNARK proof system is δ-mild (for any $\delta > 1$), then we mean that the proof size is δ factor smaller than the witness length (i.e., $|\pi| < \frac{|\omega|}{\delta} + \mathsf{poly}(\lambda, |x|)$). Moreover, we do **not** put any other succinctness/efficiency requirements on any other component of the proof system. For example, the CRS size could grow polynomially with the witness length or the circuit size of membership circuit. Concretely, we show:

Theorem 1.1 (Informal, see Corollary 6.5)**.** For any constant $\epsilon > 0$. Assuming RAM delegation, any λ^{ϵ}-mild SNARK for **NP** can be turned into a fully-succinct SNARK for **NP**.

Moreover, if we additionally assume that the underlying SNARK has a *fast extractor*, then we can perform bootstrap even 'milder' SNARKs. We say a δ-mild SNARK has a fast extractor, if

[2]The SNARG construction by Sahai-Waters [SW14] relies on indistinguishability obfuscation [BGI+01], which is also a non-falsifiable assumption [GK16]. Although, obfuscation is known to be instantiable from $2^{|x|}$-hardness of standard cryptographic hardness assumptions [JLS21]

[3]Briefly, *slight succinctness* states the proof size is sublinear in the statement and witness length (i.e., $o(|x| + |\omega|)$).

the extractor's running time is at most a poly(δ) multiplicative factor larger than the adversary's running time. E.g., if δ is a constant, then the fast extractor's running time is at most a constant times more than adversary's running time. Concretely, we show.

Theorem 1.2 (Informal, see Corollary 6.6)**.** For any $\delta > 1$. Assuming RAM delegation, any δ-mild SNARK for **NP** with *fast extractors* can be turned into a fully-succinct SNARK for **NP**.

Combining above theorems with great recent progress on designing RAM delegation [CJJ21, CJJ22, HJKS22, WW22, KLVW23], we obtain:

Corollary 1.3 (Informal)**.** Assuming either LWE, or DLIN, or sub-exponential DDH (and QR), any δ-mild SNARK for **NP** can be boosted to a fully-succinct SNARK for **NP**, as long as $\delta = \lambda^{\epsilon}$, or it has a fast extractor.

Discussion and interpreting our results. The above suggests that designing any non-trivial SNARG for **NP** with knowledge soundness is sufficient to design a fully-succinct SNARG for **NP** with knowledge soundness. Thus, a natural interpretation is that overcoming the Gentry-Wichs barrier for just non-trivial succinctness is enough to settle the fully-succinct SNARK problem, and there are no further technical barriers beyond designing non-trivial SNARKs.

Our result can also be viewed as a mild strengthening of Gentry-Wichs. As, by combining our results with [GW11], we can rule out existence of non-trivial SNARKs with poly-time black-box reduction to falsifiable assumptions. Currently, [GW11] only rules out *slightly-succinct* arguments, i.e. $|\pi| = o(|x| + |\omega|)$, but we can rule out any $|\pi| = |\omega| - \Omega(\lambda)$ with fast extractors.

Another interpretation is that known constructions for various rate-1 arguments of knowledge (such as zero-knowledge, or batch arguments) are essentially optimal, and any further improvement is tantamount to designing fully-succinct SNARKs, thus bypassing [GW11]. That is, rate-1 NIZK [GGI+15] and rate-1 (se)BARGs [DGKV22, PP22] are best possible arguments of knowledge, with a polynomial-time black-box reduction to falsifiable assumptions.

Finally, we remark that while our approach relies on a careful recursive extraction technique, our security reduction is oblivious to black-box or non-black-box nature of mild SNARK extractor. Although Campanelli et al. [CGKS23] proved impossibility of black-box extraction for adaptive knowledge soundness in the standard model, black-box extraction is possible in non-standard models. Thus, if a mild SNARK extractor needs black-box/non-black-box access to the cheating prover in the standard/non-standard model, then so does our fully-succinct SNARK extractor.

2 Technical Overview

A SNARK is a non-interactive proof system that allows a prover to convince the validity of a statement x to a verifier, by providing a proof π whose length is shorter than the witness length, $|\omega|$. They are typically defined in the common reference string model. Given the crs, a prover processes a pair of instance-witness pair (x, ω) to produce a short proof π. The verifier, on input crs, x, and π, outputs a bit to signal validity of proof. The standard notion of soundness states that a (polynomial-time) cheating prover, given crs, cannot find a instance-proof pair (x, π) such that the verifier accepts, yet x is invalid. In this work, we rely on the stronger knowledge soundness

property. It states that there exists an efficient extractor $\mathcal{E}$ that can extract a valid witness ω from a cheating prover, given the instance x, proof π, and the cheating prover's algorithm $\mathcal{P}$.

Defining succintness in SNARKs. If we allow proof size to be as large as the witness, then a SNARK can be trivially designed. Simply, set the proof $\pi = \omega$. Thus, succinctness is an essential property for SNARKs. To capture a meaningful notion of succinctness, we define the notion of δ-mildness to capture any non-trivial SNARK. We consider $\delta = \delta(\lambda, n, m)$ to be a function of security parameter λ, instance length $|x| = n$, and witness length $|\omega| = m$. In words, the proofs generated by a δ-mild SNARK are (aymptotically) δ-multiplicative-factor shorter than the witness length, m. Formally, we say a SNARK is δ-mild, iff it satisfies the following:

$$|\pi| = \frac{m}{\delta} + \mathsf{poly}(\lambda, n).$$

Note that any non-trivial SNARK is a δ-mild SNARK, for some $\delta > 1$. Similarly, a *fully-succinct* SNARK corresponds to a m-mild SNARK (i.e., $\delta = m$)[4] with an additional property that $|\mathsf{crs}|$ is also a fixed polynomial in λ, and does not scale with n or m.

In words, we define δ-mild SNARKs to capture the most general notion of non-trivial SNARKs, as we only require their proof size to be non-trivially smaller than witness length, but do not put any constraints on the CRS size or verification runtime, etc. The rest of the technical overview is divided into three parts, covering all our main results:

1. Assuming RAM delegation, any δ-mild SNARK can be upgraded to have short CRS.
2. Any λ^ϵ-mild SNARK with short CRS can be boosted to a fully-succinct SNARK for RAM, for any constant $\epsilon > 0$.
3. Any δ-mild SNARK with γ-fast extractors can be boosted to λ^ϵ-mild SNARK, as long as $\delta > 1$ and $\gamma = \delta^c$ for some constant $c > 0$.

2.1 Shortening CRS in SNARKs via RAM Delegation

The first step of our generic compiler is to design a mild SNARK with a short CRS, where its size does not grow with the classical **NP** verification but only instance and witness lengths. As we will see in the next stages of our compiler, starting with mild SNARKs with short CRS is extremely useful for iterative compositions. Towards the goal of shortening CRS, our main observation is that it suffices to compose a mild SNARK (with an extremely long CRS) with a fully-succinct SNARG for **P**.

A SNARG for **P**, more commonly regarded as RAM delegation [KP16, BHK17, CJJ22], is a proof system that lets a prover generate short proofs of correctness for deterministic computation. Thus, syntactically, they are defined same as general SNARK(G)s, except the witness is always empty. That is, a prover only takes crs and x as inputs, and to verify π, a verifier needs both x and crs. The special property of (fully-succinct) RAM delegation is that $|\mathsf{crs}|, |\pi|$ and verification time grows only poly-logarithmically with T. Here T is the time taken to run the RAM machine

[4]Technically, a SNARK is fully-succinct if $|\pi| = \mathsf{poly}(\lambda)$, and does not grow with n. However, by the folklore hash-and-prove paradigm, a SNARK with $|\pi| = \mathsf{poly}(\lambda, n)$ can be easily turned into SNARK with $|\pi| = \mathsf{poly}(\lambda)$.

(i.e., the membership circuit) on x. Clearly, knowledge soundness and standard soundness notions are the same for RAM delegation, since the witness is empty.

The idea for composing mild SNARKs with RAM delegation is very simple. Given any δ-mild SNARK that lacks a short CRS, rather than asking the prover to prove validity of x by using ω as a witness, ask the prover to prove validity of x by using ω and $\mathsf{del}.\pi$ as a witness. That is, a prover first computes a RAM proof $\mathsf{del}.\pi$ as $\mathsf{del}.\pi \leftarrow \mathsf{del}.\mathsf{Prove}(\mathsf{del}.\mathsf{crs}, (x, \omega))$. Next, it creates a mild SNARK proof π for instance x to prove that it knows $(\omega, \mathsf{del}.\pi)$ such that $\mathsf{del}.\mathsf{Verify}(\mathsf{del}.\mathsf{crs}, (x, \omega), \mathsf{del}.\pi) = 1$. By relying on succinctness of RAM delegation, we can improve the CRS size for δ-mild SNARK from $\mathsf{poly}(\lambda, |\mathcal{C}|, n, m)$ to $\mathsf{poly}(\lambda, \log |\mathcal{C}|, n, m)$, where $\mathcal{C}$ is the membership circuit for language associated with x. This is because the CRS size and verification time for RAM delegation only grows poly-logarithmically with $|\mathcal{C}|$. Moreover, this transformation preserves knowledge soundness as well as δ-mildness of underlying SNARKs. For completeness, we provide this formally in Section 6.

An exciting line of recent works [CJJ21, CJJ22, KVZ21, HJKS22, WW22, DGKV22, PP22, KLVW23] have designed RAM delegation from a variety of standard falsifiable assumptions such as LWE, k-LIN, or sub-exponential DDH (and QR). Morover, Kalai et al. [KLVW23] established a fantastic result about boosting any non-trivial RAM delegation (with fast verification) to fully-succinct RAM delegation, under the existence of rate-1 string OT (known under many standard assumptions). Thus, we view short CRS as a very mild assumption for non-trivial SNARKs.

2.2 Fully-succinct SNARKs from λ^ϵ-mild SNARKs

The next step of our work is to design a generic compiler for boosting any λ^ϵ-mild SNARK to a fully-succinct SNARK, for any constant $\epsilon > 0$. Given that a mild-SNARK can be viewed as a mechanism to compress a witness ω into a verifiable encoding π, a natural idea is to iteratively compose δ-mild SNARKs many times until we reach desired compression.

Specifically, let $\mathcal{C}$ be the circuit that verifies the **NP** language $\mathcal{L}$ for which we want to design a SNARK, and $n, m \leq 2^\lambda$ be the corresponding instance-witness lengths. The process begins by initializing $\mathcal{C}_0$ as the circuit $\mathcal{C}$ for the language $\mathcal{L}$, and sampling crs_0 as the CRS for the mild SNARK corresponding to the circuit $\mathcal{C}_0$. Next, we (iteratively) sample $\ell = \frac{\lambda}{\epsilon \log \lambda}$ many CRS $\mathsf{crs}_1, \ldots, \mathsf{crs}_\ell$ for the (iteratively) defined languages $\mathcal{L}_1, \ldots, \mathcal{L}_\ell$, where the circuit $\mathcal{C}_{i+1}$ (corresponding to $\mathcal{L}_{i+1}$) is set as the verification circuit for the mild SNARK corresponding to $\mathcal{L}_i$. That is, $\mathcal{C}_{i+1}$ has crs_i hardwired, and it accepts an instance x if there exists a mild SNARK proof π_i such that (x, π_i) is a valid proof-witness pair w.r.t. crs_i. Given $\mathsf{crs}_1, \ldots, \mathsf{crs}_\ell$, a prover can create a fully-succinct proof by iteratively running the mild SNARK prover as follows – first, generate π_0 as a SNARK proof for x under crs_0 using ω as witness; next, generate π_1 as a SNARK proof for x under crs_1 using π_0 as witness; and so on, until the proof size is below a fixed $\mathsf{poly}(\lambda)$ threshold.

Whenever $\delta = \lambda^\epsilon$, then after about $\frac{\log m}{\epsilon \log \lambda}$ iterative applications of a mild SNARK prover, we obtain a proof $\pi_{\frac{\log m}{\epsilon \log \lambda}}$ of size at most $\mathsf{poly}(\lambda, n)$. This achieves the desired goal of full-succinctness. Unfortunately, the above template doesn't work!

CRS generation takes too long! Recall that a δ-mild SNARK only guarantees that the proof size, $|\pi|$, is smaller than m, but it does not say anything about other parameters or algorithms. Thus, it is possible that CRS size and/or the verifier's running time is greater than $|\omega|$, or the time

taken to check $\mathcal{C}(x, \omega) = 1$. In such scenarios, we cannot efficiently generate the full CRS. This is because, under crs_{i+1}, the prover proves that it has a mild SNARK proof π_i for x w.r.t. crs_i. Thus, the $(i+1)^{th}$ language membership circuit (for crs_{i+1}) is the i^{th} verification circuit (w.r.t. to crs_i), and hence $|\mathsf{crs}_i|$ would grow with $|\mathcal{C}|, m, n$ and super-polynomially with i.

A shorter CRS helps? Recall that we already discussed how one can generically reduce the CRS size (as well as time needed to compute it) to only grow poly-logarithmically with $|\mathcal{C}|$. Isn't that enough to ensure the above template works? Unfortunately, the answer is no! This is because RAM delegation only brings down CRS size from $\mathsf{poly}(\lambda, |\mathcal{C}|, n, m)$ to $\mathsf{poly}(\lambda, \log |\mathcal{C}|, n, m)$. But, for full-succinctness, our goal is to have the CRS size (as well as verification time) to grow only poly-logarithmically with m. Thus, the question is how can we compose mild SNARKs such that verification time and CRS size does not grow with m or n?

Tree-based composition. Our idea is to switch an *iterative straightline composition* of SNARKs with an iterative, but *tree-based*, composition of SNARKs. Broadly speaking, the intuition is similar to how composing hash functions in a tree-based fashion [Mer88] is more efficient than a straightline iterated hashing approach [Mer79, Dam89].A similar issue was faced by Bitansky et al. [BCCT13] in the context of designing Proof Carrying Data (PCD) [CT10] and complexity-preserving SNARKs. Although Bitansky et al. assumed existence of a *fully-succinct* SNARK (which is what we want to design here), the underlying technical approach is more general. Our intuition is that a similar approach is quite meaningful for boosting non-trivial SNARKs.

