Public-Coin, Complexity-Preserving, Succinct Arguments of Knowledge for NP from Collision-Resistance

Cody Freitag[1(✉)], Omer Paneth[2], and Rafael Pass[3]

[1] Northeastern University, Boston, USA
c.freitag@northeastern.edu
[2] Tel Aviv University, Tel Aviv, Israel
omerpa@mail.tau.ac.il
[3] Tel Aviv University & Cornell Tech, Tel Aviv, Israel
rafaelp@tau.ac.il

Abstract. Succinct arguments allow a powerful (yet polynomial-time) prover to convince a weak verifier of the validity of some NP statement using very little communication. A major barrier to the deployment of such proofs is the unwieldy overhead of the prover relative to the complexity of the statement to be proved. In this work, we focus on *complexity-preserving* arguments where proving a non-deterministic time t and space s RAM computation takes time $\tilde{O}(t)$ and space $\tilde{O}(s)$.

Currently, all known complexity-preserving arguments either are private-coin, rely on non-standard assumptions, or provide only weak succinctness. In this work, we construct complexity-preserving succinct argument based solely on collision-resistant hash functions, thereby matching the classic succinct argument of Kilian (STOC '92).

1 Introduction

In an interactive proof system, a powerful prover tries to convince a weak verifier the validity of some statement over potentially many rounds of interaction. In order to be meaningful, such a protocol needs to satisfy *completeness*—an honest prover will successfully convince the verifier of a true statement—and *soundness*—a cheating prover cannot convince the verifier of a false statement (at least with high probability). Proof systems where soundness only holds with respect to computationally bounded (i.e. polynomial-time) provers are known as *arguments*. In this work, we focus on *succinct* arguments where the verifier's running time and the communication between the prover and verifier are extremely small, essentially independent of the complexity of the underlying statement.

Building upon the machinery of probabilistically checkable proofs (PCPs) [2], Kilian [58] gave the first succinct argument for NP, where soundness only relies on the existence of a collision-resistant hash function (CRH). Even though the prover in Kilian's protocol is theoretically "efficient"—the prover runs in

M. Joye and G. Leander (Eds.): EUROCRYPT 2024, LNCS 14654, pp. 112–141, 2024.
https://doi.org/10.1007/978-3-031-58737-5_5

polynomial-time as a function of the complexity of underlying NP statement—the concrete overheads of PCPs have made such an argument system prohibitively expensive in practice. Over the last 30 years, much progress has been made on improving the overheads of PCPs (see e.g. [13,15–17,39,65]) even at the cost of additional rounds of communication (see e.g. [12,14,68,69]). Despite all of this progress, there is still a fundamental barrier to making Kilian's protocol (or similar variants) practically efficient. The issue is that the prover needs to seemingly store the entire PCP in memory, meaning the high PCP costs are not only felt in the time to compute the proof but also in the space required by the prover.

Complexity-Preserving Succinct Arguments. Motivated by the above issue, Valiant [73] proposed the notion of *complexity-preserving* succinct arguments where, in order to prove a statement with an NP verifier running in time t and space s, the prover only needs to use time $\tilde{O}(t)$ and space $\tilde{O}(s)$.[1] Notably, the prover cannot use space that depends even linearly on the time t to verify the NP relation. Complexity-preserving SNARKs can be from standard SNARKs based on the idea of recursive proof composition. Valiant [73] and, subsequently, Bitansky et al. [19] constructed complexity-preserving succinct *non-interactive* arguments of knowledge (SNARKs) for NP based on plain SNARKs for NP, via recursive proof composition. The existence of SNARKs for NP, however, is a non-falsifiable "knowledge" assumption, and there are known barriers for basing SNARKs on falsifiable assumption [46].

Towards basing complexity-preserving, succinct arguments on more standard assumptions, Bitansky and Chiesa [21] show how to construct a complexity-preserving multi-prover interactive proof (MIP), which can then be compiled into a complexity-preserving succinct interactive argument using fully homomorphic encryption (FHE). Compared to Kilian's protocol, their succinct argument has two main downsides. First, they rely on the heavier cryptographic machinery of FHE rather than only collision-resistance. Second, and more notably, their transformation results in a private-coin rather than a public-coin protocol. This means that their verifier needs to keep private state hidden from the prover in order for the protocol to maintain soundness. In contrast, the verifier in Kilian's protocol maintains no private state and simply sends public random coins for each of its messages. Public-coin protocols can be verified publicly, which is crucial for increasingly important applications of proofs in the distributed setting. Indeed, this was one of the main open questions posed in [21]:

Do there exist public-coin complexity-preserving succinct arguments for NP whose security can be based on standard cryptographic assumptions?

Some recent works provide initial indications towards a positive resolution, but they all either rely on non-standard assumptions or fall short of achieving full succinctness. Block, Holmgren, Rosen, Rothblum, and Soni [23] construct a

[1] For simplicity, we suppress polynomial factors in the security parameter in the introduction.

public-coin argument for NP with succinct communication in the random oracle model (ROM) assuming hardness of discrete log.[2] However, the verifier in their protocol runs in time $\tilde{O}(t)$ so the argument is not "fully" succinct.[3] Finally, an even more recent paper by Block, Holmgren, Rosen, Rothblum, and Soni [24] construct the first public-coin, complexity-preserving, *fully succinct* argument for NP in the "plain" model (i.e., without a random oracle). This construction uses $O(\log t)$ rounds of communication and relies on the assumption that a group of hidden order can be sampled using public coins—the only candidate such groups are class groups of an imaginary quadratic field, which were first suggested for cryptographic use by [37] but have seen relatively little attention as a cryptographic assumption.

Bangalore, Bhadauria, Hazay, and Venkitasubramaniam [4] construct complexity preserving arguments based only on *black-box* use of CRH, matching the assumptions for Kilian's protocol. Their protocol, however, is not fully succinct as it requires communication $\tilde{O}(t/s)$ and verifier running time $\tilde{O}(t/s + s)$. Furthermore, they demonstrate barriers for getting a fully succinct protocol in this setting.

Even when allowing for private coins, we emphasize that the protocol of [21] assumes FHE, which is a significantly stronger assumption than CRH, required for Kilian's original (non-complexity-preserving) protocol. Viewing these assumptions qualitatively through the lens of Impagliazzo's five worlds [50], FHE is a "Cryptomania" assumption compared to the substantially weaker "Minicrypt" assumption of CRH.[4] Very recent work by [38,53,66] obtain publicly verifiable complexity-preserving SNARGs from a host of other assumptions (not all known to imply FHE), but these SNARG are only for languages in P; furthermore, these assumptions are still Cryptomania assumptions. This raises the following question:

Do (even private-coin) complexity-preserving succinct arguments for NP (or even P) exist in Minicrypt?

1.1 Our Results

We resolve both of the above open problems, showing the existence of a public-coin, complexity-preserving, succinct argument for all non-deterministic, polynomial-time RAM computation based solely on collision-resistance. Similar to [21] (as well as Kilian's protocol [58] with a suitable underlying PCP), we actually

[2] Even though SNARKs for NP can be built in the ROM [63], the construction of [19] makes non-black-box use of the underlying SNARK verifier, so it is not clear how to prove their construction secure in the ROM.

[3] Note that this is still non-trivial since their scheme has succinct communication.

[4] Strictly speaking, Impagliazzo defined Minicrypt as the potential world where one-way functions (and hence symmetric-key encryption) exist yet public-key encryption does not. Technically, one-way functions do not generically imply CRH in a black-box way [71]. However, CRH are still often considered to be a Minicrypt assumption and viewed as a much weaker assumption than the existence of public-key encryption.

give an *argument of knowledge* for NP. This is a stronger property that intuitively stipulates that if the prover convinces the verifier on some NP statement, it must actually "know" a corresponding witness.

Theorem 1.1 (Informal; see Theorem 4.1). *Assuming the existence of collision-resistant hash functions, there exists a public-coin, complexity preserving, succinct argument of knowledge for NP. On input statements for security parameter λ and time bound t, the protocol requires $O(\log_\lambda t)$ rounds of communication.*

We note that we require slightly super constant round complexity for our protocol to be sound for all non-deterministic (not a priori bounded) $\mathsf{poly}(\lambda)$-time computations. Specifically, for every bound c such that the RAM computation requires time λ^c on inputs of length λ, we give a protocol that uses $6c+6$ messages of communication. So, there is no fixed constant bounding the number of rounds for a protocol that handles all non-deterministic, polynomial-time computation. It is a fascinating open problem to give a protocol with a fixed constant number of rounds.

Our construction is based on recursive proof composition similar to [19,73]. However, we compose *interactive* arguments rather than SNARKs, so we are able to get a result based only on CRH. To implement our recursive proof composition, we make non-black box use of the underlying hash function. This allows us to circumvent the barrier of [4].

Applications of Our Main Theorem. We get the following applications using our construction from Theorem 1.1:

- **Zero-knowledge:** We can generically transform our protocol into one that satisfies zero-knowledge, based on the techniques of [8] adapted to the setting of public-coin, succinct arguments of knowledge. In a bit more detail, we have the prover first commit to its messages and then prove in zero-knowledge that it can open the commitments to messages that would cause the verifier to accept. Using the constant-round, public-coin, zero-knowledge protocol of [5,6] based on CRH, this results in a zero-knowledge protocol that preserves the succinctness and public-coin properties at the cost of a constant number of additional rounds. See the full version for full details.
- **Parallelizability (SPARKs):** Applying the transformation of [40] to our construction, we can construct a public-coin, complexity-preserving, succinct *parallelizable* argument of knowledge (SPARK) for NP in $\mathsf{poly}(\log t)$ rounds from collision-resistance. In a SPARK, the prover leverages $\mathsf{poly}(\log t)$ parallel processors in order run in nearly optimal parallel time $t + \mathsf{poly}(\log t)$ (with no multiplicative overhead). This construction resolves an open problem from [40] (Section 8.1).
- **Non-interactive SNARGs and SPARGs in the ROM:** Since our construction is a public-coin protocol, it can be made non-interactive via the Fiat-Shamir transform [42] by replacing the verifier's public-coin messages with a hash of the communication transcript so far. Soundness of this transformation holds for all non-deterministic polynomial-time computations in

the ROM by the analysis of [47].[5] This yields the first *non-interactive*, public-coin, complexity-preserving, succinct argument for NP based on *any* standard assumption in the ROM.[6] (Recall that previous such results either rely on non-standard assumptions [19,24], or have a linear-time verifier [23] and still rely on Cryptomania assumptions.)
Applying the transformation of [40] to our non-interactive protocol results in a non-interactive, complexity-preserving, succinct parallelizable argument (SPARG) in the ROM. The corresponding complexity-preserving construction of [40] relied on recursive composition of SNARKs à la [19], where security is not known to hold in the ROM.

– **Tighter memory-hard VDFs in the ROM:** We note that [43] construct a non-complexity-preserving, non-interactive SPARG for P from LWE, which yields generic constructions of verifiable delay functions (VDFs) from any inherently sequential function. Furthermore, if the underlying sequential function requires "large" memory usage,[7] this is preserved by the transformation, so they also get a memory-hard VDF assuming any memory-hard sequential function.[8] However, since their non-interactive SPARG is not space-preserving, their honest evaluation algorithm requires much more space than a potential adversary who does not need to compute the corresponding proof. As our non-interactive protocol is complexity-preserving, it yields a memory-hard VDF in the ROM based on CRH and any memory-hard sequential function with a much tighter memory gap for the honest and adversarial evaluations than [43].[9] Recall that the complexity-preserving, non-interactive SPARK for NP from [40] also implies a tighter memory-hard VDF, but they

[5] [47] proves security of Fiat-Shamir in the ROM for constant-round arguments with negligible soundness error. We can apply their analysis to our protocol since for every fixed polynomial-bound on the NP verification time, our protocol only requires a constant number of rounds.

[6] It isn't clear how to argue the resulting non-interactive protocol is an argument *of knowledge* in the ROM. In particular, Valiant [73] first showed that Micali's protocol [63] is an argument of knowledge, which requires that the CRH be extractable/implemented by a random oracle. A corresponding argument does not immediately hold in the ROM for protocols based on recursive composition.

