Delegating Computations with (almost) Minimal Time and Space Overhead

Justin Holmgren[*] Ron D. Rothblum[†]

September 13, 2018

Abstract

The problem of verifiable delegation of computation considers a setting in which a client wishes to outsource an expensive computation to a powerful, but untrusted, server. Since the client does not trust the server, we would like the server to certify the correctness of the result. Delegation has emerged as a central problem in cryptography, with a flurry of recent activity in both theory and practice. In all of these works, the main bottleneck is the overhead incurred by the server, both in time and in space.

Assuming (sub-exponential) LWE, we construct a one-round argument-system for proving the correctness of any time T and space S RAM computation, in which both the verifier and prover are highly efficient. The verifier runs in time $n \cdot \mathrm{polylog}(T)$ and space $\mathrm{polylog}(T)$, where n is the input length. Assuming $S \geq \max(n, \mathrm{polylog}(T))$, the prover runs in time $\widetilde{O}(T)$ and space $S + o(S)$, and in many natural cases even $S + \mathrm{polylog}(T)$. Our solution uses somewhat homomorphic encryption but, surprisingly, only requires homomorphic evaluation of arithmetic circuits having multiplicative depth (which is a main bottleneck in homomorphic encryption) $\log_2 \log(T) + O(1)$.

Prior works based on standard assumptions had a T^c time prover, where $c \geq 3$ (at the very least). As for the space usage, we are unaware of any work, even based on non-standard assumptions, that has space usage $S + \mathrm{polylog}(T)$.

Along the way to constructing our delegation scheme, we introduce several technical tools that we believe may be useful for future work.

[*]Princeton. Email: `justin.holmgren@princeton.edu`. This research was conducted while the author was at MIT.

[†]Technion. Email: `rothblum@cs.technion.ac.il`. This research was conducted while the author was at MIT and Northeastern.

Contents

1 Introduction

The problem of verifiable delegation of computation, or just *delegation* for short, considers a setting in which a computationally weak device, such as a smart-{phone, watch, home device}, wishes to outsource the computation of a complex function f, on a given input x, to a powerful, but *untrusted*, server (aka "the cloud"). Since the device does not trust the server, a major security concern that arises is that the alleged result y may be incorrect.

A naive solution for this problem is to simply have the client re-compute $f(x)$ and compare with y. However, that defeats the entire purpose of outsourcing the computation. Rather, we would like for the server to *prove* to the client that the computation was done correctly. For such a proof-system to be useful, it must satisfy two key properties:

1. The *verification* procedure, run by the client, must be extremely efficient. In particular it should be much more efficient than computing f by yourself.

2. *Proving* correctness should be relatively efficient. Namely, not much more than the time that it takes to compute f.

In addition, we seek solutions that are *general-purpose* and apply to any computation, rather than specific custom made solutions. Due to the high cost of interaction, we will also restrict our attention to protocols that are *non-interactive*. In such protocols the verifier sends to the prover an input x (together with a short challenge) and gets back a response y as well as a short proof π. Based on this proof, the verifier should able to quickly confirm the correctness of the computation.

The problem of delegating computation has drawn considerable interest in recent years, both of a theoretical nature and even practical implementations. In particular, a delegation scheme (with some additional properties) also forms the core component of a popular crypto currency [BCG$^+$14]. The line of work implementing such schemes has identified the efficiency of the prover as the major bottleneck toward practical solutions (see, e.g., [WB15]). This efficiency refers both to the time overhead incurred by the prover over simply performing the computation *and* to the space overhead - namely, how much additional memory does the prover need in order to prove correctness.

Solutions achieving good asymptotic performance are known under *non-standard* cryptographic assumptions. In particular, these assumptions are not *falsifiable* in the sense of [Nao03, GW11] and more generally are not well understood. This motivates the following question:

> *Under standard cryptographic assumptions, can we construct delegation schemes such that the proving correctness is almost as efficient as simply performing the computation?*

Computational Model. Since we care about the precise overhead incurred by the prover (and in particular care about polynomial factors), it is important to specify what computational model we use. In this work we focus on the standard word RAM model, which is typically viewed as a good approximation for the efficiency of real computers.

1.1 Our Contributions

Our main result is a general-purpose non-interactive delegation scheme for deterministic computations with almost optimal time and space usage for both the verifier and prover.

Theorem 1 (Informally Stated, see Theorem 6.4 and Corollary 6.5)**.** *Let $\mathcal{L}$ be computable by a time T and space S RAM machine, where $\max(\log(T), n) \leq S \leq T$, and let $\zeta > \frac{\mathrm{polylog}(T)}{S}$. Suppose that there exists a sub-exponentially secure somewhat homomorphic encryption scheme. Then, there exists a 2-message argument-system for $\mathcal{L}$ with the following efficiency:*

- *The verifier runs in time $n \cdot \mathrm{polylog}(T)$ and space $\mathrm{polylog}(T)$.*
- *The prover runs in $\tilde{O}(T) \cdot (1/\zeta)$ time and uses $(1+\zeta) \cdot S$ space.*
- *The communication complexity is $\mathrm{polylog}(T)$.*

We emphasize that ζ can be smaller than 1. In particular, assuming $S \geq (\log(T))^c$ for some sufficiently large constant c, we can set $\zeta = 1/\mathrm{polylog}(T)$ to obtain a prover that runs in time $\tilde{O}(T)$ and space $S + o(S)$.

This result builds on, and improves upon, the recent line of work on delegation from standard assumptions initiated by Kalai et al. [KRR13]. Our key improvement over these prior works is in the time and space complexity of the *prover*. In particular, all prior results on non-interactive delegation from standard assumptions had a prover running time of T^c, where at the very least $c \geq 3$.

While the space usage in Theorem 1 is already quite low, we can actually improve upon it for a large and natural class of RAM programs. Specifically, for this class we can reduce the space usage to $S + \mathrm{polylog}(T)$, which is optimal up to the specific *additive* poly-logarithmic factor. Loosely speaking, this class refers to RAM programs that have good "caching" behavior (i.e., do not suffer from too many cache misses using an ideal cache).[1] In particular, this includes Turing machine computations but also other typical classes of computations. We are unaware of any prior general-purpose delegation scheme, even based on *non-standard assumptions*, that simultaneously achieves space usage $S + \mathrm{polylog}(T)$ and $\tilde{O}(T)$ running time for the prover.

Theorem 2 (Informally Stated, see Theorem 6.4 and Corollary 6.6)**.** *Under the same setting as Theorem 1, if $\mathcal{L}$ is computable by a time T and space S "cache friendly" RAM program (e.g., a Turing machine), then there exists a 2-message argument-system for $\mathcal{L}$ with the same parameters as those in Theorem 1, except with the prover running in time $\tilde{O}(T)$ and using space $S + \mathrm{polylog}(T)$.*

It is well known that cache efficiency is critical for practically efficient computing. There is also a large body of theoretical work on the asymptotic number of cache misses incurred by a variety of algorithms (see for example [FLPR99, AV88, Vit06]). Because of these considerations, we find "cache friendly" algorithms to be an interesting target for practical delegation. We believe this class of programs strikes a good balance between the concerns of efficiency vs. generality.

Homomorphic Encryption and Multiplicative Depth. Our reliance on homomorphic encryption is actually quite mild. In particular, the *multiplicative depth*[2] of the prover's homomorphic operation is $\log_2 \log(T) + O(1)$ – a *doubly exponential* improvement over a naive usage. This

[1]More precisely, we require that the machine not suffer from too many cache misses (using an ideal cache) even if we consider the caching behavior wrt to larger blocks of memory. See Section 5.2 and specifically Definition 5.12 for the actual definition.

[2]An arithmetic circuit has multiplicative depth d if every path from the output to an input gate goes through at most d multiplication gates. It is important for us that the encryption performs homomorphic evaluation in a gate-by-gate manner, which all known schemes do. See Sections 3.2 and 11.2 for additional details.

(simultaneously) greatly benefits the efficiency of our argument scheme, as well as improves the cryptographic assumptions on which it can be based.

1. **Efficiency:** A major efficiency bottleneck in existing FHE schemes (e.g., [BGV14, GSW13]) is the *multiplicative depth* of the computation. In particular, shallow computations are supported directly by these schemes without resorting to the expensive bootstrapping process.

2. **Assumptions:** Since we only need homomorphic encryption that supports low depth operations we can rely on weaker cryptographic assumptions. First, we can use "leveled" fully homomorphic encryption (FHE), which is known based on the learning with errors (LWE) assumption. In contrast, full-fledged FHE is only known based on LWE combined with a poorly understood circular security assumption or based on obfuscation [CLTV15].

The fact that our protocol does not require homomorphic evaluation of circuits with multiplicative depth $\Omega(T)$ may initially seem quite surprising. In particular, the computation that we are delegating might be represented by an arithmetic circuit with multiplicative depth $\Omega(T)$. Jumping ahead, the reason that we can obtain such a dramatic saving in the multiplicative depth is by having the prover perform most of the work directly on the plaintext data (i.e., not under the FHE).[3]

Efficient Implementations of No-Signaling MIPs. Our basic approach for proving Theorem 1 follows the line of work on non-interactive delegation for P based on standard assumptions [KRR13, KRR14, KR15, KP16, PR17, BHK17, BKK+18]. Similarly to that line of work, our basic building block is the multi-prover interactive proof-system (MIP) of Babai et al. [BFL91] (and its PCP variant from [BFLS91]). The above works all rely on a transformation from MIPs to non-interactive arguments, originally proposed by Biehl et al. [BMW98], and shown to be secure for MIPs having a strong soundness property in [KRR13]. This strong soundness property is known as *no-signaling soundness* (to be discussed in depth below).

One of our key technical contributions is constructing a variant of the [BFL91] MIP which is both no-signaling sound and for which the prover answers can be generated extremely efficiently.

Theorem 3 (Informally Stated, see Theorem 6.1 and Corollary 6.2)**.** *Let $\mathcal{L}$ be computable by a time T and space S RAM machine, where $\max(\log(T), n) \leq S \leq T$. Let $\zeta > \frac{\mathrm{polylog}(T)}{S}$. Then, there exists a $\mathrm{polylog}(T)$-prover MIP for $\mathcal{L}$ with soundness against no-signaling strategies and the following efficiency properties:*

- *The verifier runs in time $n \cdot \mathrm{polylog}(T)$ and space $\mathrm{polylog}(T)$.*
- *Each of the provers runs in $\tilde{O}(T) \cdot (1/\zeta)$ time and uses $(1+\zeta) \cdot S$ space.*
- *The communication complexity is $\mathrm{polylog}(T)$.*

Setting $\zeta = 1/\mathrm{polylog}(T)$ we get that the provers' running time is $\tilde{O}(T)$ and space usage is $S+o(S)$. Similarly to Theorem 2, we can obtain space usage $S+\mathrm{polylog}(T)$ for *cache friendly* RAM programs (see Corollary 6.3). For comparison, the running time of the prover in prior no-signaling MIPs [KRR13, KRR14, BHK17] was T^c, for an unspecified constant c (which at the very least is $c \geq 3$).

[3]On a related note, we comment that we cannot follow the *private information retrieval* (PIR) variant of the [KRR13, KRR14, BHK17] approach, since it automatically increases the space usage of the prover to $\Omega(T)$.

In fact, Theorem 3 also improves over the most efficient known *classical* MIPs.[4] Specifically, Bitansky and Chiesa [BC12] and Blumberg et al. [BTVW14] show that variants of the [BFL91] MIP can be implemented with a time $\tilde{O}(T)$ and space $S \cdot \mathrm{polylog}(T)$ prover. Prior to our work, MIPs with space usage of $O(S)$ were not known, let alone MIPs with additive space usage $S + o(S)$ or $S + \mathrm{polylog}(T)$.[5] See Section 1.2 for further comparison and discussion on these and other related works.

Remark 1.1 (PCP Length)**.** *Since we mainly care about using this* MIP *as a building block in the construction of an efficient delegation scheme, we did not try to optimize certain parameters. In particular, the length of queries generated by the verifier is slightly super logarithmic. This means that if one views our* MIP *as a probabilistically checkable proof (*PCP*), the* PCP *proof length has* quasi-polynomial *length. We believe that this is not inherent, and that our techniques can easily yield a* PCP *of polynomial length (using techniques dating back to [BFLS91]). Since it would make our proof significantly more cumbersome, for the sake of simplicity, we avoid doing so (see Remark 2.2 for additional details).*

1.1.1 Additional Technical Contributions

In the process of establishing our main results (i.e., Theorems 1 and 2), we also obtain some technical results and introduce new techniques that we believe to be of independent interest.

Efficient Implementation of the BFLS PCP and Caching. As noted above, one of our key steps is showing that, for our variant of the [BFLS91] PCP, it is possible to compute any symbol of the proof string very efficiently. Consider for example a RAM program M and an input x. We show an efficient algorithm P for computing any individual symbol of the corresponding proof string.

At a high level, this is done by giving the prover *streaming* access to the underlying computation transcript. The prover algorithm P (which is stateful) then proceeds in T steps, where T is the running time of M on x. In the i^{th} step, P is given oracle access to the i^{th} configuration of M on x, as well as (succinct representations of) the differences between the i^{th} and $(i-1)^{th}$ configuration. In case M is a *Turing machine*, we can implement P to run in *amortized* time $\mathrm{polylog}(T)$ per step, while also maintaining an internal state of size $\mathrm{polylog}(T)$.

As mentioned above, our variant of the BFLS allows M to be a RAM machine (rather than just a Turing machine). In this case, we give a time/space trade-off for the efficiency of P, that depends on the memory access pattern of M on the given input x. If M's memory access pattern is mostly sequential (as is the case for Turing machines), then we can similarly implement each step of P in (amortized) time $\mathrm{polylog}(T)$. More generally, we count the number of *cache misses* $Q(\tau, B)$ incurred by M's memory access pattern with an ideal cache[6] of size τ and with respect to a partition of the memory into blocks of size B.

However, as mentioned above, our variant of the BFLS PCP allows M to be a RAM machine (rather than merely a Turing machine). In this case, we give a time/space trade-off that depends on the memory access pattern of M on the given input x. If M's memory access pattern is mostly

[4]The MIP of Theorem 3 refers to deterministic computations (which is inherent for no-signaling soundness [DLN+01, IKM09]). However, essentially the same MIP can be shown to achieve *classical* soundness even for non-deterministic computations (see Remark 10.15).

[5]We remark that obtaining *additive* space overhead is significantly more difficult than constant multiplicative space overhead. For example, we cannot even afford to assume that the space usage is a power of two.

[6]I.e., a cache designed when one knows the memory access pattern in advance.

sequential (as is the case for Turing machines), then we can similarly implement each step of P in (amortized) time $\text{polylog}(T)$. More generally, we count the number of *cache misses* $Q(\tau, B)$ incurred by M's memory access pattern with an ideal cache[7] of size τ and with respect to a partition of the memory into blocks of size B.

For any τ, we show that it is possible to compute each step of P in amortized time $\max_B Q(\tau, B) \cdot \frac{S}{B} \cdot \text{polylog}(T)$ and space $\tau \cdot \text{polylog}(T)$. Our analysis utilizes a classical result in caching theory, due to Sleator and Tarjan [ST85], that shows that a least-recently-used (LRU) cache is nearly as good as an ideal cache.

Avoiding the Augmented Circuit of [KRR14]. Building on the work of [KRR13, KRR14], we give a new technique for establishing the no-signaling soundness of MIPs. Most notably, we manage to avoid the complicated "augmented circuit" construction of [KRR14]. We mention that similar ideas for avoiding the KRR augmented circuit appear in the concurrent and independent works of Chiesa, Manohar, and Shinkar [CMS18] and Kiyoshima[Kiy18], however those two works consider the setting of the exponential–length Hadamard PCP and in particular do not show how to remove the augmented circuit in the setting of the [KRR14] no-signaling PCP.

We remark that although our *construction* is a significant simplification, the *analysis* remains unfortunately quite complex. In particular, our analysis relies crucially on insights developed in [KRR14]. See Sections 2.2 and 10 for more details.

Abstractions and Computational Models. Our argument schemes (Theorems 1 and 2) have many moving pieces. We are able however to identify suitable abstractions that allow us to describe the construction in a modular way. We are hopeful that these ideas will be useful for future work as well.

For example, our high level approach (following [BMW98, KRR13]) is to transform our efficient no-signaling MIP into an argument-scheme by encrypting the MIP queries and have the prover generate the answers homomorphically. This is problematic in terms of efficiency, since even if the MIP prover can be efficiently implemented on a RAM machine, homomorphic encryption schemes natively support the homomorphic evaluation of *circuits*. Thus, we have to transform the MIP prover into a circuit, which is sub-optimal.

We observe however that the prover has the main input x in the clear, and only gets the MIP query q in encrypted form. To get improved efficiency, we show that the MIP prover, given x in the clear can be used to generate a sequence of simple arithmetic instructions to be done on q. We refer to this as an *arithmetic straight-line program* (ASLP).

Stated otherwise, we show that each MIP prover is computable by (uniformly generated given x) ASLP. This notion allows us to:

1. Avoid needlessly generic reductions from time-T RAM computations to circuits of size $\Omega(T^2)$.

2. Cleanly separate the provers' work "in the clear" from the work that it does under FHE.

3. Easily keep track of the multiplicative depth of provers' FHE computations.

Another model that we utilize is tree machines, a variant of Turing machines in which the work tape is laid out as an infinite complete binary tree (rather than the standard linear tape). These

[7] I.e., a cache designed when one knows the memory access pattern in advance.

are useful for us because on the one hand, they can emulate RAMs very efficiently, and on the other hand they have a local structure that is amenable to PCP techniques (similar to Turing machines).[8] See Section 2.3 for details.

Low Degree Extensions of Read Once Branching Programs. We describe clean and versatile conditions which guarantee that a function's *minimal*[9] low-degree extension is efficiently computable. Specifically, we show that the low-degree extension of any size-S read-once branching program is computable in $O(S)$ ring operations. The proof of this is extremely simple, yet it immediately subsumes and simplifies previous lengthy and ad-hoc algorithms for computing low-degree extensions of specific functions in [BHK17] and [GR17]. In algebra-based proof-systems (e.g., [GKR15, KRR13, KRR14, KR15, KP16, BHK17, GR17]), it is quite common that one or more parties must evaluate low-degree extensions of simple functions (e.g., a gate-indicator function for a circuit), and we believe that our abstraction will prove useful in future work.

Remark 1.2 (On (Non-Minimal) Low Degree Extensions of Bounded Depth Computations)**.** *We contrast this to another approach to generically computing simple functions' (non-minimal) low-degree extensions, that has been used in the literature (e.g., [BF91, LFKN92, Sha92, GKR15, PR17]). These works rely on the observation that for any Boolean function f that is computable by a small,* low-depth *circuit, it is possible to efficiently compute a* moderately *low-degree extension of f by directly arithmetizing the circuit. This approach is not suitable for us; as observed in the context of interactive proofs by Cormode et al. [CMT12] (in their fast implementation of the [GKR15] protocol), there can be drastic efficiency benefits to using the minimal low degree extension. The same type of efficiency benefits are applicable also in our setting.*

1.2 Comparison to Prior Works

Delegation based on Standard Assumptions. As noted above, our scheme builds on the recent line of work on non-interactive delegation schemes for polynomial-time computations based on no-signaling PCPs [KRR13, KRR14, KR15, KP16, PR17, BHK17, BKK$^+$18]. As mentioned above, the prover in all of these schemes runs in time $\Omega(T^3)$.

One route to improved prover efficiency is to construct a no-signaling PCP whose proof strings can be generated in time $\tilde{O}(T)$. Such PCPs with *classical* soundness are known to exist (cf. [BS08, BCGT13, BBC$^+$17, BBHR18]) but are only known to be constructible using $\Omega(T)$ *space*. Constructing short PCPs that can be generated with minimal time *and* space overhead is a fascinating open question (let alone such PCPs that also have no-signaling soundness).

Our approach follows a different route. Rather than using a PIR scheme, we use (somewhat) homomorphic encryption. Loosely speaking, this allows the cost of our prover to be proportional to just computing a single PCP symbol, rather than the entire string. A similar observation was made by Bitansky and Chiesa [BC12] (see more details below).

[8]Prior works that constructed efficient PCPs and MIPs for RAM computations used a *non-deterministic* reduction from RAM machines to Turing machines. In contrast, since in the no-signaling context proof-systems can only support *deterministic computations* [DLN$^+$01, IKM09], we cannot utilize non-deterministic reductions. See Section 1.2 for additional details.

[9]By *minimal* low degree extension of a function $f : H^m \to \mathbb{F}$ over a field $\mathbb{F}$, we refer to the *unique* individual degree $|H|-1$ polynomial $\hat{f} : \mathbb{F}^m \to \mathbb{F}$ that agrees with f on H^m. In contrast, functions have many other "low degree extensions" in which the total degree of the functions is larger but still bounded.

Returning to the comparison with the line of work on no-signaling based delegation schemes, we remark that our main result has two drawbacks compared to the recent scheme of Brakerski et al. [BHK17]:

1. First, [BHK17] only rely on polynomial hardness assumptions (e.g., standard computational PIR) whereas we rely on sub-exponential hardness. We believe that we can show that our construction can also be based on polynomial hardness assumptions, but the prover and verifier running times would increase to $T^{1+\varepsilon}$ and T^{ε}, respectively. Also, basing our construction on polynomial-time hardness assumptions would introduce some complications in the proof (e.g., handling computational no-signaling provers) as in [BHK17], that we prefer to avoid.

2. Also, [BHK17] show that their delegation scheme has *adaptive* soundness meaning that soundness holds even if the cheating prover can decide on the computational statement to be proved after seeing the first message from the verifier. We believe that our construction can similarly be shown to have adaptive soundness but we prefer to avoid doing so, given the complications involved (which were already done in [BHK17]).

Lastly, we mention that in a very recent work, Badrinarayanan et al. [BKK+18] extended this line of work to give a delegation scheme for *bounded-space* non-deterministic computations. We believe that our techniques are applicable also in their context and should yield an extension of our result for the bounded-space non-deterministic setting.

Interactive Delegation Schemes. Kilian [Kil92] constructed a 4-message delegation protocol based on PCPs and collision resistant hash functions. Micali [Mic00] extended this to a non-interactive solution in the *random oracle model* (which is a somewhat controversial model [CGH04]) and his techniques were recently further refined by [BCS16] (using an interactive notion of PCPs). We remark that the prover's *space* complexity in all of these protocols grows (at least) linearly with the *time* complexity of the computation. These solutions are therefore highly inefficient for small space computations. In contrast, in our protocol the space complexity is only slightly larger than the *space* complexity of the original computation.

As mentioned above, Bitansky and Chiesa [BC12] construct an *interactive* delegation scheme in which the prover's time overhead is almost linear, and the space usage grows by a poly-logarithmic multiplicative factor. Our main results improve on [BC12] in several ways: (1) our delegation protocol is *non-interactive*[10], (2) our construction uses only somewhat homomorphic encryption (for $\log_2 \log(T) + O(1)$ multiplicative depth) whereas [BC12] use full-fledged FHE, (3) the space usage in our scheme is $S + o(S)$ (and in some cases even $S + \text{polylog}(T)$), see Theorem 2), whereas [BC12] have space usage $S \cdot \text{polylog}(T)$; and (4) [BC12] perform have nested homomorphic evaluations (i.e., homomorphic evaluation of a homomorphic evaluation of the computation) which we do not. However, an advantage of the [BC12] result is that they can handle also non-deterministic computations whereas we are limited to deterministic computations. An improved construction with roughly the same asymptotic behavior as [BC12] (but much better concrete efficiency) was given by Blumberg et al. [BTVW14].

Other interactive solutions from the literature provide the much stronger guarantee of *statistical soundness*, but inherently handle only certain restricted classes of computations. In particular,

[10] [BC12] also give a separate *non-interactive* scheme that relies on a non-standard, and in particular *non-falsifiable*, assumption.

Goldwasser et al. [GKR15] and Reingold et al. [RRR16] construct such protocols for computations that are highly parallel-izable (i.e., bounded depth) or use bounded space, respectively. Both of these protocols require a large number of rounds, which can be minimized via the Fiat-Shamir [FS86] transformation, obtaining heuristic soundness (see also [KRR17, CCRR18]). We remark that there has recently been a very successful line of work (c.f. [CMT12, SVP$^+$12, Tha13, ZGK$^+$17, WJB$^+$17]) implementing, optimizing and improving the [GKR15] protocol.

Additional Delegation Schemes. A separate line of work [Gro10, Lip12, BCCT12a, DFH12, GLR11, BCCT12b, GGPR12, BC12, BCI$^+$13] constructs non-interactive delegation schemes for *non-deterministic* computations based on non-falsifiable assumptions (which is inherent by [GW11]). These are non standard assumptins whose security is much less understood.

Assuming indistinguishability obfuscation (IO), one can construct a delegation scheme which preserves the underlying computation's time and space complexity (up to multiplicative polynomial factors in the security parameter). This holds not only for computations modeled as Turing machines [KLW15], but also RAM machines [CH16] and even Parallel RAM machines [CCC$^+$16]. These schemes all also enjoy a number of other desirable properties, such as public verifiability, computation privacy, and the ability to delegate a persistent and mutable database.

Still, current candidate constructions of IO are problematic, both in the astronomically large polynomial factors in current candidate constructions, and more fundamentally on the fact that their security is still far from being well understood.

We also mention that many works consider delegation in the *pre-processing* model [GGP10, CKV10, AIK10, PRV12], in which the verifier first has an expensive pre-processing stage, as complex as performing the entire computation. Later, in the online phase the verifier is very efficient. In contrast, our delegation protocol does not require any pre-processing for the verifier.

1.3 Organization

We start in Section 2 by providing a high-level technical overview of our construction and main ideas. In Section 3 we provide preliminary definitions, notations and facts. In Section 4 we formally define the computational models that we consider in this work, including RAM machines, Tree machines, and arithmetic straight line programs. In Section 5 we show technical results that will be used for our main results but we believe to be of independent interest: evaluating low degree extensions of read-once branching programs and basic facts on caching theory.

In Section 6 we formally state our main results (the statements of these results rely on notions defined in the prior sections). The proof of these results is established in the subsequent sections. In Section 7 we give the construction of a PCP. In Section 8 we establish the running time of the PCP verifier. In Section 9 we give our efficient implementation of the prover strategy. In Section 10 we show that our PCP has soundness against no-signaling strategies. Finally, in Section 11 we put everything together and prove our main results.

2 Technical Overview

In this section we give an overview of the main technical ideas that we employ.

Our approach revolves around the celebrated PCP of [BFLS91]. Our presentation of this PCP departs from the textbook description (cf. [Sud00]) because we directly arithmetize the *Turing* *machine* computation, without first converting it into a Circuit-SAT instance. This is an important conceptual step since later on we will not be able to afford the overhead incurred by the transformation to a circuit.[11] Throughout this section we refer to this PCP as the BFLS PCP.

Section Organization. We start in Section 2.1 by giving a primer on the BFLS PCP. In Section 2.2 we describe how we establish *no-signaling soundness* of the BFLS PCP without relying on the augmented circuit construct of [KRR14]. In Section 2.3 we describe a new technique that allows us to efficiently handle RAM computations (rather than just Turing machine computations). In Section 2.4 we give a high level outline of our efficient implementation of the BFLS PCP (viewed as an MIP). In Section 2.5 we describe a technique by which we decrease the *multiplicative depth* of the circuits that we are homomorphically evaluating by a *doubly exponential* factor (as compared to a naive implementation). Lastly, in Section 2.6 we describe our simple method of computing low degree extensions of simple functions.

2.1 A Primer on BFLS Style PCPs

In this section we recall the basic outline of the BFLS PCP, as applied to standard Turing machine computations.

Let $\mathcal{L} \in \mathsf{TISP}(T, S)$ be a language and let M be a time T and space S (single-tape) Turing machine for deciding $\mathcal{L}$. For simplicity we assume that T and S are powers of two and denote by $t = \log_2(T)$ and $s = \log_2(S)$.[12] We construct a PCP for deciding $\mathcal{L}$ as follows.

Let $x \in \{0,1\}^n$ be an input for M. Define a function $X : \{0,1\}^t \times \{0,1\}^s \to \Gamma \times (Q \cup \perp)$, where Q is the set of internal control states of the Turing machine (and $\perp \notin Q$), as follows. For every $i \in \{0,1\}^t$ and $j \in \{0,1\}^s$ we set $X(i,j) = (\gamma, q)$ where γ is the symbol of the j^{th} position of the work tape at time step i. As for q, in case the machine head is located at position j at time step i then q is the internal control state of the machine at time step i. Otherwise, $q = \perp$. Thus, q both indicates whether the machine head is located at position j (at time i) and, if that is the case, also indicates the internal control state of the machine.

The truth table of the function X fully describes the evaluation of the machine M on input x. However, it is extremely sensitive to small changes in the sense that a computation that is entirely incorrect can be described by a function X that is locally consistent *almost* everywhere. We would like to add redundancy to X by encoding it via an error-correcting code - specifically the *multi-linear extension* - as follows.

Let $\mathbb{F}$ be a sufficiently large finite field (in particular $|\mathbb{F}| \geq |\Gamma| \cdot (|Q| + 1)$). We associate the set $\Gamma \times (Q \cup \{\perp\})$ with some subset of $\mathbb{F}$. Let $\hat{X} : \mathbb{F}^{t+s} \to \mathbb{F}$ be the (unique) multilinear polynomial such that $\hat{X}$ agrees with X on every input in $\{0,1\}^{t+s}$ (via the above association).[13]

Remark 2.1 (A Useful Notational Convention)**.** *Throughout this overview, and also later in the technical sections, we use the following useful notational convention. Blackboard bold lowercase (e.g., $\mathbb{z}$) is used for field elements whereas standard bold lowercase (e.g., $\mathbf{z}$) is used for bits. Likewise, we use $\bar{\mathbb{z}}$ to denote vectors of field elements and $\bar{\mathbf{z}}$ to denote bit strings.*

[11] In particular, as we elaborate on later, it is not clear how to use the efficient *non-deterministic* reductions from RAMs to circuits [GS89, BCGT13] in the no-signaling setting.

[12] In the actual construction we cannot assume that S is a power of two since this would lead to potential doubling of the space usage, which we would like to avoid.

[13] For more details on the multilinear extension, and in particular a proof that such a polynomial $\hat{X}$ exists, see Section 3.1.

Remark 2.2 (An Aside: Multilinear vs. Low Degree Extensions)**.** *The fact that we use the* multilinear extension *rather than the so-called "low degree extension" leads to a* PCP *whose length is quite long. In fact, the length will be* super polynomial.[14] *However, since we will actually use this* PCP *as an* MIP*, the important measure of efficiency is not the length of the* PCP *nor even the time that it takes to write down the entire* PCP*. Rather, the main resource that we care about is the complexity of retrieving any* particular *symbol from the* PCP *proof string.*

We further remark that we believe that all of our results can be adapted with only mild complications to yield a PCP *with a polynomial length proof (by using the low degree extension encoding). However, to avoid handling these additional complications we use the multilinear extension encoding.*

While $\hat{X}$ is much more "robust" than X, we also need a method by which to verify its correctness. To do so we shall express the correctness of the computation by relations between all pairs of points in the computation transcript.

To this end, we first define four functions $\phi_{+1} : \{0,1\}^t \times \{0,1\}^t \to \{0,1\}$ and $\phi_{\to}, \phi_{\leftarrow}, \phi_{\emptyset} : \{0,1\}^s \times \{0,1\}^s \to \{0,1\}$ as follows. The function ϕ_{+1} gets as input two points $\bar{\mathbf{z}}_1, \bar{\mathbf{z}}_2 \in \{0,1\}^t$ and, viewing them as integers, outputs 1 if and only if $\bar{\mathbf{z}}_2 = \bar{\mathbf{z}}_1 + 1$.[15] Thus, we think of the function ϕ_{+1} as getting as input a pair of indices corresponding to two time steps and outputting 1 if and only if they are consecutive. Likewise, the function $\phi_{\to}$ (resp., $\phi_{\leftarrow}$) gets as input indices of two points on the tape and outputs 1 iff the second point is immediately to the right (resp., left) of the first point. Lastly, the function $\phi_{\emptyset}$ gets as input a pair of points and checks that they are equal to one another.