In order to correctly implement such an idea, we need to break down the entire computation needed to verify $x \in \mathcal{L}$, given ω, into smaller pieces. Otherwise, tree-based composition will not give desired amount of succinctness/compression. Basically, by interpreting the entire computation of $\mathcal{C}(x, \omega) = 1$ as a step-by-step computation in the RAM model, we can ensure that the size of computation for which we will use mild SNARKs is of fixed size. That is, let $\mathsf{cnfg}_1, \ldots, \mathsf{cnfg}_T$ be the configuration of the RAM machine, corresponding to $\mathcal{C}(x, \omega) = 1$, where $T = \mathsf{poly}(\lambda)$ is the total running time. (A RAM machine configuration corresponds to the RAM memory, program state, and machine head at given time step.)

A prover runs a mild SNARK prover, at the base level of the tree, to prove that each step of the computation is correctly performed, i.e. $\mathsf{cnfg}_i \rightarrow \mathsf{cnfg}_{i+1}$ (here by '$\rightarrow$' we denote one RAM computation step). Further, it accumulates these proofs iteratively (as we go up the tree) by running a mild SNARK prover to prove that two configurations $\mathsf{cnfg}_i, \mathsf{cnfg}_j$ are consistent. By consistent, we mean that we can go from $\mathsf{cnfg}_i \rightarrow \mathsf{cnfg}_j$ after $(j - i)$ RAM computation steps. The intuition is that base level (i.e., level 0) we prove the configurations to be adjacent (i.e., 1 step apart), while for next level (i.e., level 1) we prove $\mathsf{cnfg}_i, \mathsf{cnfg}_j$ are 2 steps apart, and so on. Thus, at level i, we prove the configurations to be 2^i steps apart. By continuously composing SNARKs for such step-by-step computations in a tree-based fashion, we can get around the succinctness problem. This is because the instance and witness size, for each mild SNARK proof computation, are always of fixed polynomial size, independent of n, m, T as well as the level i. Thus, the CRS size doesn't grow anymore, and the resulting SNARK will be fully-succinct.

How to extract the witness in polynomial time? Unfortunately, the above process brings up a major bottleneck. The problem is how to prove (knowledge) soundness of the above design? Soundness only states that π_0 (proof at the base level) is hard to compute if $x \notin \mathcal{L}$. But, it does not

say any of the iteratively computed proofs are hard to compute! Thus, we cannot rely on standard soundness, but need to use knowledge soundness. But this would mean we have to recursively run the extractor $\log T$ many times. Unfortunately, such a recursive extractor would not run in polynomial time, since the depth of the tree is $\log T = O(\log \lambda)$.

While the above feels a pretty big problem, this can be easily resolved by simply increasing the arity of the tree (from 2 to λ). That is, rather than proving the configurations to be 2^i steps apart (at level i), we will prove configurations to be λ^i steps apart. With this change, the depth of the SNARK tree will be constant, $\log_\lambda T = O(1)$ (for any polynomial time computation). Thus, we can perform recursive extraction from any accepting proof, given a poly-time cheating prover. With this modification, we provide a more detailed sketch.

Boosting via SNARK tree. From here on, we assume we have a description of a RAM program $\mathcal{R}$ that verifies validity of instance-witness pair (x, ω), rather than a fixed circuit $\mathcal{C}$. $\mathcal{R}$ checks that validity of witness for an instance w.r.t. $\mathcal{L}$.

The prover starts by computing all configurations for every computation step, and hashing down just the RAM memory portion for each step using a Merkle hash tree [Mer88]. At the bottom level, the instance at each node consists of two consecutive *pseudo*-configurations $(\rho, \mathsf{state}, i_{\mathsf{read}}), (\rho', \mathsf{state}', i'_{\mathsf{read}})$, where ρ is the memory digest, state is the program state, and i_{read} is the index of the bit read during that step (i.e., machine head location). We call this *pseudo*-configuration, because it only contains the digest of the memory instead of full memory. The witness includes the local read/write bit and the write index $b_{\mathsf{read}}, b_{\mathsf{write}}, i_{\mathsf{write}}$, which can be used to verify the transition between the two configurations. It also includes the Merkle tree openings $u_{\mathsf{read}}, u_{\mathsf{write}}$ for the read/write bits w.r.t. memory digest. Given the above, the language for the nodes at bottom level is as follows:

> **Instance:** $(\rho, \mathsf{state}, i_{\mathsf{read}}), (\rho', \mathsf{state}', i'_{\mathsf{read}})$.
> **Witness:** $b_{\mathsf{read}}, b_{\mathsf{write}}, i_{\mathsf{write}}, u_{\mathsf{read}}, u_{\mathsf{write}}$.
> Check if the transition is valid, and the hash openings u_{read} and u_{write}, along with the memory hash values ρ, ρ', the bits $b_{\mathsf{read}}, b_{\mathsf{write}}$, and memory indexes $i_{\mathsf{read}}, i_{\mathsf{write}}$ are all consistent.

At all other levels, each node aggregates λ nodes from the lower level (see Fig. 1). The instance contains just two pseudo-configurations: the initial and final pseudo-configuration out of the λ nodes. And, the witness includes all pseudo-configurations and their corresponding SNARK proofs of transitions. Basically, the prover proves that it knows a valid sequence of transitions between the two pseudo-configurations. Below, we describe the language for intermediate levels informally:

> **Instance:** $(\rho_1, \mathsf{state}_1, i_{1,\mathsf{read}}), (\rho_{\lambda+1}, \mathsf{state}_{\lambda+1}, i_{\lambda+1,\mathsf{read}})$.
> **Witness:** $\{\pi_j\}_{j\in[\lambda]}, \{(\rho_j, \mathsf{state}_j, i_{j,\mathsf{read}})\}_{j\in[\lambda+1]}$.
> Check if the mild SNARK verifier accepts every pair of instance $((\rho_j, \mathsf{state}_j, i_{j,\mathsf{read}}), (\rho_{j+1}, \mathsf{state}_{j+1}, i_{j+1,\mathsf{read}}))$ and proof π_j for $j \in [\lambda]$.

We highlight that all above languages have succinct description since the instance/witness sizes as well as size of the associated membership circuit for each language has a fixed polynomial size. Thus, this process can be iteratively carried out till the root of the tree. And, it ensures each

individual CRS as well as the final proof size to be fully-succinct, i.e. independent of n, m, T.

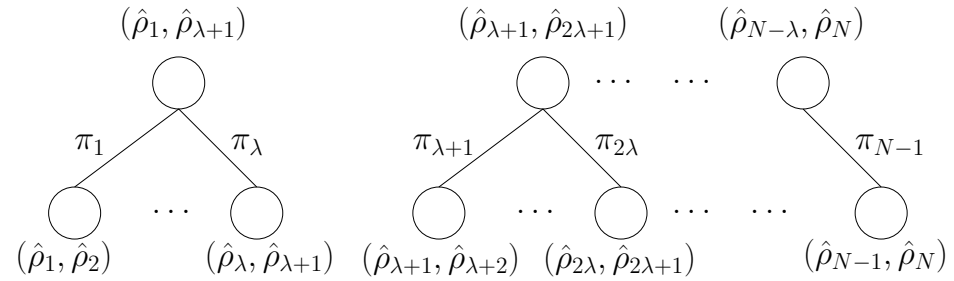

Figure 1: Level i and level $i+1$ of the SNARK tree, and $\hat{\rho}$ corresponds to a *pseudo*-configuration, $\hat{\rho} = (\rho, \mathsf{state}, i_{\mathsf{read}})$, and $N = T/\lambda^i$.

Lastly, we can design a recursive extractor efficiently to prove knowledge soundness. The extractor first extracts a valid witness for the root node, and then it considers the extraction procedure that computes a valid witness for the root node as an attacker, and continues the recursive extraction procedure. Overall, by combining all the witnesses at the bottom level (leaves of the tree), the extractor can reconstruct a valid witness that satisfies the RAM computation. (Technically, the security argument also relies on the fact that the Merkle tree is secure, but we ignore that detail for the sake of simplicity.) Clearly, the running time of extractor faces a polynomial blow-up across each level of the tree (as the i^{th} level extractor is treated as an attacker for $(i-1)^{th}$ level). However, as we discussed earlier, by designing trees with arity-λ, the height of the SNARK tree remains constant, and as a result, the extractor's runtime is polynomially bounded. Our compiler is formally presented in Section 4.

2.3 Boosting *milder* SNARKs

As a final step, we also show a generic compiler to boost any non-trivial SNARK, i.e., with any mildness parameter $\delta > 1$. However, unlike our previous compiler, we require an additional property from the underlying SNARK. To better understand this, let us first inspect what goes wrong if we simply use the previous compiler for such 'milder' SNARKs.

Consider $\delta = 1 + \epsilon$ for some constant $\epsilon > 0$. To boost such a SNARK, we need a SNARK tree of depth around $\log_{1+\epsilon} T = O(\log \lambda)$, where T is the running time of the (non-deterministic) computation. Recall this is exactly the scenario where the recursive extraction strategy fails! This is precisely whe we considered trees of larger arity, to ensure the tree depth is constant. Unfortunately, if the underlying SNARKs mildness is lower than λ^ϵ, then we can no longer guarantee that a constant number of iterative SNARK compositions will be enough for achieving full-succinctness.

Our intuition is that if the extractor runs 'sufficiently fast', then this issue goes away. Thus, to resolve this issue, we introduce a notion of δ-mild SNARKs *with fast extractors*. Roughly, it says that the ratio between the extractor's running time and the cheating prover's running time is asymptotically equal to $\mathsf{poly}(\delta)$. More formally, a δ-mild SNARK has a fast extractor if the extractor's runtime is $\mathsf{poly}(\delta) \cdot |\mathcal{A}| + \mathsf{poly}(\lambda, n, m, |\pi|)$. For such mild SNARKs, we can use our boosting compiler to design a fully-succinct SNARK by relying on $O(\log \lambda)$-depth SNARK tree.

Although one could simply the above approach, we provide an alternate approach. Basically, we show that any δ-mild SNARK, with a fast extractor $\mathcal{E}$, can be generically boosted to a λ-mild SNARK, with an extractor $\mathcal{E}'$ that runs in polynomial time. Very briefly, our approach is to compose such mild-SNARKs in a straightline iterated fashion. Note that this will be sufficient since our final goal is not to achieve a short CRS, but only a λ-mild SNARK. This can be further boosted to full-succinctness by using our previous compilers. We describe this formally in Section 5.

2.4 Related Work, Open Questions, and Roadmap

Related Work. Recently, Kalai et al. [KLVW23] designed a generic compiler to boost any non-trivial (flexible) RAM SNARGs to fully-succinct (flexible) RAM SNARGs, under a very mild assumption of rate-1 string OT. Moreover, they showed a near equivalence between such flexible RAM SNARGs and batch arguments (BARGs) for $\mathbf{NP}$. One can view their result as a generic compiler for boosting any non-trivial SNARK for $\mathbf{P}$ to a fully-succinct SNARK for $\mathbf{P}$[5]. Our work gives a matching boosting result for the non-deterministic class $\mathbf{NP}$.

Proof Carrying Data (PCD) [CT10] generalizes SNARGs to distributed computation. Their goal is for a group of participants in a distributed computation to create proofs of integrity and legitimacy for their respective (local) computations that can be combined to create a global proof of integrity and legitimacy of the entire distributed computation. PCDs are a significant generalization of Incrementally Verifiable Computations (IVC) [Val08]. IVCs can be simply viewed as PCD for straightline incremental computations (i.e., path graphs).

In a beautiful work, Bitansky et al. [BCCT13] developed new approaches for recursively composing SNARKs, and provided new techniques for constructing and using PCD. One of their central results was development of a bootstrapping mechanism for fully-succinct SNARKs with "expensive" preprocessing to design fully-succinct *complexity-preserving* SNARKs. By complexity-preserving, they meant that the prover's time/space complexity is essentially the same as that required for classical $\mathbf{NP}$ verification.

On a technical level, [BCCT13] relied on an elegant technique to recursively compose SNARKs over a tree-like structure. As discussed in the overview, our approach builds on their composition techniques. Apart from studying different problems, the key differences between our works are that we show that recursive composition works even for non-trivial SNARKs, while they relied on fully-succinct SNARKs. Moreover, a non-trivial SNARK verifier can be as large as some polynomial in the description of the classical $\mathbf{NP}$ verification circuit, thus we had to rely on additional techniques such as RAM delegation, while this was needed by Bitansky et al. since they assumed full-succinct SNARKs.

Interestingly, by combining our results with [BCCT13], we obtain that any δ-mild SNARK, with either a fast extractor or $\delta = \lambda^{\epsilon}$, is sufficient to design *complexity-preserving* SNARKs and PCDs.

Open questions. Our work leaves a lot of interesting questions for further research. We list out some of them below.

[5]Recall SNARKs and SNARGs are the same object for $\mathbf{P}$, since there is no witness.

1. Our appraoch crucially relies on knowledge soundness. A great problem is can we boost non-trivial SNARGs, without or with very weak extractability?
2. We also need SNARKs to be adaptively sound. Another great question is to similar compilers assuming non-adapative soundness.
3. To shorten the CRS size, we need to rely on efficient RAM delegation. Can this be avoided, or could we instead reduce to simpler assumptions such as CRHFs?
4. Do there exist such boosting SNARK results for other classes, other than $\mathbf{P}$ [KLVW23] and $\mathbf{NP}$ (this work)?
5. Our fully-succinct SNARKs have "fixed" multiplicative overhead in the time/space needed for computing the proof, compared to classical $\mathbf{NP}$ verification. Can we make this optimal and only incur a "fixed" additive overhead?

Answering above questions will give us more insight in the study of SNARKs and related proof systems.

Roadmap. The rest of the paper is organized as follows. First, we provide the necessary preliminaries in Section 3, and next, we describe our main boosting compiler for mild SNARKs (with short CRS) in Section 4. Later in Section 5, we show to boost even milder SNARKs, and finally, we describe how to use RAM delegation to shorten CRS in Section 6.