[7] Memory-hardness can be formalized in many ways, but the application of SPARKs does not depend on the exact formulation; see [40] for further discussion on this point.

[8] Plain VDFs are useful for generating shared randomness in the distributed setting of blockchains, and memory-hardness further resists the use of energy-wasteful ASICs for this task, based on the assumption that most modern CPUs are already heavily optimized for memory accesses.

[9] We note that all current candidate constructions of memory-hard sequential functions are proven secure in the ROM, which we then have to heuristically instantiate before applying our non-interactive, complexity-preserving, SPARK. As such, we need to assume the existence of a memory-hard sequential function. It would be very interesting to directly construct an unconditionally secure memory-hard VDF in the ROM.

rely on the existence of SNARKs and there is no proof of security in the ROM.

1.2 Technical Overview

In this section, we provide a high-level overview for the main ideas underlying our construction. Throughout the overview, we will consider a fixed non-deterministic RAM computation that runs in time t and space s. Furthermore, we'll assume for simplicity that $t, s \leq 2^\lambda$ for security parameter λ, and we will suppress λ, $\log t$, and $\log s$ terms in asymptotic statements.

We proceed in three steps to motivate our full construction. First, we provide a warm-up protocol where the prover runs in time $t \cdot \mathsf{poly}(s)$ and space $\sqrt{t} \cdot \mathsf{poly}(s)$. Then, we show how to implement this idea recursively where the prover's space, verifier's running time, and communication all grow with $\mathsf{poly}(s)$. In other words, this protocol is complexity-preserving and succinct for small-space computations. Finally, we show how to use this to handle non-deterministic computations of arbitrary space, where the prover runs in time $t \cdot \mathsf{poly}(\lambda)$ and space $s \cdot \mathsf{poly}(\lambda)$.

Warm-Up: A $(\sqrt{t} \cdot \mathsf{poly}(s))$-Space Solution. We view the RAM computation as transitioning between a sequence of size s configurations over fixed updates consisting of some polynomial $\alpha = \alpha(s)$ steps. Specifically, let M be an α-time non-deterministic function that on input a state st and witness w, outputs a new state st' of size s in time $\alpha(s)$. We consider the associated update language $\mathcal{L}_{\mathsf{Upd},\alpha}$ consisting of instances $(M, \mathsf{st}, \mathsf{st}', t)$ such that there exists a sequence of t witnesses $w_1, \ldots, w_t$ such that starting with initial state $\mathsf{st}_0 := \mathsf{st}$, computing $\mathsf{st}_j := M(\mathsf{st}_{j-1}, w_j)$ for all $j \in [t]$, results in a final state $\mathsf{st}_t = \mathsf{st}'$.

We start by recalling how Kilian's [58] 4-message succinct argument works with the setup as above. The verifier starts by sending a hash key. Next, the prover writes down a probabilistically checkable proof (PCP) for the statement that $(M, \mathsf{st}, \mathsf{st}', t) \in \mathcal{L}_{\mathsf{Upd},\alpha}$ using witnesses $w_1, \ldots, w_t$. The prover uses a hash tree [61] to commit to this PCP and sends the verifier the associated hash root, which acts as a succinct digest of the PCP. The verifier then asks the prover to open a few random locations in the PCP, with associated local opening proofs with respect to the provided hash root. The prover opens the corresponding locations in the PCP, and the verifier accepts if the PCP verifier would have accepted these responses and all of the openings are valid.

The issue with above approach is that the prover needs to store the entire PCP, which requires space $t \cdot \mathsf{poly}(s)$ even with the most efficient PCPs.[10] The use of PCPs, however, is extremely useful for reducing the necessary communication and run-time of the verifier. We want to leverage the efficiency benefits of PCPs without having to store a PCP for the entire computation. So, we observe that

[10] To the best of our knowledge, the most space-efficient PCP construction is due to [13], where they show how, given the size $O(t \cdot s)$ tableau of the computation, you can efficiently compute each bit of the PCP in low depth. It is a fascinating open question to construct complexity-preserving PCPs that don't require storing the entire computation tableau.

we can do this by only computing a PCP for a part of the computation instead of the full computation. Specifically, for some choice of k (assuming t is a multiple of k for simplicity), we split the computation into k sub-statements each of size t/k,

$$(M, \mathsf{st}_0, \mathsf{st}_{t/k}, t/k), \ldots, (M, \mathsf{st}_{(k-1)\cdot t/k}, \mathsf{st}_t, t/k),$$

where each st_i corresponds to the ith state when iteratively computing M using witnesses $w_1, \ldots, w_t$. The prover will then compute and commit to a PCP for each of the k sub-statements. The verifier will then send PCP queries as in Kilian's protocol to open up each PCP and accept if all such PCPs accept.

We proceed to analyze the efficiency of this warm-up protocol, starting with the space complexity of the prover. As the sub-statements are independent of each other, the prover can compute each PCP with associated hash root independently and then forget the full expensive PCP. At any point in time, the prover only needs to store a single PCP and associated hash tree for t/k steps of computation as well as k hash roots. Together, this only requires space $(k + (t/k)) \cdot \mathsf{poly}(s)$ to compute the first message. The dependence on t is minimized by setting $k = \sqrt{t}$, resulting in space complexity $\sqrt{t} \cdot \mathsf{poly}(s)$. Furthermore, note that the prover is deterministic, so whenever the prover needs to provide PCP openings in the next round for the verifier, the prover can recompute the PCP and hash tree as needed using an additional pass over the underlying witnesses.

In minimizing the space complexity of the prover above, we have actually lost the original succinctness of Kilian's protocol. Namely, as k is set larger in the above protocol, the communication and verifier efficiency suffer. The prover has to send the k intermediate statements and PCP openings corresponding to each statement, and the verifier then has to check all k PCP proofs. As such, the communication and verifier's run-time grow with $k = \sqrt{t}$ when minimizing the prover's space complexity. Additionally, if we choose k to be smaller to satisfy succinctness, the prover's space complexity grows to be essentially as large as Kilian's protocol. So, have we really gained anything?

Towards constructing a full-fledged succinct argument with a space-efficient prover, we separately tackle the issues of succinctness and prover's space complexity as follows:

- **Fixing succinctness via "commit-and-prove":** In order to reduce the communication and verifier's efficiency, we can generically use a standard "commit-and-prove" technique. Namely, instead of having the prover send messages for all k sub-protocols, the prover will hash the messages together and send a fixed size digest committing to the various messages in each round. In response to each digest sent, the verifier will send a single public-coin message that can be reused across all k statements. At the end of this "committed" interaction by the prover, the prover will use a succinct argument to prove that it "knows" a set of messages that would have convinced the verifier in all k sub-protocols. Using Kilian's protocol for this succinct argument, this adds 4 messages to the interaction while ensuring that the communication and verifier complexity are always succinct and independent of k.

- **Fixing prover's space complexity via recursion:** Using the blueprint above, the prover's space complexity grows with the complexity to prove a single sub-statement. So, we will instead use a more space-efficient protocol to prove each sub-statement rather than Kilian's—like the one we just built! This leads to a natural recursive approach to reduce the prover's space complexity at the cost of an increase in round complexity.

We next show how to apply both of these fixes simultaneously to construct a complexity-preserving succinct argument for bounded space computation.

Handling Small-Space, Non-deterministic Computation. Throughout, we let λ denote the security parameter and consider input statements of the form $(M, \mathsf{st}, \mathsf{st}', t) \in \mathcal{L}_{\mathsf{Upd},\alpha}$ as defined above where st and st' are size s states. By "small-space", we mean that $s \in \mathsf{poly}(\lambda)$ independent of t.

For every $r \geq 0$, we recursively construct a public-coin proof system (P_r, V_r) for the above update language $\mathcal{L}_{\mathsf{Upd},\alpha}$. For every r, we will maintain the following invariants on the efficiency of (P_r, V_r):

- P_r runs in time $t \cdot \mathsf{poly}(s, r)$ and space $(t/\lambda^r) \cdot \mathsf{poly}(s, r)$,
- V_r runs in time $\mathsf{poly}(s, r)$,
- and the communication is at most $\mathsf{poly}(\lambda)$ per message.

Our final protocol will then simply set $r = \log_\lambda t$, yielding a complexity-preserving protocol for bounded space computations.

For the base case of $r = 0$, (P_0, V_0) simply runs Kilian's succinct argument. We saw above that the prover runs in time and space $t \cdot \mathsf{poly}(s)$, and the verifier and communication complexity are at most polynomial in the statement size, which in this case is at most $\mathsf{poly}(s)$ since $\mathsf{st}, \mathsf{st}'$ are part of the statement. So, the required invariants hold.

In the general case of $r > 0$, we split the proof into two phases: (1) committing to sub-proofs, and (2) recursive proof merging:

- **Committing to sub-proofs:** P_r will split the computation of t steps into λ many sub-computations of t/λ steps each. Rather than directly proving each sub-computation using (P_{r-1}, V_{r-1}), P_r will instead succinctly commit to prover messages for P_{r-1} for all sub-computations in each round. Since V_{r-1} is public-coin, it maintains no private state, so it can send a single message per round that can be used across all sub-computations.
- **Recursive proof merging:** Next, the prover P_r will use Kilian's succinct argument to prove that it committed to prover messages in the previous phase that would cause V_{r-1} to accept on all λ sub-computations.

Completeness and argument of knowledge for (P_r, V_r) follow in a straightforward manner assuming (P_{r-1}, V_{r-1}) and Kilian's protocol are complete, arguments of knowledge. Briefly, to show that (P_r, V_r) is an argument of knowledge, we show how to use an extractor for Kilian's protocol to build a cheating prover for (P_{r-1}, V_{r-1}) by extracting out the committed prover messages (which will be unique and convincing assuming collision-resistance of the commitments).

Given this cheating prover, we can use the extractor for (P_{r-1}, V_{r-1}) to extract out witnesses for the sub-computations, which can simply be pieced together to form an overall witness for the full sequence of updates. So, the only assumption we rely on is collision-resistance. However, the running time of the extractor will grow exponentially with r, but for $\mathsf{poly}(\lambda)$-time computations, $r = \log_\lambda t$ will only be a constant.

Below, we briefly argue why each efficiency property holds separately:

- **Round complexity:** The round complexity of (P_r, V_r) adds a constant number of rounds over (P_{r-1}, V_{r-1}). So, in total, the protocol will consist of $O(r)$ rounds.
- **Communication complexity:** In the first phase, the prover commits to each of its messages, so its communication will be independent of r in each round. V_r needs to send the communication required for a single instance of V_{r-1}, *not* λ instances for each sub-computation. The communication in the second phase is bounded by the succinctness of Kilian's protocol, which is fixed and essentially independent of r and s. It follows that each message sent has fixed size at most $\mathsf{poly}(\lambda)$, so the total communication is at most $r \cdot \mathsf{poly}(\lambda)$.
- **Verifier efficiency:** This follows from the efficiency of Kilian's succinct argument, resulting in a verifier that runs in time polynomial in the statement size for the proof merging phase. This statement consists of proving committed messages are consistent with r digests according to the input statements of size s. It follows that the verifier will run in time $\mathsf{poly}(s, r)$.
- **Prover efficiency:** We first analyze the complexity required to commit to the sub-proofs. By assumption, P_{r-1} requires time $(t/\lambda) \cdot \mathsf{poly}(s, r)$ and space $(t/\lambda)/\lambda^{r-1} \cdot \mathsf{poly}(s, r) = (t/\lambda^r) \cdot \mathsf{poly}(s, r)$ to prove each sub-computation. So, to prove all λ sub-computations requires time $\lambda \cdot (t/\lambda) \cdot \mathsf{poly}(s, r) = t \cdot \mathsf{poly}(s, r)$. Since all sub-computations can be proved independently, the prover can erase the additional space needed for each sub-proof, so the total space doesn't grow.