The correctness of a computation transcript can be described by local consistency constraints. Furthermore, using the specific structure that we imposed on X, we can describe the correctness of the computation using only constraints on pairs of values $(\bar{\mathbf{a}}_1, \bar{\mathbf{b}}_1) \in \{0,1\}^t \times \{0,1\}^s$ and $(\bar{\mathbf{a}}_2, \bar{\mathbf{b}}_2) \in \{0,1\}^t \times \{0,1\}^s$ of X. For each constraint, it must hold that the points lie in subsequent layers: i.e., $\bar{\mathbf{a}}_2 = \bar{\mathbf{a}}_1 + 1$. Likewise, by the sequential nature of Turing machine computations (and again using the structure of X), $\bar{\mathbf{b}}_1$ and $\bar{\mathbf{b}}_2$ must be neighboring, i.e., $\bar{\mathbf{b}}_2 \in \{\bar{\mathbf{b}}_1 - 1, \bar{\mathbf{b}}_1, \bar{\mathbf{b}}_1 + 1\}$. Even more importantly the local constraints are universal, in the sense that they only depend on which of three options for $\bar{\mathbf{b}}_2$ holds, and do not depend on the specific values of $\bar{\mathbf{a}}_1$ and $\bar{\mathbf{b}}_1$ (where again we are ignoring edge cases). We introduce validation functions $V_{\to}, V_{\leftarrow}, V_{\emptyset} : (\Gamma \times (Q \cup \{\bot\}))^2 \to \{0,1\}$ for the three types of local constraints that we have. Each function gets as input values corresponding to a pair of points in X, and checks that these values satisfy one of the three types of constraints as follows:

- $V_{\to}$ handles constraints in which $\bar{\mathbf{b}}_2 = \bar{\mathbf{b}}_1 + 1$,
- $V_{\leftarrow}$ handles constraints in which $\bar{\mathbf{b}}_2 = \bar{\mathbf{b}}_1 - 1$; and
- $V_{\emptyset}$ handles constraints in which $\bar{\mathbf{b}}_2 = \bar{\mathbf{b}}_1$.

For every $d \in \{\to, \leftarrow, \emptyset\}$, the function V_d outputs 1 if it detects a violation of the d type of constraint, and otherwise outputs 0.

[14] In particular, this construction is closer to that of [BFL91] than that of [BFLS91].

[15] For this overview we shall ignore edge cases such as integers overflows. In particular, for the overview we do not even specify the behavior of the functions $\phi_{+1}, \phi_{\to}, \phi_{\leftarrow}$ in their edges cases. We remark that the edge cases are handled explicitly in the technical sections, see Section 7.

By the above discussion, X is a consistent description of a Turing machine computation if and only if the following holds: for every $\bar{\mathbf{a}}_1, \bar{\mathbf{a}}_2 \in \{0,1\}^t$ and $\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2 \in \{0,1\}^s$ such that $\phi_{+1}(\bar{\mathbf{a}}_1, \bar{\mathbf{a}}_2) = 1$ (i.e., $\bar{\mathbf{a}}_2$ corresponds to the time step following $\bar{\mathbf{a}}_1$) and $\phi_d(\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2) = 1$ for some $d \in \{\leftarrow, \rightarrow, \emptyset\}$ it holds that

$$V_d\Big(X\big(\bar{\mathbf{a}}_1, \bar{\mathbf{b}}_1\big), X\big(\bar{\mathbf{a}}_2, \bar{\mathbf{b}}_2\big)\Big) = 0.$$

Observe that if $\phi_d(\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2)$ is equal to 1 for some $d \in \{\leftarrow, \rightarrow, \emptyset\}$, then that d is unique. Thus, we can characterize the correctness of the computation in a more algebraic way by noting that the computation is correct if and only if for every $\bar{\mathbf{a}}_1, \bar{\mathbf{a}}_2 \in \{0,1\}^t$ and $\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2 \in \{0,1\}^s$:

$$\phi_{+1}(\bar{\mathbf{a}}_1, \bar{\mathbf{a}}_2) \cdot \sum_{d \in \{\leftarrow, \rightarrow, \emptyset\}} \phi_d(\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2) \cdot V_d\Big(X\big(\bar{\mathbf{a}}_1, \bar{\mathbf{b}}_1\big), X\big(\bar{\mathbf{a}}_2, \bar{\mathbf{b}}_2\big)\Big) = 0. \tag{1}$$

Let $\hat{\phi}_{+1} : \mathbb{F}^t \times \mathbb{F}^t \to \mathbb{F}$ and $\hat{\phi}_{\rightarrow}, \hat{\phi}_{\leftarrow}, \hat{\phi}_{\emptyset} : \mathbb{F}^s \times \mathbb{F}^s \to \mathbb{F}$ be the multilinear extensions of $\hat{\phi}_{+1}$, $\hat{\phi}_{\rightarrow}$, $\hat{\phi}_{\leftarrow}$ and $\hat{\phi}_{\emptyset}$, respectively. Let $\hat{V}_{\rightarrow}, \hat{V}_{\leftarrow}, \hat{V}_{\emptyset} : \mathbb{F}^2 \to \mathbb{F}$ be the low degree extensions of $V_{\rightarrow}$, $V_{\leftarrow}$ and $V_{\emptyset}$, respectively.

Motivated by Eq. (1), we define a polynomial $P_0 : \mathbb{F}^t \times \mathbb{F}^s \times \mathbb{F}^t \times \mathbb{F}^s \to \mathbb{F}$ as follows:

$$P_0(\bar{\mathbb{a}}_1, \bar{\mathbb{b}}_1, \bar{\mathbb{a}}_2, \bar{\mathbb{b}}_2) = \hat{\phi}_{+1}(\bar{\mathbb{a}}_1, \bar{\mathbb{a}}_2) \cdot \sum_{d \in \{\leftarrow, \rightarrow, \emptyset\}} \hat{\phi}_d(\bar{\mathbb{b}}_1, \bar{\mathbb{b}}_2) \cdot \hat{V}_d\Big(\hat{X}\big(\bar{\mathbb{a}}_1, \bar{\mathbb{b}}_1\big), \hat{X}\big(\bar{\mathbb{a}}_2, \bar{\mathbb{b}}_2\big)\Big). \tag{2}$$

The polynomial P_0 is included as part of the PCP proof string, in addition to the polynomial $\hat{X}$. The PCP verifier needs to check that P_0 is identically 0 for all binary inputs.[16] This check will be done by augmenting the PCP with one last component, often referred to as the "sumcheck polynomials".[17]

Let $\ell = 2(t+s)$. For every $i \in [\ell]$, define the polynomial $P_i : \mathbb{F}^\ell \to \mathbb{F}$ as follows:

$$P_i(\mathbb{z}_1, \ldots, \mathbb{z}_\ell) = \sum_{\mathbf{z}_i \in \{0,1\}} P_{i-1}\big(\mathbb{z}_1, \ldots, \mathbb{z}_{i-1}, \mathbf{z}_i, \mathbb{z}_{i+1}, \ldots, \mathbb{z}_\ell\big) \cdot \mathbb{z}_i^{\mathbf{z}_i}, \tag{3}$$

where we are viewing $\mathbf{z}_i \in \{0,1\}$ both as a field element and as an integer, in the natural way. Note that we are simultaneously using $\mathbf{z}_i$ and $\mathbb{z}_i$ in the same equation. This choice, which may at first appear confusing, is an intentional notational convention that we adopt. Later on, it will make our notation *much* clearer.

Remark 2.3. *The order in which we handle the variables in Eq. (3) can be chosen arbitrarily (i.e., unrelated to the different role that these variables play in P_0). Moreover, we could equally well define P_i to sum over a constant number of variables of P_{i-1}, rather than just one. Later on, in Section 2.4 (when we describe an efficient implementation of the* PCP *prover), we shall choose a different ordering and grouping that is more amenable to our efficient implementation.*

The reason that the polynomials $P_1, \ldots, P_\ell$ are useful is because

$$P_i|_{\mathbb{F}^i \times \{0,1\}^{\ell-i}} \equiv 0 \quad \Leftrightarrow \quad P_{i-1}|_{\mathbb{F}^{i-1} \times \{0,1\}^{\ell-i+1}} \equiv 0. \tag{4}$$

[16] We emphasize since P_0 has individual degree greater than 1, it is most likely the case that it is *not* the identically 0 polynomial.

[17] Indeed, these polynomials are directly related to the classical sumcheck protocol of Lund et al. [LFKN92].

The $\Leftarrow$ direction is immediate from Eq. (3). The $\Rightarrow$ direction follows by observing that the right hand side of Eq. (3), viewed as a (degree 1) polynomial in $\mathbb{z}_i$ is the identically 0 function for any $\mathbb{z}_{i+1}, \dots, \mathbb{z}_\ell \in \{0,1\}$. A polynomial is identically 0 if and only if all its coefficients are 0.

Thus, the PCP proof string includes, in addition to P_0 and $\hat{X}$, also the polynomials $P_1, \dots, P_\ell$. Observe that if Eq. (4) holds for all $i \in [\ell]$, then checking that $P_0|_{\{0,1\}^\ell} \equiv 0$ is reduced to checking that $P_\ell|_{\mathbb{F}^\ell} \equiv 0$, which turns out to be much easier to do.

We next describe a verifier for this PCP and provide intuition on why this verifier has *classical* soundness. Later, in Section 2.2, we give intuition on a key step that shows that (a variant of) this verifier also achieves no-signaling soundness.

The PCP Verifier and Classical Soundness. Given oracle access to the functions $\hat{X} : \mathbb{F}^{t+s} \to \mathbb{F}$ and $P_0, \dots, P_\ell : \mathbb{F}^\ell \to \mathbb{F}$ (which are all included as part of the PCP proof string), the verifier first runs a low degree test [RS96] for each one of these polynomials. This test basically ensures us that these functions are *close* to low degree polynomials. Let us assume for simplicity that the functions themselves are low degree polynomials (rather than being close to having low degree).[18]

Now, the verifier checks the consistency of $\hat{X}$ with P_0. This check can be done by observing that Eq. (2) describes a low degree relation between these polynomials and so it suffices to check it at a random point to be ensured that (with very high confidence) it is correct on *all* points.

The above test requires the verifier to compute the functions $\{\hat{\phi}_d\}_{d \in \{+1, \leftarrow, \rightarrow, \emptyset\}}$ and $\{\hat{V}_d\}_{d \in \{\leftarrow, \rightarrow, \emptyset\}}$. To see how this can be done efficiently, observe that the base functions ϕ_{+1}, $\phi_{\leftarrow}$, $\phi_{\rightarrow}$ and $\phi_{\emptyset}$ are incredibly simple and, in particular, are each computable by a constant-width oblivious read-once branching program. In Section 2.6 we show that the multi-linear extension of any function computable by such a branching program can be efficiently evaluated (i.e., in time that is roughly linear in the branching program size). Thus, the verifier can easily compute these functions by itself. As for the functions $\{\hat{V}_d\}_{d \in \{\leftarrow, \rightarrow, \emptyset\}}$, since these are low degree extensions of functions defined on a *constant* sized domain, they can be evaluated in a constant number of field operations by the verifier.

This leaves us only with the check that P_i and P_{i-1} are consistent, for all $i \in [\ell]$. This can again be done by observing that Eq. (3) describes a low degree relation between the two polynomials and so, yet again, if the relation holds at a random point then it holds everywhere.

So far we have established that $\hat{X}$ is the low degree extension of a *consistent* computation transcript. To complete the argument, the verifier must further check that the input embedded within $\hat{X}$ is consistent with its own main input x. This can be done by observing that the function $\hat{X}(0, \cdot) : \mathbb{F}^s \to \mathbb{F}$ is (supposed to be) the multilinear extension of the entire first layer of the computation transcript. Since that layer simply consists of x concatenated with 0's, the verifier can directly check the correctness of this polynomial by querying it at a random point.

2.2 No-signaling Soundness and Avoiding the Augmented Circuit

In order to construct an efficient argument-system, we follow the approach originally suggested by Biehl et al. [BMW98] and shown to be secure in [KRR13].

As mentioned above, this approach composes a private-information retrieval scheme (PIR), or more generally a fully homomorphic encryption scheme (FHE), together with a multi-prover interactive proof-system (MIP) to obtain an efficient non-interactive argument-system. Kalai et

[18]This assumption is justified by the fact that polynomials are also self-correctable [GLR+91].

al. [KRR13] establish the soundness of this approach if the the MIP satisfies a strong notion of soundness called *no-signaling soundness.*[19]

No-Signaling Soundness. We first recall the definition of an MIP. An MIP consists of a verifier V and ℓ non-communicating provers $P_1, \dots, P_\ell$. The verifier generates queries $(q_1, \dots, q_\ell)$, sends q_i to P_i and gets a response a_i. Given these answers the verifier decides whether to accept or reject. Completeness means that if $x \in \mathcal{L}$, there is a strategy for $P_1, \dots, P_\ell$ to convince V to accept (say, with probability 1).

Classical soundness means that for $x \notin \mathcal{L}$, no cheating provers $P_1^*, \dots, P_\ell^*$ *that are not allowed to communicate* can convince V to accept $x \notin \mathcal{L}$ other than with negligible probability. *No-signaling soundness* strengthens the classical soundness requirement by allowing the provers' answers to depend on queries sent to the other provers, but in a very restricted way. Specifically, the answer distribution of any subset of the provers should not depend *as a random variable* on the queries to the complementary set of provers.

More precisely, a prover strategy is a family of distributions $\{A_{\bar{q}}\}_{\bar{q}}$ indexed by all possible sets of queries (to all provers). A prover strategy is *no-signaling* if for every $S \subseteq [\ell]$ and every two query sets $\bar{q}, \bar{q}'$ that agree on S (i.e., $\bar{q}_S = \bar{q}'_S$) it holds that the marginal distributions $\bar{a}_S$ and $\bar{a}'_S$ are identical, where $\bar{a}$ (resp., $\bar{a}'$) refers to the ℓ answers of the provers given the query set $\bar{q}$ (resp., $\bar{q}'$). (See Section 3.3 for the formal definition.)

An MIP verifier has no-signaling soundness if it accepts false statements with only negligible probability, even when interacting with a no-signaling strategy. To show that our PCP has no-signaling soundness we re-visit the no-signaling soundness analysis of [KRR13, KRR14].

Bird's Eye View of the Augmented Circuit of [KRR14]. Kalai et al. [KRR13] showed that a variant of the BFLS PCP has no-signaling soundness. A shortcoming of their original result was that it was applicable only to *bounded space* computations. In a subsequent work, Kalai et al. [KRR14] removed this restriction by introducing a new construct they called the "augmented circuit".

The *augmented circuit* is a transformation that gets as input a layered circuit C and outputs a redundant version C' of C. More specifically, for the i^{th} layer G_i of gates in C (for $i \in [T]$), C' computes the low-degree extension $\hat{G}_i$ of those gates' values, and also performs the (seemingly redundant) check that when $\hat{G}_i$ is restricted to any line, the result is a low-degree univariate polynomial. The BFLS PCP is now applied to the augmented circuit C'.

Consider an input $x \notin \mathcal{L}$ and a no-signaling strategy $\mathcal{A} = \{A_q\}_{\bar{q}}$ that makes our verifier accept. For simplicity, let us assume that $\mathcal{A}$ makes the verifier accept with probability that is very close to 1 (i.e., $1 - \mathsf{negl}(T)$).[20] Loosely speaking, Kalai et al., and in a more explicit manner Paneth and Rothblum [PR17], show that there is a mechanism by which we can query $\mathcal{A}$ on a subset of polylog(T) points in the sub-domain of $\hat{X}$ which refers to the underlying computation (i.e., $\{0,1\}^{t+s}$) such that the provided answers will be *locally consistent* (with probability close to 1). We refer to this procedure as "reading" the subset of points and remark that it will only be done as a mental experiment in the analysis.

[19]The results of Dwork et al. [DLN+01] and Dodis et al. [DHRW16] show that this approach is in general *insecure* when using an MIP that is not sound against no-signaling strategies.

[20]To show soundness we need to rule out no-signaling strategies that succeed with much smaller probability. This is done in a similar manner to [KRR13, KRR14, BHK17] and so we do not elaborate on it here. See Section 10 for details.

Fix a layer $i \in [T]$ and suppose that if we read the value of a *random* point $\bar{\mathbb{v}}$ in $\hat{G}_i$ then with high probability we get the correct value of that point (i.e., the unique value that is determined by the input x). Kalai et al. [KRR14] use the augmented circuit to argue that in such a case it should also be true that if any fixed (i.e., worst-case) point $\bar{\mathbb{v}}$ is read, then whp we also get the correct value.

To see why this holds, consider (as a thought experiment) taking a random line $L_{\bar{\mathbb{u}}}$ passing through the fixed point u within $\hat{G}_i$. Consider reading the restriction of $\hat{G}_i$ to the line $L_{\bar{\mathbb{u}}}$. The values retrieved can be interpreted as a univariate function $f : \mathbb{F} \to \mathbb{F}$. The additional checks added in the augmented circuit, together with the local consistency of the reading mechanism ensure that the function f will be low degree. Suppose that when reading the value of $\bar{\mathbb{u}}$ we obtain an incorrect answer. Then, since distinct low degree polynomials disagree almost everywhere, with high probability over the choice of $\bar{\mathbb{v}}$, also the value corresponding to $\bar{\mathbb{v}}$ is incorrect. By the no-signaling condition, the same is true even if $\bar{\mathbb{v}}$ is read by itself (rather than via the line $L_{\bar{\mathbb{u}}}$). Thus, we reach a contradiction which means that reading the fixed point $\bar{\mathbb{u}}$ will return the correct value.

This worst-case to average-case reduction is the core component that allows [KRR14] to bypass the limitation of [KRR13] to bounded space computations.

Delegation for Unbounded Space Without the Augmented Circuit. The augmented circuit is a major barrier to obtaining efficient provers. The first reason is that it introduces an additional layer of abstraction and complicates the structure of the PCP proof string. More fundamentally though, the augmented circuit drives us further away from the base computational model that we are working with (i.e., the Turing machine, and later on the RAM machine).

With that in mind, our first observation is that, in a sense, low degree extensions of the individual layers of C are already present in the original BFLS PCP (i.e., without the augmented circuit). More specifically, for every $i \in \{0,1\}^t$, the polynomial $\hat{X}(i,\cdot) : \mathbb{F}^s \to \mathbb{F}$ is essentially the low degree extension of layer i. We would like to make similar arguments to [KRR14] with regard to these low degree extensions that are anyhow present in the PCP.

The major complication that arises as compared to the augmented circuit approach, is that we have no reason to believe that if we were to read the values of $\hat{X}(i,\cdot)$ restricted to the line $L_{\bar{\mathbb{u}}} : \mathbb{F} \to \mathbb{F}^s$ then the answers would be low degree. While the PCP verifier does explicitly test that the restriction of $\hat{X}$ to *random* lines has low degree, in the no-signaling setting we do not know how to argue that the restriction to a *non-random* line such as $L_{\bar{\mathbb{u}}}$ is low degree (in other words, we do not have a low degree test for no-signaling PCPs). It is worth pointing out that the reason that $L_{\bar{\mathbb{u}}}$ is not a random line is due to two facts (1) that it passes through the fixed, worst-case, point $\bar{\mathbb{u}}$ and (2) that it lies entirely the i^{th} layer of the circuit. The latter point is actually much more concerning for us since the analysis of [KRR13] does show that that the restriction of $\hat{X}$ to a random line passing through a *fixed* point is in fact low degree.

We resolve this issue by looking at the restriction of $\hat{X}$ to a carefully constructed *plane*, rather than a line.[21] Recall that a plane is an individual degree 1 bivariate polynomial mapping (whereas a line is a univariate degree 1 mapping) and that the restriction of a plane to each of its two coordinates is a line. Thus, rather than taking a random line passing through $\mathbb{u} \in \mathbb{F}^{t+s}$, we take a take a random plane $M : \mathbb{F}^2 \to \mathbb{F}^{t+s}$ such that $M(0,0) = \bar{\mathbb{u}}$ and $M(\cdot,0)$ is contained entirely within layer

[21]In the actual construction, due to complications that arise from the implementation of the reading mechanism (and the fact that the prover only succeed with small probability), we need to resort to multiple and slightly more complicated manifolds (rather than a plane).

i (i.e., $M(\cdot, 0) \subseteq \{i\} \times \mathbb{F}^s$).[22] Observe that for each $\beta \neq 0$, the line $M(\cdot, \beta)$ is a totally random line and so the restriction of $\hat{X}$ to that line should be low degree (since the verifier tests that random lines are low degree). For every $\alpha \neq 0$, the line $M(\alpha, \cdot)$ is a random line conditioned on passing through layer i at 0. In other words, $M(\alpha, \cdot)$ is entirely random except for a single point which is not entirely random. As mentioned above, the analysis in [KRR13] shows that also the restriction of $\hat{X}$ to such "almost random" lines will also be low degree (or more accurately, it is low degree if we ignore its value at 0). Very loosely speaking, this lets us argue that if we were to read the restriction of $\hat{X}$ to the plane M, the result would be a low degree (bivariate) polynomial F. Thus, the line $M(\cdot, 0)$ is a random line that lies entirely within layer i and so we can treat it similarly to the line $L_{\bar{u}}$ in the original analysis of [KRR14].

This concludes the overview of the no-signaling soundness analysis of our PCP. For full technical details, see Section 10.

2.3 PCPs for RAM computations via Tree Machines

Having established the no-signaling soundness of our PCP, we proceed to the main step - giving an efficient implementation of the PCP prover. The first difficulty that we encounter when trying to obtain an efficient implementation of the BFLS PCP (as described in Section 2.1), is the very fact that it refers to *Turing machine* computations rather than RAM computations.

It is well known that Turing machines can emulate RAMs with a quadratic to cubic slowdown, depending on the precise RAM variant. In our context, we cannot afford such a slowdown since we care about the precise polynomial running time of the prover, and we are aiming for a prover that has only a poly-logarithmic overhead in running time over the RAM complexity of the computation.

The trouble that we run into when trying to implement a PCP directly for RAM computations, is that RAMs do not offer the same type of local checkability that Turing machines offer. More specifically, any location in memory might potentially be affected in any time step, depending on the contents of the memory cell under the machine head. This difficulty was resolved in prior works, starting from [BCGT13], by utilizing a more efficient *non-deterministic* reduction from RAM programs to Turing machines (originally due to Gurevich and Shelach [GS89]). These works observe that since they are anyhow constructing PCPs for non-deterministic computations, they may as well utilize non-determinism when going from RAM computations to Turing machines. In contrast, since we are only constructing protocols for *deterministic* computations (and this is inherent when aiming for no-signaling soundness, see [DLN$^+$01, IKM09]) we cannot utilize non-deterministic reductions.

Our approach to resolving this difficulty is by considering a twist on the Turing machine model, called *Tree machines*. The advantage of Tree machines is that on one hand they are quite similar to Turing machines and in particular are "locally checkable" (i.e., at any given time step, only a constant number of neighboring memory location might be affected). This makes them amenable to algebraic PCP techniques. On the other hand, the key advantage of Tree machines over Turing machines is that they can emulate RAMs with only a *poly logarithmic* slowdown (using a simple deterministic reduction).

Tree machines can be thought of as Turing machines with a different tape "geometry". Rather than having one (or more) linear working tapes, we think of the tape of a tree machine as an infinite

[22] Such a plane can be constructed by sampling at random $\bar{v} \in \{0\} \times \mathbb{F}^s$ and $\bar{z} \in \mathbb{F}^{t+s}$ and setting $M(\alpha, \beta) = \bar{u} + \alpha \cdot \bar{v} + \beta \cdot \bar{z}$.

complete binary tree. At any given point in time, the machine head points to a node in the tree, and similarly to a Turing machine, the head position can move to either a parent or child of the current node. Similarly to Turing machines, the decision of where to move (and what to write at the current head position) is based on the symbol currently read by the machine head and an internal control state.

We find Tree machines to be a very useful and natural abstract model of computation. Unsurprisingly, Tree machines have been previously studied in the complexity theoretic literature at least since the 70's (cf. [PF79]), in particular, due to their efficient emulation of RAMs. Still, to the best of our knowledge Tree machines have never been used in the context of designing efficient proof-systems.

Efficient Emulation of RAMs by Tree machines. Recall that the main difficulty in emulating RAMs by Turing machines is that in a RAM the machine head might move to a place that is at some large distance d from the current position, at the cost of one instruction. For Turing machines this seems to necessitate d instructions to cover the large distance. On the other hand, it is easy to see that Tree machines can emulate such a step in $O(\log(d))$ instructions as follows.

The RAM tape of length S is emulated by considering only the first $\log(S)$ layers of the (infinite) tree-tape of the Tree machine. We view the $\log(S)^{th}$ layer of the tree as the leaves (i.e., ignoring the subsequent layers) and associate the RAM tape with this layer. Now, a transition of the RAM machine head from location v to u can be emulated in the Tree machine by moving the Tree machine head from v to the least common ancestor of u and v and then back down to u. Overall this emulation takes $O(\log(S))$ steps (rather than the S steps in a Turing machine).

We remark that for technical convenience our actual formalization of a Tree machine is slightly different than stated above. In particular, rather than storing just single symbol at each node of the tree, we store an entire infinite sequential working tape. This is done since in a RAM each memory cell can contain a super-constant sized symbol (e.g., from an alphabet that is as large as the RAM tape). See Section 4 for our actual formalization of Tree machines.

2.4 Efficient Implementation of the BFLS Prover

We now turn our attention to the prover efficiency. For sake of simplicity we address the easier case where the computation is described by a *Turing machine* M, rather than a *Tree machine*. In particular, we still refer to the construction and notation presented in Section 2.1 (which referred to a Turing machine computation). We point out complications that arise from using Tree machines only as necessary.

We describe an efficient evaluation procedure for each of the polynomials $\hat{X}, P_0, \ldots, P_\ell$, defined over a field $\mathbb{F}$ in which all operations take $\mathrm{polylog}(T)$ time and representing an element of $\mathbb{F}$ takes $\mathrm{polylog}(T)$ space. We start with $\hat{X}$, which serves as a warmup for the far more complicated evaluation of $P_0, \ldots, P_\ell$. (Recall that $\hat{X}$ is the multilinear extension of the function $X : \{0,1\}^t \times \{0,1\}^s \to \Gamma \times (Q \cup \bot)$, which corresponds to the computation transcript.)

Evaluating $\hat{X}$: Attempt 1. The simplest approach for computing $\hat{X}(\mathbb{z})$ is by first generating the entire truth table of X and then observing that $\hat{X}(\mathbb{z})$ is a linear combination of the truth table of X (and the coefficients of this linear combination can be efficiently generated from $\mathbb{z}$). The problem with this approach is that it takes both time and space $\Omega(T \cdot S)$ (whereas we are aiming for time roughly T and space roughly S).

Evaluating $\hat{X}$: Attempt 2. The key idea for improving the space usage is to generate the values of X "on-the-fly" as we are running the computation, as follows.

Let $\mathbb{y} \in \mathbb{F}^t$ and $\mathbb{u} \in \mathbb{F}^s$ such that $\mathbb{z} = (\mathbb{y}, \mathbb{u})$. Using the fact that the multilinear extension is a *tensor* code we can express $\hat{X}(\mathbb{z})$ as

$$\hat{X}(\mathbb{z}) = \sum_{i \in \{0,1\}^t} \beta_{i \to \mathbb{y}} \cdot \hat{X}(i, \mathbb{u})$$

where $\beta_{i \to \mathbb{y}}$ are easily computable coefficients.[23]

At any given time step i of the computation, given access to the entire work tape and control state of M (i.e., head position and internal state) we can compute $\hat{X}(i, \mathbb{u})$ in time roughly S and space $\mathrm{polylog}(T)$ since this value is a linear combination of evaluations of the restricted function $X(i, \cdot)$.

Overall this approach can be implemented in time roughly $T \cdot S$. As for space, beyond the space actually using by the Turing machine, we only need additional $\mathrm{polylog}(T)$ space for the various counters. Note that this still falls short of our goal of having the prover run in time roughly T.

Efficient Evaluation of $\hat{X}$. Our final observation regarding the efficient evaluation of $\hat{X}$ is that the memory contents of the Turing machine do not change so much in each time step. More specifically, they can are only changed in a constant number of location.[24] Thus, we can minimize the amount of work involved in computing the contribution of a given layer to $\hat{X}(\bar{\mathbb{z}})$ as follows.

Suppose that we have already computed $\hat{X}(i, \bar{\mathbb{u}})$ and we want to compute $\hat{X}(i+1, \bar{\mathbb{u}})$. Observe that:

$$\hat{X}(i, \bar{\mathbb{u}}) = \sum_{j \in \{0,1\}^s} \beta_{j \to \bar{\mathbb{u}}} \cdot X(i, j).$$

Let $W_i \subseteq \{0,1\}^s$ denote the positions in the work tape that are affected in transitioning from the i^{th} time step to the $(i+1)^{th}$ one. Then,

$$\hat{X}(i+1, \bar{\mathbb{u}}) - \hat{X}(i, \bar{\mathbb{u}}) = \sum_{j \in W_i} \beta_{j \to \bar{\mathbb{u}}} \cdot \big(X(i+1, j) - X(i, j) \big). \tag{5}$$

Since we know the set W_i (and its size is $O(1)$) and we also know what changes are made when transitioning to the $(i+1)^{th}$ time step (these all follow from the current head location, internal machine state and transition function of the Turing machine), we can compute $\hat{X}(i+1, \bar{\mathbb{u}})$ from $\hat{X}(i, \bar{\mathbb{u}})$ in just $\mathrm{polylog}(T)$ time and space. Thus, overall, we can evaluate $\hat{X}(\bar{\mathbb{z}})$ in time $\tilde{O}(T)$ and space $S + \mathrm{polylog}(T)$.

Efficient Evaluation of P_0. Given our efficient evaluation of $\hat{X}$ it is actually very easy to evaluate P_0 (indeed, the complications will only arise for evaluating P_i with $i \geq 1$). Recall from Eq. (2) that P_0 is defined as:

$$P_0(\bar{\mathbb{y}}_1, \bar{\mathbb{u}}_1, \bar{\mathbb{y}}_2, \bar{\mathbb{u}}_2) = \hat{\phi}_{+1}(\bar{\mathbb{y}}_1, \bar{\mathbb{y}}_2) \cdot \sum_{d \in \{\leftarrow, \rightarrow, \emptyset\}} \hat{\phi}_d(\bar{\mathbb{u}}_1, \bar{\mathbb{u}}_2) \cdot \hat{V}_d\Big(\hat{X}\big(\bar{\mathbb{y}}_1, \bar{\mathbb{u}}_1\big), \hat{X}\big(\bar{\mathbb{y}}_2, \bar{\mathbb{u}}_2\big)\Big).$$

[23]The notation is meant to remind of the fact that $\beta_{i \to \mathbb{y}}$ measures the contribution of $X(i)$ to $\hat{X}(\bar{\mathbb{y}})$, which is well-defined because the multi-linear extension is a linear code.

[24]The same is also true for Tree machines.

As was briefly discussed in Section 2.1, the functions $\hat{\phi}_{+1}$, $\{\hat{\phi}_d\}_{d\in\{\leftarrow,\rightarrow,\emptyset\}}$ and $\{\hat{V}_d\}_{d\in\{\leftarrow,\rightarrow,\emptyset\}}$ are all computable in time $\mathrm{polylog}(T)$. In combination with our procedure for evaluating $\hat{X}$, Eq. (2) can be used directly to evaluate P_0 in time $\tilde{O}(T)$ and space $S + \mathrm{polylog}(T)$.

Efficient Evaluation of P_i for $i \le t$. We proceed to the evaluation of the polynomials P_i, but start with the case that $i \le t$. We will use the notation $x\|x'$ to denote the concatenation of x and x'.