3 Preliminaries

3.1 RAM Machine

A RAM machine is modeled as a deterministic machine $\mathcal{R}$ which takes input $x \in \{0,1\}^n$, has random access to a large memory of size k upon which it can perform arbitrary reads and writes. It performs a sequence of T computation steps where in each step, it reads from a single memory cell, writes into a memory cell, and updates its local state. More concretely, the RAM machine $\mathcal{R}$ is represented as a local state, a large memory of length k: $M = (M_1, \ldots, M_k)$, and a state-transformation circuit which we denote by $\mathcal{C}^{\mathsf{RAM}}$: $\mathcal{C}^{\mathsf{RAM}}(\mathsf{state}, b_{\mathsf{read}}) \to (\mathsf{state}', i_{\mathsf{read}}, i_{\mathsf{write}}, b_{\mathsf{write}})$.

In the above definition, input consists of state and b_{read}, with state representing the state of machine $\mathcal{R}$, b_{read} denoting the bit read from memory. Output includes the $\mathsf{state}', i_{\mathsf{read}}, i_{\mathsf{write}}$, and b_{write}. state' is the updated state after transformation, i_{read} is the index of the memory cell to be read for state', i_{write} is the next index of the memory cell to write into, and b_{write} is the bit to write into that memory cell. Memory cell $M_{i_{\mathsf{write}}}$ is overwritten by bit b_{write}.

RAM machine $\mathcal{R}$ takes as input $x \in \{0,1\}^n$: The first n memory cells are initialized as string x, such that $M_i = x_i$ for all $i \in [n]$, and the remaining memory cells $\{M_i\}_{i \in \{n+1,\ldots,k\}}$ are deterministically initialized as 0 bits. The local state and transition circuit $\mathcal{C}^{\mathsf{RAM}}$ have fixed sizes.

3.2 Fully Succinct SNARK for RAM

Syntax. We define fully succinct SNARK for any RAM Machine.

$\mathsf{Setup}(1^\lambda, \mathcal{R}, n, m, T, k) \to \mathsf{crs}$. It takes as input security parameter λ, machine $\mathcal{R}$, running time bound T, instance length n, witness length m, and memory size k. It outputs crs.

$\mathsf{Prove}(\mathsf{crs}, x, \omega) \to \pi$. The prover takes as input crs, instance x, witness ω, and outputs a proof π.

$\mathsf{Verify}(\mathsf{crs}, x, \pi) \to \{0,1\}$. The verifier takes as input CRS crs, instance x, and proof π. It outputs a bit to signal whether the proof is valid.

Completeness For every $\lambda \in \mathbb{N}$, polynomial $n = n(\lambda)$, $m = m(\lambda)$, $T = T(\lambda)$, $k = k(\lambda)$, RAM Machine $\mathcal{R}$ of running time at most T, memory size k, instance $x \in \{0,1\}^n$ and witness $\omega \in \{0,1\}^m$ where $\mathcal{R}(x, \omega) = 1$, there exists a negligible function $\mathsf{negl}(\lambda)$ for which the following holds:

$$\Pr\left[\ \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \ \ : \ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}, 1^n, 1^m, T, k) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, \omega) \end{array}\ \right] \geq 1 - \mathsf{negl}(\lambda).$$

Succinctness The SNARK is fully-succinct, if there exists polynomials p_1, p_2, p_3, p_4 such that for all $\lambda \in \mathbb{N}$, polynomials $n = n(\lambda), m = m(\lambda), T = T(\lambda), k = k(\lambda)$, $x \in \{0,1\}^n$, $\omega \in \{0,1\}^m$, RAM machine $\mathcal{R}$ of running time at most T and memory size k, $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}, 1^n, 1^m, T, k)$, and $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, \omega)$, it holds that $|\mathsf{crs}| \leq p_1(\lambda, \log T)$, $|\pi| \leq p_2(\lambda, \log T)$, setup runtime is $\mathsf{poly}(\lambda, \log T)$, and verifier runtime is $\mathsf{poly}(\lambda, n, \log T)$.

Definition 3.1 (Adaptive proof of knowledge)**.** A SNARK scheme satisfies adaptive knowledge soundness if there exists a extractor $\mathcal{E}$ such that for all polynomials $n = n(\lambda)$, $m = m(\lambda)$, $k = k(\lambda)$, $T = T(\lambda)$, RAM Machines $\mathcal{R}$ of memory size k and running time T, every PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\ \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \wedge \mathcal{R}(x, \omega) = 0 \ \ : \ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}, n, m, T, k) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \omega \leftarrow \mathcal{E}(\mathcal{A}, \mathsf{crs}, x, \pi) \end{array}\ \right] \leq \mathsf{negl}(\lambda).$$

3.3 Hash Tree

Syntax. It contains following polytime algorithms.

$\mathsf{Setup}(1^\lambda, N) \to \mathsf{hk}$. The setup algorithm takes as input a security parameters λ, the size of input N. It outputs a hash key hk.

$\mathsf{Hash}(\mathsf{hk}, x) \to h$. The hash function takes as input a hash key hk, a input string of size N, and outputs a hash value h where $|h| = \mathsf{poly}(\lambda, \log N)$.

$\mathsf{Open}(\mathsf{hk}, x, i) \to u$. The opening algorithm takes as input a hash key hk, a input of size N, an index $i \in [N]$, and outputs an opening u, where $|u| = \mathsf{poly}(\lambda, \log N)$.

$\mathsf{Verify}(\mathsf{hk}, h, i, b, u) \to \{0,1\}$. The verifer algorithm takes as input a hash key hk, a hash value h, an index $i \in [N]$, bit b, opening u, and outputs 0 or 1.

$\mathsf{Write}(\mathsf{hk}, h, i, b, u) \to h'$. The writing algorithm takes as input a hash key hk, a hash value h, an index $i \in [N]$, a bit b to write into index i, opening u. It outputs an updated hash value h'. It aborts and outputs $\perp$ if u is not a valid opening with respect to h.

$\mathsf{Update}(\mathsf{hk}, i, b, u_i, j, u_j) \to u'_j$. The update function takes as input hash key hk, index i, bit value b (to be updated at index i), its opening u, and another index j, and its opening u_j. It outputs an *updated* opening u'_j, which is the opening for index j.

In words, the update algorithm takes two openings u_i, u_j for some locations i, j as well as a bit b. Its goal is to generate an updated opening for index j, under the condition that the input bit at location i is being changed to b.

Completeness. *Opening:* For every $\lambda, N \in \mathbb{N}$, $x \in \{0,1\}^N$, $i \in [N]$, $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, N)$, $h = \mathsf{Hash}(\mathsf{hk}, x)$, and $u = \mathsf{Open}(\mathsf{hk}, x, i)$, it holds that $\mathsf{Verify}(\mathsf{hk}, h, i, x_i, u) = 1$.
Writing: For every $\lambda, N \in \mathbb{N}$, $h \in \{0,1\}^\lambda$, $i \in [N]$, $b \in \{0,1\}$, $u \in \{0,1\}^U$, and $h' = \mathsf{Write}(\mathsf{hk}, h, i, b, u)$, it holds that $\mathsf{Verify}(\mathsf{hk}, h', i, b, u) = 1$.

Definition 3.2. (Collision resistance). A hash tree satisfies collision resistance if for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, N \in \mathbb{N}$, the following holds:

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{hk}, h, i, b, u) = 1 & \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, N) \\ \wedge\, \mathsf{Verify}(\mathsf{hk}, h, i, b', u') = 1 & (h, i, b, b', u, u') \leftarrow \mathcal{A}(\mathsf{hk}) \\ \wedge\, b \neq b' & \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition 3.3. (Soudness of Updating). A hash tree satisfies soundness of updating if for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, N \in \mathbb{N}$, the following holds:

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{hk}, h, i_{\mathsf{upd}}, b_{\mathsf{upd}}, u_{\mathsf{upd}}) = 1 & \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, N) \\ \wedge\, \mathsf{Write}(\mathsf{hk}, h, i, b, u) = h' \neq \perp & (h, i, b, u, i_{\mathsf{upd}}, b_{\mathsf{upd}}, u_{\mathsf{upd}}) \leftarrow \mathcal{A}(\mathsf{hk}) \\ \wedge\, i \neq i_{\mathsf{upd}} & u'_{\mathsf{upd}} = \mathsf{Update}(\mathsf{hk}, i, b, u, i_{\mathsf{upd}}, u_{\mathsf{upd}}) \\ \wedge\, \mathsf{Verify}(\mathsf{hk}, h', i_{\mathsf{upd}}, b_{\mathsf{upd}}, u'_{\mathsf{upd}}) \neq 1 & \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 3.4. ([Mer88]) From any collision resistant hash family, one can build a Hash Tree as above.

Remark 3.5. We require an additional property from the collision resistant hash family as building block: For $h = \mathsf{Hash}(\mathsf{hk}, 0^n)$, the output h must be an all zeros string. We argue that such property can be achieved using any collision resistant hash family H. We define a new collision resistant hash family as the following:

$\mathsf{Hash}(\mathsf{hk}, x) \to h$. For $x = 0^n$, it outputs h as an all zeros string. For $x \neq 0^n$, it outputs $h = 1||\mathsf{H.Hash}(\mathsf{hk}, x)$.

The rest of the functions are identical to H. Completeness and collision resistance properties immediately follow from those properties of H. Consequently, a Hash Tree designed using such a collision-resistant hash family as its building block will also inherit the property of generating an all zeros hash value when taking an all zeros input.

3.4 RAM Delegation

Syntax. A RAM delegation scheme consists of the following algorithms.

$\mathsf{Setup}(1^\lambda, \mathcal{R}, T, n) \to \mathsf{crs}$: The setup algorithm takes as input security parameter λ, machine $\mathcal{R}$, running time bound T, and input length n. It outputs CRS crs.

$\mathsf{Prove}(\mathsf{crs}, x) \to \pi$: The prover algorithm takes as input CRS crs, input $x \in \{0,1\}^n$, and outputs proof π.

$\mathsf{Verify}(\mathsf{crs}, x, \pi) \to \{0,1\}$: The verifier algorithm takes as input crs, input x, and proof π, and outputs either 0 or 1.

Completeness For all RAM machine $\mathcal{R}$, and $\lambda, T, n \in \mathbb{N}$ such that $n \leq T \leq 2^\lambda$, and $x \in \{0,1\}^n$ such that $\mathcal{R}(x)$ accepts in T steps, the following holds:

$$\Pr\left[\ \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \ \ : \ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}, T, n) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x) \end{array}\ \right] \geq 1 - \mathsf{negl}(\lambda).$$

Succinctness In the above completeness experiment, $|\mathsf{crs}| \leq \mathsf{poly}(\lambda, \log T)$. Prover algorithm runs in time $\mathsf{poly}(\lambda, T)$ and outputs a proof of length $|\pi| \leq \mathsf{poly}(\lambda, \log T)$. Verifier runs in time $\mathsf{poly}(\lambda, \log T, n)$.

Definition 3.6 (Soundness)**.** For every polynomial $n = n(\lambda)$, $T = T(\lambda)$, RAM machine $\mathcal{R}$ that takes a input of length n and runs in time T, and every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\ \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \wedge \mathcal{R}(x) \text{ does not accept in } T \text{ steps} \ \ : \ \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}, T, n) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\ \right] \leq \mathsf{negl}(\lambda).$$

Corollary 3.7. [CJJ21, CJJ22, KVZ21, HJKS22, WW22, DGKV22, PP22, KLVW23] Assuming either LWE, k-LIN over pairing groups for any constant $k \in \mathbb{N}$, or sub-exponential DDH over pairing-free groups (and QR), then for every $\lambda \in \mathbb{N}$, $T, n \in \mathsf{poly}(\lambda)$, RAM delegation exists for machine $\mathcal{R}$ with running time T and input size n.

4 Full Succinctness from Mild SNARKs with Short CRS

We start by establishing the concept of δ-mild SNARKs.

$\delta-$Mild SNARK with succinct CRS. Consider the circuit satisfiability language. Let $\mathcal{C}$ be any boolean circuit $\{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ that takes as input $x \in \{0,1\}^n, \omega \in \{0,1\}^m$ and outputs a bit $\{0,1\}$. We define the notion of δ-mild SNARKs for the circuit satisfiability language, where such SNARKs are defined identically to that as SNARKs for RAM, except the Setup algorithm takes description of circuit $\mathcal{C}$, n (in unary), and m (in binary) as inputs. Next, we define δ-mildness and succinct CRS properties formally.

δ-mildness. Let δ be a function such that $\delta = \delta(\lambda, n, m)$. A SNARK satisfies δ-mildness, if there exists a poly $p(\cdot)$ such that for every $\lambda \in \mathbb{N}$, polynomial $n = n(\lambda), m = m(\lambda)$, boolean circuit $\mathcal{C} = \{0,1\}^n \times \{0,1\}^m$, $x \in \{0,1\}^n$, $\omega \in \{0,1\}^m$ where $\mathcal{C}(x,\omega) = 1$, $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, \omega)$, it holds that

$$|\pi| \leq \frac{m}{\delta(\lambda, n, m)} + p(\lambda, n).$$

Succinct CRS A SNARK has succinct CRS, if there exists a poly $p(\cdot)$ such that for every $\lambda \in \mathbb{N}$, polynomial $n = n(\lambda), m = m(\lambda)$, boolean circuit $\mathcal{C} = \{0,1\}^n \times \{0,1\}^m$, $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, n, m, \mathcal{C})$, it holds that $|\mathsf{crs}| \leq p(\lambda, \log|\mathcal{C}|, n, m)$.

4.1 Fully-Succinct SNARKs from λ^ϵ-Mild SNARKs

We discuss how to build fully-succinct SNARK for any RAM Machine $\mathcal{R}$ with bounded running time from a λ^ϵ-mild SNARK and hash tree systems. Later, we show how to generically improve these to RAM Machines with unbounded running time.

Special Notation: Composite Hash Tree. We rely on a specialized notation that we call Composite Hash Trees. One may consider Composite Hash Tree as a set of k Hash Trees, each assigned a section of input. For example, the hash value $h \leftarrow \mathsf{Hash}(\mathsf{hk}, x)$ consists a number of k hash values $h = (h_1, \ldots, h_k)$ where each value corresponds to one section of x. As long as k is a constant number, the efficiency and soundness properties all maintain from the original Hash Tree. The desired property from Composite Hash Tree is that the reading, opening, and writing at the i-th section of string x only depends on and affects h_i. Composite Hash Tree overloads the syntax of Hash Tree, except for the following:

$\mathsf{Setup}(1^\lambda, N_1, \ldots, N_k) \to \mathsf{hk}$. The setup algorithm is modified to accept multiple inputs $N_1, \ldots, N_k$. The size of hash input is $\Sigma_{i=1}^k N_i$. Input is considered as a number of k continuous segments, where the i-th segment is of size N_i. It also sets hash key hk as a sequence of of k independent hash keys: $\mathsf{hk} = (\mathsf{hk}_1, \ldots, \mathsf{hk}_k)$.