  For the second phase of recursive proof merging, the prover's space grows polynomially with the time to verify all sub-computations are consistent with the committed prover messages. By succinctness of (P_{r-1}, V_{r-1}), the prover's messages are short—total size $r \cdot \mathsf{poly}(\lambda)$—and the running time of V_{r-1} is at most $\mathsf{poly}(s, r)$. So, the total time and space for the second phase is at most $\mathsf{poly}(s, r)$.

  Combining the complexity for the two phases, the total time of the prover remains $t \cdot \mathsf{poly}(s, r)$ while using space $(t/\lambda^r) \cdot \mathsf{poly}(s, r)$. We note that a little more care is required to show that the exact polynomial functions for each r don't grow too much, which we defer to the full proof.

Combining the above, this yields a complexity-preserving, succinct argument of knowledge for small-space $s \in \mathsf{poly}(\lambda)$ NP computations.

Handling Arbitrary Space, Non-deterministic Computation. To handle arbitrary space RAM computations, we follow the blueprint for RAM delegation

of [54] adapted to the non-deterministic setting. Specifically, rather than proving updates to the actual space of the computation, we keep track of a hash tree for the space of the computation and prove updates hold relative to the digest for the hash tree.

Using the framework of update languages as above, we consider the update function where the state only keeps track of the hash tree digest. The witness needed to update the state provides information to: (1) prove the bit read at each step of the computation is correct with respect to the hash tree digest, and (2) if the computation step causes a write to memory, computes and proves the updated digest with respect to the new memory.

The state size for this update function is some fixed $\mathsf{poly}(\lambda)$ rather than s. So, if we use the protocol above for this, we only require $(t/\lambda^r) \cdot \mathsf{poly}(\lambda, r)$ space by the prover to run (P_r, V_r) for this update function. Furthermore, we can compute these hash tree proofs needed as the witnesses for P_r using space only $s \cdot \mathsf{poly}(\lambda)$ as opposed to $\mathsf{poly}(s)$. By setting $r = \log_\lambda t$ above, the prover's total space complexity is $s \cdot \mathsf{poly}(\lambda)$, so we get a public-coin, complexity-preserving, succinct argument of knowledge for proving general non-deterministic RAM computation in $O(\log_\lambda t)$-rounds.

1.3 Related Work on Succinct Arguments

We overview the current landscape of succinct arguments, with a focus on complexity-preserving protocols. First, we overview the main techniques for constructing succinct arguments for NP from information theoretic proofs in idealized oracle models. Second, we describe the recursive composition framework underlying constructions of complexity-preserving, succinct *non-interactive* arguments for NP from non-falsifiable assumptions. Lastly, we briefly highlight the relevant work implementing recursive composition for *deterministic* computation from falsifiable assumptions. At a very high level, we note that our main construction follows from a hybrid of the above techniques by recursively composing Kilian's succinct argument in a space-efficient way, which only relies on collision-resistant hash functions.

Information-Theoretic Compilers for Succinct Arguments for NP. Almost all constructions of succinct arguments for NP go through the following general blueprint. First, construct an information-theoretic proof with oracle access to an idealized object. Next, instantiate the oracle with a cryptographic commitment to the idealized object. Below, we overview existing constructions of succinct arguments, organized by their corresponding idealized models.

- **Probabilistically Checkable Proofs (PCPs):** PCPs (introduced for positive results in [2] and for hardness of approximation results in [41]) are non-interactive, information-theoretic proofs given oracle query access to a long proof string. Kilian [58] showed how to compile PCPs into succinct arguments by committing to the proof string using a vector commitment, which can be constructed from CRH using Merkle trees [61]. Furthermore, Micali [63]

showed how to make this approach non-interactive using the Fiat-Shamir transform [42] in the ROM since the underlying protocol is public-coin.

- **Interactive Oracle Proofs (IOPs):** IOPs (independently introduced by [14] and [68]) are interactive, public-coin variants of PCPs, where in each round the prover provides oracle access to a new long string. This generalization of PCPs can result shorter and more prover-efficient proofs (see e.g. [12,69]) at the cost of extra rounds of communication. Like PCPs, IOPs can be compiled to succinct arguments using vector commitments and hence only using CRH.
  Most relevant to this work is the IOP-based complexity-preserving argument of [4], which builds off the Ligero protocol [1]. Because this protocol is based on IOPs, it only requires assuming CRH. However, their proof size is $\tilde{O}(t/s)$ and verifier runns in time $\tilde{O}(t/s + s)$, so their protocol is not fully succinct. They complement their result with a lower bound, roughly showing that this proof length is tight for complexity-preserving IOP-based protocols. Specifically, they show that any such encoding of the size $O(t)$ transcript using space s must have distance $O(s/t)$, resulting in query complexity—and hence proof length of the compiled argument—of at least $\Omega(t/s)$.
- **Linear PCPs:** Linear PCPs (introduced by [51] and further studied in [22,26,45]) are proofs given oracle access to a linear function. [51] first shows how to convert a linear PCP into a linear multi-prover interactive proof (MIP), where the proof gives oracle access to potentially many different linear functions. They then compile this linear MIP into a (private-coin) argument using a particular multi-commitment scheme for linear functions that can be built from any additive homomorphic encryption scheme. In contrast to (plain) PCPs, the benefit of linear PCPs is that you don't need to materialize the full linear function in order to commit to it. However, their argument is not succinct since their linear PCP exponentially long, so the verifier's query must be linear in the complexity of the language. Still, their approach results in very short communication from the prover and gets around the efficiency bottleneck of using (plain) PCPs.
- **Multi-prover Interactive Proofs (MIPs):** MIPs (introduced by [9] and further studied in [21,25]) are interactive proofs with oracle access to independent, arbitrary functions that act as various provers who are not allowed to talk to each other. Bitansky and Chiesa [21] show how to compile MIPs into (private-coin) succinct arguments using succinct multi-function commitments, which can be constructed from fully homomorphic encryption. Bitansky and Chiesa use MIPs over PCPs because they can construct *complexity-preserving* MIPs where, for a time t space s non-deterministic computation, each prover can be implemented in time $\tilde{O}(t)$ and space $\tilde{O}(s)$. Then, their compiler results in a complexity-preserving succinct argument. We note that it is still a very intriguing open question to construct complexity-preserving PCPs.
- **Polynomial IOPs:** Polynomial IOPs (first formalized by [32] but implicit in previous works) generalize linear PCPs by (1) using oracles for higher degree polynomials as opposed to only linear functions and (2) allowing interaction

as in IOPs. Polynomial IOPs can be compiled into arguments using polynomial commitments [57]. Constructing both polynomial IOPs and polynomial commitments have been at the forefront of practical succinct (non-interactive) argument constructions (see e.g. [11,33,44,60,70,72] for examples of polynomial IOPs and [10,23,24,27,31,32,57] for examples of polynomial commitments).

Of particular interest to this work are the complexity-preserving arguments of [23,24]. [23] construct a *publicly verifiable*, complexity-preserving, zero-knowledge argument in the ROM assuming hardness of discrete log. However, the verifier in their protocol runs in time $\tilde{O}(t)$, which is still non-trivial given their additional focus on zero knowledge and succinct communication. Building off of [23] and [24,32] construct the first public-coin, complexity-preserving, *fully succinct* argument. Their protocol requires $O(\log t)$ rounds of communication and relies on the existence of a public-coin *hidden order* group. The only candidate such groups are class groups of an imaginary quadratic field, which were first suggested for cryptographic use by [37] but have seen relatively little attention as a cryptographic assumption. Alternatively by relying on RSA groups or other private-coin hidden order groups, the protocol of [24] is only private-coin or relies on trusted setup.

Recursive Composition. Bitansky, Canetti, Chiesa, and Tromer [19]—based on the construction of incrementally verifiable computation of Valiant [73]—show how to bootstrap any (pre-processing) succinct, non-interactive, argument of knowledge [18] (SNARK) for NP into a complexity-preserving SNARK using *recursive composition*. The idea is that the prover can first use the underlying SNARK to prove that each step of the computation was performed correctly. Instead of having the verifier check each such proof, the prover will instead batch subsequent proofs together and use the same underlying SNARK to prove that it knows accepting proofs that would have caused the verifier to accept. This idea can be applied recursively until the verifier only has to check a single, succinct proof! Furthermore, the independent nature of the sub-proofs allow the prover to generate the individual proofs in pieces without blowing up its space complexity, overall resulting in a complexity-preserving SNARK.

The main downside of this approach is the fact that the existence of SNARKs is a non-falsifiable, "knowledge" assumption. Furthermore, even without the strong knowledge-soundness property, non-falsifiable assumptions are likely inherent [46]. Also, SNARKs for NP can only possibly exist with respect to restricted auxiliary-input distributions assuming indistinguishability obfuscation [20,28]. For these reasons, we focus on more standard, falsifiable assumptions in this work. In fact, our main construction can be viewed as implementing the recursive composition technique of [19] from falsifiable assumptions using interaction.

Delegation for P. In light of the inherent limitations for constructing succinct, non-interactive arguments for NP [46], there has been a long and fruitful line of work focusing on building succinct, non-interactive arguments in the common

reference string model—or delegation protocols—for deterministic computation and other subclasses of NP (see e.g. [3,29,30,35,38,48,49,52,54–56,66,67,74]). Of particular note, Holmgren and Rothblum [48] construct privately verifiable, complexity-preserving delegation protocols for P from the (sub-exponential) learning with errors assumption. The works of [34,35,49,55,74] construct publicly verifiable (yet not complexity-preserving) delegation protocols for P from various falsifiable assumptions. Underlying these constructions are novel techniques for implementing recursive composition from falsifiable assumptions, albeit restricted to deterministic computations. The works of [38,53,66] extend these works to construct incrementally verifiable computation [73] under the same assumptions, which directly yield publicly verifiable, complexity-preserving delegation protocols for P.

Comparison of Complexity-Preserving Arguments. In Table 1 below, we summarize various efficiency properties and cryptographic assumptions for existing complexity-preserving arguments for NP. For simplicity, we use $\tilde{O}(\cdot)$ notation to suppress all multiplicative $\mathsf{poly}(\lambda, |x|, \log t, \log s)$ terms. All protocols in the table have prover time $\tilde{O}(t)$ and space $\tilde{O}(s)$. We note that the various schemes may differ in the exact terms hidden in the $\tilde{O}(\cdot)$ notation, but our primary focus is on the asymptotics in terms of t and s. Finally, we note that all of the public-coin protocols can heuristically be compressed to a single message via the Fiat-Shamir transform in the random oracle model (ROM).

Table 1. Comparison of complexity-preserving arguments for non-deterministic time t and space s computation. For simplicity, we use $\tilde{O}(\cdot)$ notation to suppress all multiplicative $\mathsf{poly}(\lambda, |x|, \log t, \log s)$ terms.

Protocol	Verifier Time	Communication	Messages	Public Coin?	Assumption
[21]	$\tilde{O}(1)$	$\tilde{O}(1)$	4	No	FHE
[19,73]	$\tilde{O}(1)$	$\tilde{O}(1)$	1	Yes	SNARKs for NP
[23]	$\tilde{O}(t)$	$\tilde{O}(1)$	1	Yes	DLog + ROM
[24]	$\tilde{O}(1)$	$\tilde{O}(1)$	$O(\log t)$	Yes	Class Groups
[4]	$\tilde{O}(t/s + s)$	$\tilde{O}(t/s)$	$O(1)$	Yes	CRH
This work	$\tilde{O}(1)$	$\tilde{O}(1)$	$O(\log_\lambda t)$	Yes	CRH

2 Preliminaries

We let $\mathbb{N} = \{1, 2, 3, \ldots\}$ denote the set of natural numbers, and for any $n \in \mathbb{N}$, we write $[n]$ to denote the set $[n] = \{1, \ldots, n\}$. For integers $a, b \in \mathbb{Z}$ with $a \leq b$, we write $[a, b]$ to denote the set $\{a, a+1, \ldots, b\}$. For a set Σ, referred to as the *alphabet*, we denote Σ^* the set of strings consisting of 0 or more elements from Σ. We let Σ^n denote the set of n-character strings from Σ and $\Sigma^{\leq n}$ the set of string of length at most n. For a string $s \in \Sigma^*$, we let $|s|$ denote the length

of s. For any string $s \in \Sigma^*$ and $i \in [|s|]$, let $s[i]$ denote the ith character of s. For $i \notin [|s|]$, we assume that $s[i]$ always returns a special character $\perp$. Unless specified otherwise, we assume that a string s is defined over the binary alphabet $\{0,1\}$.