Recall from Eq. (3) and Remark 2.3 that we may choose an arbitrary variable ordering and grouping in the definition of P_i. We choose them so that for any $i \le t$, any $\bar{\mathbb{y}}_1 = \bar{\mathbb{a}}_1\|\bar{\mathbb{b}}_1 \in \mathbb{F}^i \times \mathbb{F}^{t-i}$, any $\bar{\mathbb{y}}_2 = \bar{\mathbb{a}}_2\|\bar{\mathbb{b}}_2 \in \mathbb{F}^i \times \mathbb{F}^{t-i}$, and any $\bar{\mathbb{u}}_1, \bar{\mathbb{u}}_2 \in \mathbb{F}^s$, we have

$$P_i\big(\bar{\mathbb{a}}_1\|\bar{\mathbb{b}}_1, \bar{\mathbb{u}}_1, \bar{\mathbb{a}}_2\|\bar{\mathbb{b}}_2, \bar{\mathbb{u}}_2\big) = \sum_{\bar{\mathbf{a}}_1,\bar{\mathbf{a}}_2\in\{0,1\}^i} P_0\big(\bar{\mathbf{a}}_1\|\bar{\mathbb{b}}_1, \bar{\mathbb{u}}_1, \bar{\mathbf{a}}_2\|\bar{\mathbb{b}}_2, \bar{\mathbb{u}}_2\big) \cdot \bar{\mathbb{a}}_1^{\bar{\mathbf{a}}_1} \cdot \bar{\mathbb{a}}_2^{\bar{\mathbf{a}}_2},$$

where for $\bar{\mathbb{a}} = (\mathbb{a}_1, \dots, \mathbb{a}_i)$ and $\bar{\mathbf{a}} = (\mathbf{a}_1, \dots, \mathbf{a}_i)$, the vector exponentiation notation is defined as $\bar{\mathbb{a}}^{\bar{\mathbf{a}}} \overset{\mathsf{def}}{=} \prod_j \mathbb{a}_j^{\mathbf{a}_j}$.

This variable ordering and grouping will make it easy to evaluate P_i for $i \le t$ in a similar fashion to the method we described for evaluating $\hat{X}$.

The factors $\bar{\mathbb{a}}_1^{\bar{\mathbf{a}}_1}$ and $\bar{\mathbb{a}}_2^{\bar{\mathbf{a}}_2}$ are each computable in time $\mathrm{polylog}(T)$. We will show how to *enumerate* (i.e., sequentially output on a one-way write-only tape) the *non-zero* terms in the sequence:

$$\Big\{P_0\big(\bar{\mathbf{a}}_1\|\bar{\mathbb{b}}_1, \bar{\mathbb{u}}_1, \bar{\mathbf{a}}_2\|\bar{\mathbb{b}}_2, \bar{\mathbb{u}}_2\big)\Big\}_{\bar{\mathbf{a}}_1,\bar{\mathbf{a}}_2\in\{0,1\}^i.} \tag{6}$$

in time $\tilde{O}(T)$ and space $S + \mathrm{polylog}(T)$. Using this enumeration, we can evaluate P_i in time $\tilde{O}(T)$ and space $S + \mathrm{polylog}(T)$.

Before describing the enumeration process, let us see that there are at most $O(T)$ non-zero terms in the sequence defined in Eq. (6). This holds because by Eq. (2) $P_0\big(\bar{\mathbf{a}}_1\|\bar{\mathbb{b}}_1, \bar{\mathbb{u}}_1, \bar{\mathbf{a}}_2\|\bar{\mathbb{b}}_2, \bar{\mathbb{u}}_2\big)$ is a multiple of $\hat{\phi}_{+1}(\bar{\mathbf{a}}_1\|\bar{\mathbb{b}}_1, \bar{\mathbf{a}}_2, \bar{\mathbb{b}}_2)$. But because $\hat{\phi}_{+1}$ is the *minimal* low-degree extension of ϕ_{+1}, we will only have $\hat{\phi}_{+1}(\bar{\mathbf{a}}_1\|\bar{\mathbb{b}}_1, \bar{\mathbf{a}}_2, \bar{\mathbb{b}}_2) \ne 0$ if there *exist* $\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2 \in \{0,1\}^{t-i}$ such that $\phi_{+1}(\bar{\mathbf{a}}_1\|\bar{\mathbf{b}}_1, \bar{\mathbf{a}}_2\|\bar{\mathbf{b}}_2) = 1$. This happens only if $\bar{\mathbf{a}}_2 = \bar{\mathbf{a}}_1$ or $\bar{\mathbf{a}}_2 = \bar{\mathbf{a}}_1 + 1$, i.e. for at most $2 \cdot 2^i = O(T)$ values of $(\bar{\mathbf{a}}_1, \bar{\mathbf{a}}_2)$.

For simplicity, we only show how to enumerate the terms of Eq. (6) where $\bar{\mathbf{a}}_1 = \bar{\mathbf{a}}_2$ (the terms when $\bar{\mathbf{a}}_2 = \bar{\mathbf{a}}_1 + 1$ can be enumerated similarly and separately). As in the evaluation of P_0 discussed previously, and using the definition of P_0, enumerating these terms reduces to enumerating

$$\Big\{\big(\hat{X}(\bar{\mathbf{a}}\|\bar{\mathbb{b}}_1, \bar{\mathbb{u}}_1), \hat{X}(\bar{\mathbf{a}}\|\bar{\mathbb{b}}_2, \bar{\mathbb{u}}_2)\big)\Big\}_{\bar{\mathbf{a}}\in\{0,1\}^i} \tag{7}$$

in time $\tilde{O}(T)$ and space $S + \mathrm{polylog}(T)$.

This can be done using similar techniques to the ones we have described for evaluating $\hat{X}$, as we explain next. Observe that

$$\hat{X}(\bar{\mathbf{a}}\|\bar{\mathbb{b}}_1, \bar{\mathbb{u}}_1) = \sum_{\bar{\mathbf{b}}\in\{0,1\}^{t-i}} \beta_{\bar{\mathbf{b}}\to\bar{\mathbb{b}}_1} \cdot \hat{X}(\bar{\mathbf{a}}\|\bar{\mathbf{b}}, \bar{\mathbb{u}}_1), \tag{8}$$

and

$$\hat{X}(\bar{\mathbf{a}}\|\bar{\mathbb{b}}_2, \bar{\mathbb{u}}_2) = \sum_{\bar{\mathbf{b}}\in\{0,1\}^{t-i}} \beta_{\bar{\mathbf{b}}\to\bar{\mathbb{b}}_2} \cdot \hat{X}(\bar{\mathbf{a}}\|\bar{\mathbf{b}}, \bar{\mathbb{u}}_2). \tag{9}$$

Since the coefficients $\beta_{\bar{\mathbf{b}}\to\bar{\mathbb{b}}_1}$ and $\beta_{\bar{\mathbf{b}}\to\bar{\mathbb{b}}_2}$ are computable in time polylog(T), enumerating Eq. (7) reduces to enumerating:

$$\left\{\left(\hat{X}(\bar{\mathbf{y}},\bar{\mathbb{u}}_1),\hat{X}(\bar{\mathbf{y}},\bar{\mathbb{u}}_2)\right)\right\}_{\bar{\mathbf{y}}\in\{0,1\}^i.} \tag{10}$$

in time $\tilde{O}(T)$ and space $S+\text{polylog}(T)$, as long as the enumeration is in a "nice" ordering.[25] One nice ordering is in order of lexicographically increasing (or, if $\bar{\mathbf{y}}$ is thought of as an integer in $\{0,\ldots,2^t-1\}$, *numerically* increasing) $\bar{\mathbf{y}}$.

But this is possible by the same techniques we described for evaluating $\hat{X}$. Namely, we first compute $\hat{X}(\bar{\mathbf{0}},\bar{\mathbb{u}}_1)$ and $\hat{X}(\bar{\mathbf{0}},\bar{\mathbb{u}}_2)$ (which is easily done in time $\tilde{O}(T)$ and space $S+\text{polylog}(T)$, since these refer to points in the multilinear extension of the *initial* configuration of the machine). We then incrementally compute all remaining values $\hat{X}(\bar{\mathbf{y}},\bar{\mathbb{u}}_1)$ and $\hat{X}(\bar{\mathbf{y}},\bar{\mathbb{u}}_2)$ "on the fly" via the formula given in Eq. (5) in our previous method for evaluating $\hat{X}$. This also takes time polylog(T) time per $\bar{\mathbf{y}}$, and polylog(T) additional space, for a total running time of $\tilde{O}(T)$ and total space usage of $S+\text{polylog}(T)$.

Efficient Evaluation of P_i for $i>t$. We proceed to describe the efficient evaluation of P_i for $t<i\le t+s$, which will be the most complicated procedure. We further modify the variable ordering and grouping in the definition of P_i, so that for any i with $t<i\le t+s$, any $\bar{\mathbb{y}}_1,\bar{\mathbb{y}}_2\in\mathbb{F}^t$, any $\bar{\mathbb{u}}_1=\bar{\mathbb{a}}_1\|\bar{\mathbb{b}}_1\in\mathbb{F}^{i-t}\times\mathbb{F}^{s-(i-t)}$, and any $\bar{\mathbb{u}}_2=\bar{\mathbb{a}}_2\|\bar{\mathbb{b}}_2\in\mathbb{F}^{i-t}\times\mathbb{F}^{s-(i-t)}$, we have

$$P_i\left(\bar{\mathbb{y}}_1,\bar{\mathbb{a}}_1\|\bar{\mathbb{b}}_1,\bar{\mathbb{y}}_2,\bar{\mathbb{a}}_2\|\bar{\mathbb{b}}_2,\bar{\mathbb{u}}_2\right)=\sum_{\substack{\bar{\mathbf{y}}_1,\bar{\mathbf{y}}_2\in\{0,1\}^t\\ \bar{\mathbf{a}}_1,\bar{\mathbf{a}}_2\in\{0,1\}^{i-t}}} P_0\left(\bar{\mathbf{y}}_1,\bar{\mathbf{a}}_1\|\bar{\mathbb{b}}_1,\bar{\mathbf{y}}_2,\bar{\mathbf{a}}_2\|\bar{\mathbb{b}}_2\right)\cdot\bar{\mathbb{y}}_1^{\bar{\mathbf{y}}_1}\cdot\bar{\mathbb{y}}_2^{\bar{\mathbf{y}}_2}\cdot\bar{\mathbb{a}}_1^{\bar{\mathbf{a}}_1}\cdot\bar{\mathbb{a}}_2^{\bar{\mathbf{a}}_2}. \tag{11}$$

Unlike in the case $i\le t$, we cannot now say that Eq. (11) has $O(T)$ (or even $\tilde{O}(T)$) non-zero terms. However, we will try to make simple observations to simplify our task of computing the expression in Eq. (11). In particular, by the definition of P_0, we observe that $P_0\left(\bar{\mathbf{y}}_1,\bar{\mathbf{a}}_1\|\bar{\mathbb{b}}_1,\bar{\mathbf{y}}_2,\bar{\mathbf{a}}_2\|\bar{\mathbb{b}}_2\right)\neq 0$ only if:

- As an integer, $\bar{\mathbf{y}}_2$ is equal to $\bar{\mathbf{y}}_1+1$.
- As an integer, $\bar{\mathbf{a}}_2$ is either equal to $\bar{\mathbf{a}}_1$, $\bar{\mathbf{a}}_1-1$, or $\bar{\mathbf{a}}_1+1$.

For simplicity, we will focus on the case that $\bar{\mathbf{a}}_2=\bar{\mathbf{a}}_1$ (the other cases are handled similarly and separately). Note that $\bar{\mathbb{y}}_1^{\bar{\mathbf{y}}_1}$ and $\bar{\mathbb{y}}_2^{\bar{\mathbf{y}}_2}$ are computable in time polylog(T). We will show that the T terms

$$\left\{\sum_{\bar{\mathbf{a}}\in\{0,1\}^{i-t}} P_0(\bar{\mathbf{y}},\bar{\mathbf{a}}\|\bar{\mathbb{b}}_1,\bar{\mathbf{y}}+1,\bar{\mathbf{a}}\|\bar{\mathbb{b}}_2)\cdot\bar{\mathbb{a}}_1^{\bar{\mathbf{a}}}\cdot\bar{\mathbb{a}}_2^{\bar{\mathbf{a}}}\right\}_{\bar{\mathbf{y}}\in\{0,1\}^t} \tag{12}$$

can be enumerated in time $\tilde{O}(T)$ and space $S+\text{polylog}(T)$. Assuming, as we have claimed, that the other cases can be similarly summed, this implies that P_i itself can be evaluated in time $\tilde{O}(T)$ and space $S+\text{polylog}(T)$.

Let $\sigma_{\bar{\mathbf{y}}}$ denote the $\bar{\mathbf{y}}^{th}$ term of Eq. (12). In the hopes of following our earlier approach, we first verify that $\sigma_{\bar{\mathbf{0}}}$ can be evaluated in time $\tilde{O}(T)$ and space $S+\text{polylog}(T)$. Indeed, this follows more

[25] Specifically, we want to ensure that while computing terms of Eq. (7) via Eqs. (8) and (9), that we never have too many "incomplete summations". Otherwise, to store these partial summations might require $\Omega(T)$ space

generally from the fact that the formula for $\sigma_{\bar{\mathbf{y}}}$ is a summation of 2^{i-t} terms, each of which can be evaluated in time $2^{s-(i-t)} \cdot \mathrm{polylog}(T)$ given access to $X(\bar{\mathbf{y}}, \cdot)$ and $X(\bar{\mathbf{y}}+1, \cdot)$. The latter is simulatable with constant overhead given $X(\bar{\mathbf{y}}, \cdot)$).

Now suppose we have computed $\sigma_{\bar{\mathbf{y}}}$, have oracle access to $X(\bar{\mathbf{y}}, \cdot)$, and want to compute $\sigma_{\bar{\mathbf{y}}+1}$. Due to the structure of Turing machines, $X(\bar{\mathbf{y}}+1, \cdot)$ differs from $X(\bar{\mathbf{y}}, \cdot)$ on at most a constant number of inputs. For simplicity, suppose that they differ only on a single input $\bar{\mathbf{u}}_1$, and that $X(\bar{\mathbf{y}}+2, \cdot)$ differs from $X(\bar{\mathbf{y}}+1, \cdot)$ on a single input $\bar{\mathbf{u}}_2$. The $\bar{\mathbf{a}}^{th}$ term in the summation defining $\sigma_{\bar{\mathbf{y}}}$ is

$$P_0\big(\bar{\mathbf{y}}, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_1, \bar{\mathbf{y}}+1, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_2\big) \cdot \bar{\mathbb{a}}_1^{\bar{\mathbf{a}}} \cdot \bar{\mathbb{a}}_2^{\bar{\mathbf{a}}}. \tag{13}$$

After fixing $\bar{\mathbf{a}}$, $\bar{\mathbb{a}}_1$, $\bar{\mathbb{a}}_2$, $\bar{\mathbb{b}}_1$, and $\bar{\mathbb{b}}_2$ – essentially everything except $\bar{\mathbf{y}}$ – Eq. (13) is a function only of $\hat{X}(\bar{\mathbf{y}}, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_1)$ and $\hat{X}(\bar{\mathbf{y}}+1, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_2)$. These are themselves functions of $X(\bar{\mathbf{y}}, \bar{\mathbf{a}}\|\cdot)$ and $X(\bar{\mathbf{y}}+1, \bar{\mathbf{a}}\|\cdot)$ respectively, and thus are unchanged unless $\bar{\mathbf{a}}$ is a prefix of $\bar{\mathbf{u}}_1$ or of $\bar{\mathbf{u}}_2$.

We thus want to compute $\sigma_{\bar{\mathbf{y}}+1}$ by computing the changes to the (constant) number of differing terms of $\sigma_{\bar{\mathbf{y}}}$. Suppose the $\bar{\mathbf{a}}^{th}$ term is one such term. Via our previous techniques, it is possible to compute the change to the $\bar{\mathbf{a}}^{th}$ term in time $\mathrm{polylog}(T)$ – *if* we already happen to know the values of $\hat{X}(\bar{\mathbf{y}}, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_1)$ and $\hat{X}(\bar{\mathbf{y}}+1, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_2)$ (and in this case we will also be able to compute their new values $\hat{X}(\bar{\mathbf{y}}+1, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_1)$ and $\hat{X}(\bar{\mathbf{y}}+2, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_2)$ in time $\mathrm{polylog}(T)$). If not, we can reduce to the former case by *computing* $\hat{X}(\bar{\mathbf{y}}, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_1)$ and $\hat{X}(\bar{\mathbf{y}}+1, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_2)$ in time $2^{s-(t-i)} \cdot \mathrm{polylog}(T)$ time.

Thus, we apparently have a time/space trade-off. At one extreme, we can maintain $\hat{X}(\bar{\mathbf{y}}, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_1)$ and $\hat{X}(\bar{\mathbf{y}}+1, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_2)$ for *every* $\bar{\mathbf{a}} \in \{0,1\}^{i-t}$ as we progress in the computation from $\bar{\mathbf{y}} = 0$ to $\bar{\mathbf{y}} = 2^t - 2$. This will use space $2^{i-t} \cdot \mathrm{polylog}(T)$, which can be as large as $S \cdot \mathrm{polylog}(T)$. At the other extreme, we can store none of the values, in which case our total time usage will be $\tilde{O}(T \cdot 2^{s-(i-t)})$, which can be $\Omega(S \cdot T)$.

The time/space trade-off is in many cases mitigated by noticing that the sequence of relevant $\bar{\mathbf{a}}$'s is tightly tied to the *memory access pattern* of our computation – the sequence of addresses whose values in memory are modified. In the case of Turing machines, this has an important implication. In any consecutive interval of $\approx 2^{s-(i-t)}$ timesteps, there will be at most 2 relevant values of $\bar{\mathbf{a}}$. Thus, as long as we always remember $\hat{X}(\bar{\mathbf{y}}, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_1)$ and $\hat{X}(\bar{\mathbf{y}}+1, \bar{\mathbf{a}}\|\bar{\mathbb{b}}_2)$ for the two most recent values of $\bar{\mathbf{a}}$ (thereby using $\mathrm{polylog}(T)$ space), then there will be at most $O(T/2^{s-(i-t)})$ occasions in which we need to *compute* these values. In total, this takes $\tilde{O}(T)$ time. More generally – e.g. for tree machine and RAM computations, the actual time/space trade-off is dictated by the *cache efficiency* of the computation's memory access pattern.

More generally – e.g. for tree machine and RAM computations, the actual time/space trade-off is dictated by the *cache efficiency* of the computation's memory access pattern. This concludes the overview of the efficient implementation of the PCP provers. For the full technical details, see Section 9.

This concludes the overview of the efficient implementation of the PCP prover. For the full technical details, see Section 9.

2.5 Efficient Arguments: FHE and Multiplicative Depth

Recall that, following [BMW98], the provers in our delegation scheme need to *homomorphically* generate answers to *encrypted* PCP queries. Recall that our PCP proof string consists of the polynomials $\hat{X}, P_0, \ldots, P_\ell$ (see Section 2.1 for details). For this overview we only describe how to efficiently homomorphically evaluate $\hat{X}$. The procedure for homomorphic evaluation of $P_0, \ldots, P_\ell$ relies on similar ideas but is more complicated.

A naive way of homomorphically evaluating $\hat{X}$ is as follows. Consider the circuit C that has x (the main input to the computation) hard-coded, and given as input $\mathbb{z} \in \mathbb{F}^{t+s}$, evaluates and outputs $\hat{X}(\mathbb{z})$ (using the procedure described in Section 2.4). Then, given an encrypted query $E(\mathbb{z})$, the prover can homomorphically generate $E(C(\mathbb{z})) = E(\hat{X}(\mathbb{z}))$. Note however that this requires homomorphic evaluation of circuits that are as complex as the entire computation, which we would rather avoid (both in terms of assumptions on the FHE and efficiency).

In particular, one of the key bottlenecks in existing FHE schemes (e.g., [BGV14, GSW13]) is the *multiplicative depth* of the circuit being homomorphically evaluated. This is because ciphertexts in such lattice-based schemes have some noise parameter, which doubles with each multiplication operation performed. If too many multiplications are done then the noise level explodes and we resort to Gentry's expensive bootstrapping technique [Gen09]. The circuit C that we are trying to evaluate C might have multiplicative depth T, which would be extremely costly in terms of homomorphic operations. We will show however that we can instead rely on homomorphic encryption that only supports circuits with multiplicative depth $\log_2 \log(T) + O(1)$.

Recall that $\hat{X} : \mathbb{F}^t \times \mathbb{F}^s \to F$ is the multilinear extension of the function $X : \{0,1\}^t \times \{0,1\}^s \to \Gamma \times (Q \cup \perp)$, which corresponds to the computation transcript. In Section 2.4 we outlined a procedure for efficiently evaluating $\hat{X}$ using the following formula:

$$\hat{X}(\mathbb{z}) = \sum_{i \in \{0,1\}^t} \beta_{i \to \mathbb{y}} \cdot \sum_{j \in \{0,1\}^s} \beta_{j \to \mathbb{u}} \cdot X(i,j),$$

where $\mathbb{y} \in \mathbb{F}^t$ and $\mathbb{u} \in \mathbb{F}^s$ are such that $\mathbb{z} = (\mathbb{y}, \mathbb{u})$. Our main observation is that to compute the same expression "under the hood" it suffices to only homomorphically compute the coefficients $\{\beta_{j \to \mathbb{u}}\}_{j \in \{0,1\}^s}$ and $\{\beta_{i \to \mathbb{y}}\}_{i \in \{0,1\}^t}$ and that all other operations only include addition or multiplication with a plaintext (i.e., unencrypted) value, such as $X(i,j)$.

Each value $\beta_{i \to \mathbb{y}}$ expressed as a function of $\mathbb{y}$, is a degree t multilinear polynomial with a simple arithmetic circuit of size $O(t)$ and multiplicative depth $\log(t)$. Likewise the $\beta_{j \to \mathbb{u}}$ have degree $O(s)$ as a function $\mathbb{u}$. As argued above, all other operations are additions or multiplication by a plaintext value. Therefore, the overall multiplicative depth of the circuit that we evaluate, as a function of $\mathbb{z}$, is $\log(t) = \log \log(T)$.

2.6 Evaluating Low Degree Extension of Read Once Branching Programs

As a technical tool toward the efficient implementation of both the prover and the verifier algorithms of the BFLS PCP, we show that the low degree of some very simple functions that arise in that construction can be efficiently evaluated. More specifically, we show that the low degree extension of any function that is computable by a small read-once branching program can be efficiently evaluated. As mentioned above, this result generalizes some ad-hoc techniques for evaluating the low degree extension that were used in [BHK17] and [GR17]. We believe that this technique will be useful also in future works.

To illustrate our approach, for simplicity, we describe an important special case that suffices for our construction. Namely, computing the the *multilinear* extension of any function that is computable by an *oblivious* read-once branching program P of bounded width. We remark that our general result applies to *arbitrary* read-once branching programs (of bounded size) and is applicable to the computing the (exact) low degree extensions of functions (and not just the multi-linear extension).

Recall that an oblivious width w branching program is described by a layered directed graph G (possibly with multi-edges) with $n+1$ layers, each having exactly w vertices and where edges only go from layer i to layer $i+1$. Each layer i (except for the last one) is associated with a variable $\mathsf{Var}(i) \in [n]$. The vertices in the last layer are called sinks and are labeled by a terminating symbol which is either 0 or 1. For simplicity, and without loss of generality, we assume that that the first sink (i.e., first vertex in the last layer) is labeled by 1 and all other sinks are labeled by 0. From each vertex (which is not in the last layer), there are exactly two outgoing edges: one marked with 0 and one marked by 1.

The evaluation of a branching program on an input $x \in \{0,1\}^n$ proceeds as follows. We start at the first vertex of the first layer. Then, at each step $i \in [n]$, we move from the current vertex (which is at layer i) by following the outgoing edge that is marked by $x_{\mathsf{Var}(i)}$. Once we get to the sink, we output its terminating symbol. In what follows we assume without loss of generality that P reads its input in order.[26] That is, $\mathsf{Var}(i) = i$ for every $i \in [n]$.

Let $f : \{0,1\}^n \to \{0,1\}$ be computable by an oblivious read-once branching program P of width w. One way to describe P is by a collection of binary matrices $\{M_i^{(b)} \in \mathbb{F}_2^{w \times w}\}_{i \in [n], b \in \{0,1\}}$, where $M_i^{(b)}$, describes the adjacency matrix of the (sub-)graph of the b-labeled edges connecting layer i with layer $i+1$. Therefore, the evaluation of f on an input $\bar{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ is equal to:

$$f(\bar{x}) = \Big(\prod_{i=1}^{n} M_i^{(x_i)} \Big)_{1,1},$$

where the notation $A_{1,1}$ refers to the $(1,1)^{th}$ entry of the matrix A.

In this case, the formula for $\hat{f} : \mathbb{F}^n \to \mathbb{F}$ is particularly simple. For any $\bar{\mathbb{x}} \in \mathbb{F}^n$, we simply have

$$\hat{f}(\bar{\mathbb{x}}) = \prod_{i=1}^{n} \Big(\mathbb{x}_i \cdot M_i^{(1)} + (1 - \mathbb{x}_i) \cdot M_i^{(0)} \Big).$$

To see that $\hat{f}$ is indeed the multi-linear extension of f note that $\hat{f}$ agrees with f on every input $\bar{x} \in \{0,1\}^n$. Further, $\hat{f}$ is multilinear and by the uniqueness of the multilinear extension, we conclude that it is indeed the multi-linear extension of f.

While this result is extremely simple, to the best of our knowledge it has not been pointed out before (in the context of constructing probabilistic proof-systems). For additional details, including the extension to *non-oblivious* branching programs, see Section 5.1.

3 Preliminaries

We begin with some standard notations. For any integer n, we write $[n]$ to denote the set $\{1, \ldots, n\}$. We will sometime treat binary string of length ℓ as integers in $\{0, \ldots, 2^\ell - 1\}$ in the natural way.

For two tuples $\bar{u} = (u_1, \ldots, u_n)$ and $\bar{v} = (v_1, \ldots, v_m)$, we write $\bar{u} \| \bar{v}$ to denote the tuple $(u_1, \ldots, u_n, v_1, \ldots, v_m)$. If $n = m$, we write $\bar{u} \star \bar{v}$ to denote the tuple $(u_1, v_1, \ldots, u_n, v_n)$.

[26] If P instead reads $x_{\pi(1)}, \ldots, x_{\pi(n)}$ for some permutation π of $[n]$, then define the function $f'(\bar{x}) = f(x_{\pi^{-1}(1)}, \ldots, x_{\pi^{-1}(n)})$. Then P computes f' by reading input bits in order. It is possible to evaluate $\hat{f}$ in terms of $\hat{f}'$ using the easily verified fact that $\hat{P}(\bar{\mathbb{x}}) = \hat{P}'(\mathbb{x}_{\pi(1)}, \ldots, \mathbb{x}_{\pi(n)})$.

We use $\textsc{Prefix}_i : \{0,1\}^{\geq i} \to \{0,1\}^i$ to denote the function that outputs the i-bit prefix of its input. All logarithms in this work are base 2.

Font Conventions. Throughout this work we will use the convention that blackboard bold lowercase (e.g., $\mathbb{z}$) is used for field elements whereas standard bold lowercase (e.g., $\mathbf{z}$) is used for bits. Likewise, we use $\bar{\mathbb{z}}$ to denote vectors of field elements and $\bar{\mathbf{z}}$ to denote bit strings.

3.1 Fields, Polynomials, and Low Degree Extensions

Let $\mathbb{F}$ be a finite field. For any function $f : \mathbb{F} \to \mathbb{F}$, we use $\mathsf{Deg}(f)$ to denote the (minimal) degree of a polynomial over $\mathbb{F}$ that computes f. Throughout this work, by a degree d polynomial, we actually mean a polynomial with degree *at most* d.

Given a set of at least $d+1$ pairs $\{(\mathbb{x}_i, \mathbb{y}_i)\} \subset \mathbb{F}^2$, which we will suggestively denote $\{\mathbb{x}_i \mapsto \mathbb{y}_i\}$, we write $\mathsf{Interp}_d(\{\mathbb{x}_i \mapsto \mathbb{y}_i\})$ to denote the unique degree-d polynomial f such that $f(\mathbb{x}_i) = \mathbb{y}_i$ if such an f exists; otherwise we define $\mathsf{Interp}_d(\{\mathbb{x}_i \mapsto \mathbb{y}_i\}) = \perp$.

Definition 3.1 (Multi-linear Extension)**.** *Let $f : \{0,1\}^\ell \to \mathbb{F}$ be a function. We define the* multi-linear extension *of f, denoted $\hat{f} : \mathbb{F}^\ell \to \mathbb{F}$, as the (unique) multi-linear polynomial such that $\hat{f}|_{\{0,1\}^\ell} \equiv f$.*

Fact 3.2. *For any function $f : \{0,1\}^\ell \to F$, its multi-linear extension $\hat{f} : \mathbb{F}^\ell \to F$ is given by the formula*

$$\hat{f}(\mathbb{z}_1, \dots, \mathbb{z}_\ell) = \sum_{\mathbf{z}_1,\dots,\mathbf{z}_\ell \in \{0,1\}} f(\mathbf{z}_1, \dots, \mathbf{z}_\ell) \cdot \prod_{i=1}^{\ell} \beta_{\mathbf{z}_i \to \mathbb{z}_i},$$

where $\beta_{\mathbf{z}_i \to \mathbb{z}_i} \overset{\mathsf{def}}{=} \mathbf{z}_i \cdot \mathbb{z}_i + (1 - \mathbf{z}_i) \cdot (1 - \mathbb{z}_i)$.

We will use the notation $\beta_{\mathbf{z}_i \to \mathbb{z}_i} \overset{\mathsf{def}}{=} \mathbf{z}_i \cdot \mathbb{z}_i + (1 - \mathbf{z}_i) \cdot (1 - \mathbb{z}_i)$ throughout this work.

We now extend the definition of multilinear extension to functions defined over larger alphabets.

Definition 3.3 (Low-Degree Extension)**.** *For any subset $H \subseteq \mathbb{F}$ and function $f : H^\ell \to \mathbb{F}$, there is a unique individual degree $|H| - 1$ polynomial $\hat{f} : \mathbb{F}^\ell \to \mathbb{F}$ that agrees with f on H^ℓ.*

Definition 3.1 follows as a special case of Definition 3.3 (with $H = \{0,1\}$).

Fact 3.4. *For any function $f : H^\ell \to \mathbb{F}$ with $H \subseteq \mathbb{F}$, its low-degree extension $\hat{f}$ is given by the formula*

$$\hat{f}(\mathbb{z}_1, \dots, \mathbb{z}_\ell) = \sum_{\mathbf{z}_1,\dots,\mathbf{z}_\ell \in H} f(\mathbf{z}_1, \dots, \mathbf{z}_\ell) \cdot \prod_{i=1}^{\ell} \hat{\chi}_{\mathbf{z}_i}(\mathbb{z}_i),$$

where for each $\mathbf{z} \in H$, the degree $|H| - 1$ polynomial $\hat{\chi}_{\mathbf{z}} : \mathbb{F} \to \mathbb{F}$ is the Lagrange interpolation polynomial, defined as

$$\hat{\chi}_{\mathbf{z}}(\mathbb{z}) \overset{\mathsf{def}}{=} \prod_{h \in H \setminus \{\mathbf{z}\}} \frac{\mathbb{z} - h}{\mathbf{z} - h}.$$

Lines and Planes. A line $L : \mathbb{F} \to \mathbb{F}^m$ is a polynomial mapping of degree 1 (in particular we allow degenerate lines which are just a constant function). Similarly, a plane $M : \mathbb{F}^2 \to \mathbb{F}^m$ is also a polynomial mapping of degree at most 1 in each of its two variables. Note that if M is a plane, then for every $\alpha \in \mathbb{F}$, it holds that both $M(\alpha, \cdot)$ and $M(\cdot, \alpha)$ are lines.

3.1.1 Explicit Representation of Finite Fields

A finite field ensemble is an ensemble $\mathbb{F} = \{\mathbb{F}_n\}_{n\in\mathbb{N}}$, where $\mathbb{F}_n$ is a finite field for every $n \in \mathbb{N}$. We say that a finite field ensemble has an explicit representation if for every n, there exists there exists a representation of the elements of $\mathbb{F}_n$ as bit strings of length $O(\log(|\mathbb{F}_n|))$ and there exist (probabilistic) polynomial-time Turing machines for computing the field operations (e.g., the machine gets as input $n \in \mathbb{N}$ and the representation of two fields elements $\mathbb{x}, \mathbb{y} \in \mathbb{F}_n$ and outputs the representation of $(\mathbb{x} + \mathbb{y})$). These operations include field addition, multiplication, division, equality, generating random elements, access to constants 0 and 1.

When the parameter n is clear from the context, we will abuse notation and use $\mathbb{F}$ to refer both to the ensemble $\{\mathbb{F}_n\}_{n\in\mathbb{N}}$ and to the specific finite field $\mathbb{F}_n$.