We define the additional property of completeness: For every $\lambda, k, N_1, \ldots, N_k \in \mathbb{N}$, $x^{(i)} \in \{0,1\}^{N_i}$ for all $i \in [k]$, $x = (x^{(1)}, \ldots, x^{(k)})$, and $\mathsf{hk} = (\mathsf{hk}_1, \ldots, \mathsf{hk}_k) \leftarrow \mathsf{Setup}(1^\lambda, N_1, \ldots, N_k)$,

$$\Pr\left[\ \mathsf{Hash}(\mathsf{hk}, x) = (\mathsf{Hash}(\mathsf{hk}_i, x^{(1)}), \ldots, \mathsf{Hash}(\mathsf{hk}_i, x^{(k)}))\ \right] = 1.$$

For the rest of this section, we only rely on such Composite Hash Trees.

Our Fully-Succinct SNARK Construction. We rely on δ-mild SNARK $\Gamma = (\Gamma.\mathsf{Setup}, \Gamma.\mathsf{Prove}, \Gamma.\mathsf{Verify})$ for $\delta \geq 2\lambda$, and composite hash tree $\mathsf{H} = (\mathsf{H.Setup}, \mathsf{H.Hash}, \mathsf{H.Open}, \mathsf{H.Verify}, \mathsf{H.Write}, \mathsf{H.Update})$. Below is our design.

$\mathsf{Setup}(1^\lambda, \mathcal{R}, n, m, T, k) \to \mathsf{crs}$. The setup algorithm follows these steps:

1. RAM machine $\mathcal{R}$ is as the following: $\mathcal{R}$ takes as input (x, ω), running time of $\mathcal{R}$ is T, and memory usage is capped at k. Size of input (x, ω) is $n + m$.

2. We then specify ℓ as $\ell = \lceil \frac{\log T}{\log \lambda} \rceil$. We also specify the state transformation circuit of $\mathcal{R}$ as $\mathcal{C}^{\mathsf{RAM}}$ and the memory as $M = (M_1, \ldots, M_k)$. It sets

$$\mathsf{hk} \leftarrow \mathsf{H.Setup}(1^\lambda, n, m, k - n - m).$$

The intuition is that the memory is splitted into three sections where the first two sections are used to record the input x, ω and the last section is used as a working tape. Then under the hash key $\mathsf{hk} = (\mathsf{hk}_1, \mathsf{hk}_2, \mathsf{hk}_3)$, hash value h consists of three independent values (h_1, h_2, h_3).

3. It generates a number of $\ell + 1$ common reference strings: For all $i \in \{0, \ldots, \ell\}$,

$$\mathsf{crs}_i \leftarrow \Gamma.\mathsf{Setup}(1^\lambda, 1^{n_i}, 1^{m_i}, \mathcal{C}_i),$$

where $\mathcal{C}_i$ is defined as the following, and n_i is the instance size, m_i is the witness size for $\mathcal{C}_i$.

4. It outputs $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$.

Circuit $\mathcal{C}_0$

Instance: $(\rho, \mathsf{state}, i_{\mathsf{read}})$, $(\rho', \mathsf{state}', i'_{\mathsf{read}})$.
Witness: $b_{\mathsf{read}}, b_{\mathsf{write}}, i_{\mathsf{write}}, u_{\mathsf{read}}, u_{\mathsf{write}}$.
Hardwired: $\mathcal{C}^{\mathsf{RAM}}$.
Output: $\mathcal{C}_0$ outputs 1 if and only if the followings are satisfied:

- $\mathcal{C}^{\mathsf{RAM}}(\mathsf{state}, b_{\mathsf{read}}) = (\mathsf{state}', i'_{\mathsf{read}}, i_{\mathsf{write}}, b_{\mathsf{write}})$.
- $\mathsf{H.Verify}(\mathsf{hk}, \rho, i_{\mathsf{read}}, b_{\mathsf{read}}, u_{\mathsf{read}}) = 1$.
- $\mathsf{H.Write}(\mathsf{hk}, \rho, i_{\mathsf{write}}, b_{\mathsf{write}}, u_{\mathsf{write}}) = \rho'$.

For $i \in \{1, \ldots, \ell\}$:

Circuit $\mathcal{C}_i$

Instance: $(\rho, \mathsf{state}, i_{\mathsf{read}})$, $(\rho', \mathsf{state}', i'_{\mathsf{read}})$.
Witness: $\{\pi_j\}_{j \in \lambda}$, $\{(\rho, \mathsf{state}, i_{\mathsf{read}})_j\}_{j \in [\lambda+1]}$.
Hardwired: crs_{i-1}.
Output: $\mathcal{C}_i$ outputs 1 if and only if:

- $\Gamma.\mathsf{Verify}(\mathsf{crs}_{i-1}, ((\rho, \mathsf{state}, i_{\mathsf{read}})_j, (\rho, \mathsf{state}, i_{\mathsf{read}})_{j+1}), \pi_j) = 1$, for all $j \in [\lambda]$.
- $(\rho, \mathsf{state}, i_{\mathsf{read}}) = (\rho_1, \mathsf{state}_1, i_1)$
- $(\rho', \mathsf{state}', i'_{\mathsf{read}}) = (\rho_{\lambda+1}, \mathsf{state}_{\lambda+1}, i_{\lambda+1})$

$\mathsf{Prove}(\mathsf{crs}, x, \omega) \rightarrow \pi$. The prover algorithm follows these steps:

1. It simulates $\mathcal{R}$ step by step. Previously we specified the state transformation circuit of $\mathcal{R}$ as $\mathcal{C}^{\mathsf{RAM}}$ and the memory as M. Since $\mathcal{R}$ has a maximum number of T steps, the simulation process involves a total number of $T + 1$ states. Recall definition of RAM machine where for each step of computation, the original local state state transforms into a new state state′. Such transition involves an index to read i_{read}, the bit b_{read} at

index i_{read}. It sets $\rho = \mathsf{H.Hash}(\mathsf{hk}, M)$ where $M = (M_1, \ldots, M_k)$ represents the whole memory at such state. When reading the bit b_{read} at index i_{read}, it simultaneously computes the opening $u_{\mathsf{read}} = \mathsf{H.Open}(\mathsf{hk}, M, i_{\mathsf{read}})$. Then state transformation circuit $\mathcal{C}^{\mathsf{RAM}}$ takes state and b_{read} as input and outputs $\mathsf{state}', i'_{\mathsf{read}}, i_{\mathsf{write}}, b_{\mathsf{write}}$. It updates the bit at index i_{write}, setting $M'_{i_{\mathsf{write}}}$ as b_{write}, and M'_i as M_i for all $i \in [k] \setminus \{i_{\mathsf{write}}\}$. It sets updated memory $M' = (M'_1, \ldots, M'_k)$. Next, it obtains updated hash value as $\rho' = \mathsf{H.Write}(\mathsf{hk}, h, b_{\mathsf{write}}, i_{\mathsf{write}})$.

It records $(\mathsf{state}, i_{\mathsf{read}}, b_{\mathsf{read}}, \rho, u_{\mathsf{read}})_i$, for $i \in [T+1]$ corresponding to the i-th state. It records $(b_{\mathsf{write}}, i_{\mathsf{write}}, u_{\mathsf{write}})_i$ for $i \in [T]$ as the parameters of writing while going from state_i to state_{i+1}.

2. For all $i \in [T]$, it sets instance x_i and witness ω_i as the following:

$$x_i = ((\rho, \mathsf{state}, i_{\mathsf{read}})_i, (\rho, \mathsf{state}, i_{\mathsf{read}})_{i+1}), \omega_i = (b_{\mathsf{read}}, b_{\mathsf{write}}, i_{\mathsf{write}}, u_{\mathsf{read}}, u_{\mathsf{write}})_i.$$

Then for all $i \in [T]$, it generates $\pi_i \leftarrow \Gamma.\mathsf{Prove}(\mathsf{crs}_0, x_i, \omega_i)$.

3. It generates a tree $G = (V, E)$ of $\ell + 1$ levels, as the following: The top level is set as level ℓ and the bottom level is set as level 0. Level 0 contains a number of T nodes, and level i contains approximately $\lambda^{\ell-i}$ nodes. Every node at levels $1, \ldots, \ell$ is connected to a number of λ nodes at the lower level. Specifically, the j-th node on level $i (1 \leq i \leq \ell)$ has λ child nodes: the $((j-1)\cdot\lambda+1)$-th node, the $((j-1)\cdot\lambda+2)$-th node, ..., and the $(j \cdot \lambda)$-th node on level $i-1$. Since $T \leq \lambda^\ell$, we assume without loss of generality that there is at most one node on level ℓ. Note that the graph G has a tree structure, where each internal node has λ child nodes.

4. For each node $v \in V$, it assigns of a pair of instance and proof to v: $(\mathsf{inst}_v, \mathsf{proof}_v)$. For the i-th node at level 0 (denoted as $v_{0,i}$), it assigns instance $\mathsf{inst}_{v_{0,i}} = x_i$ and proof $\mathsf{proof}_{v_{0,i}} = \pi_i$ to such node. Next, it assigns a pair of instance and proof to each node at level 1 to ℓ.

5. It starts from level 1. Each node v at level 1 has a sequence of λ child nodes: $v_1, \ldots, v_\lambda$. It then takes instances inst_{v_i} and proofs proof_{v_i} for all $i \in [\lambda]$. It parses instance inst_{v_i} as

$$\mathsf{inst}_{v_i} = ((\rho, \mathsf{state}, i_{\mathsf{read}})_i, (\rho, \mathsf{state}, i_{\mathsf{read}})_{i+1}).$$

Then for all $i \in [\lambda+1]$, it sets z_i as $(\rho, \mathsf{state}, i_{\mathsf{read}})_i$. It generates proof

$$\sigma = \Gamma.\mathsf{Prove}(\mathsf{crs}_i, (z_1, z_{\lambda+1}), (\{\sigma_i\}_{i\in[\lambda]}, \{z_i\}_{i\in[\lambda+1]})).$$

It assigns $(z_1, z_{\lambda+1})$ as the instance and σ as the proof to such internal node:

$$\mathsf{inst}_v = (z_1, z_{\lambda+1}), \mathsf{proof}_v = \sigma.$$

6. It repeats step 5 for level $2, 3, \ldots, \ell$.

7. The root node is now assigned with the instances $(\rho, \mathsf{state}, i_{\mathsf{read}})$ and $(\rho', \mathsf{state}', i'_{\mathsf{read}})$, along with the proof σ. Recall that one may consider hash value as three independent values, such that $\rho = (\rho_1, \rho_2, \rho_3)$ and $\rho' = (\rho'_1, \rho'_2, \rho'_3)$. The output is $\pi = (\rho_2, \rho', \sigma)$.

$\mathsf{Verify}(\mathsf{crs}, x, \pi) \to \{0,1\}$. It first parses crs as $(\mathsf{hk}, \mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$ and π as (ρ_2, ρ', σ). It sets ρ_3 as an all zeros string. By Remark 3.5, ρ_3 is equivalent to $\mathsf{H.Hash}(\mathsf{hk}_3, 0^{k-n-m})$, which allows the verifier to directly generate ρ_3 without evaluating the hash function. Next, it sets ρ_1 as $\mathsf{H.Hash}(\mathsf{hk}_1, x)$, and ρ as (ρ_1, ρ_2, ρ_3). It then sets state as the starting state and state$'$ as the accepting state, and i_{read} as the reading bit's index for the starting state, i'_{read} as such index for the accepting state. Without loss of generality, we take i_{read} and i'_{read} as fixed according to the definition of machine $\mathcal{R}$. The verifier outputs

$$\Gamma.\mathsf{Verify}(\mathsf{crs}_\ell, ((\rho, \mathsf{state}, i_{\mathsf{read}}), (\rho', \mathsf{state}', i'_{\mathsf{read}})), \sigma).$$

Completeness The completeness of the above design directly follows from the completeness of composite hash tree H and δ-mild SNARK Γ.

Efficiency For all $i \in \{0, \ldots, \ell\}$, we denote n_i as instance size, m_i as witness size, and $|\pi_i|$ as proof size for each individual node at level i.

Lemma 4.1. For all $i \in \{0, \ldots, \ell\}$, $|\pi_i| \leq n_0 + m_0 + 2 \cdot p(\lambda, n_0)$, $m_i \leq \lambda \cdot (n_0 + m_0 + 2 \cdot p(\lambda, n_0))$.

Proof. We prove the lemma using induction.

Base Case $(i = 0)$**.** By δ-mildness ($\delta \geq 2\lambda$), we have $|\pi_0| \leq m_0/(2\lambda) + p(\lambda, n_0)$. Thus the base case satisfies the lemma.

Inductive Step $(1 \leq i \leq \ell)$**.** Suppose that the Lemma holds for $i-1$ and we proceed to i. By inductive hypothesis, $|\pi_{i-1}|$ is at most $n_0 + m_0 + i \cdot p(\lambda, n_0)$. We note that due to the design of our tree-style SNARK, instance at each level consists of $(\rho, \mathsf{state}, i_{\mathsf{read}})$ and $(\rho', \mathsf{state}', i'_{\mathsf{read}})$, with uniform sizes across all levels. Therefore, $n_0, \ldots, n_\ell$ are all equivalent. Given that the each witness at level i consists of $\{\pi_j\}_{j \in \lambda}$ and $\{(\rho, \mathsf{state}, i_{\mathsf{read}})_j\}_{j \in [\lambda+1]}$ which combines a number of λ proofs and instances from level $i-1$, witness size m_i is bounded by the following:

$$m_i \leq \lambda \cdot (n_{i-1} + |\pi_{i-1}|) \leq \lambda \cdot (n_0 + m_0 + 2 \cdot p(\lambda, n_0)).$$

By succinctness of Γ, $|\pi_i|$ is at most:

$$|\pi_i| \leq \frac{m_i}{2\lambda} + p(\lambda, n_i) \quad \leq n_0 + m_0 + 2 \cdot p(\lambda, n_0).$$

Putting the above together completes the proof for the lemma. □

Lemma 4.2. The prover's running time is $T \cdot \mathsf{poly}(\lambda)$ where T is the running time of $\mathcal{R}(x, \omega)$.