We write $\lambda \in \mathbb{N}$ to denote the security parameter. We say that a function $p\colon \mathbb{N} \to \mathbb{N}$ is in the set $\mathsf{poly}(\lambda)$ and is *polynomially-bounded* if there exists a constant c and an index $i \in \mathbb{N}$ such that $p(\lambda) \leq \lambda^c$ for all $\lambda \geq i$. We say that a function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}$ is negligible if for every constant $c > 0$ there exists $i \in \mathbb{N}$ such that $\mathsf{negl}(\lambda) \leq \lambda^{-c}$ for all $\lambda \geq i$.

We use PPT to denote the acronym *probabilistic, polynomial time*. A uniform algorithm A is a RAM program with a fixed (constant-size) description length. A non-uniform algorithm A consists of a sequence of algorithms $\{A_\lambda\}_{\lambda\in\mathbb{N}}$, one for each security parameter λ; we assume for simplicity that A_λ always receives 1^λ as its first input. When the security parameter is clear from context, we may write $A(\cdot)$ instead of $A_\lambda(\cdot)$ for simplicity. A non-uniform PPT algorithm is one where the description size of A_λ as a function of λ is in $\mathsf{poly}(\lambda)$.

For a distribution X, we write $x \leftarrow X$ to denote the process of sampling a value x from the distribution X. For a set $\mathcal{X}$, we use $x \leftarrow \mathcal{X}$ to denote the process of sampling a value x from the uniform distribution over $\mathcal{X}$. We use $x = A(\cdot)$ to denote the output of a deterministic algorithm and $x \leftarrow A(\cdot)$ to denote the output of a randomized algorithm. For a randomized algorithm, we write $x = A(\cdot; r)$ to denote the deterministic output given sequential access to the random coins r. We write $x := y$ to denote the assignment of value y to x. For a distribution D, we define $\mathrm{Supp}\,(D)$ to denote the support of the distribution D.

RAM Computation. We model a non-deterministic RAM computation by a machine M with local word size n, and random access to a working memory string $D \in \{0,1\}^{2^n}$ and a read-only witness. The computation of M is carried out one step at a time by a polynomial-time CPU algorithm step that takes as input a description of a program M, a RAM state rst of size $O(n)$, a bit b^{mem} read from memory, and a witness bit b^{wit}. It then outputs a tuple

$$(\mathsf{rst}_{\mathsf{new}}, i^{\mathsf{mem}}, i^{\mathsf{wrt}}, b^{\mathsf{wrt}}, i^{\mathsf{wit}}) = \mathsf{step}(M, \mathsf{rst}, b^{\mathsf{mem}}, b^{\mathsf{wit}}),$$

where $\mathsf{rst}_{\mathsf{new}}$ is the updated state, i^{mem} is the next location to read from memory, i^{wrt} is the location in memory to write next, b^{wrt} is the bit to be written, and i^{wit} is the next location to read from the witness.

We write $M^D(w)$ to denote the computation of M with working memory D and witness w. We write $M(x, w)$ to denote the computation $M^D(w)$ where D is initialized to start with an encoding of the input x followed by 0s.

The program starts with initial empty state rst_0, initial memory and witness read locations $i_0^{\mathsf{mem}} = i_0^{\mathsf{wit}} = 1$. Starting from $j = 1$, the jth execution step proceeds as follows:

1. Read the memory bit $b_{j-1}^{\mathsf{mem}} := D[i_{j-1}^{\mathsf{mem}}]$ and witness bit $b_{j-1}^{\mathsf{wit}} := w[i_{j-1}^{\mathsf{wit}}]$.
2. Compute $(\mathsf{rst}_j, i_j^{\mathsf{mem}}, i_j^{\mathsf{wrt}}, b_j^{\mathsf{wrt}}, i_j^{\mathsf{wit}}) = \mathsf{step}(M, \mathsf{rst}_{j-1}, b_{j-1}^{\mathsf{mem}}, b_{j-1}^{\mathsf{wit}})$.

3. If $i_j^{\mathsf{wrt}} \neq \perp$, write a bit to memory $D[i_j^{\mathsf{wrt}}] := b_j^{\mathsf{wrt}}$.

The execution terminates when step outputs a special terminating state for rst_t, which specifies whether the computation is accepting and outputs 1 or rejecting and outputs 0. Note that we only consider machines with binary output in this work.

For a RAM computation $M^D(w)$, we define its running time t as the number of steps until M halts and its space s as the maximum index i_j^{mem} accessed. Note that the witness tape is read-only and does not count towards the space.

We say that a RAM computation $M^D(w)$ makes m passes over its witness if reads its witness from left to right at most m times. That is, $|\{j \in [t] : i_j^{\mathsf{wit}} < i_{j-1}^{\mathsf{wit}}\}| < m$, where t is the computation's running time.

Universal Languages. The universal relation $\mathcal{R}_{\mathcal{U}}$ is the set of instance-witness pairs $((M, x, t), w)$ where M is a RAM program with word size n such that $M(x, w)$ accepts and outputs 1 within t steps. We assume that the description of M contains its word size 1^n in unary, so $|M|$ is always at least n. We let $\mathcal{L}_{\mathcal{U}}$ be the corresponding language with relation $\mathcal{R}_{\mathcal{U}}$, which we call the universal language.

Interactive Machines. We consider interactive protocols with interactive RAM programs. To allow for communication between two interacting machines, we assume there is a specified part of a machine's memory for input from and output to another interactive machine. Once one machine halts after writing downs its output, we say that it has sent a message consisting of its output to the other machine. Given a pair of interactive RAM programs A and B, we denote by $\langle A(w), B\rangle(x)$ the random variable representing the output of B with common input x, when interacting with A with common input x and witness w, when the random tape of each machine is uniformly and independently chosen. We let $\mathsf{View}_B(\langle A(w), B\rangle(x))$ be the random variable representing the view of B in the interaction between A and B, consisting of its inputs, random coins, and the communication it receives from A. The message complexity of the protocol is the number of distinct messages sent between A and B before B produces its final output.

2.1 Collision-Resistant Hash Functions

We give the notion of a keyed collision-resistant hash functions (CRH) that we use in this work. We emphasize that our definition below allows for arbitrary length input and produces a fixed size digest, but it is well known that this is implied by any compressing CRH with fixed input length [36,61,62].

Definition 2.1 (Collision-Resistant Hash Function). *A* keyed collision-resistant hash function *is given by an algorithm* Hash *with the following syntax:*

- $\mathsf{dig} = \mathsf{Hash}(\mathsf{hk}, x)$*: A deterministic algorithm that on input a hash key* $\mathsf{hk} \in \{0,1\}^\lambda$ *and string* $x \in \{0,1\}^*$*, outputs a digest* $\mathsf{dig} \in \{0,1\}^\lambda$.

We require that Hash *runs in polynomial-time and satisfies the following security property:*

- ***Collision Resistance:*** *For all non-uniform polynomial-time adversaries* A*, there exists a negligible function* negl *such that for all* $\lambda \in \mathbb{N}$*, it holds that*

$$\Pr\left[\begin{matrix}\mathsf{hk} \leftarrow \{0,1\}^\lambda \\ (x,x') \leftarrow A(\mathsf{hk})\end{matrix} : \begin{matrix}x \neq x' \\ \mathsf{Hash}(\mathsf{hk},x) = \mathsf{Hash}(\mathsf{hk},x')\end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

2.2 Hash Trees

We follow the definition of hash trees from [54] with slight modifications from [40].

Definition 2.2 (Hash Tree). *A* hash tree *is a tuple of algorithms* $(\mathsf{KeyGen}, \mathsf{Hash}, \mathsf{Read}, \mathsf{Write}, \mathsf{VerRead}, \mathsf{VerWrite})$ *with the following syntax and efficiency:*

- $\mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda)$*: A randomized polynomial-time algorithm that on input the security parameter* 1^λ *outputs a hash key* hk*.*
- $(\mathsf{tree}, \mathsf{dig}) = \mathsf{Hash}(\mathsf{hk}, D)$*: A deterministic polynomial-time algorithm that on input a hash key* hk *and a database* $D \in \{0,1\}^s$ *outputs a hash tree* tree *and a string* $\mathsf{dig} \in \{0,1\}^\lambda$*. We require that* tree *has size at most* $s \cdot \mathsf{poly}(\lambda, \log s)$*.*
- $\mathsf{dig} = \mathsf{Digest}(\mathsf{hk}, D)$*: A deterministic algorithm that on input a hash key* hk *and a database* $D \in \{0,1\}^s$ *outputs a string* $\mathsf{dig} \in \{0,1\}^\lambda$*. For databases of the form* $D = x \| 0^{s-|x|}$*,* Digest *runs in time* $(|x|+1) \cdot \mathsf{poly}(\lambda, \log s)$*.*
- $(b, \pi^{\mathsf{rd}}) = \mathsf{Read}^{\mathsf{tree}}(i^{\mathsf{rd}})$*: A read-only deterministic RAM program that accesses a database* tree *and on input an index* i^{rd} *outputs a bit* b^{rd} *and a proof* π^{rd}*. The program runs in time* $\mathsf{poly}(\lambda, \log s)$*.*
- $(\mathsf{dig}_{\mathsf{new}}, \pi^{\mathsf{wrt}}) = \mathsf{Write}^{\mathsf{tree}}(i^{\mathsf{wrt}}, b^{\mathsf{wrt}})$*: A deterministic RAM program that accesses a database* tree *and on input an index* i^{wrt} *and bit* b^{wrt} *outputs a new digest* $\mathsf{dig}_{\mathsf{new}}$ *and a proof* π^{wrt}*. The program runs in time* $\mathsf{poly}(\lambda, \log s)$ *and* tree *remains fixed size.*
- $b = \mathsf{VerRead}(\mathsf{dig}, i^{\mathsf{rd}}, b^{\mathsf{rd}}, \pi^{\mathsf{rd}})$*: A deterministic polynomial-time algorithm that on input a digest* dig*, an index* i^{rd}*, a bit* b^{rd}*, and a proof* π^{rd} *outputs a bit* b*.*
- $b = \mathsf{VerWrite}(\mathsf{dig}, i^{\mathsf{wrt}}, b^{\mathsf{wrt}}, \mathsf{dig}', \pi^{\mathsf{wrt}})$*: A deterministic polynomial-time algorithm that on input a digest* dig*, an index* i^{wrt}*, a bit* b^{wrt}*, a new digest* dig'*, and a proof* π^{wrt} *outputs a bit* b*.*

We require that $(\mathsf{KeyGen}, \mathsf{Hash}, \mathsf{Read}, \mathsf{Write}, \mathsf{VerRead}, \mathsf{VerWrite})$ *satisfy the following properties:*

- ***Digest Consistency:*** *For every* $\lambda \in \mathbb{N}$ *and* $D \in \{0,1\}^s$*, it holds that*

$$\Pr\left[\begin{matrix}\mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (\cdot, \mathsf{dig}) = \mathsf{Hash}(\mathsf{hk}, D) \\ \mathsf{dig}' = \mathsf{Digest}(\mathsf{hk}, D)\end{matrix} : \mathsf{dig} = \mathsf{dig}'\right] = 1.$$