3.2 Semantic Security and Homomorphic Encryption (FHE)

A *public-key encryption* scheme consists of three probabilistic polynomial-time algorithms Gen, Enc and Dec. The key Generation algorithm Gen, when given as input a security parameter 1^λ, outputs a pair $(\mathsf{pk}, \mathsf{sk})$ of public and secret keys. The encryption algorithm, Enc, on input a public key pk and a message $m \in \{0,1\}^{\mathsf{poly}(\lambda)}$, outputs a ciphertext $\hat{m}$, and the decryption algorithm, Dec, when given the ciphertext $\hat{m}$ and the secret key sk, outputs the original message m (with overwhelming probability). We allow the decryption process to fail with negligible probability (over the randomness of all algorithms).

Let $s = s(\lambda) \in \mathbb{N}$ and $\delta = \delta(\lambda) \in [0,1]$ be parameters. A public-key encryption scheme has (S, δ)-security if for every family of circuits $\{C_\lambda\}_{\lambda\in\mathbb{N}}$ of size $S(\lambda)$, for all sufficiently large λ and for any two messages $m, m' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ such that $|m| = |m'|$,

$$\left| \Pr_{(\mathsf{pk},\mathsf{sk})\in_R \textsc{Gen}(1^\lambda)} \left[C_\lambda(\mathsf{pk}, \textsc{Enc}_{\mathsf{pk}}(m)) = 1\right] - \Pr_{(\mathsf{pk},\mathsf{sk})\in_R \textsc{Gen}(1^\lambda)} \left[C_\lambda(\mathsf{pk}, \textsc{Enc}_{\mathsf{pk}}(m')) = 1\right] \right| < \delta(\lambda)$$

where the probability is also over the random coin tosses of Enc.

We say that an encryption scheme has sub-exponential security if there exists a constant $\varepsilon > 0$ such that it is $(2^{O(\lambda^\varepsilon)}, 2^{-\Omega(\lambda^\varepsilon)})$-secure.

Homomorphic Encryption. A public-key encryption scheme (Gen, Enc, Dec) is said to be homomorphic wrt to a circuit class $\mathcal{C}$ if there exists an algorithm, called Eval, such that on input the public-key pk, a circuit $C \in \mathcal{C}$ and a ciphertext $\hat{m}$ that is an encryption of a message m with respect to pk, outputs a string ψ such that the following two conditions hold:

- **Homomorphic Evaluation:** $\textsc{Dec}_{\mathsf{sk}}(\psi) = C(m)$, except with negligible probability (over the coins of all algorithms).
- **Compactness:** The length of ψ is polynomial in λ, $|m|$ and $|C(m)|$ but is otherwise independent of the size of C.

Furthermore, in this work we assume that homomorphic evaluation is done in a gate-by-gate manner. (All candidate homomorphic encryption schemes have this property.)

The scheme is *fully homomorphic* if it is homomorphic wrt the class of all polynomial-size circuits.

Remark 3.5. *[Homomorphic Operations over Large Fields] Most candidate homomorphic encryption schemes support addition and multiplication operations over the binary field $\mathbb{GF}(2)$. In our construction we will need to homomorphically evaluate addition and multiplication over a larger field $\mathbb{F}$. Clearly such operations can be emulated by $\mathbb{GF}(2)$ operations but here we care about the efficiency of this transformation.*

For this we restrict our attention to binary *fields (i.e., with characteristic 2). Recall that the elements of such a field can be viewed as a vector space over $\mathbb{GF}(2)$. Addition over such a field $\mathbb{F}$ corresponds to coordinate-wise XOR of vectors, which can be trivially emulated by $\mathbb{GF}(2)$ additions (and no multiplications). Less obvious is the fact that multiplication over $\mathbb{F}$ can be emulated by $\mathbb{GF}(2)$ operations* with multiplicative depth 1. *This is because multiplication of a and b over $\mathbb{F}$, viewed as an operation over $\mathbb{GF}(2)$ is a $\mathbb{GF}(2)$*-linear transformation *if we fix either a or b.*

3.3 No-Signaling PCPs

A probabilistically checkable proof (PCP) is a protocol, between a prover P and verifier V, for convincing the verifier V that a common input $x \in \{0,1\}^n$ belongs to a language $\mathcal{L}$, with soundness depending on a statistical security parameter κ. This protocol is in an idealized model; specifically, the prover produces a proof string π of some length $\ell(n,\kappa)$, and the verifier is allowed to make some number $k(n,\kappa)$ of (randomized) queries to π. Based on the answers to these queries the verifier decides whether to accept or reject.

In this work we exclusively consider PCPs with verifiers whose queries are non-adaptive; in this case, we represent V by two algorithms V_0 and V_1. The first algorithm, V_0, takes as input $(x, 1^\kappa)$ and outputs (Q, st), where Q is the set of queries asked by V and st is some private state. The second algorithm, V_1, "continues" the execution of V, taking as input $(x, 1^\kappa, \mathsf{st}, \pi_Q)$ and outputs a bit representing an accept or reject.

As usual (perfect) completeness means that for any $x \in \mathcal{L}$ and security parameter κ, there is a string π such that $V^\pi(x, 1^\kappa)$ accepts with probability 1. The classical notion of soundness means that for any $x \in \{0,1\}^n \setminus \mathcal{L}$ and for any π, it holds that $V^\pi(x, 1^\kappa)$ accepts with probability at most $\varepsilon(\kappa)$, where ε is called the soundness error.

In this work we shall consider a stronger notion of soundness called *no-signaling (*NS*) soundness.* Let $\Sigma = \Sigma(n,\kappa)$ denote the *alphabet* of π.

Definition 3.6 (NS Prover). *A $(k_{\max}, \delta)$*-NS prover P^* over alphabet Σ *is a set of distributions $\{P^*(Q)\}_{|Q| \le k_{\max}}$, where each $P^*(Q)$ is distributed over Σ^Q, such that for all sets of queries $Q' \subseteq Q$ with $|Q| \le k_{\max}$, the following two distributions on $A_{Q'} \in \Sigma^{Q'}$ have statistical distance at most δ.*

1. *Sample $A_{Q'} \leftarrow P^*(Q')$.*
2. *Sample $A_Q \leftarrow P^*(Q)$ and set $A_{Q'} = (A_Q)_{Q'}$, namely, the restriction of A_Q to the coordinates in Q'.*

Definition 3.7 (NS-PCPs). *A* PCP for a language $\mathcal{L}$ with soundness error ε against $(k_{\max}, \delta)$-NS provers *(where ε, $k_{\max}$, and δ are functions of n and κ) is composed of a prover algorithm P and verifier algorithms (V_0, V_1) such that:*

1. **(Completeness:)** *For every $x \in \mathcal{L}$ and security parameter κ, it holds that*

$$\Pr_{(Q,\mathsf{st})\leftarrow V_0(x,1^\kappa)}[V_1(x,1^\kappa,\mathsf{st},P(x,1^\kappa)_Q)=1]=1.$$

2. **(Soundness:)** *For every $x \in \{0,1\}^n \setminus \mathcal{L}$, every security parameter κ, and every $(k_{\max},\delta)$-NS prover P^*, it holds that:*

$$\Pr_{\substack{(Q,\mathsf{st})\leftarrow V_0(x,1^\kappa)\\ A_Q\leftarrow P^*_\lambda(Q)}}[V_1(\mathsf{st},A_Q)=1]\leq\varepsilon(n,\kappa).$$

3.4 Argument Systems

An argument scheme is a protocol, between a prover P and a verifier V, for convincing the verifier that a common input $x \in \{0,1\}^n$ belongs to some language $\mathcal{L}$ with certainty that depends on a computational security parameter λ. In contrast to PCPs, argument schemes work in a non-idealized model. However, their soundness holds only against computationally bounded provers.

Definition 3.8 (Argument Schemes). *A (non-adaptive, two-message) (s,ε)-sound argument scheme for a language $\mathcal{L}$ consists of an algorithm P and a pair of probabilistic polynomial-time algorithms (V_0,V_1) such that:*

1. *For all $x \in \mathcal{L}$,*

$$\Pr_{(q,\mathsf{st})\leftarrow V_0(x,1^\lambda)}\big[V_1(\mathsf{st},P(x,q))=1\big]=1.$$

2. *For all $x \in \{0,1\}^n \setminus \mathcal{L}$, and all size $s(n)$ circuits P^*,*

$$\Pr_{(q,\mathsf{st})\leftarrow V_0(x,1^\lambda)}\big[V_1(\mathsf{st},P^*(q))=1\big]\leq\varepsilon(n).$$

An argument-system has polynomial-soundness if it is $(\mathsf{poly}(n),1/\mathsf{poly}(n))$-sound for every polynomial poly. We use $\mathsf{argument\text{-}system}$ as a shorthand for a $(\mathsf{poly}(n),1/\mathsf{poly}(n))$-sound argument-system.

4 Models of Computation

A common theme in our construction is the use of specialized computational models that are geared specifically toward our needs. Since the actual formalism is vital for our construction, we go into full details about these model, which might feel a bit tedious at times. The reader may want to initially skip this section and revisit it as necessary later.

Section Organization. First, in Section 4.1 we introduce a model of computation which we call "arithmetic straight line program". This model will conveniently allow us to separate the computation done by the prover "in the clear" from those that are done "under the FHE". In Section 4.2 we formalize the RAM model that we consider. In Section 4.3 we introduce a model of computation which we call "Tree machines" which on the one hand can simulate RAM computations very efficiently (i.e., with logarithmic overhead in time) but on the other, are amenable to PCP techniques. In Section 4.4 we show how to emulate a RAM machine by a Tree machine.

4.1 Arithmetic Straight Line Program (ASLP)

Loosely speaking, an *arithmetic straight line program (*ASLP*)* is a program, defined over some ring $\mathcal{R}$, that is composed of a sequence of arithmetic instructions which can simply add or multiply two registers together.

Definition 4.1. *An* Arithmetic Straight Line Program *(*ASLP*) for inputs of length n, over a ring $\mathcal{R}$ and with k registers $R[1], \ldots, R[k]$, is a sequence of instructions $A = (\mathsf{ins}_1, \ldots, \mathsf{ins}_T)$, where each instruction ins_t takes one of the following forms. Either:*

1. *$R[i] \leftarrow \texttt{INPUT}[j]$ for some $i \in [k]$ and $j \in [n]$; or*
2. *$R[i] := \alpha$, where $i \in [k]$ and $\alpha \in \mathcal{R}$; or*
3. *$R[i] \leftarrow R[j] + R[\ell]$ for some $i, j, \ell \in [k]$; or*
4. *$R[i] \leftarrow R[j] \times R[\ell]$ for some $i, j, \ell \in [k]$; or*
5. *$R[i] \leftarrow \alpha \times R[j]$ for some $i, j \in [k]$ and $\alpha \in \mathcal{R}$; or*
6. *$\texttt{OUTPUT} \leftarrow R[i]$ for some $i \in [k]$.*

We assume that there are m different OUTPUT *instructions. Every such* ASLP *A defines in a natural way a function $A : \mathcal{R}^n \to \mathcal{R}^m$, where the m different outputs appear in the order in which they were generated.*

When the ring $\mathcal{R}$ is clear from the context, we omit it from the notation.

Remark 4.2 (Multiplication by a Constant)**.** *Clearly the instruction $R[i] \leftarrow \alpha \times R[j]$ can be easily emulated by other instructions. The reason that we add this instruction explicitly though is that in the context of fully homomorphic encryption, multiplication by a known constant is much cheaper than multiplication of two encrypted values. In particular, multiplication by a known constant does not increase the multiplicative depth (to be defined next) of the* ASLP*.*

We next define the multiplicative depth as the maximal number of sequential multiplication operations that was used to compute the value of any register.

Definition 4.3 (Multiplicative Depth of ASLP)**.** *The* multiplicative depth *of an* ASLP $(\mathsf{ins}_1, \ldots, \mathsf{ins}_T)$ *with k registers is defined as $\max_{i\in[k],t\in[T]}\{d_{i,t}\}$, where the values $\{d_{i,t}\}$ are defined inductively in t as follows. We define $d_{i,0} \overset{\mathsf{def}}{=} 0$ for all i, and for $t > 0$ we define*

$$d_{i,t} \overset{\mathsf{def}}{=} \begin{cases} 0 & \text{if } \mathsf{ins}_t \text{ is } R[i] \leftarrow \texttt{INPUT}[j] \text{ for some } j \\ 0 & \text{if } \mathsf{ins}_t \text{ is } R[i] \leftarrow \alpha \text{ for some } \alpha \\ \max(d_{j,t-1}, d_{\ell,t-1}) & \text{if } \mathsf{ins}_t \text{ is } R[i] \leftarrow R[j] + R[\ell] \\ 1 + \max(d_{j,t-1}, d_{\ell,t-1}) & \text{if } \mathsf{ins}_t \text{ is } R[i] \leftarrow R[j] \times R[\ell] \\ d_{j,t-1} & \text{if } \mathsf{ins}_t \text{ is } R[i] \leftarrow \alpha \times R[j] \\ d_{i,t-1} & \text{if } \mathsf{ins}_t \text{ is } \texttt{OUTPUT} \leftarrow R[j] \text{ for some } j. \end{cases}$$

Uniformity of ASLPs. We will mainly use ASLPs to describe computations of a function $f_\alpha(x)$, indexed by α, on a value x. First, a bounded time and space RAM machine gets input $\alpha \in \{0,1\}^*$ and outputs an ASLP P_α that computes f_α. We call such a combined program a *uniform* ASLP (see Definition 4.4 for the precise definition).

Definition 4.4. *Let $\mathcal{C}$ be a complexity class. A function family $\{f_\alpha\}_{\alpha\in\{0,1\}^*}$ is* computable by a $\mathcal{C}$-uniform ASLP over $\mathcal{R}$ with k registers and multiplicative depth d *if there is a function in $\mathcal{C}$ that on input $\alpha \in \{0,1\}^*$ outputs an ASLP P_α, with $k(\alpha)$ registers and multiplicative depth d, such that $P_\alpha(x) = f_\alpha(x)$.*

We think of the function f_α as also being parameterized by the description of the ring $\mathcal{R}$ and so the RAM machine has explicit access to a full description of $\mathcal{R}$.

Uniform ASLPs can be emulated by RAM machines as follows.

Fact 4.5 (Emulating ASLP by RAM)**.** *If a function family $\{f_\alpha\}_{\alpha\in\{0,1\}^*}$ is computable by a $\mathsf{TISP}(T,S)$-uniform ASLP over the ring $\mathcal{R}$ with k registers, then $\{f_\alpha\}_{\alpha\in\{0,1\}}$ is computable in time upper bounded by $O(T)$ ring operations and in space $S + O(k \cdot S_\mathcal{R})$, where $S_\mathcal{R}$ is the space to store a single ring element.*

Remark 4.6. *The multiplicative depth of an ASLP is upper bounded by the length of the ASLP. For a $\mathsf{TIME}(T)$-uniform ASLP, the length of the ASLP (and therefore also the multiplicative depth) are upper bounded by T.*

4.1.1 Evaluation of the Multilinear Extension via ASLPs

We proceed to describe some procedures for efficient evaluation of the multilinear extension via ASLPs.

By Fact 3.2, computing the multi-linear extension of an *arbitrary* function $f : \{0,1\}^\ell \to R$ at a point $\bar{\mathbb{z}} \in R^\ell$ reduces to enumerating the coefficients $\left\{\beta_{\bar{\mathbf{z}}\to\bar{\mathbb{z}}} \overset{\mathsf{def}}{=} \prod_{i=1}^{\ell} \beta_{\mathbf{z}_i\to\mathbb{z}_i}\right\}_{\bar{\mathbf{z}}}$. There are two ways of doing this; a time-efficient way, and a depth-efficient way.

Fact 4.7. *For any ring R, let $f_{\ell,R} : R^\ell \to R^{2^\ell}$ denote the function $f_{\ell,R}(\bar{\mathbb{z}}) = (\beta_{\bar{\mathbf{z}}\to\bar{\mathbb{z}}})_{\bar{\mathbf{z}}\in\{0,1\}^\ell}$. The function family $f = \{f_{\ell,R}\}$ is computable both by:*

- *A $\mathsf{TISP}\big(O(2^\ell\cdot\log(\ell)), O(\ell)\big)$-uniform ASLP with $O(\ell)$ registers and multiplicative depth $\log(\ell)+O(1)$; and*
- *A $\mathsf{TISP}\big(O(2^\ell), O(\ell)\big)$-uniform ASLP with $O(\ell)$ registers and multiplicative depth $\ell + O(1)$.*

Fact 3.2 directly gives a way to evaluate multi-linear extensions.

Corollary 4.8. *There is an oracle word-RAM algorithm $\mathcal{A}$ such that for any ring R, any positive integer ℓ, and any function $f : \{0,1\}^\ell \to R$, executing $\mathcal{A}$ on input (R,ℓ) with oracle f results in an ASLP that computes $\hat{f}$. Furthermore:*

1. *The process of generating the ASLP takes time $O(2^\ell \cdot \log(\ell) \cdot \log|R|)$ and space $O(\ell + \log|R|)$.*
2. *The ASLP itself has length $O(2^\ell \cdot \log(\ell))$, uses $O(\ell)$ registers, and has multiplicative depth $\log(\ell) + O(1)$.*

Corollary 4.9 (See also [Tha13, Remark 1]). *There is an oracle word-RAM algorithm $\mathcal{A}$ such that for any ring R, any positive integer ℓ, and any function $f : \{0,1\}^\ell \to R$, executing $\mathcal{A}$ on input (R, ℓ) with oracle f results in an* ASLP *that computes $\hat{f}$. Furthermore:*

1. *The process of generating the* ASLP *takes time $O(2^\ell \cdot \log |R|)$ and space $O(\ell + \log |R|)$.*
2. *The* ASLP *itself has length $O(2^\ell)$, uses $O(\ell)$ registers, and has multiplicative depth $\ell + O(1)$.*

4.2 RAM Machines

When studying the fine-grained prover overhead for general-purpose computation, the choice of computational model is important. Most models of computation can simulate each other with polynomial blow-up in running-time and constant factor blow-up in space usage, but we want to simulate real-world computations with nearly constant overhead. We therefore focus on the word RAM model of computation, which is standard in practice.

In this section we formally define the RAM machine model that we consider. Our definition allows for flexibility in the concrete set of (word) operations that the RAM supports.

Definition 4.10 (RAM Machine). *A* RAM machine *M relative to a finite operation set*[27] *$\mathcal{O}$ of functions $\{\star_i : \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}\}_i$ is a finite list of ℓ instructions $(\mathsf{ins}_1, \ldots, \mathsf{ins}_\ell)$, each taking one of the following forms (for some $i, j, k \in \mathbb{N}$).*

- *Input: $R[i] \leftarrow$ `INPUT`$[R[j]]$*
- *Load/Store instructions: One of*
  - $R[i] \leftarrow c$ *for some constant* $c \in \mathbb{Z}_{\geq 0}$
  - $R[i] \leftarrow R[j]$
  - $R[i] \leftarrow R[R[j]]$
  - $R[R[i]] \leftarrow R[j]$
- *Word operations: $R[i] \leftarrow R[j] \star R[k]$ for some operation $\star \in \mathcal{O}$.*
- *Control flow instructions, either*
  - `GOTO` *i, or*
  - `GOTO` i `IF` $R[j] \neq 0$.
- *Halting instructions:* `ACCEPT` *or* `REJECT`*.*

4.2.1 Execution Semantics

Loosely speaking, the execution of a RAM machine M on a given input string x proceeds as follows. We start with a memory tape which is set to an infinite sequence of 0's. We use $R[i]$ to refer to the i^{th} memory cell. At each time step, we execute the relevant RAM instruction. For example, $R[i] \leftarrow \texttt{INPUT}[R[j]]$ means that i^{th} memory cell $R[i]$ obtains the value of the $(R[j])$-th bit of x.

[27]The set of allowed operations is a common source of variation between different RAM models, but for us it only matters that the set is finite and each operation is computable in polynomial time.

The other operations are similarly defined in the natural way. Throughout the execution we keep track of a *program counter* $\mathsf{pc} \in [\ell]$ that points to the current instruction being executed by the RAM program. The program counter advances linearly, except in the case of `GOTO` instructions (or the halting instructions `ACCEPT` and `REJECT`). We say that $M(x) = 1$ (resp., $M(x) = 0$) if the RAM machine halts at the `ACCEPT` (resp, `REJECT`) instruction, given input x. We proceed to the formalization of the foregoing discussion.

Definition 4.11. *A* configuration *of a RAM machine $M = (\mathsf{ins}_1, \ldots, \mathsf{ins}_\ell)$ is a tuple $(\mathsf{pc}, \mathcal{T})$, where*

- $\mathsf{pc} \in [\ell]$ *is a "program counter" indicating the next instruction to be executed, and*
- $\mathcal{T} : \mathbb{N} \to \mathbb{Z}_{\geq 0}$ *is the memory contents of M.*

If $\mathsf{ins}_{\mathsf{pc}} = \texttt{ACCEPT}$, the configuration is said to be an accepting *configuration. If $\mathsf{ins}_{\mathsf{pc}} = \texttt{REJECT}$, the configuration is said to be a* rejecting *configuration. In either case, the configuration is said to be a* halting *configuration.*

Definition 4.12. *The* initial *configuration of a RAM machine $M = (\mathsf{ins}_1, \ldots, \mathsf{ins}_\ell)$ is the configuration $(1, \mathcal{T}_0)$, where $\mathcal{T}_0(\alpha) = 0$ for all $\alpha \in \mathbb{N}$.*

The following definition formalizes the "evolution" of a RAM machine computation.

Definition 4.13. *A non-halting configuration $(\mathsf{pc}, \mathcal{T})$ for a RAM machine $M = (\mathsf{ins}_1, \ldots, \mathsf{ins}_\ell)$ is said to* yield *the configuration $(\mathsf{pc}', \mathcal{T}')$ on input $x = x_1 \cdots x_n$, denoted $(\mathsf{pc}, \mathcal{T}) \overset{M,x}{\to} (\mathsf{pc}', \mathcal{T}')$, if the following holds:*

- *If $\mathsf{ins}_{\mathsf{pc}}$ is of the form $R[i] \leftarrow \texttt{INPUT}[R[j]]$, then*

$$\mathcal{T}'(\alpha) = \begin{cases} x_{\mathcal{T}(j)} & \text{if } \alpha = i \text{ and } 1 \leq \mathcal{T}(j) \leq n \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$

  and $\mathsf{pc}' = \mathsf{pc} + 1.$[28]

- *If $\mathsf{ins}_{\mathsf{pc}}$ is of the form $R[i] \leftarrow c$, then*

$$\mathcal{T}'(\alpha) = \begin{cases} c & \text{if } \alpha = i \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$

  and $\mathsf{pc}' = \mathsf{pc} + 1.$

- *If $\mathsf{ins}_{\mathsf{pc}}$ is of the form $R[i] \leftarrow R[j]$, then*

$$\mathcal{T}'(\alpha) = \begin{cases} \mathcal{T}(j) & \text{if } \alpha = i \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$

  and $\mathsf{pc}' = \mathsf{pc} + 1.$

[28]The behavior of this instruction on $R[j] > n$ can be used to determine the value of n via a binary search.

- *If* $\mathsf{ins_{pc}}$ *is of the form* $R[i] \leftarrow R[R[j]]$, *then*
$$\mathcal{T}'(\alpha) = \begin{cases} \mathcal{T}(\mathcal{T}(j)) & \text{if } \alpha = i \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$
and $\mathsf{pc}' = \mathsf{pc} + 1$.

- *If* $\mathsf{ins_{pc}}$ *is of the form* $R[R[i]] \leftarrow R[j]$, *then*
$$\mathcal{T}'(\alpha) = \begin{cases} \mathcal{T}(j) & \text{if } \alpha = \mathcal{T}(i) \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$
and $\mathsf{pc}' = \mathsf{pc} + 1$.

- *If* $\mathsf{ins_{pc}}$ *is of the form* $R[i] \leftarrow R[j] \star R[k]$ *for some operation* $\star \in \mathcal{O}$, *then*
$$\mathcal{T}'(\alpha) = \begin{cases} \mathcal{T}(j) \star \mathcal{T}(k) & \text{if } \alpha = i \\ \mathcal{T}(\alpha) & \text{otherwise,} \end{cases}$$
and $\mathsf{pc}' = \mathsf{pc} + 1$.

- *If* $\mathsf{ins_{pc}}$ *is of the form* `GOTO` i, *then* $\mathcal{T}' = \mathcal{T}$, *and* $\mathsf{pc}' = i$.

- *If* $\mathsf{ins_{pc}}$ *is of the form* `GOTO` i `IF` $R[j] \neq 0$, *then* $\mathcal{T}' = \mathcal{T}$, *and*
$$\mathsf{pc}' = \begin{cases} i & \text{if } \mathcal{T}(j) \neq 0 \\ \mathsf{pc} + 1 & \text{otherwise.} \end{cases}$$

When M and x are clear from the context we omit them from the notation and simply write $\mathcal{C} \to \mathcal{C}'$.

Definition 4.14. *If* $\mathcal{C}_0$ *is the initial configuration of* M, *and if there is a finite sequence of configurations* $\mathcal{C}_0 \overset{M,x}{\to} \cdots \overset{M,x}{\to} \mathcal{C}_T$ *with* $\mathcal{C}_T$ *a halting configuration, then* $(\mathcal{C}_0, \ldots, \mathcal{C}_T)$ *is said to be the* transcript *of* M *on* x, *and we say that* M halts *at* $\mathcal{C}_T$ *on* x. *We denote this symbolically by writing* $\mathcal{C}_0 \overset{M,x}{\Rightarrow} \mathcal{C}_T$.

Definition 4.15. *The evaluation of* M *on* x *is defined as*
$$M(x) \overset{\mathsf{def}}{=} \begin{cases} 1 & \text{if } M \text{ halts at } \mathcal{C} \text{ on } x \text{ for some accepting configuration } \mathcal{C} \\ 0 & \text{if } M \text{ halts at } \mathcal{C} \text{ on } x \text{ for some rejecting configuration } \mathcal{C} \\ \perp & \text{otherwise.} \end{cases}$$

Recall that we defined the notation $s \| s'$ to denote the concatenation of s and s'. We next define a *combined access pattern* as the list of all memory and input locations that are accessed by the RAM program on a given input. Formally, this is defined as follows.

Definition 4.16. *If the transcript for a RAM machine* $M = (\mathsf{ins}_1, \ldots, \mathsf{ins}_\ell)$ *on an input* x *is* $((\mathsf{pc}_0, \mathcal{T}_0), \ldots, (\mathsf{pc}_T, \mathcal{T}_T))$, *then the* combined access pattern *(access pattern for short) of* M *on* x *is the concatenated tuple* $\mathbf{a}_1 \| \cdots \| \mathbf{a}_T$, *where for each* $t \in [T]$,

- *If* $\mathsf{ins}_{\mathsf{pc}_t}$ *is of the form* $R[i] \leftarrow \mathit{INPUT}[R[j]]$*, then* $\mathbf{a}_t$ *is the tuple* $(j, \mathcal{T}_t(j), i)$*.*
- *If* $\mathsf{ins}_{\mathsf{pc}_t}$ *is of the form* $R[i] \leftarrow c$*, then* $\mathbf{a}_t$ *is the singleton tuple* (i)*.*
- *If* $\mathsf{ins}_{\mathsf{pc}_t}$ *is of the form* $R[i] \leftarrow R[j]$*, then* $\mathbf{a}_t = (j, i)$*.*
- *If* $\mathsf{ins}_{\mathsf{pc}_t}$ *is of the form* $R[i] \leftarrow R[R[j]]$*, then* $\mathbf{a}_t = (j, \mathcal{T}_t(j), i)$*.*
- *If* $\mathsf{ins}_{\mathsf{pc}_t}$ *is of the form* $R[R[i]] \leftarrow R[j]$*, then* $\mathbf{a}_t = (j, i, \mathcal{T}(i))$*.*
- *If* $\mathsf{ins}_{\mathsf{pc}_t}$ *is of the form* $R[i] \leftarrow R[j] \star R[k]$*, then* $\mathbf{a}_t = (j, k, i)$*.*
- *If* $\mathsf{ins}_{\mathsf{pc}_t}$ *is of the form* $\mathit{GOTO}\ i\ \mathit{IF}\ R[j] \neq 0$*, then* $\mathbf{a}_t$ *is the singleton tuple* (j)*.*
- *Otherwise,* $\mathbf{a}_t$ *is the empty tuple.*

Remark 4.17. *Note that the list of memory locations in Definition 4.16 does not distinguish between memory and input accesses. This is done for technical reasons that slightly simplify our emulation of RAM machines by Tree machines in Section 4.4.*

4.2.2 Time and Space Complexity

We next define the time and space complexity measures for RAM machines.

Definition 4.18. *When a (finite) transcript* $(\mathcal{C}_0, \ldots, \mathcal{C}_T)$ *exists*[29] *for a RAM machine* M *on input* x*, then the* running time *of* M *on* x *is defined as* $\mathsf{TIME}(M, x) \stackrel{\mathsf{def}}{=} T$*.*

Definition 4.19. *The* space usage *of a tape* $\mathcal{T}$ *of a RAM machine is defined as*

$$\mathsf{SPACE}(\mathcal{T}) \stackrel{\mathsf{def}}{=} \max\{i : \mathcal{T}(i) \neq 0\}.$$

The space usage *or a configuration* $\mathcal{C} = (\mathsf{pc}, \mathcal{T})$ *is defined as* $\mathsf{SPACE}(\mathcal{C}) = \mathsf{SPACE}(\mathcal{T})$*.*

When a transcript $(\mathcal{C}_0, \ldots, \mathcal{C}_T)$ *exists for a RAM machine* M *on input* x*, the* space usage *of* M *on* x *is defined as*

$$\mathsf{SPACE}(M, x) \stackrel{\mathsf{def}}{=} \max_{i \in [T]} \mathsf{SPACE}(\mathcal{C}_i).$$

So far we have not bounded the amount of information that can be stored in each one of the memory cells. The standard RAM model allows $O(\log n)$ bits of information per cell, where n is the input length. Likewise, we have not placed any restrictions on the complexity of the RAM operations. We define a word RAM to as a RAM that uses bounded size words as its memory and whose supported binary operations are computable in polynomial time. More formally:

Definition 4.20 (Word RAM)**.** *Let* $w = w(n) \in \mathbb{N}$*. A RAM machine* M *with operations* $\mathcal{O}$ *is a* $w(\cdot)$-bit word RAM *if:*

- *Each* $\star \in \mathcal{O}$ *is computable (by, say, a Turing machine) in polynomial time (in the bit length of its input).*
- *For each* $x \in \{0,1\}^n$ *with corresponding transcript* $((\mathsf{pc}_0, \mathcal{T}_0), \ldots, (\mathsf{pc}_T, \mathcal{T}_T))$*, it holds that* $\mathcal{T}_t(i) < 2^{w(n)}$*, for all* $t \in \{0, \ldots, T\}$ *and all* $i \in \mathbb{N}$*.*

For natural RAM programs the word size is typically $\Theta(\log(S))$.

[29] Recall that a machine does not necessarily halt for every input.

4.3 Tree Machines

When designing MIPs, it is convenient to work with a more "stepwise-local" computation model like a Turing machine. Unfortunately Turing machines can only simulate RAM machines with quadratic overhead, so they are not suitable for studying the fine-grained prover overhead. Instead we introduce a strengthening of the Turing machine model in which the work tape, rather than being linear, is a binary tree of nodes, each of which has its own sequential tape. The machine head can move either within a given tape, or to a neighboring tape.