Proof. Since instance and witness at level 0 only contain local parameters, by Lemma 4.1, we have $m_i \leq \mathsf{poly}(\lambda)$ for some universal polynomial $\mathsf{poly}(\cdot)$ and for all $i \in \{0, \ldots, \ell\}$. Instance size $n_i = n_0$ and is also upper bounded by $\mathsf{poly}(\lambda)$ for all $i \in \{0, \ldots, \ell\}$. Thus, prover's running time at each node is bounded by $\mathsf{poly}(\lambda)$ for some universal $\mathsf{poly}(\cdot)$. The overall running time of the prover is at most $T \cdot \mathsf{poly}(\lambda)$ since the total number of nodes in the data structure is linear in T. □

Lemma 4.3. The Setup algorithm's running time is $\mathsf{poly}(\lambda)$.

Proof. The Setup algorithm of our design includes setting up hash key hk and Γ CRS $\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell$. The hk setup is fast. To analyze the setup time of $\mathsf{crs}_i \leftarrow \Gamma.\mathsf{Setup}(1^\lambda, 1^{n_i}, 1^{m_i}, \mathcal{C}_i)$, we must understand circuit size $|\mathcal{C}_i|$. Observe that $|C_0| = \mathsf{poly}(\lambda, n_0, m_0)$, and $|C_i| = \mathsf{poly}(\lambda, |\mathsf{crs}_{i-1}|, n_i, m_i)$ for all $i \in [\ell]$. By Lemma 4.1, it holds that for all $i \in \{0, \ldots, \ell\}$, $m_i, n_i \leq \mathsf{poly}(\lambda)$ for some universal polynomial $\mathsf{poly}(\cdot)$. Thus, $|\mathcal{C}_0| = \mathsf{poly}(\lambda)$. By succinct CRS property of Γ, it holds that $|\mathsf{crs}_i| \leq \mathsf{poly}(\lambda, \log|\mathcal{C}_i|, n_i, m_i)$ for all $i \in [\ell]$, which implies that $|\mathcal{C}_i| \leq \mathsf{poly}(\lambda, \log|\mathcal{C}_{i-1}|)$. Thus $\forall i \in [\ell]$, $|\mathcal{C}_i| \leq \mathsf{poly}(\lambda)$ for some universal polynomial $\mathsf{poly}(\cdot)$. The overall Setup time is $\mathsf{poly}(\lambda)$ for $\ell = \frac{\log T}{\log \lambda}$. □

Lemma 4.4. Assume that Γ is a δ-mild SNARK ($\delta \geq 2\lambda$) with short CRS, and H is a Hash Tree, then the above design is a fully succinct SNARK.

Proof. Following from our design, $\mathsf{crs} = (\mathsf{hk}, \mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$, where hk represents the hash key. Thus, $|\mathsf{hk}| \leq \mathsf{poly}(\lambda)$ by design of hash tree. Following from the analysis of Lemma 4.3, it holds that $|\mathsf{crs}_i| \leq \mathsf{poly}(\lambda, \log|\mathcal{C}_i|, n_i, m_i) = \mathsf{poly}(\lambda)$ for some universal polynomial $\mathsf{poly}(\cdot)$, and the overall CRS size satisfies $|\mathsf{crs}| \leq \mathsf{poly}(\lambda)$ for $\ell = \frac{\log T}{\log \lambda}$. By Lemma 4.1, proof size also satisfies full succinctness, such that $|\pi| \leq \mathsf{poly}(\lambda)$.

□

Soundness The soundness analysis is as below.

Theorem 4.5. Assuming that the mildly succinct SNARK scheme Γ satisfies the adaptive proof of knowledge property, and the Hash Tree H satisfies soundness and collision-resistance, then the above SNARK scheme also satisfies the proof of knowledge property.

Proof. We will show how to construct an extractor $\mathcal{E}$ for our scheme. Here, we assume $\mathcal{E}$ as an extractor with non-black-box access to $\mathcal{A}$. We first build recursive extractor $\mathcal{E}'$. $\mathcal{E}'$ also has non-black-box access to the attacker.

$\mathcal{E}'(\mathcal{A}, \mathsf{crs}, x, \pi) \rightarrow (G = (V, E), \{(\mathsf{inst}_v, \mathsf{proof}_v, \mathsf{wit}_v)\}_{v \in V})$. Below is a step-by-step explanation:

1. It parses crs as $(\mathsf{hk}, \mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$, proof π as (ρ_2, ρ', σ). Using hash key $\mathsf{hk} = (\mathsf{hk}_1, \mathsf{hk}_2, \mathsf{hk}_3)$, it sets $\rho_1 = \mathsf{H.Hash}(\mathsf{hk}_1, x), \rho = (\rho_1, \rho_2, h_3)$, and $\mathsf{state}, i_{\mathsf{read}}, \mathsf{state}', i'_{\mathsf{read}}$ as the starting and ending state with their reading index.
2. It sets tree $G = (V, E)$ as defined in the prover's algorithm. Next, it assigns an instance and proof pair, an extractor algorithm to each internal node of G. Denote the only node on level ℓ as v_ℓ. It first assigns instance and proof to such node: $\mathsf{inst}_{v_\ell} = ((\rho, \mathsf{state}, i_{\mathsf{read}}), (\rho', \mathsf{state}', i'_{\mathsf{read}}))$, $\mathsf{proof}_{v_\ell} = \sigma$. Then, it sets extractor algorithm extr_{v_ℓ} as: $\mathsf{extr}_{v_\ell}(\mathcal{A}, \mathsf{crs}_\ell, \mathsf{inst}_{v_\ell}, \mathsf{proof}_{v_\ell}) \rightarrow \mathsf{wit}_{v_\ell}$. The extractor takes as input $\mathsf{crs}_\ell, \mathsf{inst}_{v_\ell}, \mathsf{proof}_{v_\ell}$, and outputs a witness
$$\mathsf{wit}_{v_\ell} \leftarrow \Gamma.\mathcal{E}(\mathcal{A}, \mathsf{crs}_\ell, \mathsf{inst}_{v_\ell}, \mathsf{proof}_{v_\ell}).$$
3. It assigns instance, proof, extractor for each node at level $i = \ell - 1$. For any node v at level i, ideally the extractor at v's parental node u, extr_u outputs the extracted witness as $\{\sigma_j\}_{j \in [\lambda]}, \{(\rho, \mathsf{state}, i_{\mathsf{read}})_j\}_{j \in [\lambda+1]}$. Consider that v is the j-th child node of the parental

node, it assigns instance and proof: $\mathsf{inst}_v = ((\rho, \mathsf{state}, i_{\mathsf{read}})_j, (\rho, \mathsf{state}, i_{\mathsf{read}})_{j+1}), \mathsf{proof}_v = \sigma_j$. Upon instance and proof, it sets the extractor for node v:

$\mathsf{extr}_v(\mathsf{extr}_u, \mathsf{crs}_i, \mathsf{inst}_v, \mathsf{proof}_v) \to \mathsf{wit}_v$. It takes as input $\mathsf{crs}_i, \mathsf{inst}_v, \mathsf{proof}_v$, and outputs a witness

$$\mathsf{wit}_v \leftarrow \Gamma.\mathcal{E}(\mathsf{extr}_u, \mathsf{crs}_i, \mathsf{inst}_v, \mathsf{proof}_v).$$

4. It repeats step 3 from level $i = \ell - 1$ to $i = 0$.
5. It outputs $G = (V, E), \{(\mathsf{inst}_v, \mathsf{proof}_v, \mathsf{wit}_v)\}_{v \in V}$.

Claim 4.6. For every efficient and valid attacker $\mathcal{A}$, $\mathcal{E}'(\mathcal{A}, \mathsf{crs}, x, \pi)$ is a polynomially bounded algorithm.

Proof. Efficient attacker $\mathcal{A}$ gives polynomials $n = n(\lambda), m = m(\lambda), T = T(\lambda), k = k(\lambda)$. Thus, $\ell = \lceil \frac{\log T}{\log \lambda} \rceil = O(1)$. For all $i \in \{0, \ldots, \ell\}$, denote V_i as the the set of all nodes at the i-th level of G. Given that the sizes of instances and witnesses are equivalent across the same level, for nodes at level i, let n_i and m_i represent the sizes of the instance and witness respectively. Following from Lemma 4.1, for all $i \in \{0, \ldots, \ell\}$, $m_i = \mathsf{poly}(\lambda)$, and instance only contains local parameters where $n_i = \mathsf{poly}(\lambda)$. For each $i \in \{0, \ldots, \ell\}$, extractor at any node v at level i runs in time $\mathsf{poly}(|\mathsf{crs}_i|, n_i, m_i, |\mathsf{extr}_u|)$, where u is the parental node of v and $|\mathsf{extr}_u|$ is the running time of extr_u. Since number of levels ℓ is a constant value, and the size of extractor only has a polynomial blow up across each level, we conclude that the running time of $\mathcal{E}'(\mathcal{A}, \mathsf{crs}, x, \pi)$ is polynomially bounded. □

Lemma 4.7. Assuming that Γ satisfies adaptive proof of knowledge, then for all polynomials $n = n(\lambda)$, $m = m(\lambda)$, $k = k(\lambda)$, $T = T(\lambda)$, RAM Machines $\mathcal{R}$ of memory size k and running time T, every PPT attacker $\mathcal{A}$, when $\mathcal{E}$ has non-black-box access to $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds for all $i \in \{0, \ldots, \ell\}$:

$$\Pr\left[\begin{array}{lcl} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 & & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}, 1^n, 1^m, T, k) \\ \wedge\, \exists v \in V_i \text{ s.t.} & : & (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \quad \mathcal{C}_i(\mathsf{inst}_v, \mathsf{wit}_v) = 0 & & (G, \{(\mathsf{inst}_v, \mathsf{proof}_v, \mathsf{wit}_v)\}_{v \in V}) \leftarrow \mathcal{E}'(\mathcal{A}, \mathsf{crs}, x, \pi) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where in the above equation, graph $G = (V, E)$ consists of $\ell + 1$ levels, and the set V_i contains the nodes on level i of G.

Proof. We construct a proof by induction on i.

Base Case $(i = \ell)$**:** We first show that the lemma holds for $i = \ell$. Assume that there exists some $v \in V_\ell$ such that $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1, \mathcal{C}_\ell(\mathsf{crs}_\ell, \mathsf{inst}_v, \mathsf{wit}_v) = 0$, where $\mathsf{wit}_v \leftarrow \mathsf{extr}_v(\mathsf{crs}_i, \mathsf{inst}_v, \mathsf{proof}_v)$. We build a reduction algorithm $\mathcal{B}$ that breaks the proof of knowledge property of Γ.

$\mathcal{A}$ starts by outputting RAM Machine $\mathcal{R}$ and λ, n, m, T, k. Reduction algorithm $\mathcal{B}$ sets up hash key $\mathsf{hk} \leftarrow \mathsf{H.Setup}(1^\lambda, n, m, k - n - m)$ and sets hash value $h_3 = \mathsf{H.Hash}(\mathsf{hk}_3, 0^{k-n-m})$. Next, for all $i \in \{0, \ldots, \ell - 1\}$, $\mathcal{B}$ generates circuits $\mathcal{C}_i$ and sets the corresponding CRS $\mathsf{crs}_i \leftarrow \Gamma.\mathsf{Setup}(1^\lambda, 1^{n_i}, 1^{m_i}, \mathcal{C}_i)$ where definition of $\mathcal{C}_i$ follows from the Setup algorithm and n_i, m_i are the

instance and witness size of $\mathcal{C}_i$. $\mathcal{B}$ then sets circuit $\mathcal{C}_\ell$ and queries the Γ challenger with $\lambda, n_\ell, m_\ell, \mathcal{C}_\ell$. The challenger outputs crs_ℓ. $\mathcal{B}$ outputs $\mathsf{crs} = (\mathsf{hk}, h_3, \mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$. Upon crs, $\mathcal{A}$ outputs (x, π).

Consider that $\mathcal{E}'(\mathcal{A}, \mathsf{crs}, x, \pi)$ outputs graph $G = (V, E)$ and $\{(\mathsf{inst}_v, \mathsf{proof}_v, \mathsf{wit}_v)\}_{v \in V}$. From the assumption, with non-negligible probability, there exists some $v \in V_\ell$ such that $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$ and $\mathcal{C}_\ell(\mathsf{crs}_\ell, \mathsf{inst}_v, \mathsf{wit}_v) = 0$. $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$ implies that $\Gamma.\mathsf{Verify}(\mathsf{crs}_\ell, \mathsf{inst}_v, \mathsf{proof}_v) = 1$. Meanwhile, recall that wit_v was extracted as the following: $\mathsf{wit}_v = \mathsf{extr}_v(\mathsf{crs}_\ell, \mathsf{inst}_v, \mathsf{proof}_v) = \Gamma.\mathcal{E}(\mathcal{A}, \mathsf{crs}_\ell, \mathsf{inst}_v, \mathsf{proof}_v)$. Thus $\mathcal{B}$ breaks the proof of knowledge property by outputting $\mathsf{inst}_v, \mathsf{proof}_v$.

Inductive Step $(0 \leq i < \ell)$**:** Suppose that the lemma holds for all $j < i \leq \ell$. We show that the above lemma also holds for $i = j$. Assume that there exists some $v \in V_j$ such that $\mathcal{C}_j(\mathsf{crs}_j, \mathsf{inst}_v, \mathsf{wit}_v) = 0$, where $\mathsf{wit}_v \leftarrow \mathsf{extr}_v(\mathsf{extr}_u, \mathsf{crs}_j, \mathsf{inst}_v, \mathsf{proof}_v)$ for u as the parental node. We build the following reduction algorithm $\mathcal{B}$ which breaks the proof of knowledge property of Γ.