– ***Correctness of Read:*** *For every* $\lambda \in \mathbb{N}$, $D \in \{0,1\}^s$, $i^{\mathsf{rd}} \in [s]$, $m \geq 0$, *and sequence of pairs* $(i_j, b_j) \in [s] \times b$ *for* $j \in [m]$, *let* $D' \in \{0,1\}^s$ *be the database equal to* D *followed by updates* b_j *to index* i_j *for each* $j \in [m]$. *Then, it holds that*

$$\Pr\left[\begin{array}{l|l} \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda) & \\ (\mathsf{tree}, \mathsf{dig}_0) = \mathsf{Hash}(\mathsf{hk}, D) & \\ \forall j \in [m], & \mathsf{VerRead}(\mathsf{dig}_m, i^{\mathsf{rd}}, b^{\mathsf{rd}}, \pi^{\mathsf{rd}}) = 1 \\ \quad (\cdot, \mathsf{dig}_j) = \mathsf{Write}^{\mathsf{tree}}(i_j, b_j) & \wedge\ b^{\mathsf{rd}} = D'[i^{\mathsf{rd}}] \\ (b^{\mathsf{rd}}, \pi^{\mathsf{rd}}) = \mathsf{Read}^{\mathsf{tree}}(i^{\mathsf{rd}}) & \end{array}\right] = 1.$$

– ***Correctness of Write:*** *For every* $\lambda \in \mathbb{N}$, $D \in \{0,1\}^s$, $i^{\mathsf{wrt}} \in [s]$, $b^{\mathsf{wrt}} \in \{0,1\}$, $m \geq 0$, *and sequence of pairs* $(i_j, b_j) \in [s] \times \{0,1\}$ *for* $j \in [m]$, *let* $D' \in \{0,1\}^s$ *be the database equal to* D *followed by updates* b_j *to index* i_j *for each* $j \in [m]$. *Let* D'_{new} *be the database equal to* D' *followed by one more update* b^{wrt} *to index* i^{wrt}. *Then, it holds that*

$$\Pr\left[\begin{array}{l|l} \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda) & \\ (\mathsf{tree}, \mathsf{dig}_0) = \mathsf{Hash}(\mathsf{hk}, D) & \mathsf{VerWrite}(\mathsf{dig}_m, i^{\mathsf{wrt}}, \\ \forall j \in [m], (\cdot, \mathsf{dig}_j) = \mathsf{Write}^{\mathsf{tree}}(i_j, b_j) & \quad b^{\mathsf{wrt}}, \mathsf{dig}_{\mathsf{new}}, \pi^{\mathsf{wrt}}) = 1 \\ (\mathsf{dig}_{\mathsf{new}}, \pi^{\mathsf{wrt}}) = \mathsf{Write}^{\mathsf{tree}}(i^{\mathsf{wrt}}, b^{\mathsf{wrt}}) & \wedge\ \mathsf{dig}_{\mathsf{new}} = \mathsf{dig}' \\ (\cdot, \mathsf{dig}') = \mathsf{Hash}(\mathsf{hk}, D'_{\mathsf{new}}) & \end{array}\right] = 1.$$

– ***Soundness of Read:*** *For every non-uniform PPT* A, *there exists a negligible function* negl *such that for all* $\lambda \in \mathbb{N}$, *it holds that*

$$\Pr\left[\begin{array}{l|l} \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda) & b \neq b' \\ (\mathsf{dig}, i, b, \pi, b', \pi') \leftarrow A(\mathsf{hk}) & \wedge\ \mathsf{VerRead}(\mathsf{dig}, i, b, \pi) = 1 \\ & \wedge\ \mathsf{VerRead}(\mathsf{dig}, i, b', \pi') = 1 \end{array}\right] \leq \mathsf{negl}(\lambda).$$

– ***Soundness of Write:*** *For every non-uniform PPT* A, *there exists a negligible function* negl *such that for all* $\lambda \in \mathbb{N}$, *it holds that*

$$\Pr\left[\begin{array}{l|l} \mathsf{hk} \leftarrow \mathsf{KeyGen}(1^\lambda) & \mathsf{dig}_{\mathsf{new}} \neq \mathsf{dig}'_{\mathsf{new}} \\ (\mathsf{dig}, i, b, \mathsf{dig}_{\mathsf{new}}, \pi, & \wedge\ \mathsf{VerWrite}(\mathsf{dig}, i, b, \mathsf{dig}_{\mathsf{new}}, \pi) = 1 \\ \quad \mathsf{dig}'_{\mathsf{new}}, \pi') \leftarrow A(\mathsf{hk}) & \wedge\ \mathsf{VerWrite}(\mathsf{dig}, i, b, \mathsf{dig}'_{\mathsf{new}}, \pi') = 1 \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Based on Merkle trees [61], we can construct hash trees that satisfy the above definition from any CRH.

Theorem 2.3 ([40,61]). *Assuming the existence of a keyed collision-resistant hash function, there exists a hash tree as per Definition 2.2.*

2.3 Arguments of Knowledge

We define arguments of knowledge for the universal language $\mathcal{L}_{\mathcal{U}}$. We use a definition from [40] which is equivalent to the more standard definition of [7] (see Remark 2.5). We choose to work with this definition since it is more convenient in settings involving composition (see Remark 2.6). We note that in contrast to the notion of universal arguments [6], our arguments of knowledge property only considers polynomial time computations.

Definition 2.4 (Argument of Knowledge). *A pair of interactive RAM programs (P, V) is an* argument of knowledge for $\mathcal{R}_{\mathcal{U}}$ *if the following hold:*

- ***Prover Efficiency:*** *There exists a polynomial q such that for every $\lambda \in \mathbb{N}$ and $((M, x, t), w) \in \mathcal{R}_{\mathcal{U}}$, the prover P on common input $(1^\lambda, (M, x, t))$ and witness w runs in time $q(\lambda, |M, x|, t)$.*
- ***Completeness:*** *For every $\lambda \in \mathbb{N}$ and $(y, w) \in \mathcal{R}_{\mathcal{U}}$, it holds that*

$$\Pr\left[\langle P(w), V\rangle(1^\lambda, y) = 1\right] = 1.$$

- ***Argument of Knowledge:*** *For every polynomial p, there exists a probabilistic oracle machine $\mathcal{E}$ and a polynomial q such that for every non-uniform polynomial-time prover $P^\star$, there exists a negligible function* negl *such that for every $\lambda \in \mathbb{N}$, and instance $y = (M, x, t)$ such that $|M, x| \leq p(\lambda)$ and $t \leq p(\lambda)$, the following hold.*
  Let $V[\rho]$ denote the verifier V using randomness $\rho \in \{0,1\}^{\ell(\lambda)}$ where $\ell(\lambda)$ is a bound on the number of random bits used by V. Then:
  1. *The expected running time of $\mathcal{E}^{P^\star}(1^\lambda, y, \rho)$ is bounded by $q(\lambda)$ where the expectation is over $\rho \leftarrow \{0,1\}^{\ell(\lambda)}$ and the random coins of $\mathcal{E}$, and oracle calls to $P^\star$ cost only a single step.*
  2. *It holds that*

$$\Pr\left[\begin{matrix}\rho \leftarrow \{0,1\}^{\ell(\lambda)} \\ w \leftarrow \mathcal{E}^{P^\star}(1^\lambda, y, \rho)\end{matrix} : \begin{matrix}(y, w) \notin \mathcal{R}_{\mathcal{U}} \\ \wedge\ \langle P^\star, V[\rho]\rangle(1^\lambda, y) = 1\end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

Remark 2.5 (Equivalence to definition of [7]). The "standard" definition of an argument of knowledge is due to Bellare and Goldreich [7] (BG). The BG extractor always succeeds in extracting a valid witness (in contrast to succeeding in accordance with a uniformly sampled view given by the verifier's randomness ρ) but runs in expected time $\mathsf{poly}(\lambda)/(\epsilon - \mathsf{negl}(\lambda))$ for negligible function negl where ϵ is the probability that $\langle P^\star, V\rangle(1^\lambda, y) = 1$. The existence of a BG extractor implies the existence of an EFKP extractor (as defined above) and vice versa [40]. This implication from BG to EFKP is shown via the intermediate notion of witness-extended emulation from Lemma 3.1 of [59] and Lemma A.6 of [40]. To construct a BG extractor from an EFKP extractor, you first sample randomness ρ for the verifier, check if $\langle P^\star, V[\rho]\rangle(1^\lambda, y)$ is accepting, run the EFKP extractor with ρ if so, and repeat if the transcript is rejecting or the extractor fails.

Remark 2.6 (On composition for the definition of [40]). A key challenge when composing arguments of knowledge is bounding the running time of the final extractor. One notion that composes well is "precise" arguments of knowledge [64] where, for any given view of the cheating prover (defined by the verifier's randomness ρ as above), the extractor's running time is a fixed polynomial in the running time of the cheating prover on that particular view. This notion, however, is quite strong and not known to hold for arguments of knowledge from standard assumptions.

In a more standard—and also achievable—argument of knowledge notion called witness-extended emulation [59], the extractor is not given a view, but instead must output a uniformly distributed view of the verifier and a corresponding witness if the verifier accepts the view. Furthermore, an extractor for witness-extended emulation only needs to run in expected polynomial time and may use rewinding. However, the view chosen by the extractor may not be consistent with the external view when used as a sub-protocol.

The argument of knowledge notion of [40] gives the extractor a uniformly sampled view and requires that the extractor run in expected polynomial-time over the choice of the view (although to compose well, this polynomial must be independent of the cheating prover's strategy). This relaxes the strict efficiency requirement of [64] since the extractor need not run in fixed polynomial time, but also (conceptually) strengthens the notion of [59] as the extractor must work for a given view rather than outputting one itself. Existentially, the definitions of [40] and [59] are equivalent however; see the above remark.

Public-Coin Protocols. We say that an argument (P, V) is public-coin—or equivalently that V is a public-coin verifier—if all of V's messages simply consist of random coins, and V maintains no other private state. Specifically, the final output of V is a function only of the protocol's transcript.

Efficiency. We consider two efficiency requirements of arguments for $\mathcal{R}_{\mathcal{U}}$: *succinct* and *complexity-preserving* arguments. Roughly speaking, in succinct arguments the communication complexity and verification time are logarithmic in the running time of the computation that is being proved. In complexity-preserving arguments, the time and space complexity of the honest prover are close to the complexity of the original computation. When measuring efficiency, we model the prover P and verifier V as interactive RAM programs. We formalize these notions below.

Definition 2.7 (Succinct Arguments). *Let (P, V) be an argument for $\mathcal{R}_{\mathcal{U}}$. We say that (P, V) is* succinct *if there exists a polynomial q such that for any $\lambda \in \mathbb{N}$ and $(y = (M, x, t), w) \in \mathcal{R}_{\mathcal{U}}$, the following always hold during the experiment $\langle P(w), V\rangle(1^{\lambda}, y)$:*

- ***Succinct Verification:*** *The verifier V runs in time at most $q(\lambda, |M, x|, \log t)$.*
- ***Succinct Communication:*** *The length of the transcript is at most $q(\lambda, \log |M, x|, \log t)$.*

Succinct arguments of knowledge are known based on CRH.

Theorem 2.8 ([58]). *Assuming the existence of a keyed collision-resistant hash function, there exists a 4-message, public-coin, succinct argument of knowledge for $\mathcal{R}_{\mathcal{U}}$.*

Definition 2.9 (Complexity-Preserving Arguments). *Let (P, V) be an argument for $\mathcal{R}_{\mathcal{U}}$. We say that (P, V) is* complexity-preserving *if there exists*

a polynomial q such that for any $\lambda \in \mathbb{N}$ and $(y = (M, x, t), w) \in \mathcal{R}_{\mathcal{U}}$ such that $M(x, w)$ uses space s, the prover P in the experiment $\langle P(w), V\rangle(1^\lambda, y)$ runs in time at most $t \cdot q(\lambda, |M, x|, \log t)$ and uses space at most $s \cdot q(\lambda, |M, x|, \log t)$.

3 Arguments of Knowledge for Bounded Space Computation

As a step towards our main result, we construct an argument of knowledge for the update language in this section. In the update language, we consider a non-deterministic polynomial-time Turing machine M updating a state of size s. The language contains tuples $(M, \mathsf{st}, \mathsf{st}', t)$ such that M moves from state st to state st' via a sequence of t updates. In Sect. 4, we turn this argument for the update language into a complexity preserving succinct argument of knowledge for $\mathcal{R}_{\mathcal{U}}$.