Definition 4.21. *A* tree machine *is a tuple* $(Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$*, where*

- Q *is the set of states,*
- Σ *is the input alphabet, not containing the blank symbol* ϵ*,*
- $\Gamma \supseteq \Sigma \cup \{\epsilon\}$ *is the tape alphabet,*
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\uparrow, \swarrow, \searrow, \otimes, \odot\}$ *is the transition function*[30]*,*
- $q_0 \in Q$ *is the start state,*
- $q_{\mathsf{acc}} \in Q$ *is the accept state, and*
- $q_{\mathsf{rej}} \in Q$ *is the reject state, where* $q_{\mathsf{rej}} \neq q_{\mathsf{acc}}$*.*

4.3.1 Execution Semantics

A tree machine is evaluated on a given input x in the following natural way. We consider an infinite rooted binary tree, where each vertex is associated with an infinite memory tape. The i-th input bit is placed in the first cell of the i-th vertex of the binary tree[31] (where the indexing is done in a breadth first manner). At any given time, the machine head points to one memory cell of one of the vertices of the tree. Similarly to a Turing machine, each computation step depends only on the memory cell located under the machine head and an internal machine state. Given these, the machine can write on the current head location, and move *either* sequentially on the tape associated with the current vertex, or to one of the three neighboring vertices (i.e., the parent and two children). Intuitively, this lets the tree machine emulate a RAM machine with only logarithmic overhead in time (this will be done formally in Construction 4.32). We proceed to the formalization of the above discussion.

Definition 4.22. *A* configuration *of a tree machine* $\mathsf{TM} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$ *is a tuple* $\big(q, (v, z), \mathcal{T}\big)$*, where*

- $q \in Q$ *is the control state of* TM*,*
- $(v, z) \in \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$ *is the head position of* TM*;* v *indexes nodes of a binary tree in breadth-first order (i.e., the root is labeled* 1*, and any node labeled* ℓ *has children labeled* 2ℓ *and* $2\ell + 1$*)*[32]*, and* z *indexes a position on that node's sequential tape.*

[30] The symbol $\otimes$ (resp., $\odot$), borrowing notation from physics, is meant to evoke an arrow pointing *into* (resp., out of) the page (the viewer sees either the tail end (with feathers) or the pointy end (with a dot) of the arrow).

[31] This is mainly done for technical convenience and to avoid having an additional tree for the input bits.

[32] The observant reader will notice that we have not accounted for the vertex labeled 0; this is intentional and allows us to handle some edge cases in a simpler way later on.

- $\mathcal{T} : \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \to \Gamma$ *is a function representing the tape contents of* TM.

A configuration in which $q = q_{\mathsf{acc}}$ *is called an* accepting configuration*, and when* $q = q_{\mathsf{rej}}$*, it is called a* rejecting configuration*. Either case is a* halting configuration*.*

Definition 4.23. *The* initial configuration *of a tree machine* $\mathsf{TM} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$ *on input* $x = (x_1, \dots, x_n) \in \Sigma^n$ *is* $(q_0, (0, 0), \mathcal{T}_0)$*, where*

$$\mathcal{T}_0(v, z) \overset{\mathsf{def}}{=} \begin{cases} x_i & \text{if } v = i \text{ and } z = 0 \\ \epsilon & \text{otherwise.} \end{cases}$$

Observe that, as mentioned earlier, in a tree machine the input bits are stored in memory location 0 of the first n nodes of the tree.

Definition 4.24. *For a given tree machine* $\mathsf{TM} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$*, and any non-halting configuration* $(q, (v, z), \mathcal{T})$*, we say that* $(q, (v, z), \mathcal{T})$ yields $(q', (v', z'), \mathcal{T}')$*, denoted* $(q, (v, z), \mathcal{T}) \overset{\mathsf{TM}}{\to} (q', (v', z'), \mathcal{T}')$*, if*

1. $(q', \mathcal{T}'(v, z), d) = \delta(q, \mathcal{T}(v, z))$*; and*
2. $v' = \begin{cases} \lfloor v/2 \rfloor & \text{if } d = \uparrow \\ 2v & \text{if } d = \swarrow \\ 2v + 1 & \text{if } d = \searrow \\ v & \text{otherwise; and} \end{cases}$
3. $z' = \begin{cases} z + 1 & \text{if } d = \otimes \\ \max(0, z - 1) & \text{if } d = \odot \\ z & \text{otherwise; and} \end{cases}$
4. $\mathcal{T}'(u) = \mathcal{T}(u)$*, for all* $u \neq (v, z)$*.*

For any halting configuration $(q, (v, z), \mathcal{T})$*, we say that* $(q, (v, z), \mathcal{T})$ *yields* $(q, (v, z), \mathcal{T})$*.*

We also write $(q', (v', z'), \mathcal{T}') = \mathcal{S}((q, (v, z), \mathcal{T}))$*, saying that* $(q', (v', z'), \mathcal{T}')$ *is the* successor *of* $(q, (v, z), \mathcal{T})$*.*

Definition 4.25. *If* $\mathcal{C}_0$ *is the initial configuration of* TM *on input* x*, and if there exists a finite sequence of configurations* $\mathcal{C}_0 \overset{\mathsf{TM}}{\to} \cdots \overset{\mathsf{TM}}{\to} \mathcal{C}_T$ *with* $\mathcal{C}_{T-1}$ *a non-halting configuration and* $\mathcal{C}_T$ *a halting configuration, then* $(\mathcal{C}_0, \dots, \mathcal{C}_T)$ *is said to be the* transcript *of* TM *on* x*, and we say that* TM halts *at* $\mathcal{C}_T$ *on* x*.*

Definition 4.26. *The evaluation of* TM *on* x *is defined as*

$$\mathsf{TM}(x) \overset{\mathsf{def}}{=} \begin{cases} 1 & \text{if } \mathsf{TM} \text{ halts at } \mathcal{C} \text{ on } x \text{ for some accepting configuration } \mathcal{C} \\ 0 & \text{if } \mathsf{TM} \text{ halts at } \mathcal{C} \text{ on } x \text{ for some rejecting configuration } \mathcal{C} \\ \perp & \text{otherwise} \end{cases}$$

Definition 4.27. *If the transcript of a tree machine* TM *on an input* x *is* $(\mathcal{C}_0, \dots, \mathcal{C}_{T-1})$*, we say that the* running time *of* TM *on* x *is* T*, which we denote by* $\mathsf{TIME}(\mathsf{TM}, x)$*.*

Remark 4.28 (Space Usage of Tree Machine). *Loosely speaking, we could define the space usage of a Tree machine as the sum over all nodes of the maximal location in the respective node's tape that we accessed.*

In the technical parts below however we never consider the space usage of the Tree machine, but rather only of the RAM machine that it emulates. The reason is that the more modular alternative introduces overhead that we cannot afford.

Definition 4.29. *If the transcript of a tree machine* TM *on an input x is* $(\mathcal{C}_0, \dots, \mathcal{C}_{T-1})$ *with* $\mathcal{C}_i = (q_i, (v_i, z_i), \mathcal{T}_i)$, *we say that the* access pattern *of* TM *on x is* $((v_0, z_0), \dots, (v_T, z_T))$.

Loosely speaking, we say that a configuration is (d, ℓ)-bounded if it does not involve memory cells beyond a depth d of the tree, nor memory cells that are at distance more than ℓ in any of the tapes associated with vertices at depth less than d. More formally, we define bounded configurations as follows.

Definition 4.30 (Bounded Configuration). *A configuration* $(q, (v^*, z^*), \mathcal{T})$ *is said to be* (d, ℓ)-bounded *if* $v^* < 2^d$, $z^* < \ell$, *and whenever* $v \geq 2^d$ *or* $z \geq \ell$, *it holds that* $\mathcal{T}(v, z) = \epsilon$.

We also give an alternate formalization of a bounded configuration. Jumping ahead, this formalization will be convenient for some algebraic manipulations in our eventual construction.

Definition 4.31 (Configuration Function Representation). *We define the* function representation *of a* (d, ℓ)*-bounded configuration* $(q, (v^*, z^*), \mathcal{T})$ *of a tree machine* $\mathsf{TM} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$ *as the function*

$$\mathcal{C} : \{0,1\}^d \times \{0,1\}^{\lceil \log \ell \rceil} \to \bar{\Gamma},$$

where $\bar{\Gamma} = (Q \sqcup \{\perp\}) \times \Gamma \times E$ *and* $E = \{\mathsf{leaf}, \perp\} \times \{\mathsf{front}, \mathsf{back}, \perp\}$ *is the (finite) set of possible edge cases. Namely,*

$$\mathcal{C}(u) = \begin{cases} (q, \mathcal{T}(u), e(u)) & \text{if } u = (v^*, z^*) \\ (\perp, \mathcal{T}(u), e(u)) & \text{otherwise,} \end{cases}$$

where $e((v, z)) = (e_1(v), e_2(z))$ *and*

$$e_1(v) = \begin{cases} \mathsf{leaf} & \text{if } v \geq 2^{d-1} \\ \perp & \text{otherwise} \end{cases}$$

and

$$e_2(z) = \begin{cases} \mathsf{front} & \text{if } z = 0 \\ \mathsf{back} & \text{if } z = \ell - 1 \\ \perp & \text{otherwise.} \end{cases}$$

4.4 Emulating RAM machines with Tree machines

In this section we describe how to emulate a word RAM by a tree machine with logarithmic overhead in time .

Construction 4.32 (Emulating RAM by Tree Machine - Sketch). *Given a word RAM machine M, we construct a tree machine* $\mathsf{Tree}(M)$ *with input alphabet* $\{0, 1\}$ *as follows.*

$\mathsf{Tree}(M)$ *stores the program counter of* M *in its internal control state, and stores the memory contents* $\mathcal{T}'$ *of* M *in its own tape* $\mathcal{T}$ *by storing the* i^{th} *bit of* $\mathcal{T}'(j)$ *in* $\mathcal{T}(j,i)$ *for* $i \in \{1,\dots,w(n)\}$*. The* j^{th} *input bit is stored in* $\mathcal{T}(j,0)$ *for* $j \in \{1,\dots,n\}$*.*

$\mathsf{Tree}(M)$ *reserves* $\mathcal{T}(0,\cdot)$ *as a linear "work tape", and simulates two "virtual" tape heads: one is dedicated to this tape, and the other traverses the rest of the tree tape (i.e.,* $\mathcal{T}(v,z)$ *for* $v > 0$*) freely. The simulation simply runs the real head back and forth between the two head positions. This necessitates augmenting the tape alphabet with an additional bit used to mark the positions of the two heads. For the second head, we also mark the* path *to its position starting from the root of the tree, by leaving "breadcrumbs": each vertex along the path indicates the direction toward the second head.*

$\mathsf{Tree}(M)$ *now simulates* M *instruction-by-instruction. The linear tape* $\mathcal{T}(0,\cdot)$ *is used primarily to perform the word operations specified by* M*. It also serves as a place to store indirect addresses (i.e.,* $R[j]$ *in instructions of the form* $R[i] \leftarrow R[R[j]]$ *or* $R[R[j]] \leftarrow R[i]$*).*

$\mathsf{Tree}(M)$*'s state space* Q *is augmented if necessary to reveal the most recent tape motion; that is, we ensure there is a well-defined function* $\mathsf{dir} : Q \to \{\uparrow, \swarrow, \searrow, \otimes, \odot\}$ *such that whenever* $(q',\gamma',d) = \delta(q,\gamma)$*, it holds that* $\mathsf{dir}(q') = d$*. Additionally, we ensure that* $\mathsf{Tree}(M)$ *always moves its tape head to the position* $(0,0)$ *before halting.*

Proposition 4.33 (Correctness)**.** *For any* $x \in \{0,1\}^*$*, it holds that* $\mathsf{Tree}(M)(x) = M(x)$*.*

Let M be a w-bit word RAM machine relative to a set of operations $\mathcal{O}$. We denote by $W = W(n) = \mathsf{poly}(w(n))$ the maximal time complexity for any operation in $\mathcal{O}$ on inputs of length w.

Let $T = T(n)$ and $S = S(n)$ respectively denote the time and space usage of M on input $x \in \{0,1\}^n$. Define $S' \overset{\mathsf{def}}{=} \max(S,n)$.

Proposition 4.34 (Efficiency)**.** *The following efficiency constraints hold:*

- *The running time* T' *of* $\mathsf{Tree}(M)$ *on* x *is at most* $O(T \cdot \log S' \cdot W^2)$*, and*
- *Each configuration* $\mathcal{C}_i$ *in the transcript of* $\mathsf{Tree}(M)$ *on* x *is* $(\lceil \log S' \rceil, W)$*-bounded.*

Propositions 4.33 and 4.34 follow directly from the construction.

Proposition 4.35. *The access pattern of* $\mathsf{Tree}(M)$ *on input* x *is a* $W \cdot \lceil \log(S'+1) \rceil$*-blowup (as in Definition 5.7) of the combined access pattern of* M *on* x*.*

More generally, let $((v_0,z_0),\dots,(v_T,z_T))$ *denote the access pattern of* $\mathsf{Tree}(M)$ *on* x*, and let* $(a_1,\dots,a_{T'})$ *denote the combined access pattern of* M *on* x*.*

- *For any* $i \le \lceil \log W \rceil$*,*
$$((v_0, \textsc{Prefix}_i(z_0)),\dots,(v_T, \textsc{Prefix}_i(z_T)))$$
is a $2^i \cdot \lceil \log(S'+1) \rceil$*-blowup of*
$$(a_1,\dots,a_{T'}).$$
- *For any* $i \le \lceil \log(S'+1) \rceil$*,*
$$(\textsc{Prefix}_i(v_0),\dots,\textsc{Prefix}_i(v_T))$$
is an $(i+1)$*-blowup of*
$$(\textsc{Prefix}_i(a_1),\dots,\textsc{Prefix}_i(a_{T'})).$$

Proof Sketch. The more general statement follows from the fact that, when emulating a step of M that accesses either the a_i-th input bit or the a_i-th word of memory, $\mathsf{Tree}(M)$ accesses its tape $\mathcal{T}$ only at (v, z) for which v is an "ancestor" (i.e., prefix) of a_i. $\square$

4.4.1 Emulating the Emulator: Succinct Configurations and Emulation of $\mathsf{Tree}(M)$

Let M be a RAM machine and let x be an input for M. Let T and S denote the running time and space usage, respectively, of M on x. We show that the tree machine $\mathsf{Tree}(M)$'s execution on x is "emulatable" by a RAM machine with similar time and space usage to M.

Recall that $\mathsf{Tree}(M)$ is $(\lceil \log S' \rceil, W)$-bounded (see Proposition 4.34). Thus, using the function representation (see Definition 4.31), each configuration $\mathcal{C}$ of $\mathsf{Tree}(M)$ is a function $\mathcal{C} : \{0,1\}^s \to \bar{\Gamma}$, where $s \overset{\mathsf{def}}{=} \lceil \log S' \rceil + \lceil \log W \rceil$ and $\bar{\Gamma}$ is unimportant for the current discussion.

Proposition 4.36. *There exists a data structure $\mathcal{DS}$, of size $S + O(W)$, that represents the configurations of $\mathsf{Tree}(M)$ (executed on input x) as follows. At any point in time, the data structure's state implicitly represents a single configuration $\mathcal{DS}.\mathcal{C}$ of $\mathsf{Tree}(M)$ and supports the following operations (given read-only random access to x).*

- $\textsc{InitConfigDS}(M, x)$: *returns, in time $O(1)$, an initialized data structure $\mathcal{DS}$ such that $\mathcal{DS}.\mathcal{C}$ is the initial configuration of $\mathsf{Tree}(M)$ on input x.*
- $\mathcal{DS}.\textsc{ActiveCells}()$: *returns $\{\bar{\mathbf{u}} : \mathcal{DS}.\mathcal{C}(\bar{\mathbf{u}}) \neq \mathsf{next}(\mathcal{DS}.\mathcal{C})(\bar{\mathbf{u}})\}$ in $O(1)$ time, where $\mathsf{next}(\mathcal{DS}.\mathcal{C})$ denotes the successor configuration of $\mathcal{DS}.\mathcal{C}$.*
- $\mathcal{DS}.\textsc{Eval}(R, \bar{\mathbf{u}})$: *where R is a ring*[33] *and $\bar{\mathbf{u}} \in \{0,1,?\}^s$. When R is implicit, we omit it as an input. If there are k "?" symbols in $\bar{\mathbf{u}}$, then $\textsc{Eval}$ outputs an ASLP that on input $\bar{\mathbb{z}} \in R^k$ does the following.*

  1. *For each $i \in [k]$, replace the i^{th} "?" symbol of $\bar{\mathbf{u}}$ by $\mathbb{z}_i$, and denote the resulting element of R^s by $\bar{\mathbb{u}}$.*
  2. *Compute and output the multi-linear extension of $\mathcal{DS}.\mathcal{C}$ evaluated on $\bar{\mathbb{u}}$.*

  The process of generating the ASLP takes time $\tilde{O}(2^k)$ and space $\mathsf{poly}(k)$, and the ASLP itself has length $O(k \cdot 2^k)$, uses $O(1)$ registers, and has multiplicative depth $\log(k) + O(1)$.

  We typically think of using $\mathcal{DS}.\textsc{Eval}$ to evaluate the low-degree extension of the current configuration $\mathcal{DS}.\mathcal{C}$ on an input $\bar{\mathbb{u}} \in R^s$, some of whose coordinates are known to be 0 or 1. This knowledge is represented by a vector $\bar{\mathbf{u}} \in \{0,1,?\}^s$. When this is the case (and $\bar{\mathbf{u}}$ is clear from context), we abuse notation and write $\mathcal{DS}.\textsc{Eval}(\bar{\mathbb{u}})$ to denote the result of (1) generating an ASLP P from $\mathcal{DS}.\textsc{Eval}(\bar{\mathbf{u}})$, and (2) evaluating P on the vector obtained from $\bar{\mathbb{u}}$ by removing those coordinates that were already known to be 0 or 1 (i.e., those that do not correspond to a "?" in $\bar{\mathbf{u}}$).
- $\mathcal{DS}.\textsc{Advance}()$: *update $\mathcal{DS}.\mathcal{C}$ to the configuration $\mathsf{next}(\mathcal{DS}.\mathcal{C})$ in $O(1)$ time, where next is the successor operation as defined above.*

[33] We assume that there is an efficient injective mapping from $\bar{\Gamma}$ to R

Proof. Before we describe the data format of $\mathcal{DS}$, first recall from Construction 4.32 that $\mathsf{Tree}(M)$ emulates M instruction by instruction, so that each transition $\mathcal{C} \to \mathcal{C}'$ of $\mathsf{Tree}(M)$ is part of the emulation of some transition $\bar{\mathcal{C}} \to \bar{\mathcal{C}}'$ of M. We now define $\mathcal{DS}$ so that when (implicitly) representing $\mathcal{C}$, it (explicitly) contains:

- The configuration $\bar{\mathcal{C}}$ (which requires $S + O(1)$ words of space).
- Additional state which pertains to the emulation of a single instruction of M, namely
  - The control state q of $\mathsf{Tree}(M)$. This requires $O(1)$ space.
  - The contents of $\mathsf{Tree}(M)$'s work tape $\mathcal{T}(0,\cdot)$ (which requires space $O(W)$)
  - The positions of $\mathsf{Tree}(M)$'s two "virtual" tape heads (which requires $O(1)$ words of space). Note that this implicitly defines the positions of all the "breadcrumbs" mentioned in Construction 4.32.
  - The position of $\mathsf{Tree}(M)$'s real tape head (which requires $O(1)$ words of space).

The definitions of ActiveCells, Eval, and Get are then straight-forward. □

Corollary 4.37. *The data structure $\mathcal{DS}$ defined in Proposition 4.36 also supports the operations* NextActiveCells(i) *and* NextEval$(\llbracket R \rrbracket, \bar{\mathrm{u}}, i)$ *operations that are the same as* ActiveCells *and* Eval *(respectively) but applied to the i^{th}* successor *configuration of $\mathcal{C}$ (and without modifying the internal representation of $\mathcal{DS}$). These operations take $O(i)$ time and space. We sometimes omit the parameter i in which case we mean to use as default $i = 1$ (i.e., the immediate successor).*

Proof. Since calling Advance i times takes $O(i)$ time, we can just store all the changes that Advance would have made without actually performing them. □

5 Technical Tools

In this section we introduce some new technical tools that will be useful in our main construction and its analysis. In Section 5.1 we show a general result on how to efficiently evaluate the multi-linear extension of functions that can be evaluated by small read-once branching programs. This result generalizes some ad-hoc approaches for computing the multilinear extension of specific functions, e.g. in [BHK16] and [GR17]. Then, in Section 5.2 we discuss basic notions and results from caching theory.

5.1 Computing Low Degree Extensions of Read-Once Branching Programs

In this section we show a general result by which the multi-linear extension of function computable by read-once branching programs can be efficiently computed. This result also extends to the more general setting of *low degree extension.* While in this work we will only apply the result to computing multi-linear extensions, we believe that this result will be useful for future work and so we prove a more general statement here.

Definition 5.1. *A* branching program *P for inputs of length n over an alphabet H and output space Y is an acyclic directed graph (V, E) with a designated "source" vertex s and a positive number of sink vertices $t_1, \ldots, t_\ell$ such that:*

- *Each non-sink vertex v is labeled with an index* $\mathsf{Var}(v) \in [n]$*, and has* $|H|$ *outgoing edges* $\{v \to \mathsf{Succ}_h(v)\}_{h \in H}$.
- *Each sink vertex t is labeled with an output value* $y_t \in Y$.

The size $|P|$ *of P is the number of vertices. The branching program is said to be* read-once *if for every path from source to a sink, the vertices on that path are labeled distinctly.*

A read-once branching program is oblivious if it is a layered graph with $n+1$ layers and all the vertices v within layer $i \in [n]$ are labeled with the same index, which we simply denote by $\mathsf{Var}(i)$. All vertices in the $(n+1)^{th}$ layer are sinks. The width of an oblivious read-once branching program is the maximal number of vertices of any layer in the graph.

Evaluating a Branching Program. Given any string $x \in H^n$, we can iterate the following simple rule to map any vertex v to an output in Y, which we will denote by $P_v(x)$.

If at a non-sink vertex v, move to $\mathsf{Succ}_{x_{\mathsf{Var}(v)}}(v)$*. If at a sink vertex t, output* y_t.

By mapping x to y_t, the branching program P can thus be thought of as a function $P = P_s : H^n \to Y$.

Uniformity of Branching Programs. A function family $\{f_\alpha\}_{\alpha \in \{0,1\}^*}$ is computable by a (T, S)-uniform read-once branching program, if there exists a standard word RAM machine that on input $x \in \{0,1\}^*$ runs in time $T(x)$ and space $S(x)$ and outputs a read-once branching program P such that $P(x) = f_\alpha(x)$.

5.1.1 Low Degree Extension of Read-Once Branching Programs

The following result shows how to efficiently evaluate the low degree extension of functions computable by a read-once branching program.

Theorem 5.2. *Given any read-once branching program $B : H^n \to Y$ and any field $\mathbb{F}$ that contains H and Y, it is possible to evaluate the low-degree extension $\hat{B} : \mathbb{F}^n \to \mathbb{F}$ in time dominated by $O(|B| \cdot |H|^2)$ field operations.*

Moreover, viewing $\hat{B}$ as a function family indexed by B and $\mathbb{F}$, it holds that $\hat{B}$ is computable by a $\mathsf{TIME}(|B| \cdot |H|^2)$*-uniform* ASLP*s of length $O(|B| \cdot |H|)$ and of multiplicative depth $O(n \cdot \log(|H|))$.*

Proof. By Fact 4.5, it suffices to show only the moreover clause. Without loss of generality, we assume that the vertex set V is $[S]$.

1. For each vertex $v \in V$, in reverse topological order (so that we visit sink vertices first and source vertices last):
   (a) If v is a sink vertex, output the instruction $R[v] \leftarrow y_v$.
   (b) Otherwise, output a sequence of $O(|H|)$ instructions that, loosely speaking, computes

$$R[v] \leftarrow \sum_{h \in H} \hat{\chi}_h(\mathbb{z}_{\mathsf{Var}(v)}) \cdot R[\mathsf{Succ}_h(v)],$$

where $\hat{\chi}_h : \mathbb{F} \to \mathbb{F}$ is the Lagrange interpolation polynomial for h:

$$\hat{\chi}_h(\mathbb{z}) = \prod_{h' \in H \setminus \{h\}} \frac{\mathbb{z} - h'}{h - h'}.$$

Such a sequence of instructions can be constructed in terms of the ASLP of Claim 5.2.3.

2. Output the instruction OUTPUT $\leftarrow R[s]$.

To show correctness, we must show that the algorithm computes $\hat{f}$ correctly on H^n, and that the algorithm computes a polynomial of degree $|H|-1$ in each of $\mathbb{z}_1, \ldots, \mathbb{z}_n$. Let $d(v)$ denote the "depth" of a vertex v: specifically, the length of the longest directed path from v to a sink vertex.

Claim 5.2.1. *If $\bar{\mathbf{z}} \in H^n$, then the above algorithm on input $\bar{\mathbf{z}}$ outputs $f(\mathbf{z}_1, \ldots, \mathbf{z}_n)$.*

Proof. We show that for every vertex v, the v^{th} iteration of the loop on line 1 assigns the $P_v(\mathbf{z}_1, \ldots, \mathbf{z}_n)$ to $R[v]$. This is by induction on $d(v)$. If $d(v) = 0$, then v is a sink, and $R[v]$ is assigned y_v, which is $P_v(\mathbf{z}_1, \ldots, \mathbf{z}_n)$ by definition.

If $d(v) > 0$, then $R[v]$ is assigned $\sum_{h \in H} \hat{\chi}_h(\mathbf{z}_{\mathsf{Var}(v)}) \cdot R[\mathsf{Succ}_h(v)]$. But $\mathbf{z}_{\mathsf{Var}(v)} \in H$, so this is simply equal to $R[\mathsf{Succ}_{\mathbf{z}_{\mathsf{Var}(v)}}(v)]$, which by the inductive hypothesis is $P_{\mathsf{Succ}_{\mathbf{z}_{\mathsf{Var}(v)}}(v)}(x)$. □

Claim 5.2.2. *The output of the algorithm on input $\mathbb{z}_1, \ldots, \mathbb{z}_n \in \mathbb{F}$ is degree $|H|-1$ in $\mathbb{z}_1, \ldots, \mathbb{z}_n$.*

Proof. We show that for every vertex v, $A(v)$ has degree $|H|-1$ in each variable. This is by induction on $d(v)$. If $d(v) = 0$, then v is a sink vertex and $A(v)$ is a constant independent of $\mathbb{z}_1, \ldots, \mathbb{z}_n$, i.e. has degree 0.

If $d(v) > 0$, then by definition $A(v) = \sum_{h \in H} \hat{\chi}_h(\mathbb{z}_{\mathsf{Var}(v)}) \cdot A(\mathsf{Succ}_h(v))$. For every h, $d(\mathsf{Succ}_h(v)) < d(v)$, so by the inductive hypothesis $A(\mathsf{Succ}_h(v))$ has degree $|H|-1$ in each of $\mathbb{z}_1, \ldots, \mathbb{z}_n$. Furthermore, because the branching program is read-once, $A(\mathsf{Succ}_h(v))$ is independent of $\mathbb{z}_{\mathsf{Var}(v)}$ (i.e. it has degree 0 in $\mathbb{z}_{\mathsf{Var}(v)}$). Since $\hat{\chi}_h$ has degree $|H|-1$, it holds that $A(v)$ has degree $|H|-1$ in each of $\mathbb{z}_1, \ldots, \mathbb{z}_n$. □

Finally, we must show that Line 1b can in fact be implemented with $O(|H| \cdot S)$ instructions. This follows from the following claim.

Claim 5.2.3 (Linear-Size ASLPs for Batch Computing Lagrange Interpolation Polynomials)**.** *There is an algorithm that takes as input a description of a field $\mathbb{F}$, as well as a subset $H = \{h_1, \ldots, h_{|H|}\} \subseteq \mathbb{F}$, and in time $O(|H|^2)$ outputs an* ASLP *of length $O(|H|)$ (and therefore multiplicative depth $O(|H|)$) that on input $\mathbb{z} \in \mathbb{F}$ outputs $\big(\hat{\chi}_{h_1}(\mathbb{z}), \ldots, \hat{\chi}_{h_{|H|}}(\mathbb{z})\big)$, where*

$$\hat{\chi}_{h_i}(\mathbb{z}) = \prod_{j \neq i} \frac{\mathbb{z} - h_j}{h_i - h_j}.$$

Proof. The algorithm does the following:

1. Output $O(|H|)$ instructions that jointly compute all "prefix products" $\left\{\prod_{j=1}^{i-1}(\mathbb{z}-h_j)\right\}_{i\in[|H|]}$, storing the i^{th} prefix product in register $R[i]$. Specifically, one such sequence of instructions is

$$\begin{aligned}
&R[1] \leftarrow 1\\
&(R[i] \leftarrow h_{i-1})_{i\in\{2,\dots,|H|\}}\\
&R\big[|H|+1\big] \leftarrow \texttt{INPUT}[1]\\
&\Big(R[i] \leftarrow R\big[|H|+1\big] - R[i]\Big)_{i\in\{2,\dots,|H|\}}\\
&\big(R[i] \leftarrow R[i] \times R[i-1]\big)_{i\in\{2,\dots,|H|\}}
\end{aligned}$$

2. Similarly, output $O(|H|)$ instructions that jointly compute all "suffix products" $\left\{\prod_{j=i+1}^{|H|}(\mathbb{z}-h_j)\right\}_{i\in[|H|]}$, storing the i^{th} suffix product in register $R\big[|H|+i\big]$.

3. Finally, output $O(|H|)$ instructions that jointly compute and store $\hat{\chi}_{h_i}(\mathbb{z})$ in register $R\big[2|H|+i\big]$ for every $i \in \big[|H|\big]$ by computing $R\big[2|H|+i\big] \leftarrow R[i]\cdot R\big[|H|+i\big]\cdot\prod_{j\neq i}\frac{1}{h_i-h_j}$. Specifically, one sequence of instructions that accomplishes this is

$$\begin{aligned}
&\left(R\big[2|H|+i\big] \leftarrow \prod_{j\neq i}\frac{1}{h_i-h_j}\right)_{i\in[|H|]} \qquad (14)\\
&\big(R\big[2|H|+i\big] \leftarrow R\big[2|H|+i\big] \times R[i]\big)_{i\in[|H|]}\\
&\big(R\big[2|H|+i\big] \leftarrow R\big[2|H|+i\big] \times R\big[|H|+i\big]\big)_{i\in[|H|]}
\end{aligned}$$

□

Remark 5.3. *We note that the complexity of* computing *the $O(|H|)$ instructions of Eq. (14) is $O(|H|^2)$. For certain choices of H and $\mathbb{F}$ it is possible to compute this sequence of instructions more efficiently, namely in time equivalent to $O(|H|)$ operations over $\mathbb{F}$. In particular, if H is of the form $\{h_i \overset{\text{def}}{=} h_0 + i\cdot\delta\}_{i\in\{1,\dots,|H|\}}$ for some $h_0\in\mathbb{F}$, $\delta\in\mathbb{F}\setminus\{0\}$, then $\prod_{i=1}^{k}(h_k-h_i)=\delta^k\cdot k!$, which is computable in $O(1)$ operations given $\prod_{i=1}^{k-1}(h_{k-1}-h_i)=\delta^{k-1}\cdot(k-1)!$. For a given field $\mathbb{F}$, such sets H exist of all sizes up to and including the characteristic of $\mathbb{F}$.*

This concludes the proof of Theorem 5.2. □

In the case of *oblivious* read-once branching programs, it is possible to achieve much better multiplicative depth.

Theorem 5.4. *Viewing $\hat{B}$ as a function family indexed by B and $\mathbb{F}$, it holds that $\hat{B}$ is computable by a $\mathsf{TIME}\Big(O\big(n\cdot(w^3+|H|^2)\big)\Big)$-uniform ASLPs of length $O\big(n\cdot(w^3+|H|)\big)$ and multiplicative depth $\lceil\log(n)\rceil+|H|-2$.*

Proof. Let B be a width w oblivious read-once branching program. We assume without loss of generality that all the layers of B have width w. Let $s\in[w]$ denote the index of the starting vertex of B within layer 1. Let $T = [w]$ denote the indices of all the sink vertices within layer $n+1$ (recall that we assumed that all the vertices within layer $n+1$ are sinks).

For every $i \in [n]$ let $B_i : H \to \{0,1\}^{w \times w}$ be a function that on input $h \in H$ outputs a matrix $M_{i,h} \in \{0,1\}^{w \times w}$ that is the adjacency matrix corresponding to a transition of the branching program from layer i to layer $i+1$ when the $\mathsf{Var}(i)^{th}$ input symbol is h. More precisely, the $(j,k)^{th}$ entry of $M_{i,h}$ is 1 if there is an h-labeled edge from the j^{th} vertex in layer i to the k^{th} vertex in layer $i+1$; otherwise, the entry is 0.