$\mathcal{A}$ starts by outputting $\lambda, n, m, \mathcal{R}, T, k$. $\mathcal{B}$ sets $\mathsf{hk} \leftarrow \mathsf{H.Setup}(1^\lambda, n, m, k - n - m)$, $h_3 = \mathsf{H.Hash}(\mathsf{hk}_3, 0^{k-n-m})$. Next, $\mathcal{B}$ generates circuits $\mathcal{C}_0, \ldots, \mathcal{C}_j$ and $\mathsf{crs}_0, \ldots, \mathsf{crs}_{j-1}$ using the setup algorithm of Γ. then, $\mathcal{B}$ queries the Γ challenger with $\lambda, n_j, m_j, \mathcal{C}_j$ and the challenger outputs crs_i. $\mathcal{B}$ then generates circuit $\mathcal{C}_{j+1}, \ldots, \mathcal{C}_\ell$ and $\mathsf{crs}_{j+1}, \ldots, \mathsf{crs}_\ell$ accordingly. Next $\mathcal{B}$ outputs $\mathsf{crs} = (\mathsf{hk}, h_3, \mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$. $\mathcal{A}$ then outputs (x, π).

Consider $\mathcal{E}'(\mathcal{A}, \mathsf{crs}, x, \pi)$ outputs $G = (V, E)$ and $\{(\mathsf{inst}_v, \mathsf{proof}_v, \mathsf{wit}_v)\}_{v \in V}$. Again we denote u as a parental node of v. Then extr_u outputs wit_u which contains the node v's instance inst_v and proof proof_v. Following from the inductive hypothesis, with all but negligible probability, $\mathcal{C}_{j+1}(\mathsf{crs}_{j+1}, \mathsf{inst}_u, \mathsf{wit}_u) = 1$, which implies that $\Gamma.\mathsf{Verify}(\mathsf{crs}_j, \mathsf{inst}_v, \mathsf{proof}_v) = 1$. According to the above assumption, with non-negligible probability, there exists some $v \in V_j$ such that $\mathcal{C}_j(\mathsf{crs}_j, \mathsf{inst}_v, \mathsf{wit}_v) = 0$, where $\mathsf{wit}_v = \mathsf{extr}_v(\mathsf{extr}_u, \mathsf{crs}_j, \mathsf{inst}_v, \mathsf{proof}_v) = \Gamma.\mathcal{E}(\mathsf{extr}_u, \mathsf{crs}_j, \mathsf{inst}_v, \mathsf{proof}_v)$. Thus, $\mathcal{B}$ breaks the proof of knowledge property by outputting $\mathsf{inst}_v, \mathsf{proof}_v$. □

The recursive extractor $\mathcal{E}'$ falls short as a sufficient extractor, as it only produces the instance and witness for individual nodes. To build a comprehensive extractor, we introduce algorithm $\mathcal{E}$, which given (x, π), outputs a valid witness ω satisfying $\mathcal{R}(x, \omega) = 1$.

$\mathcal{E}(\mathcal{A}, \mathsf{crs}, x, \pi) \to \omega$. We bring a detailed breakdown:

1. It runs $\mathcal{E}'(\mathcal{A}, \mathsf{crs}, x, \pi)$ and obtains $G = (V, E)$, $\{(\mathsf{inst}_v, \mathsf{proof}_v, \mathsf{wit}_v)\}_{v \in V}$.
2. For the i-th bottom node v_i, it parses the instance inst_{v_i} as $(\rho, \mathsf{state}, i_{\mathsf{read}})_i, (\rho', \mathsf{state}', i'_{\mathsf{read}})_i$, and witness wit_{v_i} as $b_{\mathsf{read}}, b_{\mathsf{write}}, i_{\mathsf{write}}, u_{\mathsf{read}}, u_{\mathsf{write}}$.
3. It initializes ω as a zero string of length m, $\omega = 0^m$. It then iterates from $i = 1$ to $i = T$. At each node v_i, if the reading index i_{read} is fresh and falls within the memory range designated for the witness, it captures the bit b_{read} as the $(i_{\mathsf{read}} - n)$-th bit of ω. Namely, if the memory block i_{read} is unused in the previous steps and $n + 1 \leq i_{\mathsf{read}} \leq n + m$, it sets $\omega_{i_{\mathsf{read}} - n}$ as b_{read}.
4. It outputs ω.

Lemma 4.8. Assuming that Γ satisfies adaptive proof of knowledge, for all polynomials $n = n(\lambda)$, $m = m(\lambda)$, $k = k(\lambda)$, $T = T(\lambda)$, RAM Machines $\mathcal{R}$ of memory size k and running time T, every

PPT attacker $\mathcal{A}$, when $\mathcal{E}$ has non-black-box access to $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 & \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}, 1^n, 1^m, T, k) \\ \wedge\, \mathcal{R}(x, \omega) = 0 & : \ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & \omega \leftarrow \mathcal{E}(\mathcal{A}, \mathsf{crs}, x, \pi) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

Proof. Next, we assume that there exists some PPT adversary $\mathcal{A}$ breaking the proof of knowledge property of the above scheme. Common reference string crs is set from $\mathsf{Setup}(1^\lambda, R, 1^n, 1^m, T, k)$. Given crs, $\mathcal{A}$ outputs (x, π), such that $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$ and $\mathcal{R}(x, \mathcal{E}(\mathcal{A}, \mathsf{crs}, x, \pi))$ rejects.

Our proof follows from a case by case reduction:

Type 1**:** Consider that in the above experiment, one runs $\mathcal{E}'(\mathcal{A}, \mathsf{crs}, x, \pi)$ and obtains $G = (V, E)$, $\{(\mathsf{inst}_v, \mathsf{proof}_v, \mathsf{wit}_v)\}_{v \in V})$. For type 1 attacker $\mathcal{A}$, there exists at least one node v in G at a certain level i, where instance inst_v and extracted witness wit_v at such node do not satisfy Γ relation: $\mathcal{C}_i(\mathsf{inst}_v, \mathsf{wit}_v) = 0$.

Type $2, 3$**:** Recall that each node $v \in V$ is associated with an instance defined by $((\rho, \mathsf{state}, i_{\mathsf{read}}), (\rho', \mathsf{state}', i'_{\mathsf{read}}))$ and a witness as $(b_{\mathsf{read}}, b_{\mathsf{write}}, i_{\mathsf{write}}, u_{\mathsf{read}}, u_{\mathsf{write}})$. Arrange the nodes at the bottom level from left to right as a sequence, labeled as $v_1, \ldots, v_T$. For every $i \in [T]$, denote $v_i.\rho, v_i.i_{\mathsf{read}}, v_i.b_{\mathsf{read}}, v_i.u_{\mathsf{read}}$, $v_i.i_{\mathsf{write}}, v_i.b_{\mathsf{write}}, v_i.u_{\mathsf{write}}$ as the respective terms in the given instance and witness. For type 2 and 3 attacker $\mathcal{A}$, there exists at least one pair of indexes $i^*, j^* \in [T]$ and $i^* < j^*$, such that all of the following conditions hold:

- $v_{i^*}.i_{\mathsf{read}} = v_{j^*}.i_{\mathsf{read}}$ and $v_{i^*}.b_{\mathsf{read}} \neq v_{j^*}.b_{\mathsf{read}}$,
- For all $i \in \{i^*, \ldots, j^* - 1\}$, $v_i.i_{\mathsf{write}} \neq v_{i^*}.i_{\mathsf{read}}$.

Intuition of type 2 and 3 is that the attacker reads the same memory index as inconsistent bits, even though this index has not been modified between the two readings.

Consider that one applies the updating algorithm $\mathsf{H.Update}$ to update the opening $v_{i^*}.u_{\mathsf{read}}$ for bit $v_{i^*}.b_{\mathsf{read}}$ at index $v_{i^*}.i_{\mathsf{read}}$ starting from the i^*-th step, along with every writing operation, until the $(j^* - 1)$-th step. Namely, define u_i as the updated opening at the i-th step. And u_{i^*} is obtained as

$$u_{i^*} = \mathsf{H.Update}(\mathsf{hk}, v_{i^*}.i_{\mathsf{write}}, v_{i^*}.b_{\mathsf{write}}, v_{i^*}.u_{\mathsf{write}}, v_{i^*}.i_{\mathsf{read}}, v_{i^*}.u_{\mathsf{read}}).$$

Next, for all $i \in \{i^* + 1, \ldots, j^* - 1\}$:

$$u_i = \mathsf{H.Update}(\mathsf{hk}, v_i.i_{\mathsf{write}}, v_i.b_{\mathsf{write}}, v_i.u_{\mathsf{write}}, v_{i^*}.i_{\mathsf{read}}, u_{i-1}).$$

For type 2 attacker $\mathcal{A}$, the update algorithm fails to provide a valid opening u_{j^*-1}, such that $\mathsf{H.Verify}(\mathsf{hk}, v_{j^*}.\rho, v_{i^*}.i_{\mathsf{read}}, v_{i^*}.b_{\mathsf{read}}, u_{j^*-1}) = 0$. For type 3 attacker $\mathcal{A}$, the update algorithm successfully outputs a valid opening where $\mathsf{H.Verify}(\mathsf{hk}, v_{j^*}.\rho, v_{i^*}.i_{\mathsf{read}}, v_{i^*}.b_{\mathsf{read}}, u_{j^*-1}) = 1$.

Type 4**:** For type 4 attacker $\mathcal{A}$, there exists at least one pair of indexes $i^*, j^* \in [T]$ and $i^* < j^*$, such that the following conditions hold:

- $v_{i^*}.i_{\mathsf{write}} = v_{j^*}.i_{\mathsf{read}}$ and $v_{i^*}.b_{\mathsf{write}} \neq v_{j^*}.b_{\mathsf{read}}$,
- For all $i \in \{i^*+1, \ldots, j^*-1\}$, $v_i.i_{\mathsf{write}} \neq v_{i^*}.i_{\mathsf{write}}$.

The intuition is that type 4 attacker writes a specific bit to a memory location, but then reads a different bit from that same location.

Type 5: For type 5 attacker $\mathcal{A}$, there exists some index $i^* \in [T]$, such that the following conditions hold:

- $v_{i^*}.i_{\mathsf{read}} \in [n]$ and $v_{i^*}.b_{\mathsf{read}} \neq x_{v_{i^*}.i_{\mathsf{read}}}$,
- For all $i \in \{1, \ldots, i^*-1\}$, $v_i.i_{\mathsf{write}} \neq v_{i^*}.i_{\mathsf{read}}$.

Type 5 attacker reads a bit on the input tape that does not match the actual input x.

Type 6: For type 6 attacker $\mathcal{A}$, there exists some index $i^* \in [T]$, such that the following conditions hold:

- $v_{i^*}.i_{\mathsf{read}} \in \{n+m+1, \ldots, k\}$ and $v_{i^*}.b_{\mathsf{read}} \neq 0$,
- For all $i \in \{1, \ldots, i^*-1\}$, $v_i.i_{\mathsf{write}} \neq v_{i^*}.i_{\mathsf{read}}$.

Type 6 attacker reads non-zero bits on a working tape that should have been initialized to all zeros.

Claim 4.9. Assuming that Γ satisfies adaptive proof of knowledge, then type 1 attacker $\mathcal{A}$ has at most negligible advantage.

Proof. Assuming that Γ satisfies adaptive proof of knowledge, then algorithm $\mathcal{E}'$ satisfies the property defined in Lemma 4.7. Assume that type 1 attacker has non-negligible advantage, we design a reduction algorithm $\mathcal{B}$ that breaks such property.

$\mathcal{A}$ starts by outputting λ, n, m, T, k and RAM Machine $\mathcal{R}$. $\mathcal{B}$ outputs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}, 1^n, 1^m, T, k)$. $\mathcal{A}$ then outputs x and π. $\mathcal{B}$ outputs $\mathcal{A}$, x, and π.

By definition of type 1 attacker $\mathcal{A}$, let $(G, \{(\mathsf{inst}_v, \mathsf{proof}_v, \mathsf{wit}_v)\}_{v \in V}) \leftarrow \mathcal{E}'_{\mathcal{A}}(\mathsf{crs}, x, \pi)$ for $G = (V, E)$. By definition of a type 1 attacker, there exists some $v \in V$ at some level i, such that $\mathcal{C}_i(\mathsf{inst}_v, \mathsf{wit}_v) = 0$. Thus $\mathcal{B}$ breaks the property in Lemma 4.7 with non-negligible advantage.

□

Claim 4.10. Assume that the Hash Tree H satisfies soundness of updating, then type 2 attacker $\mathcal{A}$ has at most negligible advantage.

Proof. Assume that type 2 attacker $\mathcal{A}$ has non-negligible advantage, we design a reduction algorithm $\mathcal{B}$ that breaks the soundness of updating property of Hash Tree H.

$\mathcal{A}$ starts by outputting RAM Machine $\mathcal{R}$. Next, $\mathcal{B}$ queries the hash challenger and obtains hash key hk. $\mathcal{B}$ then generates h_3, $\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell$ and outputs crs as $(\mathsf{hk}, h_3, \mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$. Upon crs, $\mathcal{A}$ outputs x, π. Following from the definition of type 2 attacker, there exists some index i where $i^* \leq i \leq j^* - 1$, such that the updating of opening fails at the i-th node on the bottom level. Namely, the following conditions hold for some $i \in \{i^*, \ldots, j^*-1\}$:

– $\mathsf{H.Verify}(\mathsf{hk}, v_i.\rho, v_{i^*}.i_{\mathsf{read}}, v_{i^*}.b_{\mathsf{read}}, u_{i-1}) = 1$,

– $\mathsf{H.Verify}(\mathsf{hk}, v_{i+1}.\rho, v_{i^*}.i_{\mathsf{read}}, v_{i^*}.b_{\mathsf{read}}, u_i) = 0$.

For simplicity, we define u_{i^*-1} as $u_{i^*-1} = v_{i^*}.u_{\mathsf{read}}$ in the above equations. Recall that $u_i = \mathsf{H.Update}(\mathsf{hk}, v_i.i_{\mathsf{write}}, v_i.b_{\mathsf{write}}, v_i.u_{\mathsf{write}}, v_{i^*}.i_{\mathsf{read}}, u_{i-1})$. Also, it follows from the failure of type 1 attacker that $v_{i+1}.\rho = \mathsf{H.Write}(\mathsf{hk}, v_i.\rho, v_i.i_{\mathsf{write}}, v_i.b_{\mathsf{write}}, v_i.u_{\mathsf{write}})$. Thus $\mathcal{B}$ breaks the soundness of updating by outputting the above.