Definition 3.1 (Update Language). *Let α be a polynomial. The update language $\mathcal{L}_{\mathsf{Upd},\alpha}$ with relation $\mathcal{R}_{\mathsf{Upd},\alpha}$ consists of instance-witness pairs of the form $((M, \mathsf{st}, \mathsf{st}', t), w)$ where M is an α-time non-deterministic Turing machine, st, $\mathsf{st}' \in \{0,1\}^s$, $t \in \mathbb{N}$, and $w = (w_1, \ldots, w_t) \in \{0,1\}^{\alpha(s)\times t}$ such that the following procedure accepts:*

- *Set $\mathsf{st}_0 = \mathsf{st}$.*
- *For $i = 1, \ldots, t$:*
  - *Emulate $M(\mathsf{st}_{i-1}, w_i)$. If M does not halt within $\alpha(s)$ steps then reject.*
  - *Obtain M's output st_i. If $|\mathsf{st}_i| \neq s$ then reject.*
- *Accept if $\mathsf{st}_t = \mathsf{st}'$ and reject otherwise.*

We construct a sequence of arguments for $\mathcal{L}_{\mathsf{Upd},\alpha}$.

Theorem 3.2. *Assume the existence of a keyed collision-resistant hash function. There exists $d \in \mathbb{N}$ such that for every polynomial α, there exists sequence of interactive RAM programs $\{(P_r, V_r)\}_{r\geq 0}$ such that the following hold:*

- ***Completeness:*** *For every $\lambda \in \mathbb{N}$, $r \geq 0$, and $(y, w) \in \mathcal{R}_{\mathsf{Upd},\alpha}$, it holds that*

$$\Pr\left[\langle P_r(w), V_r\rangle(1^\lambda, y) = 1\right] = 1.$$

- ***Argument of Knowledge:*** *For every polynomial p, there exists a probabilistic oracle machine $\mathcal{E}$ and a polynomial q such that for every non-uniform polynomial-time prover $P^\star$ and function $r = r(\lambda)$ such that $(\lambda \cdot r)^r \leq p(\lambda)$, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$, instance $y = (M, \mathsf{st}, \mathsf{st}', t)$ such that $|y|, t \leq p(\lambda)$, the following hold.*
  Let $V_r[\rho]$ denote the verifier V_r using randomness $\rho \in \{0,1\}^{\ell(\lambda)}$. Then:
  1. *The expected running time of $\mathcal{E}^{P^\star}(1^\lambda, y, \rho, r)$ is bounded by $q(\lambda)$.*
  2. *It holds that*

$$\Pr\left[\begin{array}{l}\rho \leftarrow \{0,1\}^{\ell(\lambda)} \\ w \leftarrow \mathcal{E}^{P^\star}(1^\lambda, y, \rho, r)\end{array} : \begin{array}{l}(y, w) \notin \mathcal{R}_{\mathsf{Upd},\alpha} \\ \wedge\ \langle P^\star, V_r[\rho]\rangle(1^\lambda, y) = 1\end{array}\right] \leq \mathsf{negl}(\lambda).$$

– ***Efficiency:*** *There exists a polynomial q such that for every $\lambda \in \mathbb{N}$, $r \geq 0$, and $(y = (M, \mathsf{st}, \mathsf{st}', t), w) \in \mathcal{R}_{\mathsf{Upd},\alpha}$ such that $|\mathsf{st}| = s$, the following efficiency properties always hold in the experiment $\langle P_r(w), V_r\rangle(1^\lambda, y)$ assuming $(\lambda \cdot |M| \cdot \alpha(s) \cdot t \cdot r)^d \leq 2^\lambda$:*
  - ***Efficiently Computable:*** *There exists a polynomial-time Turing machine that on input 1^r outputs the descriptions of the interactive RAM programs (P_r, V_r).*
  - ***Round Complexity:*** *The interaction between P_r and V_r consists of $6r+4$ messages.*
  - ***Prover Efficiency:*** *The prover P_r runs in time at most $t \cdot q(\lambda, |M|, s, r)$ and space at most $(t/\lambda^r) \cdot q(\lambda, |M|, s, r)$ and makes at most 1 pass over its witness per message it sends.*
  - ***Verifier Efficiency:*** *The verifier V_r is public-coin and runs in time at most $q(\lambda, |M|, s, r)$.*
  - ***Communication Efficiency:*** *The length of each message sent by P_r or V_r is at most $q(\lambda)$.*

The full construction is provided in Sect. 3.1 with associated proofs deferred to the full version.

3.1 Construction

Let α be any polynomial. We construct a sequence of interactive RAM programs $(P_r, V_r)_{r\geq 0}$ satisfying the properties stated in Theorem 3.2.

Let $(P_{\mathsf{base}}, V_{\mathsf{base}})$ be a 4-message, public-coin, succinct argument of knowledge for $\mathcal{R}_{\mathcal{U}}$ (see Theorem 2.8). In the base case (P_0, V_0), we use $(P_{\mathsf{base}}, V_{\mathsf{base}})$ by converting an the instance $y = (M, \mathsf{st}, \mathsf{st}', t) \in \mathcal{L}_{\mathsf{Upd},\alpha}$ to an input for $\mathcal{L}_{\mathcal{U}}$ as follows. We define M' to be the machine that takes as input $(\mathsf{st}, \mathsf{st}', t)$, sets $\mathsf{st}_0 := \mathsf{st}$, computes $\mathsf{st}_i := M(\mathsf{st}_{i-1}, w_i)$ for $i = 1, \ldots, t$, and outputs 1 if $\mathsf{st}' = \mathsf{st}_t$ and 0 otherwise. Note that M' runs in non-deterministic time $t \cdot \alpha(s)$ given witness $w \in \{0,1\}^{\alpha(s)\times t}$. The prover P_0 and verifier V_0 in the interaction $\langle P_0(w), V_0\rangle(1^\lambda, (M, \mathsf{st}, \mathsf{st}', t))$ simply emulate $\langle P_{\mathsf{base}}(w), V_{\mathsf{base}}\rangle(1^\lambda, (M', (\mathsf{st}, \mathsf{st}', t), t \cdot \alpha(s))$, and V_0 returns the output of V_{base}.

For $r \geq 1$, the construction of (P_r, V_r) relies on a keyed collision-resistant hash function Hash and on the protocol (P_{r-1}, V_{r-1}). At a high level, the protocol has two phases: In the first phase the prover splits the update statements into λ smaller sub-statements and proves all sub-statements in parallel using the protocol (P_r, V_r). To keep the communication from growing too much, in each round, the prover only provides a short commitment the messages of the λ parallel executions. In the second phase, the prover uses the protocol $(P_{\mathsf{base}}, V_{\mathsf{base}})$ to succinctly prove that the λ committed transcripts are all accepting.

We proceed to give a formal description of the protocol (P_r, V_r). The prover P_r and verifier V_r receive as common input a security parameter 1^λ and an instance $y = (M, \mathsf{st}, \mathsf{st}', t) \in \mathcal{L}_{\mathsf{Upd},\alpha}$. The prover additionally receives a witness $w = (w_1, \ldots, w_t) \in \{0,1\}^{\alpha(s)\times t}$ such that $(y, w) \in \mathcal{R}_{\mathsf{Upd},\alpha}$.

– V_r samples a hash key $\mathsf{hk} \leftarrow \{0,1\}^\lambda$ and sends it to P_r.

- Set $\tau = \lceil t/\lambda \rceil$. P_r sets $\mathsf{st}_0 := \mathsf{st}$ and for $i = 1, \ldots, t$, it computes $\mathsf{st}_i := M(\mathsf{st}_{i-1}, w_i)$. During this computation, P_r only saves the states $\mathsf{st}_0, \mathsf{st}_t$ and the $\lambda - 1$ intermediate states $\mathsf{st}_{i \cdot \tau}$ for $i \in [\lambda - 1]$. Every other state is erased as soon as the next state is computed. P_r computes $\mathsf{dig}_0 = \mathsf{Hash}(\mathsf{hk}, (\mathsf{st}_1, \ldots, \mathsf{st}_{\lambda-1}))$ and sends dig_0 to V_r.
- For $i \in [\lambda - 1]$, let $y_i = (M, \mathsf{st}_{(i-1)\cdot\tau}, \mathsf{st}_{i\cdot\tau}, \tau)$ and let $y_\lambda = (M, \mathsf{st}_{(\lambda-1)\cdot\tau}, \mathsf{st}_t, t - (\lambda - 1) \cdot \tau)$. For $i \in [\lambda - 1]$, let $\mathsf{wit}_i = (w_{(i-1)\cdot\tau+1}, \ldots, w_{i\cdot\tau})$ and let $\mathsf{wit}_\lambda = (w_{(i-1)\cdot\tau+1}, \ldots, w_t)$.
- For $j = 1, \ldots, 3r - 1$ corresponding to each back-and-forth round of (P_{r-1}, V_{r-1}):
  - V_r samples $\mathsf{vmsg}_j \leftarrow \{0,1\}^\ell$, where ℓ is a bound on the length of length of each messages sent by V_{r-1} on input $(1^\lambda, y_i)$. V_r sends vmsg_j to P_r.
  - For $i = 1, \ldots, \lambda$, P_r emulates P_{r-1} with input $(1^\lambda, y_i)$, witness wit_i and verifier messages $\mathsf{vmsg}_1, \ldots, \mathsf{vmsg}_j$ and obtains P_{r-1}'s next message $\mathsf{pmsg}_{j,i}$. The prover saves the message $\mathsf{pmsg}_{j,i}$ and erases the memory used to emulate P_{r-1} as soon as $\mathsf{pmsg}_{j,i}$ is computed. P_r computes $\mathsf{dig}_j = \mathsf{Hash}(\mathsf{hk}, (\mathsf{pmsg}_{j,1}, \ldots, \mathsf{pmsg}_{j,\lambda}))$ and sends dig_j to V_r.
- Let $x_{\mathsf{mrg}} = (1^\lambda, r, t, M, \mathsf{hk}, \mathsf{st}_0, \mathsf{st}_t, \vec{\mathsf{dig}}, \vec{\mathsf{vmsg}})$ for $\vec{\mathsf{dig}} = (\mathsf{dig}_0, \ldots, \mathsf{dig}_{3r-1})$ and $\vec{\mathsf{vmsg}} = (\mathsf{vmsg}_1, \ldots, \mathsf{vmsg}_{3r-1})$. Let $w_{\mathsf{mrg}} = (\vec{\mathsf{st}}, \mathsf{Pmsg})$ where $\vec{\mathsf{st}} = (\mathsf{st}_\tau, \ldots, \mathsf{st}_{(\lambda-1)\cdot\tau})$ and $\mathsf{Pmsg} = (\mathsf{pmsg}_{j,i})_{j \in [3r-1], i \in [\lambda]}$. The prover emulates P_{base} and the verifier emulates V_{base} in the interaction

$$\langle P_{\mathsf{base}}(w_{\mathsf{mrg}}), V_{\mathsf{base}} \rangle (1^\lambda, (\mathsf{Merge}, x_{\mathsf{mrg}}, t_{\mathsf{mrg}})),$$

  where Merge is the non-deterministic machine of Fig. 1 and t_{mrg} is the time to compute $\mathsf{Merge}(x_{\mathsf{mrg}}, w_{\mathsf{mrg}})$. The verifier returns the output of V_{base}.

4 Complexity-Preserving Succinct Arguments of Knowledge

In this section, we build a complexity-preserving succinct argument of knowledge for $\mathcal{L}_{\mathcal{U}}$ using the argument for the update language given in Sect. 3. At a high level, we follow the blueprint of [19] where each step of the update language emulates one step of the RAM computation. Instead of accessing memory, the memory content is provided as part of the witness. A hash tree is used to verify memory accesses (Definition 2.2).

Theorem 4.1. *Assume the existence of a keyed collision-resistant hash function. There exists a public-coin, succinct, complexity-preserving, argument of knowledge (P, V) for $\mathcal{R}_{\mathcal{U}}$. On common input $(1^\lambda, (M, x, t))$, the message complexity of the protocol is at most $6 \log_\lambda t + 6$, and P makes at most $3 \log_\lambda t + 3$ passes over its witness.*

To prove the theorem, we provide a construction in Sect. 4.1 with associated proofs deferred to the full version.