Consider the function $f : \mathbb{F}^n \to \mathbb{F}$ defined as:

$$f(\mathbb{z}) = \sum_{t \in [w]} \left(\prod_{i \in [n]} \left(\sum_{h \in H} \chi_h(\mathbb{z}_i) \cdot B_i(h) \right) \right)_{s,t} \cdot y_t,$$

where by $A_{(s,t)}$ we refer to the $(s,t)^{th}$ entry of the matrix A.

By inspection, for every $\mathbf{z} \in H^n$, the function f evaluates the branching program on $\mathbf{z}$. On the other hand, f is an individual degree $|H|-1$ polynomial. Thus, by the uniqueness of the low degree extension, $f \equiv \hat{B}$.

The uniform ASLP for evaluating f proceeds as follows. For every sink $t \in [w]$, the expression $\left(\prod_{i \in [n]} \left(\sum_{h \in H} \chi_h(\mathbb{z}_i) \cdot B_i(h) \right) \right)_{s,t} \cdot y_t$ is computed by a balanced binary multiplication tree of depth $\lceil \log(n) \rceil$. Leaves of this tree correspond to a computation of $\sum_{h \in H} \chi_h(\mathbb{z}_i) \cdot B_i(h)$. This expression can be computed by a $\mathsf{TIME}(O(|H|^2))$-uniform ASLP of length $O(|H|)$ and multiplicative depth $|H|-2$. Finally, all the results (corresponding to different values of $t \in [w]$ are summed.

□

Remark 5.5. *The w^3 term in Theorem 5.4 can be improved to w^ω, where ω is the matrix multiplication exponent. However, we do not try to optimize the polynomial dependence on w in this work, since we will only use Theorem 5.4 with $w = O(1)$.*

5.2 Caching

We recall some basic notions and results from caching theory. A cache is a dictionary (i.e., it supports standard GET and PUT operations), with a bound τ on the number of mappings it can store. PUT operations are allowed to delete a mapping to avoid exceeding this bound. The main performance objective of a cache is to minimize the number of failed GET operations that result from these deletions.

There are many different types of caches corresponding to different replacement policies – how it chooses which mapping to delete. Two particularly important types of caches are the *ideal cache* (which is easily analyzed), and the *least recently used (LRU) cache*, which is efficiently implementable.

An *ideal* cache behaves optimally and clairvoyantly with respect to future GET operations.

Definition 5.6. *An access pattern $(a_1, \ldots, a_T)$ is said to* incur Q misses *on an* ideal cache of size τ *(written $Q = Q_{\mathsf{ideal}}((a_1, \ldots, a_T), \tau)$) if Q is the minimum value of $\sum_{t=1}^{T} \left| A_t \setminus A_{t-1} \right|$, where the minimum is taken over sets*[34] *$A_0, \ldots, A_T$ satisfying the following constraints for each t.*

[34] A_t represents the set of keys mapped by the cache at time t

- $A_0 = \emptyset$ *(the cache starts empty), and*
- $|A_t| \leq \tau$ *(the cache can store at most τ entries), and*
- $a_t \in A_t$ *(the cache must have a_t at time t)*

We state without proof several useful facts about how (ideal) cache misses behave under composition.

Definition 5.7. *An α-blowup of an access pattern $(a_1, \ldots, a_T)$ is another access pattern $(a'_1, \ldots, a'_{T'})$ that is obtained by replacing each a_i by a (possibly empty) sequence of addresses, each of which lies in a set A_{a_i} whose size is at most α. We emphasize that A_{a_i} depends only on the value a_i, and not directly on i.*

Proposition 5.8. *If an access pattern $(a_1, \ldots, a_T)$ incurs Q misses on an ideal cache of size τ, then any α-blowup of $(a_1, \ldots, a_T)$ incurs at most $\alpha \cdot Q$ misses on an ideal cache of size $\alpha \cdot \tau$.*

An LRU cache is an "online" cache – i.e., the set A_t of mappings stored at time t is determined by $a_1, \ldots, a_{t-1}$. In particular, A_t is obtained from A_{t-1} by adding a_t and, if necessary to ensure $|A_t| \leq \tau$, removing the mapping which was least recently accessed. This is efficiently implementable with a doubly linked list and a hash table.

Definition 5.9. *An access pattern $(a_1, \ldots, a_T)$ is said to incur Q misses on an LRU cache of size τ if $Q = \sum_{t=1}^{T} \left| A_t \setminus A_{t-1} \right|$ (written $Q = Q_{\mathsf{LRU}}((a_1, \ldots, a_T), \tau)$), where $A_0, \ldots, A_T$ are defined as*

- $A_0 = \emptyset$.
- *For $t > 0$,*

$$A_t = \begin{cases} A_{t-1} & \text{if } a_t \in A_{t-1} \\ A_{t-1} \cup \{a_t\} & \text{if } a_t \notin A_{t-1} \text{ and } |A_{t-1}| < \tau \\ (A_{t-1} \cup \{a_t\}) \setminus \{\arg\min_{a \in A_{t-1}} \max\{i : i < t \wedge a_i = a\}\} & \text{otherwise.} \end{cases}$$

A theorem of Sleator and Tarjan [ST85] shows that an LRU cache is nearly as good as a mythical ideal cache.

Theorem 5.10 ([ST85, Theorem 6])**.** *If an access pattern $(a_1, \ldots, a_T)$ incurs Q misses on an ideal cache of size τ, then it incurs at most $2Q$ misses on an LRU cache of size 2τ.*

Definition 5.11 (Cache Block Size)**.** *We say that an access pattern $(a_1, \ldots, a_T)$ incurs Q misses on an ideal (respectively, LRU) cache of size τ and* block size $\mathcal{B}$ *if $\left(\lfloor \frac{a_1}{\mathcal{B}} \rfloor, \ldots, \lfloor \frac{a_T}{\mathcal{B}} \rfloor\right)$ incurs Q misses on an ideal (respectively, LRU) cache of size τ.*

When using a cache for bounded memoization, it is convenient to define the following "get or compute" operation.

Algorithm 1 Looks up key k in cache, or computes $f(k)$

```
1: procedure GetOrCompute(cache, $k$, $f$)
2:     if cache.Get($k$) = $\perp$ then
3:         cache.Put($k$, $f(k)$)
4:     return cache.Get($k$)
5: end procedure
```

We end this section with a technical definition that will be useful for our theorem statements. This definition refers to the "cache friendliness" of a RAM program M on a given input x. Loosely speaking, this quantity corresponds to the number of cache misses during the evaluation of M on x. The actual definition refers also to prefixes of memory addresses.

Definition 5.12 (Cache Friendliness)**.** *For $x \in \{0,1\}^*$ and RAM program M, let $(a_1, \dots, a_{O(T)})$ denote the combined access pattern (see Definition 4.16) of M on input x. For any $\tau \in \mathbb{N}$, we define the τ-cache friendliness of M on x, denoted $\mathsf{CF}_\tau(x, M)$, as the maximum, over $r \in [s]$, the number of misses incurred by $(\textsc{Prefix}_r(a_1), \dots, \textsc{Prefix}_r(a_{O(T)}))$ on an ideal cache of size τ, divided by 2^r.*

Proposition 5.13. *For any RAM machine M, input x and cache size $\tau \leq O(T)$ it holds that $\mathsf{CF}_\tau(x, M) \leq O(T/\tau)$.*

Proof. For r such that $2^r \leq \tau$, the prefixes of all memory addresses fit in the cache and so the number of cache misses is at most 2^r (just for filling up the cache to begin with). For r such that $2^r > \tau$, we can use the trivial bound of $O(T)$ on the number of cache misses.

Overall, we get that $\mathsf{CF}_\tau(M, x) \leq \max\left(\frac{2^r}{2^r}, \frac{O(T)}{\tau}\right) = O(T/\tau)$. □

For *Turing machines* computations, we can achieve a better bound.

Proposition 5.14 (On Cache Friendliness of Turing Machines)**.** *For a (multi-tape) Turing machine M and any input x, it holds that $\mathsf{CF}_\tau(x, M) = O(T/2^s)$, with $\tau = O(1)$.*

Proof Sketch. Consider first the case of a single-tape Turing machine. Let $(a_1, \dots, a_{O(T)}$ be the combined access pattern of M on x and fix $r \in [s]$. Consider an LRU cache of size 2.

Due to the Turing machine sequential access pattern, after any given cache miss, it takes at least 2^{s-r} steps before making another cache miss. Thus, the number of cache misses is at most $O(T/2^{s-r})$ and the cache friendliness is therefore $\mathsf{CF}_2(x, M) = O(T/2^s)$.

This argument extends easily to k-tape Turing machines, using a cache of size $2k$. □

6 Our Main Results

Let $\mathcal{L}$ be any language that is decidable in time $T(n)$ and space $S(n)$ by a standard w-bit word RAM machine M, where we assume that:

- $\max(\log(n), \log(S)) \leq w \leq O(\log(T))$,
- $n \leq T \leq 2^{O(n)}$, and
- $\log(T) \leq S \leq T$.

In this section we state our main results: the existence of efficient PCPs with no-signaling soundness and argument-systems for $\mathcal{L}$.

Recall that $\mathsf{CF}_\tau(M, x)$ is the "cache-friendliness" of the RAM machine M on input x, given a cache of size τ (see Definition 5.12).

6.1 Efficient PCPs

We first state a very general theorem that shows the explicit dependence of the running time of the prover on a cache size and "cache friendliness" of the program. Later, we provide corollaries for the most interesting setting of parameters.

Recall that $\mathsf{CF}_\tau(M, x)$ is the "cache-friendliness" of the RAM machine M on input x, given a cache of size τ (see Definition 5.12).

Theorem 6.1. *Let λ be a security parameter and let $\mathbb{F}$ be an explicit field ensemble of size $|\mathbb{F}| = \Theta(\log(T))$. There is a q-query* PCP *for $\mathcal{L}$, over the alphabet $\mathbb{F}$, with soundness error ε against $(k_{\max}, \delta)$-*NS *strategies, where $q = \lambda \cdot \mathrm{polylog}(T)$, $\varepsilon = 2^{-\lambda}$, $\delta = 2^{-\lambda \cdot \mathrm{polylog}(T)}$ and $k_{\max} = \lambda \cdot \mathrm{polylog}(T)$. The* PCP *has the following efficiency properties:*

1. *The* PCP *verifier can be implemented in time $n \cdot \lambda \cdot \mathrm{polylog}(T)$ and space $\lambda \cdot \mathrm{polylog}(T)$.*

2. *For any cache size τ, any symbol in the* PCP *proof string can be evaluated in time $\big(T + \max(S, n) \cdot \mathsf{CF}_\tau(M, x)\big) \cdot \mathrm{polylog}(T)$ and space $S + \tau \cdot \mathrm{polylog}(T)$.*

   Moreover, any symbol in the PCP *proof string can be evaluated by a $\mathsf{TISP}\Big(\big(T + \max(S, n) \cdot \mathsf{CF}_\tau(M, x)\big) \cdot \mathrm{polylog}(T), S + \tau \cdot \mathrm{polylog}(T)\Big)$-uniform* ASLP *over $\mathbb{F}$ with $\tau \cdot \mathrm{polylog}(T)$ registers and multiplicative depth $\log\log(T) + O(1)$.*

3. *The communication complexity with each prover is $\mathrm{polylog}(T)$.*

Sections 7 to 11 are devoted to the proof of Theorem 6.1. Using Theorem 6.1 we can derive corollaries corresponding to the most interesting setting of parameters. First, for any $\zeta > \frac{\mathrm{polylog}(T)}{S}$ we can set the cache size to be $\tau = \frac{\zeta \cdot S}{\mathrm{polylog}(T)}$. Using Proposition 5.13 we get a bound on the cache friendliness of *any* RAM program and so we can derive the following corollary (which for simplicity of presentation is restricted to the setting of sublinear space $S \geq n$):

Corollary 6.2. *Suppose that $S \geq n$ and let $\zeta > \frac{\mathrm{polylog}(T)}{S}$. Then, every symbol in the* PCP *proof-string of Theorem 6.1 can be evaluated in time $\tilde{O}(T) \cdot (1/\zeta)$ and space $(1 + \zeta) \cdot S$.*

Moreover, every symbol in the PCP *proof string can be evaluated by a $\mathsf{TISP}\big(\tilde{O}(T) \cdot (1/\zeta), (1 + \zeta) \cdot S\big)$-uniform* ASLP *with $\zeta \cdot S$ registers and multiplicative depth $\log\log(T) + O(1)$.*

In particular, setting $\zeta = 1/\mathrm{polylog}(T)$ (and assuming that $S \geq (\log(T))^c$ for a sufficiently large constant $c \geq 1$) we obtain that every symbol in the PCP proof-string can be computed in time $\tilde{O}(T)$ and space $S + o(S)$.

Alternatively, for programs that are "cache friendly" (e.g., Turing machines - see Proposition 5.14), we derive the following improved efficiency:

Corollary 6.3. *Suppose M has cache friendliness $\mathsf{CF}_\tau(M, x) \leq \frac{T \cdot \mathrm{polylog}(T)}{\max(S,n)}$, for $\tau = \mathrm{polylog}(T)$, and every $x \in \{0, 1\}^n$. Then, every point in the* PCP *proof-string of Theorem 6.1 can be evaluated in time $\tilde{O}(T)$ and space $S + \mathrm{polylog}(T)$.*

Moreover, every symbol in the PCP *proof string can be evaluated by a $\mathsf{TISP}\big(\tilde{O}(T), S + \mathrm{polylog}(T)\big)$-uniform* ASLP *with $\mathrm{polylog}(T)$ registers and multiplicative depth $\log\log(T) + O(1)$.*

6.2 Argument-Systems

Applying the transformation of [KRR13], from no-signaling PCPs to argument-schemes, to the PCPs of Theorem 6.1 and Corollaries 6.2 and 6.3 we derive our main results on the existence of efficient 2-message argument-systems.

Theorem 6.4. *Assume the existence of a sub-exponential secure homomorphic encryption scheme that supports evaluation of arithmetic circuits of size* $\big(T + \max(S,n) \cdot \mathsf{CF}_\tau(M,x)\big) \cdot \mathrm{polylog}(T)$ *and multiplicative depth* $\log\log(T) + O(1)$*. Then, there exists a 2-message argument scheme for* $\mathcal{L}$ *with the following efficiency properties:*

1. *The verifier of the argument-system runs in time* $n \cdot \mathrm{polylog}(T)$ *and space* $\mathrm{polylog}(T)$*.*

2. *The prover runs in time* $\big(T+\max(S,n)\cdot\mathsf{CF}_\tau(M,x)\big)\cdot\mathrm{polylog}(T)$ *and space* $S+\tau\cdot\mathrm{polylog}(T)$*.*

3. *The communication complexity is* $\mathrm{polylog}(T)$*.*

Theorem 6.4 is proved in Section 11.

In analogy to Corollary 6.2 we can derive a result assuming worst-case cache behavior of the program as follows.

Corollary 6.5. *Suppose that* $S \geq n$ *and let* $\zeta > \frac{\mathrm{polylog}(T)}{S}$*.*

Assume the existence of a sub-exponential secure homomorphic encryption scheme that supports evaluation of arithmetic circuits of size $\tilde{O}(T)\cdot(1/\zeta)$ *and multiplicative depth* $\log\log(T)+O(1)$*. Then, the prover in Theorem 6.4 can be implemented in time* $\tilde{O}(T)\cdot(1/\zeta)$ *and space* $(1+\zeta)\cdot S$*.*

In contrast, if we assume the program is cache friendly, in analogy to Corollary 6.3 we can derive a more efficient argument-system.

Corollary 6.6. *Suppose* M *has cache friendliness* $\mathsf{CF}_\tau(M,x) \leq \frac{T\cdot\mathrm{polylog}(T)}{\max(S,n)}$*, for* $\tau = \mathrm{polylog}(T)$*, and every* $x \in \{0,1\}^n$*.*

Assume the existence of a sub-exponential secure homomorphic encryption scheme that supports evaluation of arithmetic circuits of size $\tilde{O}(T)$ *and multiplicative depth* $\log\log(T)+O(1)$*. Then, the prover in Theorem 6.4 can be implemented in time* $\tilde{O}(T)$ *and space* $S+\mathrm{polylog}(T)$*.*

7 The Construction

Let M be a standard word RAM machine that on an input of length n runs in time $T = T(n)$ and space $S = S(n)$ with a word size $w = w(n)$. As in Section 4.4, let $W = \mathsf{poly}(w)$ be the maximal complexity of implementing any (standard) word operation by a single-tape Turing machine on w-bit inputs, let $S' = \max(n,S)$, and let $\mathsf{Tree}(M)$ be the tree machine from Construction 4.32 that simulates M in time $T' = \Theta(T\cdot W^2\cdot\log S')$. Throughout this section, we think of configurations of $\mathsf{Tree}(M)$ as represented by functions $\mathcal{C} : \{0,1\}^{\lceil\log S'\rceil}\times\{0,1\}^{\lceil\log W\rceil}\to\bar{\Gamma}$ as in Definition 4.31. For convenience of notation we define $s \overset{\mathsf{def}}{=} \lceil\log S'\rceil + \lceil\log W\rceil$ and $t \overset{\mathsf{def}}{=} \lceil\log T'\rceil$.

We now construct a PCP for the language accepted by M. The PCP is parameterized by a finite field ensemble $\mathbb{F} = \{\mathbb{F}(n)\}_{n\in\mathbb{N}}$, of size $|\mathbb{F}(n)| \geq |\bar{\Gamma}|$, and for each n we require that the description of $\mathbb{F}(n)$ includes a canonical association of $\bar{\Gamma}$ with a subset of $|\mathbb{F}(n)|$. We will describe the functionality of the prover and verifier, and defer analysis of their complexities to Sections 8 and 9. In Section 10 we show that this construction, with appropriate parameters (e.g., the field size), has no-signaling soundness.

Notations and Conventions. Recall that we use the convention that blackboard bold lowercase (e.g., $\mathbb{z}$) is used for field elements whereas standard bold lowercase (e.g., $\mathbf{z}$) is used for bits. Likewise, we use $\bar{\mathbb{z}}$ to denote vectors of field elements and $\bar{\mathbf{z}}$ to denote bit strings. For two vectors $\bar{u} = (u_1, \ldots, u_k)$ and $\bar{v} = (v_1, \ldots, v_k)$, recall that we write $\bar{u} \star \bar{v}$ to denote the vector $(u_1, v_1, \ldots, u_k, v_k)$.

7.1 Preliminaries: A Locally-Checkable Transcript for Tree Machines

In this section we show that the correctness of a Tree machine computation can be expressed via local constraints. This is analogous to the Cook-Levin theorem, which shows that Turing machine computations can be expressed as a combination of local constraints. As is typical in the PCP literature, later on, we will make these constraints "robust" using algebraic techniques.

For every "direction" $d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}$ we define a function

$$\phi_d : \{0,1\}^{2s} \to \{0,1\}$$

that takes as input a pair of head locations $\bar{\mathbf{u}} \star \bar{\mathbf{u}}'$ of the Tree machine $\mathsf{Tree}(M)$ and returns 1 if a single step from $\bar{\mathbf{u}}$ in direction d reaches $\bar{\mathbf{u}}'$ and 0 otherwise. In other words,

$$\phi_d^{-1}(1) = \begin{cases} \{(\bar{\mathbf{v}}, \bar{\mathbf{z}}) \star (\lfloor \bar{\mathbf{v}}/2 \rfloor, \bar{\mathbf{z}})\} & \text{if } d = \uparrow \\ \{(\bar{\mathbf{v}}, \bar{\mathbf{z}}) \star (2\bar{\mathbf{v}}, \bar{\mathbf{z}})\} & \text{if } d = \swarrow \\ \{(\bar{\mathbf{v}}, \bar{\mathbf{z}}) \star (2\bar{\mathbf{v}}+1, \bar{\mathbf{z}})\} & \text{if } d = \searrow \\ \{(\bar{\mathbf{v}}, \bar{\mathbf{z}}) \star (\bar{\mathbf{v}}, \bar{\mathbf{z}}+1)\} & \text{if } d = \otimes \\ \{(\bar{\mathbf{v}}, \bar{\mathbf{z}}) \star (\bar{\mathbf{v}}, \bar{\mathbf{z}}-1)\} & \text{if } d = \odot \\ \{(\bar{\mathbf{v}}, \bar{\mathbf{z}}) \star (\bar{\mathbf{v}}, \bar{\mathbf{z}})\} & \text{if } d = \emptyset. \end{cases}$$

Remark 7.1. *The reader might wonder why the format of the input to ϕ_d is $(\bar{\mathbb{u}} \star \bar{\mathbb{u}}')$ rather than simply $(\bar{\mathbb{u}}, \bar{\mathbb{u}}')$ (recall that in the former expression, the coordinates of $\bar{\mathbb{u}}$ and $\bar{\mathbb{u}}'$ are interleaved). The reason for this convention (which will also be used extensively below) will become more clear later, but jumping ahead, we remark that it is made for notational convenience regarding the order in which variables are "eliminated" in the so-called sumcheck polynomials.*

Proposition 7.2. *There exist functions $\left\{V_d : \bar{\Gamma}^2 \to \{0,1\}\right\}_{d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}}$ such that for any pair of configurations $\mathcal{C}, \mathcal{C}' : \{0,1\}^s \to \bar{\Gamma}$, we have $\mathcal{C} \to \mathcal{C}'$ iff for every $\bar{\mathbf{u}}'$ in $\{0,1\}^s$, it holds that $\mathcal{C}'(\bar{\mathbf{u}}')$ is the (unique) value in $\bar{\Gamma}$ such that for every $d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}$ and $\bar{\mathbf{u}} \in \{0,1\}^s$:*

$$\phi_d(\bar{\mathbf{u}} \star \bar{\mathbf{u}}') \cdot V_d\Big(\mathcal{C}(\bar{\mathbf{u}}), \mathcal{C}'(\bar{\mathbf{u}}')\Big) = 0.$$

Proof. This follows from the following decomposition of what it means for a configuration $\mathcal{C}$ to yield another configuration $\mathcal{C}'$.

1. (Memory Consistency) For every $(\bar{\mathbf{v}}, \bar{\mathbf{z}})$, let $(q, \gamma, e) = \mathcal{C}(\bar{\mathbf{v}}, \bar{\mathbf{z}})$ and let $(q', \gamma', e') = \mathcal{C}'(\bar{\mathbf{v}}, \bar{\mathbf{z}})$.
    - If $q = \perp$ or $q \in \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$, then $\gamma' = \gamma$.
    - Otherwise, $\delta(q, \gamma) = (\tilde{q}, \gamma', \widetilde{\mathsf{dir}})$ for some $\tilde{q}$ and some $\widetilde{\mathsf{dir}}$.
2. (Control State Consistency) For every $(\bar{\mathbf{v}}, \bar{\mathbf{z}})$ and $(\bar{\mathbf{v}}', \bar{\mathbf{z}}')$, let $(q, \gamma, e) = \mathcal{C}(\bar{\mathbf{v}}, \bar{\mathbf{z}})$ and $(q', \gamma', e') = \mathcal{C}'(\bar{\mathbf{v}}', \bar{\mathbf{z}}')$.

- If $q = \bot$ and $q' \neq \bot$, then
  - If $\bar{\mathbf{v}}' = \lfloor \bar{\mathbf{v}}/2 \rfloor$ and $\bar{\mathbf{z}}' = \bar{\mathbf{z}}$, then $\mathsf{dir}(q') \neq \uparrow$.
  - If $\bar{\mathbf{v}}' = 2\bar{\mathbf{v}}$ and $\bar{\mathbf{z}}' = \bar{\mathbf{z}}$ then $\mathsf{dir}(q') \neq \swarrow$.
  - If $\bar{\mathbf{v}}' = 2\bar{\mathbf{v}} + 1$ and $\bar{\mathbf{z}}' = \bar{\mathbf{z}}$ then $\mathsf{dir}(q') \neq \searrow$.
  - If $\bar{\mathbf{v}}' = \bar{\mathbf{v}}$ and $\bar{\mathbf{z}}' = \bar{\mathbf{z}} + 1$ then $\mathsf{dir}(q') \neq \otimes$.
  - If $\bar{\mathbf{v}}' = \bar{\mathbf{v}}$ and $\bar{\mathbf{z}}' = \bar{\mathbf{z}} - 1$ then $\mathsf{dir}(q') \neq \odot$.
- If $q \in \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$, then

$$q' = \begin{cases} q & \text{if } \bar{\mathbf{v}}' = \bar{\mathbf{v}} \text{ and } \bar{\mathbf{z}}' = \bar{\mathbf{z}} \\ \bot & \text{otherwise.} \end{cases}$$

- If $q \notin \{\bot, q_{\mathsf{acc}}, q_{\mathsf{rej}}\}$ and $(\tilde{q}, \tilde{\gamma}, d) = \delta(q, \gamma)$, then $q' = \tilde{q}$ if:
  - $\bar{\mathbf{v}}' = \lfloor \bar{\mathbf{v}}/2 \rfloor$, $\bar{\mathbf{z}}' = \bar{\mathbf{z}}$, and $d = \uparrow$; or
  - $\bar{\mathbf{v}}' = 2\bar{\mathbf{v}}$, $\bar{\mathbf{z}}' = \bar{\mathbf{z}}$, and $d = \swarrow$; or
  - $\bar{\mathbf{v}}' = 2\bar{\mathbf{v}} + 1$, $\bar{\mathbf{z}}' = \bar{\mathbf{z}}$, and $d = \searrow$; or
  - $\bar{\mathbf{v}}' = \bar{\mathbf{v}}$, $\bar{\mathbf{z}}' = \bar{\mathbf{z}} + 1$, and $d = \otimes$; or
  - $\bar{\mathbf{v}}' = \bar{\mathbf{v}}$, $\bar{\mathbf{z}}' = \bar{\mathbf{z}} - 1$, and $d = \odot$

3. (Edge-case Consistency) For every $(\bar{\mathbf{v}}, \bar{\mathbf{z}})$, let $(q, \gamma, (e_1, e_2)) = \mathcal{C}(\bar{\mathbf{v}}, \bar{\mathbf{z}})$ and let $(q', \gamma', (e_1', e_2')) = \mathcal{C}'(\bar{\mathbf{v}}, \bar{\mathbf{z}})$.
   - $e_1' = e_1$ and $e_2' = e_2$.
   - If $e_1 = \mathsf{leaf}$ and $q' \neq \bot$, then $\mathsf{dir}(q') \neq \uparrow$.
   - If $e_2 = \mathsf{front}$ and $q' \neq \bot$, then $\mathsf{dir}(q') \neq \otimes$.
   - If $e_2 = \mathsf{back}$ and $q' \neq \bot$, then $\mathsf{dir}(q') \neq \odot$. □

7.2 The Prover

On input x and security parameter 1^κ, the prover P generates the string formed by writing down the truth tables of several polynomials that are defined below. As mentioned in the remarks following Lemma 11.1, the efficiency of P does not directly determine the efficiency of our argument scheme. Rather, the essential metric is the complexity of computing the i^{th} symbol of $P(x, 1^\kappa)$ given $(i, x, 1^\kappa)$. This will be analyzed in Section 9. As such, we now describe only the functionality of P (i.e., the proof string that it outputs).

Let $\mathcal{C}_0, \ldots, \mathcal{C}_{T'-1}$ denote the transcript of $\mathsf{Tree}(M)$ on input x, with each configuration $\mathcal{C}_{\mathbf{y}}$ represented as described in Definition 4.31. Extend this transcript to have length 2^t by setting $\mathcal{C}_{T'-1} = \mathcal{C}_{T'} = \cdots = \mathcal{C}_{2^t-1}$.

Motivated by Proposition 7.2, we make the following definition.

Definition 7.3. *For any $\bar{\mathbf{y}} \in [T' - 1]$, we say that $\mathcal{C}_{\bar{\mathbf{y}}}(\bar{\mathbf{u}}')$ is* determined by *$\{\mathcal{C}_{\bar{\mathbf{y}}-1}(\bar{\mathbf{u}})\}_{\bar{\mathbf{u}} \in U}$, where U is the (constant-sized) set of $\bar{\mathbf{u}}$ for which some $\phi_d(\bar{\mathbf{u}} \star \bar{\mathbf{u}}') \neq 0$.*

Define $X : \{0,1\}^t \times \{0,1\}^s \to \mathbb{F}$ such that $X(\bar{\mathbf{y}}, \cdot) = \mathcal{C}_{\bar{\mathbf{y}}}(\cdot)$, for every $\bar{\mathbf{y}} \in \{0,1\}^t$. Let $\hat{X} : \mathbb{F}^{t+s} \to \mathbb{F}$ be the multi-linear extension of X.

Let $\phi_{+1} : \{0,1\}^{2t} \to \{0,1\}$ be the indicator function for

$$\{\bar{\mathbf{y}} \star \bar{\mathbf{y}}' : \bar{\mathbf{y}}' = \bar{\mathbf{y}} + 1\}$$

and let $\hat{\phi}_{+1} : \mathbb{F}^{2t} \to \mathbb{F}$ be its multi-linear extension.

For every $d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}$, let $\hat{\phi}_d : \mathbb{F}^{2s} \to \mathbb{F}$ be the multi-linear extension of ϕ_d and let $\hat{V}_d : \mathbb{F}^2 \to \mathbb{F}$ be the low-degree extension of V_d, where ϕ_d and V_d are as in Proposition 7.2.

The PCP proof string consists of $\hat{X}$, as well as two sequences of "sum-check polynomials" $P_0^{(\mathsf{win})}, \dots, P_{t+s}^{(\mathsf{win})} : \mathbb{F}^{2t+2s} \to \mathbb{F}$ and $P_0^{(\mathsf{enc})}, \dots, P_{t+s}^{(\mathsf{enc})} : \mathbb{F}^{t+s} \to \mathbb{F}$, defined as follows.

- The polynomial $P_0^{(\mathsf{win})} : \mathbb{F}^{2t+2s} \to \mathbb{F}$ is defined as

$$P_0^{(\mathsf{win})}(\bar{\mathbb{y}} \star \bar{\mathbb{y}}', \bar{\mathbb{u}} \star \bar{\mathbb{u}}') \stackrel{\mathsf{def}}{=} \hat{\phi}_{+1}(\bar{\mathbb{y}} \star \bar{\mathbb{y}}') \cdot \sum_{d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}} \hat{\phi}_d(\bar{\mathbb{u}} \star \bar{\mathbb{u}}') \cdot \hat{V}_d\Big(\hat{X}(\bar{\mathbb{y}}, \bar{\mathbb{u}}), \hat{X}(\bar{\mathbb{y}}', \bar{\mathbb{u}}')\Big).$$

- For $1 \le i \le t+s$, the polynomial $P_i^{(\mathsf{win})} : \mathbb{F}^{2t+2s} \to \mathbb{F}$ is defined as

$$P_i^{(\mathsf{win})}(\mathbb{z}_1, \dots, \mathbb{z}_{2t+2s}) \stackrel{\mathsf{def}}{=} \sum_{z_1, \dots, z_{2i} \in \{0,1\}} P_0^{(\mathsf{win})}(z_1, \dots, z_{2i}, \mathbb{z}_{2i+1}, \dots, \mathbb{z}_{2t+2s}) \cdot \mathbb{z}_1^{z_1} \cdots \mathbb{z}_{2i}^{z_{2i}},$$

where we are treating 0 and 1 both as field elements (as input to $P_0^{(\mathsf{win})}$) and as integers (e.g., $\mathbb{z}_1^{z_1}$).

- The polynomial $P_0^{(\mathsf{enc})} : \mathbb{F}^{t+s} \to \mathbb{F}$ is defined as

$$P_0^{(\mathsf{enc})}(\bar{\mathbb{y}}, \bar{\mathbb{u}}) \stackrel{\mathsf{def}}{=} Z_{\bar{\Gamma}}(\hat{X}(\bar{\mathbb{y}}, \bar{\mathbb{u}})),$$

where $Z_{\bar{\Gamma}} : \mathbb{F} \to \mathbb{F}$ is a degree $|\bar{\Gamma}| - 1$ polynomial such that $Z_{\bar{\Gamma}}(\mathbb{z}) = 0$ iff $\mathbb{z}$ is in $\bar{\Gamma}$ (i.e., $Z_{\bar{\Gamma}}(\mathbb{z}) = \prod_{\gamma \in \bar{\Gamma}}(\mathbb{z} - \gamma)$).