□

Claim 4.11. Assume that Hash Tree H satisfies collision resistance, then type 3 attaccker $\mathcal{A}$ has at most negligible advantage.

Proof. Assume that type 3 attacker $\mathcal{A}$ has non-negligible advantage, we design the following reduction algorithm $\mathcal{B}$.

$\mathcal{A}$ outputs $\mathcal{R}$. $\mathcal{B}$ queries the hash challenger and obtains hk. $\mathcal{B}$ generates h_3 and $\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell$ by itself and outputs $\mathsf{crs} = (\mathsf{hk}, h_3, \mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$. $\mathcal{A}$ outputs x and π. By definition of type 3 attacker, the following conditions hold:

– $\mathsf{H.Verify}(\mathsf{hk}, v_{j^*}.\rho, v_{i^*}.i_{\mathsf{read}}, v_{i^*}.b_{\mathsf{read}}, u_{j^*-1}) = 1$,

– $\mathsf{H.Verify}(\mathsf{hk}, v_{j^*}.\rho, v_{i^*}.i_{\mathsf{read}}, v_{j^*}.b_{\mathsf{read}}, v_{j^*}.u_{\mathsf{read}}) = 1$.

Since $v_{i^*}.b_{\mathsf{read}} \neq v_{j^*}.b_{\mathsf{read}}$, $\mathcal{B}$ breaks the collision resistance property by outputting the above.

□

Claim 4.12. Assume that Hash Tree H satisfies soundness of updating and collision resistance, then type $4, 5, 6$ attacker $\mathcal{A}$ has at most negligible advantage.

Proof. We omit the proof here as it relies on the Merkle Hash Tree and is nearly identical to proof of Claim 4.10 and Claim 4.11. □

Claim 4.13. With all but negligible probability, retrieved witness ω satisfies that $\mathcal{R}(x, \omega)$ accepts.

Proof. Since type $1, 2, 3, 4, 5, 6$ attacker only have negligible advantage, we conclude that with all but negligible probability, the RAM Machine properly transfers from the starting state to an accepting state, with consistent memory reading and writing. Witness ω is retrieved as the bits which are accessed and within the range for witness on memory tape. Thus we conclude that $\mathcal{R}(x, \omega)$ accepts with all but negligible probability. □

□

This completes the proof of our main theorem. □

Remark 4.14. We remark that the above the bootstrapping process can be extended to a λ^ϵ-mild SNARK of proof size $\frac{m}{\lambda^\epsilon} + p(\lambda, n)$ for any constant $\epsilon > 0$. This immediately follows by composing a λ^ϵ-mild SNARK a number of $\lceil 2/\epsilon \rceil$ times: It gives us a 2λ-mild SNARK with proof size $\frac{m}{\lambda^2} + p'(\lambda, n) \leq \frac{m}{2\lambda} + p'(\lambda, n)$ where $p'(\lambda, n) \leq 2 \cdot p(\lambda, n)$.

Remark 4.15. We remark that our design of fully succinct SNARK for RAM with bounded running time can be readily extended to unbounded RAM: For RAM machine $\mathcal{R}$ of a maximum running time 2^λ, we apply RAM delegation scheme del to prove that $\mathcal{R}(x,\omega) = 1$: $\mathsf{del}.\pi \leftarrow \mathsf{del}.\mathsf{Prove}(\mathsf{del}.\mathsf{crs}, (x,\omega))$. Next, define RAM machine $\mathcal{R}'$ such that $\mathcal{R}'(x,(\omega,\mathsf{del}.\pi)) = 1$ if and only if $\mathsf{del}.\mathsf{Verify}(\mathsf{del}.\mathsf{crs}, (x,\omega), \mathsf{del}.\pi) = 1$. By succinctness of RAM delegation, del.Verify runs in time $\mathsf{poly}(\lambda, n, m)$ for a universal $\mathsf{poly}(\cdot,\cdot,\cdot)$. By applying our SNARK for bounded RAM over $\mathcal{R}'$, we obtain a fully succinct SNARK for RAM with unbounded running time.

5 Full Succinctness from Milder SNARKs with Fast Extractors

In this section, we show to build fully-succinct SNARKs from milder SNARKs. Towards that goal, we introduce the notion of fast extractors.

γ-Efficient Extractor. We say a SNARK has a γ-efficient extractor, if the extractor $\mathcal{E}$'s running time grows as $\gamma \cdot |\mathcal{A}| + q(\lambda, n, m, |\pi|)$, for $\gamma = \gamma(\lambda, n, m, |\pi|)$ and some poly $q(\cdot)$. That is, γ for an extractor $\mathcal{E}(\mathcal{A}, \mathsf{crs}, x, \pi)$ is defined as the (asymptotic) ratio of the extractor's running time to the running time of attacker $\mathcal{A}$.

5.1 Fullly Succinct SNARK from $(1+\epsilon)$-Mild SNARK with γ-Efficient Extractor

Assuming the existence of $(1+\epsilon)$-mild SNARK $\Gamma = (\Gamma.\mathsf{Setup}, \Gamma.\mathsf{Prove}, \Gamma.\mathsf{Verify})$ with a γ-efficient extractor (and succinct CRS) for any $\epsilon, \gamma > 0$, we design a δ-mild SNARK for $\delta \geq 2\lambda$:

$\mathsf{Setup}(1^\lambda, 1^n, 1^m, \mathcal{C}) \to \mathsf{crs}$. It sets $\ell = 2 \cdot \lceil \frac{\log \lambda}{\log(1+\epsilon)} \rceil$. For $i \in [\ell]$, we set $\mathcal{C}_i$ as the following boolean circuit and m_i as the witness size for such circuit. It generates $\mathsf{crs}_i = \Gamma.\mathsf{Setup}(1^\lambda, 1^n, 1^{m_i}, \mathcal{C}_i)$. It outputs $\mathsf{crs} = \{\mathsf{crs}_1, \ldots, \mathsf{crs}_\ell\}$.

Circuit $\mathcal{C}_i$

Instance: x.
Witness: ω.
Hardwired: crs_{i-1}.

- For $i = 1$, $\mathcal{C}_i$ outputs $\mathcal{C}(x, \omega)$.
- For $i \geq 2$, $\mathcal{C}_i$ outputs $\Gamma.\mathsf{Verify}(\mathsf{crs}_{i-1}, x, \omega) = 1$.

$\mathsf{Prove}(\mathsf{crs}, x, \omega) \to \pi$. It parses crs as above. It sets ω_1 as $\omega_1 \leftarrow \Gamma.\mathsf{Prove}(\mathsf{crs}_1, x, \omega)$. Next for all $i \in \{2, \ldots, \ell\}$, it computes ω_i as $\Gamma.\mathsf{Prove}(\mathsf{crs}_i, x, \omega_{i-1})$. It outputs $\pi = \omega_\ell$.

$\mathsf{Verify}(\mathsf{crs}, x, \pi) \to \{0,1\}$. It outputs $\Gamma.\mathsf{Verify}(\mathsf{crs}_\ell, x, \pi)$.

Completeness The recursive composition inherits the completeness of Γ.

Lemma 5.1. Assuming that Γ satisfies $(1+\epsilon)$-mildness, then the above SNARK satisfies 2λ-mildness.

Proof. The proof size of $(1+\epsilon)$-mild SNARK is given as $m/(1+\epsilon)+p(\lambda,n)$. By composing this mild SNARK $\ell = 2 \cdot \lceil \frac{\log \lambda}{\log(1+\epsilon)} \rceil$ times in the above construction, the resulting proof size becomes $|\pi| \leq m/\lambda^2 + 2 \cdot p(\lambda, n)$.

□

Theorem 5.2. Assuming that Γ satisfies $(1+\epsilon)$-mildness with a γ-efficient extractor ($\gamma > 0$), then the above SNARK also satisfies adaptive proof of knowledge property, as long as γ and $(1+\epsilon)$ are polynomially related such that $\gamma = (1+\epsilon)^c$ where c is a constant.

Proof. We define extractor $\mathcal{E}$ of as the following:

$\mathcal{E}(\mathcal{A}, \mathsf{crs}, x, \pi) \to \omega$. The extractor algorithm follows these steps:

1. For $i = \ell$, it sets $\mathcal{E}_\ell$ as $\Gamma.\mathcal{E}$ with non-black-box access to $\mathcal{A}$, and witness ω_ℓ as the output: $\omega_\ell = \Gamma.\mathcal{E}(\mathcal{A}, \mathsf{crs}_i, x, \omega_i)$.
2. For $i = \ell - 1, \ldots, 1$, set $\mathcal{E}_i$ as $\Gamma.\mathcal{E}$ with non-black-box access to $\mathcal{E}_{i+1}$, and ω_i as $\Gamma.\mathcal{E}(\mathcal{E}_{i+1}, \mathsf{crs}_i, x, \omega_{i+1})$.
3. $\mathcal{E}(\mathcal{A}, \mathsf{crs}, x, \pi)$ outputs ω_1.

We analyze the running time of the extractor $\mathcal{E}$ using induction (assume that $\gamma > 1$):

Base Case $(i = \ell)$. $\mathcal{E}_\ell$ runs in time $\gamma \cdot |\mathcal{A}| + q(\lambda, n, m, |\pi|) \leq \gamma \cdot |\mathcal{A}| + (2\gamma - 1) \cdot q(\lambda, n, m, |\pi|)$.

Inductive Step $(1 \leq i \leq \ell - 1)$. The running time of $\mathcal{E}_i$ is

$$\gamma \cdot |\mathcal{E}_{i+1}| + q(\lambda, n, m, |\pi|) \leq \gamma \cdot (\gamma^{\ell-i} \cdot |\mathcal{A}| + (2\gamma^{\ell-i} - 1) \cdot q(\lambda, n, m, |\pi|)) \leq \gamma^{\ell+1-i} \cdot |\mathcal{A}| + (2\gamma^{\ell+1-i} - 1) \cdot q(\lambda, n, m, |\pi|).$$

Thus for $\gamma > 1$, the running time of $\mathcal{E}$ is at most $\gamma^\ell \cdot |\mathcal{A}| + (2\gamma^\ell - 1) \cdot q(\lambda, n, m, |\pi|)$. Recall that $\ell = 2 \cdot \lceil \frac{\log \lambda}{\log(1+\epsilon)} \rceil$, we conclude that $|\mathcal{E}| = \mathsf{poly}(\lambda^{\frac{2 \log \gamma}{\log(1+\epsilon)}}, |\mathcal{A}|, n, m)$. Thus for $\gamma = (1+\epsilon)^c$ where c is a constant, the extractor runs in polynomial time. We omit the analysis for $\gamma \leq 1$ as the extractor is even more efficient in this case. We omit the soundness proof for the argument of knowledge property. Readers are encouraged to refer to the inductive proof of Lemma 4.7, which also employs a leveled structure, although it is more complex. □

Corollary 5.3. Following from Lemma 4.4, Lemma 5.1, and Theorem 5.4, there exists a fully succinct SNARK assuming the existence of $(1+\epsilon)$-mild SNARK with $\gamma-$efficient extractor such that $\gamma = \mathsf{poly}(1+\epsilon)$.

Theorem 5.4. Assuming that Γ satisfies $(1+\epsilon)$-mildness with a γ-efficient extractor ($\gamma > 0$), then the above SNARK also satisfies adaptive proof of knowledge property, as long as γ and $(1+\epsilon)$ are polynomially related such that $\gamma = (1+\epsilon)^c$ where c is a constant.

Corollary 5.5. Following from Lemma 4.4, Lemma 5.1, and Theorem 5.4, there exists a fully succinct SNARK assuming the existence of $(1+\epsilon)$-mild SNARK with $\gamma-$efficient extractor such that $\gamma = \mathsf{poly}(1+\epsilon)$.

6 Mild SNARKs without Succinct CRS

Finally, we show how to use $\delta-$mild SNARK and RAM delegation to design a $\delta-$mild SNARK with succinct CRS.

Our construction. Let $\mathsf{del} = (\mathsf{del.Setup}, \mathsf{del.Prove}, \mathsf{del.Verify})$ be a RAM delegation scheme, and $\Gamma = (\Gamma.\mathsf{Setup}, \Gamma.\mathsf{Prove}, \Gamma.\mathsf{Verify})$ be a δ-mild SNARK. Below we provide our construction:

$\mathsf{Setup}(1^\lambda, 1^n, 1^m, \mathcal{C}) \to \mathsf{crs}$. Let $\mathcal{R}$ be the following RAM machine – $\mathcal{R}$ takes as input $(x, \omega) \in \{0,1\}^{n+m}$ and accepts if and only if $\mathcal{C}(x, \omega) = 1$. Running time of $\mathcal{R}$ is $T = \mathsf{poly}(|\mathcal{C}|)$ for some fixed polynomial $\mathsf{poly}(\cdot)$, and input size is $n + m$. The setup algorithms samples $\mathsf{del.crs} \leftarrow \mathsf{del.Setup}(1^\lambda, \mathcal{R}, T, n + m)$.

Let $|\mathsf{del}.\pi|$ denote the proof size generated by del using del.crs and a valid instance-witness pair. Consider the following boolean circuit $\mathcal{C}' = \{0,1\}^n \times \{0,1\}^{m+|\mathsf{del}.\pi|} \to \{0,1\}$:

Circuit $\mathcal{C}'$
Instance: It takes as input instance $x \in \{0,1\}^n$.
Witness: It takes as input witness $(\omega, \mathsf{del}.\pi) \in \{0,1\}^{m+|\mathsf{del}.\pi|}$.
Hardwired: It has del.crs hardwired.

- It outputs $\mathsf{del.Verify}(\mathsf{del.crs}, (x, \omega), \mathsf{del}.\pi)$.

The setup algorithm samples $\Gamma.\mathsf{crs}$ as $\Gamma.\mathsf{crs} \leftarrow \Gamma.\mathsf{Setup}(1^\lambda, 1^n, 1^{m+|\mathsf{del}.\pi|}, \mathcal{C}')$. And, it outputs crs as $(\mathsf{del.crs}, \Gamma.\mathsf{crs})$.