Machine Merge

Inputs: A security parameter 1^λ, a recursion depth $r \in \mathbb{N}$, a number of steps $t \in \mathbb{N}$, a machine M, a hash key hk, an initial state st_0, a final state st_t, a sequence of digests $\vec{\mathsf{dig}} = (\mathsf{dig}_0, \ldots, \mathsf{dig}_{3r-2})$, and a sequence of verifier messages $\vec{\mathsf{vmsg}} = (\mathsf{vmsg}_1, \ldots, \mathsf{vmsg}_{3r-3})$.

Witness: A sequence of intermediate states $\vec{\mathsf{st}} = (\mathsf{st}_\tau, \ldots, \mathsf{st}_{(\lambda-1)\cdot\tau})$, and prover messages $\mathsf{Pmsg} = (\mathsf{pmsg}_{j,i})_{j\in[3r-2],i\in[\lambda]}$.

Verification: $\mathsf{Merge}(x_{\mathsf{mrg}} = (1^\lambda, r, t, M, \mathsf{hk}, \mathsf{st}_0, \mathsf{st}_t, \vec{\mathsf{dig}}, \vec{\mathsf{vmsg}}), w_{\mathsf{mrg}} = (\vec{\mathsf{st}}, \mathsf{Pmsg}))$ outputs 1 if all of the following checks pass:

1. $\mathsf{dig}_0 = \mathsf{Hash}(\mathsf{hk}, \vec{\mathsf{st}})$.
2. For each $j \in [3r-2]$, $\mathsf{dig}_j = \mathsf{Hash}(\mathsf{hk}, (\mathsf{pmsg}_{j,1}, \ldots, \mathsf{pmsg}_{j,\lambda}))$.
3. For each $i \in [\lambda - 1]$, V_{r-1} outputs 1 on common input $(1^\lambda, (M, \mathsf{st}_{(i-1)\cdot\tau}, \mathsf{st}_{i\cdot\tau}, \tau))$ given the transcript of message $(\mathsf{vmsg}_1, \mathsf{pmsg}_{1,i}, \ldots, \mathsf{vmsg}_{3r-1}, \mathsf{pmsg}_{3r-1,i})$. For $i = \lambda$, the above holds but for common input consisting of statement $(M, \mathsf{st}_{(\lambda-1)\cdot\tau}, \mathsf{st}_t, t - (\lambda-1)\cdot\tau)$.

Fig. 1. The non-deterministic machine Merge used to verify the validity of all intermediate $\mathcal{L}_{\mathsf{Upd},\alpha}$ statements.

4.1 Construction

We construct a proof system (P, V) for $\mathcal{R}_\mathcal{U}$ satisfying the properties stated in Theorem 4.1. In the construction we make use of the update function UpdHT given in Fig. 2. At a high level, each call to UpdHT runs a step of M and updates a hash tree digest over the memory. Let α be the polynomial specifying the running time of UpdHT as a function of its state size. Our construction relies on a hash tree HT (see Theorem 2.3) and the arguments $(P_r, V_r)_{r\geq 0}$ for $\mathcal{L}_{\mathsf{Upd},\alpha}$ given by Theorem 3.2 with associated constant d.

The prover P and verifier V receive as common input the security parameter 1^λ and an instance $y = (M, x, t) \in \mathcal{L}_\mathcal{U}$. The prover additionally receives a witness w such that $(y, w) \in \mathcal{R}_\mathcal{U}$.

Let $|\mathsf{st}|$ denote the state size for UpdHT. We run the following protocol using security parameter $\lambda' := \max(\lambda, d \cdot \lceil \log(|\mathsf{UpdHT}| \cdot \alpha(|\mathsf{st}|) \cdot \log t) \rceil)$, which ensures that $(\lambda' \cdot |\mathsf{UpdHT}| \cdot \alpha(|\mathsf{st}|) \cdot t \cdot \log_{\lambda'} t)^d \leq 2^{\lambda'}$ as required for the efficiency properties of Theorem 3.2 to hold. Note that this will not effect the asymptotic succinctness or complexity-preserving properties of our protocol as $\lambda' \in O(\lambda + \log|M, x| + \log t)$. For simplicity, we simply write λ instead of λ' in the protocol description.

The proof system is defined as follows:

- V samples a hash key $\mathsf{hk} \leftarrow \mathsf{HT.KeyGen}(1^\lambda)$ and sends it to P.
- P computes $(\mathsf{tree}^{\mathsf{mem}}, \mathsf{dig}_0^{\mathsf{mem}}) = \mathsf{HT.Hash}(\mathsf{hk}, D)$, where D is the initial memory that starts with an encoding of x followed by 0s. V computes the same digest via $\mathsf{dig}_0^{\mathsf{mem}} = \mathsf{HT.Digest}(\mathsf{hk}, D)$. Both P and V set $\mathsf{st}_0 = (\mathsf{rst}_0, \mathsf{dig}_0^{\mathsf{mem}}, i_0^{\mathsf{mem}})$ where rst_0 is the initial RAM state and $i_0^{\mathsf{mem}} = 1$.

Update Function UpdHT

Hardcoded: The input machine M.

State: Each state consists of a RAM state rst, a memory hash tree digest $\mathsf{dig}^{\mathsf{mem}}$, and a memory index i^{mem}. An invalid state is indicated by $\perp$.

Witness: Each witness consists of a witness bit b^{wit}, a read memory bit b^{mem}, a memory read proof π^{mem}, a new memory digest $\mathsf{dig}^{\mathsf{mem}}_{\mathsf{new}}$, and a write proof π^{wrt}.

Computation: The function $\mathsf{UpdHT}(\mathsf{st}, w)$ computes a new state $\mathsf{st}_{\mathsf{new}}$ as follows:

1. Verify that $\mathsf{st} \neq \perp$. Parse $\mathsf{st} = (\mathsf{rst}, \mathsf{dig}^{\mathsf{mem}}, i^{\mathsf{mem}})$ and $w = (b^{\mathsf{wit}}, b^{\mathsf{mem}}, \pi^{\mathsf{mem}}, \mathsf{dig}^{\mathsf{mem}}_{\mathsf{new}}, \pi^{\mathsf{wrt}})$.
2. Verify that $\mathsf{HT.VerRead}(\mathsf{dig}^{\mathsf{mem}}, i^{\mathsf{mem}}, b^{\mathsf{mem}}, \pi^{\mathsf{mem}}) = 1$.
3. Compute $(\mathsf{rst}_{\mathsf{new}}, i^{\mathsf{mem}}_{\mathsf{new}}, i^{\mathsf{wrt}}, b^{\mathsf{wrt}}, i^{\mathsf{wit}}_{\mathsf{new}}) = \mathsf{step}(M, \mathsf{rst}, b^{\mathsf{mem}}, b^{\mathsf{wit}})$.
4. If $i^{\mathsf{wrt}} = \perp$, verify that $\mathsf{dig}^{\mathsf{mem}}_{\mathsf{new}} = \mathsf{dig}^{\mathsf{mem}}$ and $\pi^{\mathsf{wrt}} = \perp$. Otherwise, verify that $\mathsf{HT.VerWrite}(\mathsf{dig}^{\mathsf{mem}}, i^{\mathsf{wrt}}, b^{\mathsf{wrt}}, \mathsf{dig}^{\mathsf{mem}}_{\mathsf{new}}, \pi^{\mathsf{wrt}}) = 1$.
5. If all verifications pass, output $\mathsf{st}_{\mathsf{new}} = (\mathsf{rst}_{\mathsf{new}}, \mathsf{dig}^{\mathsf{mem}}_{\mathsf{new}}, i^{\mathsf{mem}}_{\mathsf{new}})$. Otherwise, output $\mathsf{st}_{\mathsf{new}} = \perp$.

Fig. 2. The update function UpdHT for proving the correctness of non-deterministic RAM computation in small state using a hash tree HT.

– For $j = 1, \ldots, t$, P computes the next state st_j and witness wit_j given the current state st_{j-1} as follows:
  1. Let $\mathsf{st}_{j-1} = (\mathsf{rst}_{j-1}, \mathsf{dig}^{\mathsf{mem}}_{j-1}, i^{\mathsf{mem}}_{j-1})$.
  2. Compute $(b^{\mathsf{mem}}_{j-1}, \cdot) = \mathsf{HT.Read}^{\mathsf{tree}^{\mathsf{mem}}}(i^{\mathsf{mem}}_{j-1})$.
  3. Compute $(\mathsf{rst}_j, i^{\mathsf{mem}}_j, i^{\mathsf{wrt}}_j, b^{\mathsf{wrt}}_j, i^{\mathsf{wit}}_j) = \mathsf{step}(M, \mathsf{rst}_{j-1}, b^{\mathsf{mem}}_{j-1}, w[i^{\mathsf{wit}}_{j-1}])$.
  4. If $i^{\mathsf{wrt}}_j = \perp$, set $\mathsf{dig}^{\mathsf{mem}}_j = \mathsf{dig}^{\mathsf{mem}}_{j-1}$ and $\pi^{\mathsf{wrt}}_j = \perp$. Otherwise, set $(\mathsf{dig}^{\mathsf{mem}}_j, \cdot) = \mathsf{HT.Write}^{\mathsf{tree}^{\mathsf{mem}}}(i^{\mathsf{wrt}}_j, b^{\mathsf{wrt}}_j)$.
  5. P saves $\mathsf{st}_j = (\mathsf{rst}_j, \mathsf{dig}^{\mathsf{mem}}_j, i^{\mathsf{mem}}_j)$ for the next iteration and erases st_{j-1} and all other working memory aside from its inputs and $\mathsf{tree}^{\mathsf{mem}}$.

P sends st_t to V and erases all of its working memory other than its inputs.

– Let $r = \lceil \log_\lambda t \rceil$. P and V emulate P_r and V_r, respectively, in the interaction

$$\langle P_r(\mathsf{wit}_1, \ldots, \mathsf{wit}_t), V_r \rangle (1^\lambda, (\mathsf{UpdHT}, \mathsf{st}_0, \mathsf{st}_t, t)).$$

In order to compute each of its messages, P_r makes a single pass over the witness $(\mathsf{wit}_1, \ldots, \mathsf{wit}_t)$. To emulate this, P computes each message for P_r as follows:

- Compute $(\mathsf{tree}^{\mathsf{mem}}, \mathsf{dig}^{\mathsf{mem}}_0) = \mathsf{HT.Hash}(\mathsf{hk}, D)$ and initialize $\mathsf{st}_0 = (\mathsf{rst}_0, \mathsf{dig}^{\mathsf{mem}}_0, i^{\mathsf{mem}}_0)$.
- For $j = 1, \ldots, t$, P does the following:

1. Let $\mathsf{st}_{j-1} = (\mathsf{rst}_{j-1}, \mathsf{dig}^{\mathsf{mem}}_{j-1}, i^{\mathsf{mem}}_{j-1})$.
2. Compute $(b^{\mathsf{mem}}_{j-1}, \pi^{\mathsf{mem}}_{j-1}) = \mathsf{HT.Read}^{\mathsf{tree}^{\mathsf{mem}}}(i^{\mathsf{mem}}_{j-1})$.
3. Compute $(\mathsf{rst}_j, i^{\mathsf{mem}}_j, i^{\mathsf{wrt}}_j, b^{\mathsf{wrt}}_j, i^{\mathsf{wit}}_j) = \mathsf{step}(M, \mathsf{rst}_{j-1}, b^{\mathsf{mem}}_{j-1}, w[i^{\mathsf{wit}}_{j-1}])$.
4. If $i^{\mathsf{wrt}}_j = \bot$, set $\mathsf{dig}^{\mathsf{mem}}_j = \mathsf{dig}^{\mathsf{mem}}_{j-1}$ and $\pi^{\mathsf{wrt}}_j = \bot$. Otherwise, set $(\mathsf{dig}^{\mathsf{mem}}_j, \pi^{\mathsf{wrt}}_j) = \mathsf{HT.Write}^{\mathsf{tree}^{\mathsf{mem}}}(i^{\mathsf{wrt}}_j, b^{\mathsf{wrt}}_j)$.
5. P saves $\mathsf{st}_j = (\mathsf{rst}_j, \mathsf{dig}^{\mathsf{mem}}_j, i^{\mathsf{mem}}_j)$ and witness $\mathsf{wit}_j = (w[i^{\mathsf{wit}}_{j-1}], b^{\mathsf{mem}}_{j-1}, \pi^{\mathsf{mem}}_{j-1}, \mathsf{dig}^{\mathsf{mem}}_j, \pi^{\mathsf{wrt}}_j)$ and erases st_{j-1} and all other working memory aside from its inputs and $\mathsf{tree}^{\mathsf{mem}}$.
6. P emulates P_r providing access to wit_j until P_r reads from the next witness wit_{j+1}, at which point P erases wit_j and continues in the loop in order to compute wit_{j+1}.