- For $1 \le i \le t+s$, the polynomial $P_i^{(\mathsf{enc})} : \mathbb{F}^{t+s} \to \mathbb{F}$ is defined as

$$P_i^{(\mathsf{enc})}(\mathbb{z}_1, \dots, \mathbb{z}_{t+s}) \stackrel{\mathsf{def}}{=} \sum_{z_1, \dots, z_i \in \{0,1\}} P_0^{(\mathsf{enc})}(z_1, \dots, z_i, \mathbb{z}_{i+1}, \dots, \mathbb{z}_{t+s}) \cdot \mathbb{z}_1^{z_1} \cdots \mathbb{z}_i^{z_i}$$

(where we are again treating 0 and 1 both as field elements and integers).

7.3 The Verifier

Similarly to [KRR14], our verifier performs several tests κ times independently, where κ is the statistical security parameter, and accepts if and only if all tests pass. We denote the base verifier, which corresponds to a single set of tests, by V, and the actual κ-times repeated verifier by $V^{\otimes \kappa}$.

The tests themselves also build on [KRR14]. In particular, we will rely on some of the tests that the [KRR14] verifier makes (and later, in Section 10, we will reuse parts of [KRR14]'s soundness analysis). We first list the [KRR14] tests that we re-use in Section 7.3.1 and then state a few additional tests that we need for our analysis in Section 7.3.2.

7.3.1 Tests from [KRR14]

The following tests are based on the verifier of [KRR14]. We only use these tests in a "blackbox" manner by relying on certain results from [KRR14] (enumerated in Section 10.1.1). The base verifier V makes the following tests:

1. **Low Degree Test for $\hat{X}$:** Choose a random line $L : \mathbb{F} \to \mathbb{F}^{t+s}$. Check that $\hat{X} \circ L$ has degree $t + s$.

2. **Low Degree Test for $P_i^{(\mathsf{win})}$:** For every $i \in \{0, \ldots, t + s - 1\}$ choose a random line $L : \mathbb{F} \to \mathbb{F}^{2t+2s}$. Check that $P_i^{(\mathsf{win})} \circ L$ has degree $2|\bar{\Gamma}| \cdot (t + s)$.

3. **Sum Check for $P_i^{(\mathsf{win})}$:** For every $i \in \{1, \ldots, t + s\}$, choose at random $\mathbb{z}_1, \ldots, \mathbb{z}_{2t+2s} \in \mathbb{F}$. Check that:
$$P_i^{(\mathsf{win})}(\mathbb{z}_1, \ldots, \mathbb{z}_{2t+2s}) \overset{\text{def}}{=} \sum_{z,z' \in \{0,1\}} P_{i-1}^{(\mathsf{win})}(\mathbb{z}_1, \ldots, \mathbb{z}_{2i-1}, z, z', \mathbb{z}_{2i+2}, \ldots, \mathbb{z}_{2t+2s}) \cdot \mathbb{z}_{2i}^{z} \cdot \mathbb{z}_{2i+1}^{z'}.$$

4. **Consistency of X and $P_0^{(\mathsf{win})}$:** Choose at random $\bar{\mathbb{y}}, \bar{\mathbb{y}}' \in \mathbb{F}^t$ and $\bar{\mathbb{u}}, \bar{\mathbb{u}}' \in \mathbb{F}^s$. Check that
$$P_0^{(\mathsf{win})}(\bar{\mathbb{y}} \star \bar{\mathbb{y}}', \bar{\mathbb{u}} \star \bar{\mathbb{u}}') \overset{\text{def}}{=} \hat{\phi}_{+1}(\bar{\mathbb{y}} \star \bar{\mathbb{y}}') \cdot \sum_{d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}} \hat{\phi}_d(\bar{\mathbb{u}} \star \bar{\mathbb{u}}') \cdot \hat{V}_d\Big(\hat{X}(\bar{\mathbb{y}}, \bar{\mathbb{u}}), \hat{X}(\bar{\mathbb{y}}', \bar{\mathbb{u}}')\Big).$$

5. **Low Degree Test for $P_i^{(\mathsf{enc})}$:** For every $i \in \{0, \ldots, t + s - 1\}$ choose a random line $L : \mathbb{F} \to \mathbb{F}^{2t+2s}$. Check that $P_i^{(\mathsf{enc})} \circ L$ has degree $(|\bar{\Gamma}| - 1) \cdot (t + s)$.

6. **Sum Check for $P_i^{(\mathsf{enc})}$:** For every $i \in \{1, \ldots, t + s\}$, choose at random $\mathbb{z}_1, \ldots, \mathbb{z}_{t+s} \in \mathbb{F}$. Check that:
$$P_i^{(\mathsf{enc})}(\mathbb{z}_1, \ldots, \mathbb{z}_{t+s}) \overset{\text{def}}{=} \sum_{z \in \{0,1\}} P_{i-1}^{(\mathsf{enc})}(\mathbb{z}_1, \ldots, \mathbb{z}_{i-1}, z, \mathbb{z}_{i+1}, \ldots, \mathbb{z}_{t+s}) \cdot \mathbb{z}_i^{z}.$$

7. **Consistency of X and $P_0^{(\mathsf{enc})}$:** Choose at random $\bar{\mathbb{y}} \in \mathbb{F}^t$ and $\bar{\mathbb{u}} \in \mathbb{F}^s$. Check that
$$P_0^{(\mathsf{enc})}(\bar{\mathbb{y}}, \bar{\mathbb{u}}) \overset{\text{def}}{=} Z_{\bar{\Gamma}}(\hat{X}(\bar{\mathbb{y}}, \bar{\mathbb{u}})).$$

Remark 7.4. *The verifier in [KRR14] actually performs some additional more restricted types of low degree tests (specifically, using lines that are orthogonal and/or parallel to certain axes). As noted therein (see [KRR14, Section 5.2]) these additional tests are redundant and were presented there only for convenience.*

7.3.2 Additional Verifier Tests

In addition to the tests mentioned above, the verifier V performs the following tests:

1. **Axis-Parallel Low-Degree Test on $\hat{X}$:** For every $i \in \{1, \ldots, s\}$), sample a uniformly random $\bar{\mathbb{u}}_0 \leftarrow \mathbb{F}^{t+s}$, and define a line $L : \mathbb{F} \to \mathbb{F}^{t+s}$ as $L(\alpha) = \bar{\mathbb{u}}_0 + \alpha \cdot \bar{\mathbf{e}}_{t+i}$, where $\bar{\mathbf{e}}_j$ denotes the j^{th} standard basis vector. Check that $\hat{X} \circ L$ is a linear function (i.e., a degree 1 polynomial).

2. **Layer-Parallel Low-Degree Test on $\hat{X}$:** Sample a uniformly random $\bar{\mathbb{u}}_0 \leftarrow \mathbb{F}^{t+s}$ and $\bar{\mathbb{u}}_1 \in \{0\} \times \mathbb{F}^s$, and define a line $L : \mathbb{F} \to \mathbb{F}^{t+s}$ as $L(\alpha) = \bar{\mathbb{u}}_0 + \alpha \cdot \bar{\mathbb{u}}_1$. Check that $\hat{X} \circ L$ is a degree s polynomial.

3. **Accepting Final State Test:** Sample a uniformly random line $L : \mathbb{F} \to \mathbb{F}^{t+s}$ such that $L(0) = (2^t - 1, 0) \in \{0,1\}^t \times \{0,1\}^s$.[35] Check that $\mathsf{Interp}_{t+s}(\{\alpha \mapsto \hat{X}(L(\alpha))\}_{\alpha \neq 0})(0)$ is $(q_{\mathsf{acc}}, \gamma, e)$ for some γ and some e.

4. **Correct Initial Input Test:** Sample a uniformly random $\bar{\mathbb{u}} \leftarrow \mathbb{F}^s$, and sample a uniformly random line $L : \mathbb{F} \to \mathbb{F}^{t+s}$ such that $L(0) = (0, \bar{\mathbb{u}}) \in \{0,1\}^t \times \mathbb{F}^s$. Check that $\mathsf{Interp}_{t+s}(\{\alpha \mapsto \hat{X}(L(\alpha))\}_{\alpha \neq 0})(0) = \hat{\mathcal{C}}_0(\bar{\mathbb{u}})$.[36]

In Section 8 we show how to compute the foregoing verifier tests efficiently.

8 Verifier Efficiency

As in Section 7, let M be a standard word RAM machine that on a length-n input runs in time $T = T(n)$ and space $S = S(n)$ with a word size $w = w(n) \le O(\log(T(n)))$. Suppose for simplicity that:

- $\max(\log(n), \log S(n)) \le w(n) \le O\Big(\log\big(T(n)\big)\Big)$,
- $n \le T(n) \le 2^{O(n)}$, and
- $\log\big(T(n)\big) \le S(n) \le T(n)$.

We recall some notations that were set up in Section 7. We use $W = \mathsf{poly}(w)$ to refer to the maximal complexity of implementing any (standard) word operation by a single-tape Turing machine on w-bit inputs. We denote $S' = \max(n, S)$, and $T' = \Theta(T \cdot W^2 \cdot \log S')$. Lastly, $s \overset{\mathsf{def}}{=} \lceil \log S' \rceil + \lceil \log W \rceil$ and $t \overset{\mathsf{def}}{=} \lceil \log T' \rceil$.

In this section, we describe the efficiency of the *verifier* in our PCP construction (Section 7.2) for the language accepted by M, when instantiated with any explicit finite field ensemble $\mathbb{F}$. Let $T_{\mathbb{F}} = T_{\mathbb{F}}(n)$ denote the time complexity of performing a field operation in $\mathbb{F}$. Then, we have:

Lemma 8.1. *The verifier V can be implemented in time $O\Big(\big(n + \mathrm{polylog}(T)\big) \cdot \kappa \cdot T_{\mathbb{F}}\Big)$ and space $O\big(\kappa \cdot \mathsf{poly}(|\mathbb{F}|)\big)$.*

The only non-trivial part in the verifier's checks is evaluating the polynomials $\hat{\phi}_{+1}$, $\hat{\phi}_d$ for $d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}$, and $\hat{\mathcal{C}}_0$. Given these implementations, all additional arithmetic computations involved can be implemented in time $\kappa \cdot \mathrm{polylog}(T') \cdot T_{\mathbb{F}}$ and space $\kappa \cdot \mathsf{poly}(|\mathbb{F}|)$. Thus, our main step is to show how compute the functions $\hat{\phi}_{+1}$, $\hat{\phi}_d$ and $\hat{\mathcal{C}}_0$ efficiently. These efficient implementations rely on results established in Section 5.1.

[35] Recall that the point $(2^t - 1, 0)$ refers to the result of the Tree machine computation.

[36] Recall that $\mathcal{C}_0$ refers to the initial configuration of the Tree machine (see Definition 4.23), and $\hat{\mathcal{C}}_0$ is its multi-linear extension.

8.1 Computing $\hat{\phi}_{+1}$, $\hat{\phi}_d$ and $\hat{\mathcal{C}}_0$

The functions $\hat{\phi}_{+1}$ and $\hat{\phi}_d$ are low degree extensions of very simple indicator functions. Specifically, the following propositions show that these indicator functions can be computed by small (and uniform) read-once branching programs. In this section, especially when discussing the uniformity of our algorithms, we emphasize that ϕ_{+1} and ϕ_d should be thought of as *families* of functions that are respectively indexed by t and s. Similarly, $\hat{\phi}_{+1}$ and $\hat{\phi}_d$ should be thought of as families of functions that are respectively indexed by $(t, \mathbb{F})$ and $(s, \mathbb{F})$.

Proposition 8.2. *$\phi_{+1} : \{0,1\}^{2t} \to \{0,1\}$ is computable by a $\mathsf{TIME}\big(O(t)\big)$-uniform oblivious ROBP of width $O(1)$.*

Proposition 8.3. *For each $d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}$, $\phi_d : \{0,1\}^{2s} \to \{0,1\}$ is computable by a $\mathsf{TIME}\big(O(s)\big)$-uniform oblivious ROBP of width $O(1)$.*

Proof Sketch for Propositions 8.2 and 8.3. The propositions consist of several claims that we need to establish.

- (Computing ϕ_{+1}, $\phi_{\otimes}$, and $\phi_{\odot}$) To determine whether $\bar{\mathbf{y}}' = \bar{\mathbf{y}} + 1$, it suffices to check that they are of the form $\bar{\mathbf{y}}' = \bar{\mathbf{p}} \| 1 \| 0^c$ and $\bar{\mathbf{y}} = \bar{\mathbf{p}} \| 0 \| 1^c$ for some prefix string p and some "number of carries" c. The functions $\phi_{\otimes}$ and $\phi_{\odot}$ are equivalent to ϕ_{+1}: they simply check that $\bar{\mathbf{z}}' = \bar{\mathbf{z}} + 1$ or that $\bar{\mathbf{z}} = \bar{\mathbf{z}}' + 1$. There is a straight-forward implementation of this check by a read-once branching program.

- (Computing $\phi_{\uparrow}$) To check whether $\bar{\mathbf{v}}' = \lfloor \bar{\mathbf{v}}/2 \rfloor$, it suffices to check that the bits of $\bar{\mathbf{v}}'$ are the same as those of $\bar{\mathbf{v}}$, but shifted one place to the right (with a 0 in the leftmost position). There is a straight-forward implementation of this check by a read-once branching program.

- (Computing $\phi_{\swarrow}$ and $\phi_{\searrow}$) To check whether $\bar{\mathbf{v}}' = 2\bar{\mathbf{v}}$ (resp., $2\bar{\mathbf{v}} + 1$), it suffices to check that the bits of $\bar{\mathbf{v}}'$ are the same as those of $\bar{\mathbf{v}}$, but shifted one place to the left (with a 0 (resp., 1) in the rightmost position). There is a straight-forward implementation of this check by a read-once branching program.

- (Computing $\phi_{\emptyset}$) Simply need to check that two strings are identical. There is a straight-fcorward implementation of this check by a read-once branching program. □

Applying, Theorem 5.2 we derive the following two immediate corollaries.

Corollary 8.4. *The function $\hat{\phi}_{+1} : \mathbb{F}^{2t} \to \mathbb{F}$ is computable by a $\mathsf{TIME}\big(O(t)\big)$-uniform ASLP with multiplicative depth $\log(t) + O(1)$.*

Corollary 8.5. *For every $d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}$, the function $\hat{\phi}_d : \mathbb{F}^{2s} \to \mathbb{F}$ is computable by a $\mathsf{TIME}\big(O(s)\big)$-uniform ASLP with multiplicative depth $\log(s) + O(1)$.*

The following proposition shows that also $\hat{\mathcal{C}}_0$ is efficiently computable in both time and space. Moreover, viewing $\hat{\mathcal{C}}_0$ as a family of functions indexed by the input x and the field $\mathbb{F}$, we have

Proposition 8.6. *The function $\hat{\mathcal{C}}_0 : \mathbb{F}^s \to \mathbb{F}$, viewed as a function family indexed by $x \in \{0,1\}^n$ and a field $\mathbb{F}$, is both:*

- *Computable by a* $\mathsf{TISP}(O\big(n\cdot\text{polylog}(|\mathbb{F}|)\big), O\big(s\cdot\text{polylog}(|\mathbb{F}|)\big)$*-uniform* ASLP *with* $O(1)$ *registers and* $s + O(1)$ *multiplicative depth; and*

- *Computable by a* $\mathsf{TISP}\Big(O\big(n\cdot s\cdot\text{polylog}(|\mathbb{F}|)\big), O\big(s\cdot\text{polylog}(|\mathbb{F}|)\big)\Big)$*-uniform* ASLP *with* $O(s)$ *registers and* $\log(s) + O(1)$ *multiplicative depth.*

We remark that we use the second bound exclusively in our construction of an efficient argument scheme (**??**), where we care much more about multiplicative depth due to our reliance on homomorphic encryption.

Proof. We view $\mathcal{C}_0 : \{0,1\}^{\lceil \log S' \rceil} \times \{0,1\}^{\lceil \log W \rceil} \to \bar{\Gamma}$ as the sum (over $\mathbb{F}$) of two $\mathbb{F}$-valued functions: $\mathcal{C}_0^{(M)}$, which handles edge cases and depends only on the machine M; and $\mathcal{C}_0^{(x)}$, which depends only on the input x.

Specifically, we define

$$\mathcal{C}_0^{(M)}(\bar{\mathbf{v}}, \bar{\mathbf{z}}) = (q, \epsilon, \beta), \text{ where } q \text{ and } \beta \text{ are defined as}$$

- $q = q_0$ if $\bar{\mathbf{v}} = 0$ and $\bar{\mathbf{z}} = 0$, and $q = \perp$ otherwise.
- $\beta = (\beta_1, \beta_2)$, where $\beta_1 = \mathsf{leaf}$ if $\bar{\mathbf{v}} \geq 2^{\lceil \log S' \rceil - 1}$ and $\beta_1 = \perp$ otherwise, and $\beta_2 = \mathsf{front}$ if $\bar{\mathbf{z}} = 0$, $\beta_2 = \mathsf{back}$ if $\bar{\mathbf{z}} = W - 1$ and $\beta_2 = \perp$ otherwise.

We define

$$\mathcal{C}_0^{(x)}(\bar{\mathbf{v}}, \bar{\mathbf{z}}) = \begin{cases} (\perp, x_i, (\beta_1, \mathsf{front})) - (\perp, \epsilon, (\beta_1, \mathsf{front})) & \text{if } \bar{\mathbf{v}} = i \in [n] \text{ and } \bar{\mathbf{z}} = 0 \\ 0 & \text{otherwise} \end{cases}$$

where as before, $\beta_1 = \mathsf{leaf}$ if $\bar{\mathbf{v}} \geq 2^{\lceil \log S' \rceil - 1}$ and $\beta_1 = \perp$ otherwise.

It can be easily verified that $\mathcal{C}_0 \equiv \mathcal{C}_0^{(M)} + \mathcal{C}_0^{(x)}$ and therefore also $\hat{\mathcal{C}}_0 \equiv \hat{\mathcal{C}}_0^{(M)} + \hat{\mathcal{C}}_0^{(x)}$. It remains to establish that the low-degree extensions $\hat{\mathcal{C}}_0^{(M)}$ and $\hat{\mathcal{C}}_0^{(x)}$ are both computable in the stated time and space.

Based on our description of $\mathcal{C}_0^{(M)}$ above, there is a straight-forward implementation of $\mathcal{C}_0^{(M)}$ by a $\mathsf{TIME}\big(O\big(s \cdot \text{polylog}(|\mathbb{F}|)\big)\big)$-uniform oblivious ROBP of width $O(1)$. Therefore by Theorem 5.4, the function $\hat{\mathcal{C}}_0^{(M)}$ is computable by a $\mathsf{TIME}\big(O(s \cdot \text{polylog}(|\mathbb{F}|))\big)$-uniform ASLP with multiplicative depth $\log(s) + O(1)$.

As for evaluating $\hat{\mathcal{C}}_0^{(x)}$, first observe that $\mathcal{C}_0^{(x)}$ is sparse: $\mathcal{C}_0^{(x)}(\bar{\mathbf{v}}, \bar{\mathbf{z}}) \neq 0$ only for $\bar{\mathbf{v}} \in [n]$ and $\bar{\mathbf{z}} = 0$. Thus,

$$\hat{\mathcal{C}}_0^{(x)}(\bar{\mathbb{v}}, \bar{\mathbb{z}}) = \sum_{\bar{\mathbf{v}}=1}^{n} \mathcal{C}_0^{(x)}(\bar{\mathbf{v}}, 0) \cdot \beta_{(\bar{\mathbf{v}}, \bar{0}) \to (\bar{\mathbb{v}}, \bar{\mathbb{z}})} = \beta_{\bar{0} \to \bar{\mathbb{z}}} \cdot \sum_{\bar{\mathbf{v}}=1}^{n} \mathcal{C}_0^{(x)}(\bar{\mathbf{v}}, 0) \cdot \beta_{\bar{\mathbf{v}} \to \bar{\mathbb{v}}},$$

which by Corollaries 4.8 and 4.9 is computable both by:

- A $\mathsf{TISP}\big(O(n \cdot \text{polylog}(|\mathbb{F}|)), O(\text{polylog}(|\mathbb{F}|))\big)$-uniform ASLP with $O(1)$ registers and multiplicative depth s; and
- A $\mathsf{TISP}\Big(O\big(n \cdot s \cdot \text{polylog}(|\mathbb{F}|)\big), O(\text{polylog}(|\mathbb{F}|))\Big)$-uniform ASLP with $O(s)$ registers and multiplicative depth
$$\max(\log(s), \log\log(W)) + O(1) = \log(s) + O(1).$$

Combining the asymptotic expressions yields the proposition as stated. □

Lemma 8.1 follows by combining Corollaries 8.4 and 8.5 and Proposition 8.6.

9 Prover Efficiency

As in Section 7, let M be a standard word RAM machine that on a length-n input runs in time $T = T(n)$ and space $S = S(n)$ with a word size $w = w(n) \le O(\log(T))$. Suppose for simplicity that:

- $\max(\log(n), \log(S)) \le w \le O(\log(T))$,
- $n \le T \le 2^{O(n)}$, and
- $\log(T) \le S \le T$.

We recall some notations that were set up in Section 7. We use $W = \mathsf{poly}(w)$ to refer to the maximal complexity of implementing any (standard) word operation by a single-tape Turing machine on w-bit inputs. We denote $S' = \max(n, S)$, and $T' = \Theta(T \cdot W^2 \cdot \log S')$. Lastly, $s \overset{\text{def}}{=} \lceil \log S' \rceil + \lceil \log W \rceil$ and $t \overset{\text{def}}{=} \lceil \log T' \rceil$.

In this section, we describe the efficiency of the *prover* in our PCP construction (Section 7.2) for the language accepted by M, when instantiated with an explicit field ensemble $\mathbb{F}$ satisfying $|\mathbb{F}| \le T(n)$ (so field operations are computable in time $\mathrm{polylog}(T)$).

Specifically, we show how to evaluate $\hat{X}$, $P_0^{(\mathsf{win})}, \ldots, P_{t+s}^{(\mathsf{win})}$, and $P_0^{(\mathsf{enc})}, \ldots, P_{t+s}^{(\mathsf{enc})}$, as defined in Section 7, with efficiency that depends on the cache behaviour of M. Although we don't explicitly write it, we think of $\hat{X}$, $\{P_i^{(\mathsf{enc})}\}_i$, and $\{P_i^{(\mathsf{win})}\}_i$ as *families* of polynomials that are indexed by inputs x to M, as well as $i \in \{0, \ldots, t+s\}$ in the case of $\{P_i^{(\mathsf{enc})}\}$ and $\{P_i^{(\mathsf{win})}\}$.

Recall that $\mathsf{CF}_\tau(M, x)$ is the "cache-friendliness" of the RAM machine M on input x, given a cache of size τ (see Definition 5.12).

Lemma 9.1. *Let $\tau = \tau(n)$ be a cache parameter. Given as input $x \in \{0,1\}^n$, each of the polynomials $\hat{X}$, $\{P_i^{(\mathsf{enc})}\}_{i \in [t+s]}$, and $\{P_i^{(\mathsf{win})}\}_{i \in [t+s]}$ can be evaluated in time $\big(T + \max(S, n) \cdot \mathsf{CF}_\tau(M, x)\big) \cdot \mathrm{polylog}(T)$ and space $S + \tau \cdot \mathrm{polylog}(T)$.*

Moreover, these polynomials can be evaluated by a $\mathsf{TISP}\Big(\big(T + \max(S, n) \cdot \mathsf{CF}_\tau(M, x)\big) \cdot \mathrm{polylog}(T), S + \tau \cdot \mathrm{polylog}(T)\Big)$-uniform ASLP with $\tau \cdot \mathrm{polylog}(T)$ registers and multiplicative depth $\log\log(T) + O(1)$.

Our analysis will also rely on the efficient algorithms for computing $\hat{\phi}_{+1}$, $\hat{\phi}_d$, and $\hat{\mathcal{C}}_0$ that were given in Section 8.1 (specifically, Corollaries 8.4 and 8.5 and Proposition 8.6).

9.1 Evaluating $\hat{X}$

In this section we show how to evaluate the polynomial $\hat{X}$ efficiently at any given point. As a matter of fact, we shall prove a stronger statement (Proposition 9.2 below) that will be useful in the evaluation of other parts of the PCP proof in the subsequent subsections.

Recall that $t = \lceil \log(T') \rceil$ and $s = \lceil \log S' \rceil + \lceil \log W \rceil$, where $T' = \tilde{O}(T)$ is the running time of the tree machine and $S' = \max(S, n)$.

Proposition 9.2. *For any $i \leq t$, any $\bar{\mathbb{a}} \in \mathbb{F}^{t-i}$, and any $\bar{\mathbb{b}} \in \mathbb{F}^s$, it is possible to enumerate $\{\hat{X}(\bar{\mathbf{p}} \| \bar{\mathbb{a}}, \bar{\mathbb{b}})\}_{\bar{\mathbf{p}} \in \{0,1\}^i}$ in order of lexicographically increasing $\bar{\mathbf{p}}$, using time $\tilde{O}(T)$ and space $S + \mathrm{polylog}(T)$.*

Furthermore, for ℓ distinct $\bar{\mathbb{b}}_1, \ldots, \bar{\mathbb{b}}_\ell$, it is possible to enumerate $\{(\hat{X}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}_1), \ldots, \hat{X}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}_\ell))\}_{\bar{\mathbf{a}} \in \{0,1\}^i}$ in order of lexicographically increasing $\bar{\mathbf{a}}$, using time $\ell \cdot \tilde{O}(T)$ and space $S + \ell \cdot \mathrm{polylog}(T)$.

Moreover, it is possible to do this enumeration with a $\mathsf{TISP}\big(\ell \cdot \tilde{O}(T), S + \log \ell + \mathrm{polylog}(T)\big)$-uniform ASLP that has $O(\ell + \log(T))$ registers, and multiplicative depth $\log\log(T) + O(1)$.

Proof. We first focus on the case $\ell = 1$, and observe that

$$\hat{X}(\bar{\mathbf{p}} \| \bar{\mathbb{a}}, \bar{\mathbb{b}}) = \sum_{\bar{\mathbf{a}} \in \{0,1\}^{t-i}} \beta_{\bar{\mathbf{a}} \to \bar{\mathbb{a}}} \cdot \hat{X}(\bar{\mathbf{p}} \| \bar{\mathbf{a}}, \bar{\mathbb{b}})$$

and by Fact 4.7, the coefficients $\{\beta_{\bar{\mathbf{a}} \to \bar{\mathbb{a}}}\}_{\bar{\mathbf{a}} \in \{0,1\}^{t-i}}$ are enumerable by a $\mathsf{TISP}\big(O(2^{t-i} \cdot \log(t-i)), O(t-i)\big)$-uniform ASLP with $O(t-i)$ registers and multiplicative depth $\log(t-i)+O(1)$. Thus, it suffices to establish the following claim.

Claim 9.2.1. *$\{\hat{X}(\bar{\mathbf{y}}, \bar{\mathbb{b}})\}_{\bar{\mathbf{y}} \in \{0,1\}^t}$, or equivalently $\{\hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbb{b}})\}_{\bar{\mathbf{y}} \in \{0,1\}^t}$, is enumerable in order of lexicographically increasing $\bar{\mathbf{y}}$ by a $\mathsf{TISP}\big(\tilde{O}(T), S + \mathrm{polylog}(T)\big)$-uniform ASLP with $O(s)$ registers and multiplicative depth $\log(s) + O(1)$.*

Proof. Proposition 8.6 states that there is a $\mathsf{TISP}\Big(O\big(n \cdot s \cdot \mathrm{polylog}(|\mathbb{F}|)\big), O\big(s \cdot \mathrm{polylog}(|\mathbb{F}|)\big)\Big)$-uniform ASLP that computes $\hat{\mathcal{C}}_0(\bar{\mathbb{b}})$ with $O(s)$ registers and multiplicative depth $\log(s) + O(1)$. For each $\bar{\mathbf{y}} \geq 1$, observe that given $\hat{\mathcal{C}}_{\bar{\mathbf{y}}-1}(\bar{\mathbb{b}})$ and the (constant-sized) set of differences between $\mathcal{C}_{\bar{\mathbf{y}}-1}$ and $\mathcal{C}_{\bar{\mathbf{y}}}$, we can compute $\hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbb{b}})$ via term-wise differences in the formula $\hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbb{b}}) = \sum_{\bar{\mathbf{b}} \in \{0,1\}^s} \mathcal{C}_{\bar{\mathbf{y}}}(\bar{\mathbf{b}}) \cdot \beta_{\bar{\mathbf{b}} \to \bar{\mathbb{b}}}$. This computation can be performed within an ASLP using $O(s)$ instructions, $O(1)$ temporary registers, and multiplicative depth $\log(s) + O(1)$.

But all of the term-wise differences between $\mathcal{C}_{\bar{\mathbf{y}}-1}$ and $\mathcal{C}_{\bar{\mathbf{y}}}$ for $\bar{\mathbf{y}} \in [2^t]$ are enumerable (outside the ASLP) in time $O(T') = \tilde{O}(T)$ and space $S + O(W)$ using Proposition 4.36. In total, it is possible to enumerate $\{\hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbb{b}})\}_{\bar{\mathbf{y}} \in \{0,1\}^t}$ with a $\mathsf{TISP}(O(n \cdot s \cdot \mathrm{polylog}(|\mathbb{F}|) + T' \cdot s, S + O(W))$-uniform ASLP with $O(s)$ registers and multiplicative depth $\log(s) + O(1)$. ☐

The case of $\ell > 1$ follows in a similar manner. ☐

As a direct corollary of Proposition 9.2 we can also efficiently compute $\hat{X}$ at any given point.

Corollary 9.3. *The polynomial $\hat{X}$ is computable in time $\tilde{O}(T)$ and space $S + \mathrm{polylog}(T)$. Moreover, the polynomial is computable by a $\mathsf{TISP}\Big(\tilde{O}(T), S + \mathrm{polylog}(T)\Big)$-uniform ASLP with $O(t)$ registers and $\log(t) + O(1)$ multiplicative depth.*

9.2 Evaluating $P_i^{(\mathsf{enc})}$ and $P_i^{(\mathsf{win})}$ for $i \leq t$

We proceed to show how to efficiently evaluate the polynomials $P_i^{(\mathsf{enc})}(\bar{\mathbb{y}}, \bar{\mathbb{u}})$ for values of i that are at most t.

Proposition 9.4. *For $i \le t$, the polynomial $P_i^{(\mathsf{enc})}$ can be evaluated in time $\tilde{O}(T)$ and space $S + \operatorname{polylog}(T)$.*

Moreover, $P_i^{(\mathsf{enc})}$ is computable by a $\mathsf{TISP}\big(\tilde{O}(T), S + \operatorname{polylog}(T)\big)$-uniform ASLP with $O(t)$ registers and $\log(t) + O(1)$ multiplicative depth.