$\mathsf{Prove}(\mathsf{crs}, x, \omega) \to \pi$. It parses crs as above. First, it computes a RAM proof as $\mathsf{del}.\pi \leftarrow \mathsf{del.Prove}(\mathsf{del.crs}, (x, \omega))$, and then it creates a mild SNARK proof as $\Gamma.\pi \leftarrow \Gamma.\mathsf{Prove}(\Gamma.\mathsf{crs}, x, (\omega, \mathsf{del}.\pi))$. It outputs $\pi = \Gamma.\pi$.

$\mathsf{Verify}(\mathsf{crs}, x, \pi) \to \{0,1\}$. It outputs $\Gamma.\mathsf{Verify}(\Gamma.\mathsf{crs}, x, \pi)$.

Completeness The completeness of the above design follows from the completeness of RAM machine delegation scheme del and mild SNARK scheme Γ.

Efficiency We analyze the efficiency of the above design:

Lemma 6.1. Assume that $\Gamma = (\Gamma.\mathsf{Setup}, \Gamma.\mathsf{Prove}, \Gamma.\mathsf{Verify})$ is a $\delta-$mild SNARK for function $\delta(\lambda, n, m)$ with extractor $\Gamma.\mathcal{E}$. Then for every $\lambda \in \mathbb{N}$, polynomial $n = n(\lambda)$, $m = m(\lambda)$, boolean circuit $\mathcal{C} = \{0,1\}^n \times \{0,1\}^m$, any $x \in \{0,1\}^n$, and $\omega \in \{0,1\}^m$ such that $\mathcal{C}(x, \omega) = 1$, the following holds:

- For $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^m, \mathcal{C})$, there exists a universal polynomial $\mathsf{poly}(\cdot, \cdot, \cdot, \cdot)$ such that $|\mathsf{crs}| \leq \mathsf{poly}(\lambda, \log |\mathcal{C}|, n, m)$.
- For $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, \omega)$, there exists a universal polynomial $\mathsf{poly}(\cdot)$ such that $p'(\cdot) = \mathsf{poly}(p(\cdot))$, where $|\pi| \leq \frac{m}{\delta(\lambda,n,m)} + p'(\lambda, n)$.

Proof. Common reference string $\mathsf{crs} = (\mathsf{del.crs}, \Gamma.\mathsf{crs})$. $\mathsf{del.crs}$ was set as $\mathsf{del.Setup}(1^\lambda, \mathcal{R}, T, n + m)$ where $\mathcal{R}$ is the above RAM Machine and $T = \mathsf{poly}(|\mathcal{C}|)$ is the running time of $\mathcal{R}$. By efficiency of RAM delegation, size of $\mathsf{del.crs}$ is at most $\mathsf{poly}(\lambda, \log T)$. Next, $\Gamma.\mathsf{crs}$ was set as $\Gamma.\mathsf{Setup}(1^\lambda, 1^n, 1^{m+|\mathsf{del}.\pi|}, \mathcal{C}')$. Here, $\mathcal{C}'$ is the circuit of RAM Delegation's Verifier $\mathsf{del.Verify}$, which runs in time $\mathsf{poly}(\lambda, \log T, n, m)$. Thus $|\mathcal{C}'| \leq \mathsf{poly}(\lambda, \log T, n, m)$. Then, size of $\Gamma.\mathsf{crs}$ is at most $\mathsf{poly}(\lambda, \log T, n, m)$. Thus overall crs is of size $\mathsf{poly}(\lambda, \log T, n, m) = \mathsf{poly}(\lambda, \log |\mathcal{C}|, n, m)$ for some universal polynomial $\mathsf{poly}(\cdot, \cdot, \cdot, \cdot)$.

Recall that proof π is generated from $\Gamma.\mathsf{Prove}(\Gamma.\mathsf{crs}, x, (\omega, \mathsf{del}.\pi))$. By succinctness of Γ, proof size $|\pi|$ is at most $\frac{m+|\mathsf{del}.\pi|}{\delta(\lambda,n,m)} + p(\lambda, n)$. By succinctness of RAM delegation del, $|\mathsf{del}.\pi| \leq \mathsf{poly}(\lambda, \log T)$. Thus proof size $|\pi| \leq \frac{m+|\mathsf{del}.\pi|}{\delta(\lambda,n,m)} + p(\lambda, n)^c$ for some constant c, which implies that there exists a universal polynomial $\mathsf{poly}(\cdot)$ such that $|\pi| \leq \frac{m}{\delta(\lambda,n,m)} + p'(\lambda, n)$ for $p'(\cdot) = \mathsf{poly}(p(\cdot))$. □

By Lemma 6.1, our design satisfies both δ-mildness and succinct CRS properties.

Theorem 6.2. Assume that the mild SNARK scheme Γ satisfies adaptive proof of knowledge, and the RAM delegation scheme del is sound, then the above design of SNARK satisfies adaptive proof of knowledge.

Proof. We first define the extractor of the above design:

$\mathcal{E}(\mathcal{A}, \mathsf{crs}, x, \pi) \to \omega$. The algorithm takes as input common reference string crs, statement x, and proof π. It parses crs as $(\mathsf{del.crs}, \Gamma.\mathsf{crs})$, and runs extractor to compute $(\omega, \mathsf{del}.\pi) \leftarrow \Gamma.\mathcal{E}(\mathcal{A}, \Gamma.\mathsf{crs}, x, \pi)$. Finally, it outputs ω.

The security proof follows from a case by case reduction to the soundness of RAM delegation scheme del and adaptive proof of knowledge property of wsnark. Suppose there exists a PPT attacker $\mathcal{A}$ that wins the argument of knowledge game with non-negligible advantage. We divide the potential adversary into the following cases:

Type 1 : Upon common reference string crs, type 1 attacker $\mathcal{A}$ outputs (x, π) such that $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$, but $\mathcal{C}(x, \omega) \neq 1$ and $\mathsf{del.Verify}(\mathsf{del.crs}, (x, \omega), \mathsf{del}.\pi) \neq 1$ where $(\omega, \mathsf{del}.\pi)$ is extracted by $\Gamma.\mathcal{E}(\mathcal{A}, \Gamma.\mathsf{crs}, x, \pi)$.

Type 2 : Type 2 attacker is considered as the complimentary of type 1. It outputs (x, π), such that $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$, $\mathcal{C}(x, \omega) \neq 1$ and $\mathsf{del.Verify}(\mathsf{del.crs}, (x, \omega), \mathsf{del}.\pi) = 1$ where $(\omega, \mathsf{del}.\pi) \leftarrow \Gamma.\mathcal{E}(\mathcal{A}, \Gamma.\mathsf{crs}, x, \pi)$.

Claim 6.3. Assuming that Γ satisfies proof of knowledge, then type 1 attacker $\mathcal{A}$ has at most negligible advantage.

Proof. Assume that type 1 attacker $\mathcal{A}$ has non-negligible advantage, we design a reduction algorithm $\mathcal{B}$ that breaks proof of knowledge property of Γ.

$\mathcal{A}$ starts by outputting the circuit $\mathcal{C}$. Reduction algorithm $\mathcal{B}$ sets up $\mathsf{del.crs}$ using $\mathsf{del.Setup}(1^\lambda, 1^n, 1^m, \mathcal{C})$. Next $\mathcal{B}$ sets circuit $\mathcal{C}'$ following from the definition in the setup algorithm above, and queries wsnark challenger with circuit $\mathcal{C}'$. Challenger outputs $\Gamma.\mathsf{crs}$ accordingly. $\mathcal{B}$ then sends $(\mathsf{del.crs}, \Gamma.\mathsf{crs})$ to $\mathcal{A}$. $\mathcal{A}$ then outputs (x, π) such that $\Gamma.\mathsf{Verify}(\Gamma.\mathsf{crs}, x, \pi) = 1$ and $\mathcal{C}(x, \omega) \neq 1$. Next consider

that $\mathcal{B}$ extracts $(\omega, \mathsf{del}.\pi) \leftarrow \Gamma.\mathcal{E}(\mathcal{A}, \Gamma.\mathsf{crs}, x, \pi)$. Then, by definition of type 1 attacker, it follows that $\mathsf{del.Verify}(\mathsf{del.crs}, (x, \omega), \mathsf{del}.\pi) \neq 1$, which implies that $\mathcal{C}'(x, (\omega, \mathsf{del}.\pi)) \neq 1$. Thus, $\mathcal{B}$ outputs statement x and proof π and breaks the proof of knowledge property of Γ. □

Claim 6.4. Assuming that RAM delegation scheme del satisfies soundness property, then type 2 attacker $\mathcal{A}$ has at most negligible advantage.

Proof. Assume that type 2 attacker $\mathcal{A}$ has non-negligible advantage, we design a reduction algorithm $\mathcal{B}$ that breaks the soundness property of del.

$\mathcal{A}$ starts by outputting the circuit $\mathcal{C}$. Reduction algorithm $\mathcal{B}$ sets RAM machine $\mathcal{R}$ as defined in Setup algorithm according to circuit $\mathcal{C}$. Consider that the running time bound on $\mathcal{R}$ is T. $\mathcal{B}$ then queries the challenger with $1^\lambda, \mathcal{R}, T, n+m$. Challenger outputs del.crs. Next $\mathcal{B}$ sets up circuit $\mathcal{C}'$ and generates $\Gamma.\mathsf{crs} = \Gamma.\mathsf{Setup}(1^\lambda, 1^n, 1^m, \mathcal{C}')$. $\mathcal{B}$ sends $(\mathsf{del.crs}, \mathsf{wsnark.crs})$ to $\mathcal{A}$. $\mathcal{A}$ outputs tuple (x, π). $\mathcal{B}$ then extracts $(\omega, \mathsf{del}.\pi)$ from extractor $\Gamma.\mathcal{E}(\mathcal{A}, \Gamma.\mathsf{crs}, x, \pi)$, such that $\Gamma.\mathsf{Verify}(\Gamma.\mathsf{crs}, x, \pi) = 1$ and $\mathcal{C}(x, \omega) \neq 1$. Note that $\mathcal{C}(x, \omega) \neq 1$ implies that $\mathcal{R}(x, \omega) \neq 1$. Then, $\mathcal{B}$ outputs $(x, \omega), \mathsf{del}.\pi$ and breaks the soundness property of del. □

This completes the proof of our theorem. □

Corollary 6.5. Based on recent results about RAM delegation (Corollary 3.7) and our SNARKs design (Lemma 4.4, Theorem 4.5, Remark 4.14, Lemma 6.1, and Theorem 6.2), the following holds: Assuming the existence of λ^ϵ-mild SNARKs without CRS succinctness requirement, and assuming either LWE, k-LIN over pairing groups for any constant $k \in \mathbb{N}$, or sub-exponential DDH over pairing-free groups (and QR), there exists fully succinct SNARKs.

Corollary 6.6. By extending Corollary 6.5 using Corollary 5.5, we have: Assuming the existence of $(1+\epsilon)$-mild SNARKs with $\gamma-$efficient extractor for $\gamma = \mathsf{poly}(1+\epsilon)$, and without CRS succinctness requirement, and assuming either LWE, k-LIN over pairing groups for any constant $k \in \mathbb{N}$, or sub-exponential DDH over pairing-free groups (and QR), there exists fully succinct SNARKs.

References

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 252–283. Springer, Heidelberg, August 2023.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of computer and system sciences*, 37(2):156–189, 1988.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, October 2017.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 326–349. Association for Computing Machinery, January 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 111–120. ACM Press, June 2013.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 315–333. Springer, Heidelberg, March 2013.

[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, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, August 2001.

[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 Annual ACM Symposium on Theory of Computing*, pages 474–482. ACM Press, June 2017.

[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 Annual ACM Symposium on Theory of Computing*, pages 709–721. ACM Press, June 2018.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73. ACM Press, November 1993.

[CGKS23] Matteo Campanelli, Chaya Ganesh, Hamidreza Khoshakhlagh, and Janno Siim. Impossibilities in succinct arguments: Black-box extraction and more. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23: 14th International Conference on Cryptology in Africa*, volume 14064 of *Lecture Notes in Computer Science*, pages 465–489, July 2023.

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 394–423, Virtual Event, August 2021. Springer, Heidelberg.

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd Annual Symposium on Foundations of Computer Science*, pages 68–79. IEEE Computer Society Press, February 2022.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS*, volume 10, pages 310–331, 2010.

[Dam89] Ivan Bjerre Damgård. A design principle for hash functions. In *Conference on the Theory and Application of Cryptology*, pages 416–427. Springer, 1989.

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

[GGI+15] Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *Journal of Cryptology*, 28(4):820–843, October 2015.

[GK16] Shafi Goldwasser and Yael Tauman Kalai. Cryptographic assumptions: A position paper. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 505–522, January 2016.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340. Springer, Heidelberg, December 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, Heidelberg, May 2016.

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

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of

Lecture Notes in Computer Science, pages 520–549. Springer, Heidelberg, May / June 2022.

[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, *53rd Annual ACM Symposium on Theory of Computing*, pages 708–721. ACM Press, June 2021.

[JKLV24] Zhengzhong Jin, Yael Kalai, Alex Lombardi, and Vinod Vaikuntanathan. SNARGs under LWE via propositional proofs. In *56th Annual ACM Symposium on Theory of Computing*, pages 1750–1757. ACM Press, June 2024.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM Symposium on Theory of Computing*, pages 60–73. ACM Press, June 2021.

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

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and ram delegation. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1545–1552, 2023.

[KP16] Yael Tauman Kalai and Omer Paneth. Delegating RAM computations. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 91–118. Springer, Heidelberg, October / November 2016.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1115–1124. ACM Press, June 2019.

[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: 19th Theory of Cryptography Conference, Part I*, volume 13042 of *Lecture Notes in Computer Science*, pages 330–368. Springer, Heidelberg, November 2021.

[Mer79] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* Stanford university, 1979.

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, Heidelberg, August 1988.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[Nao03] Moni Naor. On cryptographic assumptions and challenges (invited talk). In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 96–109. Springer, Heidelberg, August 2003.

[NWW23] Shafik Nassar, Brent Waters, and David J Wu. Monotone policy bargs from bargs and additively homomorphic encryption. *Cryptology ePrint Archive*, 2023.

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

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484, 2014.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, March 2008.

[WB15] Michael Walfish and Andrew J Blumberg. Verifying computations without reexecuting them. *Communications of the ACM*, 58(2):74–84, 2015.

[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, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 433–463. Springer, Heidelberg, August 2022.