- Once P_r has computed its next message, P sends it to V and erases all of its working memory other than its inputs.

To emulate V_r, V simply needs to run its code given the transcript of messages from P. At the end of the emulated interaction, V outputs 1 if rst_t corresponds to an accepting state and V_r outputs 1.

Acknowledgments. Cody Freitag is supported by a Khoury College Distinguished Postdoctoral Fellowship. His work was partially done while at Cornell Tech and Boston University, and he is supported in part by the NSF Graduate Research Fellowship under Grant No. DGE-2139899, DARPA Award HR00110C0086, AFOSR Award FA9550-18-1-0267, NSF CNS-2128519, and DARPA under Agreement No. HR00112020023.

Omer Paneth is a member of the Checkpoint Institute of Information Security and is supported by an Azrieli Faculty Fellowship, Len Blavatnik and the Blavatnik Foundation and ISF grant 1789/19. Supported in part by AFOSR Award FA9550-23-1-0312. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or AFOSR.

Rafael Pass is supported in part by AFOSR Awards FA9550-18-1-0267, FA9550-23-1-0387, FA9550-23-1-0312, ISF Grant No. 2338/23 and an Algorand Foundation award. This material is based upon work supported by DARPA under Agreement No. HR00110C0086. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government, DARPA, AFOSR or the Algorand Foundation.

References

1. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: lightweight sublinear arguments without a trusted setup. In: CCS, pp. 2087–2104. ACM (2017)
2. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, STOC, pp. 21–31 (1991)
3. Badrinarayanan, S., Kalai, Y.T., Khurana, D., Sahai, A., Wichs, D.: Succinct delegation for low-space non-deterministic computation. In: STOC, pp. 709–721. ACM (2018)

4. Bangalore, L., Bhadauria, R., Hazay, C., Venkitasubramaniam, M.: On black-box constructions of time and space efficient sublinear arguments from symmetric-key primitives. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022. LNCS, vol. 13747, pp. 417–446. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22318-1_15
5. Barak, B.: How to go beyond the black-box simulation barrier. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS, pp. 106–115 (2001)
6. Barak, B., Goldreich, O.: Universal arguments and their applications. SIAM J. Comput. **38**(5), 1661–1694 (2008)
7. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 390–420. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_28
8. Ben-Or, M., et al.: Everything provable is provable in zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 37–56. Springer, New York (1990). https://doi.org/10.1007/0-387-34799-2_4
9. Ben-Or, M., Goldwasser, S., Kilian, J., Wigderson, A.: Multi-prover interactive proofs: how to remove intractability assumptions. In: STOC, pp. 113–131. ACM (1988)
10. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast Reed-Solomon interactive oracle proofs of proximity. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, Prague, Czech Republic, 9–13 July 2018. LIPIcs, vol. 107, pp. 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). https://doi.org/10.4230/LIPIcs.ICALP.2018.14
11. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 701–732. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_23
12. Ben-Sasson, E., Chiesa, A., Gabizon, A., Riabzev, M., Spooner, N.: Interactive oracle proofs with constant rate and query complexity. In: ICALP. LIPIcs, vol. 80, pp. 40:1–40:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
13. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: On the concrete efficiency of probabilistically-checkable proofs. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) Symposium on Theory of Computing Conference, STOC 2013, Palo Alto, CA, USA, 1–4 June 2013, pp. 585–594. ACM (2013). https://doi.org/10.1145/2488608.2488681
14. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_2
15. Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.P.: Short PCPs verifiable in polylogarithmic time. In: 20th Annual IEEE Conference on Computational Complexity (CCC 2005), San Jose, CA, USA, 11–15 June 2005, pp. 120–134. IEEE Computer Society (2005). https://doi.org/10.1109/CCC.2005.27
16. Ben-Sasson, E., Kaplan, Y., Kopparty, S., Meir, O., Stichtenoth, H.: Constant rate PCPs for circuit-sat with sublinear query complexity. J. ACM **63**(4), 32:1–32:57 (2016)
17. Ben-Sasson, E., Sudan, M.: Short PCPs with polylog query complexity. SIAM J. Comput. **38**(2), 551–607 (2008)
18. Bitansky, N., et al.: The hunting of the SNARK. J. Cryptol. **30**(4), 989–1066 (2017)

19. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) Symposium on Theory of Computing Conference, STOC 2013, Palo Alto, CA, USA, 1–4 June 2013, pp. 111–120. ACM (2013). https://doi.org/10.1145/2488608.2488623
20. Bitansky, N., Canetti, R., Paneth, O., Rosen, A.: On the existence of extractable one-way functions. SIAM J. Comput. **45**(5), 1910–1952 (2016)
21. Bitansky, N., Chiesa, A.: Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 255–272. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_16
22. Bitansky, N., Chiesa, A., Ishai, Y., Paneth, O., Ostrovsky, R.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_18
23. Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12551, pp. 168–197. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64378-2_7
24. Block, A.R., Holmgren, J., Rosen, A., Rothblum, R.D., Soni, P.: Time- and space-efficient arguments from groups of unknown order. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 123–152. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_5
25. Blumberg, A.J., Thaler, J., Vu, V., Walfish, M.: Verifiable computation using multiple provers. IACR Cryptology ePrint Archive, p. 846 (2014)
26. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear PCPs. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 67–97. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_3
27. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_12
28. Boyle, E., Pass, R.: Limits of extractability assumptions with distributional auxiliary input. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9453, pp. 236–261. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48800-3_10
29. Brakerski, Z., Brodsky, M.F., Kalai, Y.T., Lombardi, A., Paneth, O.: SNARGs for monotone policy batch NP. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023. LNCS, vol. 14082, pp. 252–283. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38545-2_9
30. Brakerski, Z., Holmgren, J., Kalai, Y.T.: Non-interactive delegation and batch NP verification from standard computational assumptions. In: STOC, pp. 474–482. ACM (2017)
31. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society (2018)
32. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 677–706. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_24

33. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 738–768. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_26
34. Choudhuri, A.R., Garg, S., Jain, A., Jin, Z., Zhang, J.: Correlation intractability and SNARGs from sub-exponential DDH. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023. LNCS, vol. 14084, pp. 635–668. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38551-3_20
35. Choudhuri, A.R., Jain, A., Jin, Z.: SNARGs for P from LWE. In: 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS, pp. 68–79 (2021)
36. Damgård, I.B.: A design principle for hash functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_39
37. Damgård, I., Fujisaki, E.: A statistically-hiding integer commitment scheme based on groups with hidden order. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 125–142. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36178-2_8
38. Devadas, L., Goyal, R., Kalai, Y., Vaikuntanathan, V.: Rate-1 non-interactive arguments for batch-NP and applications. In: FOCS, pp. 1057–1068. IEEE (2022)
39. Dinur, I.: The PCP theorem by gap amplification. J. ACM **54**(3), 12 (2007)
40. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: SPARKs: succinct parallelizable arguments of knowledge. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 707–737. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_25
41. Feige, U., Goldwasser, S., Lovász, L., Safra, S., Szegedy, M.: Interactive proofs and the hardness of approximating cliques. J. ACM **43**(2), 268–292 (1996)
42. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12
43. Freitag, C., Pass, R., Sirkin, N.: Parallelizable delegation from LWE. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022. LNCS, vol. 13748, pp. 623–652. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22365-5_22
44. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptology ePrint Archive, p. 953 (2019)
45. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37
46. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: STOC, pp. 99–108. ACM (2011)
47. Goldreich, O., Krawczyk, H.: On the composition of zero-knowledge proof systems. In: International Colloquium on Automata, Languages, and Programming, ICALP, pp. 268–282 (1990)
48. Holmgren, J., Rothblum, R.: Delegating computations with (almost) minimal time and space overhead. In: FOCS, pp. 124–135. IEEE Computer Society (2018)
49. Hulett, J., Jawale, R., Khurana, D., Srinivasan, A.: SNARGs for P from Subexponential DDH and QR. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022. LNCS, vol. 13276, pp. 520–549. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-07085-3_18

50. Impagliazzo, R.: A personal view of average-case complexity. In: Proceedings of the Tenth Annual Structure in Complexity Theory Conference, pp. 134–147. IEEE Computer Society (1995)
51. Ishai, Y., Kushilevitz, E., Ostrovsky, R.: Efficient arguments without short PCPs. In: CCC, pp. 278–291. IEEE Computer Society (2007)
52. Jawale, R., Kalai, Y.T., Khurana, D., Zhang, R.Y.: SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. In: STOC 2021: 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC, pp. 708–721 (2021)
53. Kalai, Y., Lombardi, A., Vaikuntanathan, V., Wichs, D.: Boosting batch arguments and RAM delegation. In: STOC, pp. 1545–1552. ACM (2023)
54. Kalai, Y., Paneth, O.: Delegating RAM computations. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 91–118. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_4
55. Kalai, Y.T., Paneth, O., Yang, L.: How to delegate computations publicly. In: STOC, pp. 1115–1124. ACM (2019)
56. Kalai, Y.T., Raz, R., Rothblum, R.D.: How to delegate computations: the power of no-signaling proofs. In: STOC, pp. 485–494. ACM (2014)
57. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11
58. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Proceedings of the 24th Annual ACM Symposium on Theory of Computing, STOC, pp. 723–732 (1992)
59. Lindell, Y.: Parallel coin-tossing and constant-round secure two-party computation. J. Cryptol. **16**(3), 143–184 (2003)
60. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: zero-knowledge snarks from linear-size universal and updatable structured reference strings. In: CCS, pp. 2111–2128. ACM (2019)
61. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_32
62. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_21
63. Micali, S.: CS proofs (extended abstracts). In: 35th Annual Symposium on Foundations of Computer Science, FOCS, pp. 436–453 (1994)
64. Micali, S., Pass, R.: Local zero knowledge. In: STOC, pp. 306–315. ACM (2006)
65. Mie, T.: Short PCPPs verifiable in polylogarithmic time with O(1) queries. Ann. Math. Artif. Intell. **56**(3–4), 313–338 (2009)
66. Paneth, O., Pass, R.: Incrementally verifiable computation via rate-1 batch arguments. In: FOCS, pp. 1045–1056. IEEE (2022)
67. Paneth, O., Rothblum, G.N.: On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10678, pp. 283–315. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70503-3_9
68. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. SIAM J. Comput. **50**(3) (2021)
69. Ron-Zewi, N., Rothblum, R.D.: Local proofs approaching the witness length [extended abstract]. In: FOCS, pp. 846–857. IEEE (2020)

70. Setty, S.: Spartan: efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 704–737. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_25
71. Simon, D.R.: Finding collisions on a one-way street: can secure hash functions be based on general assumptions? In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 334–345. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054137
72. Szepieniec, A., Zhang, Y.: Polynomial IOPs for linear algebra relations. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022. LNCS, vol. 13177, pp. 523–552. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-97121-2_19
73. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 1–18. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78524-8_1
74. Waters, B., Wu, D.J.: Batch arguments for np and more from standard bilinear group assumptions. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022. LNCS, vol. 13508, pp. 433–463. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15979-4_15