Proof. Let $i \le t$, $\bar{\mathbb{y}} \in \mathbb{F}^t$ and $\bar{\mathbb{u}} \in \mathbb{F}^s$ be arbitrary. Write $\bar{\mathbb{y}} = \bar{\mathbb{a}} \| \bar{\mathbb{b}}$ for $\bar{\mathbb{a}} \in \mathbb{F}^i$. Recall that we use vector exponentiation, e.g. $\bar{\mathbb{a}}^{\bar{\mathbf{a}}}$ as shorthand for $\prod_i \mathbb{a}_i^{\mathbf{a}_i}$. By definition of $P_i^{(\mathsf{enc})}$,

$$P_i^{(\mathsf{enc})}(\bar{\mathbb{a}} \| \bar{\mathbb{b}}, \bar{\mathbb{u}}) = \sum_{\bar{\mathbf{a}} \in \{0,1\}^i} Z_{\bar{\Gamma}}\big(\hat{X}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}, \bar{\mathbb{u}})\big) \cdot \bar{\mathbb{a}}^{\bar{\mathbf{a}}}.$$

The polynomial $Z_{\bar{\Gamma}}$ (which was defined as $Z_{\bar{\Gamma}}(\mathbb{z}) = \prod_{\gamma \in \bar{\Gamma}} (\mathbb{z} - \gamma)$) is computable within an ASLP in $O(1)$ instructions, and $\{\bar{\mathbb{a}}^{\bar{\mathbf{a}}}\}_{\bar{\mathbf{a}} \in \{0,1\}^i}$ can be enumerated within an ASLP using $O(i \cdot 2^i)$ instructions, $O(i)$ registers, and $\log(i) + O(1)$ multiplicative depth in a similar fashion to Fact 4.7. Thus, it suffices to enumerate the $2^i = O(T')$ values $\big\{\hat{X}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}, \bar{\mathbb{u}})\big\}_{\bar{\mathbf{a}} \in \{0,1\}^i}$. This can be done with a $\mathsf{TISP}\big(\tilde{O}(T), S + \operatorname{polylog}(T)\big)$-uniform ASLP with $O(t)$ registers and $\log(t) + O(1)$ multiplicative depth by Proposition 9.2. □

An analogous proposition holds for $P_i^{(\mathsf{win})}$, but the proof is slightly more involved:

Proposition 9.5. *For $i \le t$, the polynomial $P_i^{(\mathsf{win})}$ can be evaluated in time $\tilde{O}(T)$ and space $S + \operatorname{polylog}(T)$.*

Moreover, $P_i^{(\mathsf{win})}$ is computable by a $\mathsf{TISP}\big(\tilde{O}(T), S + \operatorname{polylog}(T)\big)$-uniform ASLP with $O(t)$ registers and multiplicative depth $\log(t) + O(1)$.

Proof. We show how to compute $P_i^{(\mathsf{win})}(\bar{\mathbb{y}} \star \bar{\mathbb{y}}', \bar{\mathbb{u}} \star \bar{\mathbb{u}}')$ for arbitrary $\bar{\mathbb{y}}, \bar{\mathbb{y}}' \in \mathbb{F}^t$ and $\bar{\mathbb{u}}, \bar{\mathbb{u}}' \in \mathbb{F}^s$. Write $\bar{\mathbb{y}} = \bar{\mathbb{a}} \| \bar{\mathbb{b}}$ and $\bar{\mathbb{y}}' = \bar{\mathbb{a}}' \| \bar{\mathbb{b}}'$ for $\bar{\mathbb{a}}, \bar{\mathbb{a}}' \in \mathbb{F}^i$ (and recall that $i \le t$). Then $P_i^{(\mathsf{win})}(\bar{\mathbb{y}} \star \bar{\mathbb{y}}', \bar{\mathbb{u}} \star \bar{\mathbb{u}}')$ is, by definition, equal to

$$\begin{aligned}
&P_i^{(\mathsf{win})}\big((\bar{\mathbb{a}} \star \bar{\mathbb{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}'), \bar{\mathbb{u}} \star \bar{\mathbb{u}}'\big) \\
&= \sum_{\bar{\mathbf{a}}, \bar{\mathbf{a}}' \in \{0,1\}^i} P_0^{(\mathsf{win})}\Big((\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}'), \bar{\mathbb{u}} \star \bar{\mathbb{u}}'\Big) \cdot \bar{\mathbb{a}}^{\bar{\mathbf{a}}} \cdot (\bar{\mathbb{a}}')^{\bar{\mathbf{a}}'} \\
&= \sum_{\bar{\mathbf{a}}, \bar{\mathbf{a}}' \in \{0,1\}^i} \hat{\phi}_{+1}\big((\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}')\big) \cdot \sum_{d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}} \hat{\phi}_d(\bar{\mathbb{u}} \star \bar{\mathbb{u}}') \cdot \hat{V}_d\Big(\hat{X}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}, \bar{\mathbb{u}}), \hat{X}(\bar{\mathbf{a}}' \| \bar{\mathbb{b}}', \bar{\mathbb{u}}')\Big) \cdot \bar{\mathbb{a}}^{\bar{\mathbf{a}}} \cdot (\bar{\mathbb{a}}')^{\bar{\mathbf{a}}'} \\
&= \sum_{d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}} \hat{\phi}_d(\bar{\mathbb{u}} \star \bar{\mathbb{u}}') \cdot \sum_{\bar{\mathbf{a}}, \bar{\mathbf{a}}' \in \{0,1\}^i} \hat{\phi}_{+1}\big((\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}')\big) \cdot \hat{V}_d\Big(\hat{X}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}, \bar{\mathbb{u}}), \hat{X}(\bar{\mathbf{a}}' \| \bar{\mathbb{b}}', \bar{\mathbb{u}}')\Big) \cdot \bar{\mathbb{a}}^{\bar{\mathbf{a}}} \cdot (\bar{\mathbb{a}}')^{\bar{\mathbf{a}}'}.
\end{aligned}$$

Because $\hat{\phi}_d$ is computable in time $O(s \cdot T_{\mathbb{F}})$ (in fact by a $\mathsf{TIME}(O(s))$-uniform ASLP with multiplicative depth $\log(s) + O(1)$, see Corollary 8.5), it suffices to separately compute for each d the following sub-sum:

$$\sum_{\bar{\mathbf{a}}, \bar{\mathbf{a}}' \in \{0,1\}^i} \hat{\phi}_{+1}\big((\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}')\big) \cdot \hat{V}_d\Big(\hat{X}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}, \bar{\mathbb{u}}), \hat{X}(\bar{\mathbf{a}}' \| \bar{\mathbb{b}}', \bar{\mathbb{u}}')\Big) \cdot \bar{\mathbb{a}}^{\bar{\mathbf{a}}} \cdot (\bar{\mathbb{a}}')^{\bar{\mathbf{a}}'}.$$

We again simplify our task by noting that $\hat{\phi}_{+1}((\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}'))$ is non-zero only if for some $\bar{\mathbf{b}}, \bar{\mathbf{b}}' \in \{0,1\}^{t-i}$ we have $\phi_{+1}\big((\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbf{b}} \star \bar{\mathbf{b}}')\big) = 1$, which can happen only if $\bar{\mathbf{a}}' \in \{\bar{\mathbf{a}}, \bar{\mathbf{a}} + 1\}$. Thus, the above summation has at most $2^{i+1} = O(T)$ *non-zero* terms. To enumerate these terms, it suffices to enumerate $\left\{\left(\hat{X}(\bar{\mathbf{a}}\|\bar{\mathbb{b}}, \bar{\mathbb{u}}), \hat{X}(\bar{\mathbf{a}}'\|\bar{\mathbb{b}}', \bar{\mathbb{u}}')\right)\right\}_{\substack{\bar{\mathbf{a}}\in\{0,1\}^i \\ \bar{\mathbf{a}}'\in\{\bar{\mathbf{a}},\bar{\mathbf{a}}+1\}}}$ because

- $\hat{\phi}_{+1}$ is computable by a $\mathsf{TIME}(O(t))$-uniform ASLP with multiplicative depth $\log(t) + O(1)$ (see Corollary 8.4);
- $\hat{V}_d$ is computable with an ASLP with $O(1)$ instructions (via Lagrange interpolation polynomials, with coefficients computed externally to the ASLP); and
- $\bar{\mathbb{a}}^{\bar{\mathbf{a}}}$ and $(\bar{\mathbb{a}}')^{\bar{\mathbf{a}}'}$ are each computable within an ASLP using $t = O(\log T')$ instructions with multiplicative depth $\log(t) + O(1)$.

The enumeration of $\left\{\left(\hat{X}(\bar{\mathbf{a}}\|\bar{\mathbb{b}}, \bar{\mathbb{u}}), \hat{X}(\bar{\mathbf{a}}'\|\bar{\mathbb{b}}', \bar{\mathbb{u}}')\right)\right\}_{\substack{\bar{\mathbf{a}}\in\{0,1\}^i \\ \bar{\mathbf{a}}'\in\{\bar{\mathbf{a}},\bar{\mathbf{a}}+1\}}}$ in turn reduces to enumerating the sequence $\left\{\left(\hat{X}(\bar{\mathbf{a}}\|\bar{\mathbb{b}}, \bar{\mathbb{u}}), \hat{X}(\bar{\mathbf{a}}\|\bar{\mathbb{b}}', \bar{\mathbb{u}}')\right)\right\}_{\bar{\mathbf{a}}\in\{0,1\}^i}$, because we can simply keep the last element as we are enumerating. By Proposition 9.2, the latter enumeration is possible using a $(\tilde{O}(T), S+\text{polylog}(T))$-uniform ASLP with $O(t)$ registers and multiplicative depth $\log(t) + O(1)$. □

9.3 Evaluating $P_i^{(\mathsf{enc})}$ for $t < i \leq t + s$

In this section, we show how to efficiently evaluate $P_i^{(\mathsf{enc})}(\bar{\mathbb{y}}, \bar{\mathbb{u}})$ for arbitrary $\bar{\mathbb{y}} \in \mathbb{F}^t$ and $\bar{\mathbb{u}} \in \mathbb{F}^s$ for $i > t$.

The main algorithmic bottleneck in evaluating $P_i^{(\mathsf{enc})}$ turns out to be evaluating the *low-degree extensions* of each of the $\mathsf{Tree}(M)$'s configurations $\mathcal{C}_1, \ldots, \mathcal{C}_T$ on points in $\{0,1\}^r \times \mathbb{F}^{s-r}$. Each of these evaluations can be done in $O(2^{s-r})$ operations over $\mathbb{F}$ (see Proposition 4.36, but we wish to evaluate $P_i^{(\mathsf{enc})}$ in time less than $T \cdot 2^{s-r}$. Additionally, we want to use minimal space beyond what is required to evaluate M on x.

We use a *bounded memoization* technique to reduce the number of necessary evaluations at the expense of somewhat larger space usage. We show that the profitability of this trade-off is intimately related to the memory access pattern of M on x, and in particular the number of cache misses it incurs (with various block sizes). For a quick review of standard definitions and results about caching, the reader is invited to Section 5.2.

Recall from Definition 4.16 that the combined access pattern of a RAM machine M on an input x is a tuple whose entries are all addresses accessed by M, where input accesses are treated as if they were memory accesses. Conflating input accesses with memory accesses is somewhat unnatural, and is done for convenience, as $\mathsf{Tree}(M)$ stores the j^{th} bit of x and the j^{th} memory cell's contents in the same region of its tape.

The performance of our algorithm is determined by the number of misses incurred by the combined access pattern on an ideal cache (see Definition 5.6). This may seem even less natural, but it is bounded[37] by the number of misses incurred by input accesses *plus* the number of misses incurred by memory accesses. Both of the latter quantities are natural parameters that are optimized in practical programs.

[37]up to a factor of 2 both in the number of misses and in the cache size considered

Proposition 9.6. *There is an algorithm for evaluating $P^{(\mathsf{enc})}_{t+r}$, for any $r \in [s]$, whose efficiency is controlled by a "cache size" parameter τ as follows.*

Let $\mathsf{addrs} = (\mathsf{addr}_1, \dots, \mathsf{addr}_{O(T)})$ *denotes the combined access pattern of M on x. Let Q denote the number of misses incurred by* $(\textsc{Prefix}_r(\mathsf{addr}_1), \dots, \textsc{Prefix}_r(\mathsf{addr}_{O(T)}))$ *on an ideal cache of size τ. The algorithm runs in time $(T + Q \cdot 2^{s-r}) \cdot \mathrm{polylog}(T)$ and space $S + \tau \cdot \mathrm{polylog}(T)$.*

Moreover, $P^{(\mathsf{enc})}_{t+r}$ is computable by a $\mathsf{TISP}\big((T+Q\cdot 2^{s-r})\cdot \mathrm{polylog}(T), S+\tau\cdot\mathrm{polylog}(T)\big)$*-uniform* ASLP *with $\tau \cdot \mathrm{polylog}(T)$ registers and multiplicative depth $\log(t) + O(1)$.*

Proof. Consider an arbitrary input $(\bar{\mathbb{y}}, \bar{\mathbb{u}}) \in \mathbb{F}^t \times \mathbb{F}^s$ to $P^{(\mathsf{enc})}_{t+r}$. Writing $\bar{\mathbb{u}} = \bar{\mathbb{a}} \| \bar{\mathbb{b}}$ with $\bar{\mathbb{a}} \in \mathbb{F}^r$, we have, by definition, that

$$P^{(\mathsf{enc})}_{t+r}(\bar{\mathbb{y}}, \bar{\mathbb{a}}\|\bar{\mathbb{b}}) = \sum_{\bar{\mathbf{y}} \in \{0,1\}^t} \sum_{\bar{\mathbf{a}} \in \{0,1\}^r} Z_{\bar{\Gamma}}\big(\hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}}\|\bar{\mathbb{b}})\big) \cdot \bar{\mathbb{y}}^{\bar{\mathbf{y}}} \cdot \bar{\mathbb{a}}^{\bar{\mathbf{a}}}.$$

In Algorithm 2 below, we show how to enumerate the $2^t = O(T')$ terms $\{\sigma_{\bar{\mathbf{y}}}\}_{\bar{\mathbf{y}} \in \{0,1\}^t}$, where

$$\sigma_{\bar{\mathbf{y}}} \overset{\mathsf{def}}{=} \sum_{\bar{\mathbf{a}} \in \{0,1\}^r} Z_{\bar{\Gamma}}\big(\hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}}\|\bar{\mathbb{b}})\big) \cdot \bar{\mathbb{a}}^{\bar{\mathbf{a}}}, \tag{15}$$

in total time $(T + Q\cdot 2^{s-r}) \cdot \mathrm{polylog}(T)$ and space $S + O\big(\tau \cdot \mathrm{polylog}(T)\big)$. Actually, Algorithm 2 can be viewed as producing, in time $(T + Q \cdot 2^{s-r}) \cdot \mathrm{polylog}(T)$ and space $S + \tau \cdot \mathrm{polylog}(T)$, an ASLP that enumerates $\{\sigma_{\bar{\mathbf{y}}}\}_{\bar{\mathbf{y}} \in \{0,1\}^t}$ with $\tau \cdot \mathrm{polylog}(T)$ registers and multiplicative depth $\log(s) + O(1)$.

We use this as a subroutine to evaluate $P^{(\mathsf{enc})}_{t+r}$ with Algorithm 3, whose correctness, efficiency, and multiplicative depth are immediate (given the analysis of Algorithm 2).

.

Algorithm 2 Enumerates (as streaming output) $\sigma_0, \dots, \sigma_{2^t-1}$

```
1: $\mathsf{configDS} \leftarrow \textsc{InitConfigDS}(M, x)$
2: Initialize $\mathsf{cache}$ as a new LRU cache of size $\tau' = 2W \cdot \tau \cdot \log S'$.
3: $\sigma \leftarrow \sum_{\bar{\mathbf{a}} \in \{0,1\}^r} Z_{\bar{\gamma}}\big(\mathsf{configDS}.\textsc{Eval}(\bar{\mathbf{a}}\|\bar{\mathbb{b}})\big) \cdot \bar{\mathbb{a}}^{\bar{\mathbf{a}}}$.    ▷ $\sigma = \sum_{\bar{\mathbf{a}} \in \{0,1\}^r} Z_{\bar{\gamma}}\big(\hat{\mathcal{C}}_0(\bar{\mathbf{a}}\|\bar{\mathbb{b}})\big) \cdot \bar{\mathbb{a}}^{\bar{\mathbf{a}}}$
4: Output "$\sigma_0 = \sigma$".
5: for $\bar{\mathbf{y}} \in \{0, \dots, 2^t - 2\}$ do
6:     $U_{\bar{\mathbf{y}}} \leftarrow \mathsf{configDS}.\textsc{ActiveCells}()$    ▷ i.e., $U_{\bar{\mathbf{y}}} = \Big\{\bar{\mathbf{u}} : \mathcal{C}_{\bar{\mathbf{y}}}(\bar{\mathbf{u}}) \neq \mathcal{C}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{u}})\Big\}$
7:     $A_{\bar{\mathbf{y}}} \leftarrow \{\textsc{Prefix}_r(\bar{\mathbf{u}}) : \bar{\mathbf{u}} \in U_{\bar{\mathbf{y}}}\}$    ▷ $A_{\bar{\mathbf{y}}} \supseteq \Big\{\bar{\mathbf{a}} : \hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}}\|\bar{\mathbb{b}}) \neq \hat{\mathcal{C}}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{a}}\|\bar{\mathbb{b}})\Big\}$
8:     for all $\bar{\mathbf{a}} \in A_{\bar{\mathbf{y}}}$ do
9:         $\mathsf{cur} \leftarrow \mathsf{cache}.\textsc{GetOrCompute}\big(\bar{\mathbf{a}}, \mathsf{configDS}.\textsc{Eval}(\cdot\|\bar{\mathbb{b}})\big)$    ▷ $\mathsf{cur} = \hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}}\|\bar{\mathbb{b}})$
10:        $\mathsf{new} \leftarrow \mathsf{cur}$
11:        for all $\bar{\mathbf{b}} \in \{0,1\}^{s-r}$ such that $\bar{\mathbf{a}}\|\bar{\mathbf{b}} \in U_{\bar{\mathbf{y}}}$ do    ▷ computes $\mathsf{new} = \hat{\mathcal{C}}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{a}}\|\bar{\mathbb{b}})$
12:            $\mathsf{new} \leftarrow \mathsf{new} + \big(\mathsf{configDS}.\textsc{NextEval}(\bar{\mathbf{a}}\|\bar{\mathbf{b}}) - \mathsf{configDS}.\textsc{Eval}(\bar{\mathbf{a}}\|\bar{\mathbf{b}})\big) \cdot \beta_{\bar{\mathbf{b}} \to \bar{\mathbb{b}}}$
13:        $\sigma \leftarrow \sigma + (Z_{\bar{\gamma}}(\mathsf{new}) - Z_{\bar{\gamma}}(\mathsf{cur})) \cdot \bar{\mathbb{a}}^{\bar{\mathbf{a}}}$
14:        $\mathsf{cache}.\textsc{Put}(\bar{\mathbf{a}}, \mathsf{new})$
15:    Output "$\sigma_{\bar{\mathbf{y}}+1} = \sigma$".
16:    $\mathsf{config}.\textsc{Advance}()$    ▷ from $\mathcal{C}_{\bar{\mathbf{y}}}$ to $\mathcal{C}_{\bar{\mathbf{y}}+1}$
```

Algorithm 3 Evaluates $P_i^{(\mathsf{enc})}$ for $i > t$

```
1: $\mathsf{sum} \leftarrow 0$, $\bar{\mathbf{y}} \leftarrow 0$
2: for $\sigma \leftarrow \sigma_0, \ldots, \sigma_{2^t-1}$ do        ▷ Enumerated using Algorithm 2
3:     $\mathsf{sum} \leftarrow \mathsf{sum} + \sigma \cdot (\bar{\mathbb{y}})^{\bar{\mathbf{y}}}$
4:     $\bar{\mathbf{y}} \leftarrow \bar{\mathbf{y}} + 1$
5: return sum
```

Correctness of Algorithm 2. The for-loop of Line 5 maintains several invariants that hold at the beginning of the $\bar{\mathbf{y}}^{th}$ iteration, for every $\bar{\mathbf{y}}$:

1. Every (key, value) pair in cache is of the form $\big(\bar{\mathbf{a}}, \hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}}\|\bar{\mathbb{b}})\big)$, for some $\bar{\mathbf{a}} \in \{0,1\}^r$.

2. σ is $\sigma_{\bar{\mathbf{y}}}$ as defined in Eq. (15).

3. $\mathsf{configDS}.\mathcal{C} = \mathcal{C}_{\bar{\mathbf{y}}}$, i.e., the (succinct) representation of the $\bar{\mathbf{y}}^{th}$ configuration of $\mathsf{Tree}(M)$ on input x.

Lines 6 and 7 compute the possible terms in the definition of σ that could change when $\bar{\mathbf{y}}$ increments, and the for-loop of Line 8 iterates over these terms. The for-loop at Line 11 computes how each term actually changes (if at all), and Line 13 applies this change to σ to maintain Invariant 2. Line 14 applies these changes to the entries of cache to maintain Invariant 1.

The correctness of Algorithm 2 follows from the fact that Invariant 2 is maintained throughout the course of the algorithm, and, in each step, we output $\sigma = \sigma_{\bar{\mathbf{y}}+1}$.

Space Usage of Algorithm 2. The data structure configDS uses $S + O(W)$ space (see Proposition 4.36) and cache uses $O\big(\tau' \cdot (r + \log(|\mathbb{F}|))\big)$ space, where recall that $\tau' = 2W \cdot \tau \cdot \log S'$. The rest of Algorithm 2 only uses space $O(r \cdot \log(|\mathbb{F}|))$. In total, Algorithm 2 uses space $S + O(W) + O\big(\tau' \cdot (r + \log(|\mathbb{F}|))\big) + O(r \cdot \log(|\mathbb{F}|) = S + O\big(\tau \cdot s \cdot W \cdot (r + \log(|\mathbb{F}|))\big)$.

Running Time of Algorithm 2. It is straightforward to verify that the running time of Algorithm 2 excluding Line 9 is $O\big((2^s + 2^t \cdot s) \cdot T_{\mathbb{F}}\big) = O\big(T' \cdot s \cdot T_{\mathbb{F}}\big)$. (Note that Line 3 takes time $O(2^s \cdot T_{\mathbb{F}})$ and $|U_{\bar{\mathbf{y}}}| \leq 2$ for all $\bar{\mathbf{y}}$, which bounds the number of iterations of the for-loops at Lines 8 and 11.)

The total running time of Line 9 (over all loop iterations) is the running time of $\mathsf{config}.\textsc{Eval}(\cdot\|\bar{\mathbb{b}})$ (which is $O(T_{\mathbb{F}} \cdot 2^{s-r})$) *multiplied by* the number of misses incurred on an LRU cache of size τ' by the sequence of values for $\bar{\mathbf{a}}$ at Line 9 over the entire execution of Algorithm 2.

Claim 9.6.1. *The number of misses incurred on an LRU cache of size τ' by the sequence of values for $\bar{\mathbf{a}}$ at Line 9 over the entire execution of Algorithm 2 is at most $2W \cdot \log(S') \cdot Q$.*

Proof. Recall that we denote by Q_{LRU} (resp., Q_{ideal}) the number of cache misses incurred on LRU (resp., ideal) cache (see Section 5.2).

Let $\mathfrak{a}$ denote the sequence of values taken by $\bar{\mathbf{a}}$ in the $O(T')$ executions of Line 9 in Algorithm 2. By Theorem 5.10 it holds that:

$$Q_{\mathsf{LRU}}(\mathfrak{a}, \tau') \leq 2 \cdot Q_{\mathsf{ideal}}(\mathfrak{a}, \tau'/2). \tag{16}$$

Let $(\bar{\mathbf{u}}_0, \ldots, \bar{\mathbf{u}}_{2^t-1})$ denote the access pattern of $\mathsf{Tree}(M)$ on input x. Note that by construction[38]

$$\mathfrak{a} = \big(\textsc{Prefix}_r(\bar{\mathbf{u}}_0), \textsc{Prefix}_r(\bar{\mathbf{u}}_1), \textsc{Prefix}_r(\bar{\mathbf{u}}_1), \ldots, \textsc{Prefix}_r(\bar{\mathbf{u}}_{2^t-2}), \textsc{Prefix}_r(\bar{\mathbf{u}}_{2^t-2}), \textsc{Prefix}_r(\bar{\mathbf{u}}_{2^t-1})\big).$$

Thus,

$$Q_{\mathsf{ideal}}(\mathfrak{a}, \tau'/2) = Q_{\mathsf{ideal}}((\textsc{Prefix}_r(\bar{\mathbf{u}}_0), \ldots, \textsc{Prefix}_r(\bar{\mathbf{u}}_{T'})), \tau'/2). \tag{17}$$

Let $\mathsf{addrs} = (\mathsf{addr}_1, \ldots, \mathsf{addr}_{O(T)})$ denote the combined access pattern of the RAM machine M on x. Then, combining Proposition 5.8 and Proposition 4.35 we get that the right hand side of Eq. (17) is upper bounded by

$$W \cdot \log(S') \cdot Q_{\mathsf{ideal}}\left((\textsc{Prefix}_r(\mathsf{addr}_1), \ldots, \textsc{Prefix}_r(\mathsf{addr}_{O(T)})), \frac{\tau'}{2W \cdot \log(S')}\right) \tag{18}$$

which in turn is equal to $W \cdot \log(S') \cdot Q$. The claim follows by combining Eqs. (16) to (18). This concludes the proof of Claim 9.6.1. □

This concludes the proof of Proposition 9.6. □

9.4 Evaluating $P_i^{(\mathsf{win})}$ for $t < i \leq t + s$

We now move on to $P_i^{(\mathsf{win})}$, showing how to efficiently compute $P_i^{(\mathsf{win})}(\bar{\mathbb{y}} \star \bar{\mathbb{y}}', \bar{\mathbb{u}} \star \bar{\mathbb{u}}')$ for arbitrary $\bar{\mathbb{y}}, \bar{\mathbb{y}}' \in \mathbb{F}^t$ and $\bar{\mathbb{u}}, \bar{\mathbb{u}}' \in \mathbb{F}^s$.

Proposition 9.7. *There is an algorithm for evaluating $P_{t+r}^{(\mathsf{win})}$, for any $0 < r \leq s$, which is given a cache size parameter τ, and exhibits the following efficiency.*

Let $\mathsf{addrs} = (\mathsf{addr}_1, \ldots, \mathsf{addr}_{O(T)})$ denote the combined access pattern of the RAM machine M on x. Let Q denote the number of misses incurred by $\big(\textsc{Prefix}_r(\mathsf{addr}_1), \ldots, \textsc{Prefix}_r(\mathsf{addr}_{O(T)})\big)$ on an ideal cache of size τ. The algorithm runs in time $(T + Q \cdot 2^{s-r}) \cdot \mathrm{polylog}(T)$ and space $S + \tau \cdot \mathrm{polylog}(T)$.

Moreover, $P_{t+r}^{(\mathsf{win})}$ is computable by a $\mathsf{TISP}\big((T + Q \cdot 2^{s-r}) \cdot \mathrm{polylog}(T), S + \tau \cdot \mathrm{polylog}(T)\big)$-uniform ASLP with $\tau \cdot \mathrm{polylog}(T)$ registers and multiplicative depth $\log(t) + O(1)$.

Proof. Let $0 < r \leq s$ be such that $i = t + r$. Fix inputs $\bar{\mathbb{y}}, \bar{\mathbb{y}}' \in \mathbb{F}^t$, and $\bar{\mathbb{u}}, \bar{\mathbb{u}}' \in \mathbb{F}^s$ be inputs for $P_{t+r}^{(\mathsf{win})}$. Let $\bar{\mathbb{a}}, \bar{\mathbb{a}}' \in \mathbb{F}^r$ and $\bar{\mathbb{b}}, \bar{\mathbb{b}}' \in \mathbb{F}^{s-r}$ such that $\bar{\mathbb{u}} = \bar{\mathbb{a}} \| \bar{\mathbb{b}}$ and $\bar{\mathbb{u}}' = \bar{\mathbb{a}}' \| \bar{\mathbb{b}}'$. By definition, $P_{t+r}(\bar{\mathbb{y}} \star \bar{\mathbb{y}}', \bar{\mathbb{u}} \star \bar{\mathbb{u}}')$ is equal to

$$\begin{aligned}
&\sum_{\bar{\mathbf{y}}, \bar{\mathbf{y}}', \bar{\mathbf{a}}, \bar{\mathbf{a}}'} P_0\Big(\bar{\mathbf{y}} \star \bar{\mathbf{y}}', (\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}')\Big) \cdot (\bar{\mathbb{y}} \star \bar{\mathbb{y}}')^{\bar{\mathbf{y}} \star \bar{\mathbf{y}}'} \cdot (\bar{\mathbb{a}} \star \bar{\mathbb{a}}')^{\bar{\mathbf{a}} \star \bar{\mathbf{a}}'} \\
&= \sum_{\bar{\mathbf{y}}, \bar{\mathbf{y}}'} \hat{\phi}_{+1}(\bar{\mathbf{y}} \star \bar{\mathbf{y}}') \cdot (\bar{\mathbb{y}} \star \bar{\mathbb{y}}')^{\bar{\mathbf{y}} \star \bar{\mathbf{y}}'} \cdot \sum_{\bar{\mathbf{a}}, \bar{\mathbf{a}}'} \sum_d \hat{\phi}_d\big((\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}')\big) \cdot \hat{V}_d\Big(\hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}), \hat{\mathcal{C}}_{\bar{\mathbf{y}}'}(\bar{\mathbf{a}}' \| \bar{\mathbb{b}}')\Big) \cdot (\bar{\mathbb{a}} \star \bar{\mathbb{a}}')^{\bar{\mathbf{a}} \star \bar{\mathbf{a}}'} \\
&= \sum_d \sum_{\bar{\mathbf{y}}=0}^{2^t-2} (\bar{\mathbb{y}} \star \bar{\mathbb{y}}')^{\bar{\mathbf{y}} \star (\bar{\mathbf{y}}+1)} \cdot \sum_{\bar{\mathbf{a}}, \bar{\mathbf{a}}'} \hat{\phi}_d\big((\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}')\big) \cdot \hat{V}_d\Big(\hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}), \hat{\mathcal{C}}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{a}}' \| \bar{\mathbb{b}}')\Big) \cdot (\bar{\mathbb{a}} \star \bar{\mathbb{a}}')^{\bar{\mathbf{a}} \star \bar{\mathbf{a}}'},
\end{aligned}$$

[38] This is basically because the algorithm enumerates at every step both the current and next head positions.

where the last equality uses the fact that

$$\hat{\phi}_{+1}(\bar{\mathbf{y}} \star \bar{\mathbf{y}}') = \phi_{+1}(\bar{\mathbf{y}} \star \bar{\mathbf{y}}') = \begin{cases} 1 & \text{if } \bar{\mathbf{y}}' = \bar{\mathbf{y}} + 1 \\ 0 & \text{otherwise.} \end{cases}$$

Fix $d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}$. We show how to efficiently enumerate the sequence $\{\sigma_{d,\bar{\mathbf{y}}}\}_{\bar{\mathbf{y}} \in \{0,\ldots,2^t-2\}}$, where

$$\sigma_{d,\bar{\mathbf{y}}} \overset{\mathsf{def}}{=} \sum_{\bar{\mathbf{a}},\bar{\mathbf{a}}'} \hat{\phi}_d\big((\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}')\big) \cdot \hat{V}_d\Big(\hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}), \hat{\mathcal{C}}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{a}}' \| \bar{\mathbb{b}}')\Big) \cdot (\bar{\mathbb{a}} \star \bar{\mathbb{a}}')^{\bar{\mathbf{a}} \star \bar{\mathbf{a}}'} \tag{19}$$

(in order of increasing $\bar{\mathbf{y}} \in \{0, \ldots, 2^t - 2\}$). Specifically, Algorithm 4 shows how to efficiently enumerate $\{\sigma_{d,\bar{\mathbf{y}}}\}_{\bar{\mathbf{y}}}$ in order of increasing $\bar{\mathbf{y}}$, taking time $O\big((s \cdot (S' \cdot W + T') + Q \cdot 2^{s-r} \cdot W \cdot \log S') \cdot T_{\mathbb{F}}\big)$ and using space $S + O\big(\tau \cdot W \log S' \cdot (\log(|\mathbb{F}|) + r)\big)$. We can then evaluate $P^{(\mathsf{win})}_{t+r}$ in a straightforward way with Algorithm 5. Before describing these algorithms, we first introduce the following useful notation:

Definition 9.8. *For any $d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}$, we write $\bar{\mathbf{a}} \sim_d \bar{\mathbf{a}}'$ for $\bar{\mathbf{a}}, \bar{\mathbf{a}}' \in \{0,1\}^i$ iff for some $\bar{\mathbf{b}}, \bar{\mathbf{b}}' \in \{0,1\}^{s-i}$, it holds that $\phi_d((\bar{\mathbf{a}} \| \bar{\mathbf{b}}) \star (\bar{\mathbf{a}}' \| \bar{\mathbf{b}}')) \neq 0$.*

We proceed to describe Algorithms 4 and 5.

Correctness of Algorithm 4. The for-loop of Line 6 maintains several invariants:

1. Every (key, value) pair in cache is of the form $\big(\bar{\mathbf{a}}, \hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}} \| \bar{\mathbb{b}})\big)$, and every (key, value) pair in cache' is of the form $\big(\bar{\mathbf{a}}', \hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}}' \| \bar{\mathbb{b}}')\big)$.
2. σ is $\sigma_{d,\bar{\mathbf{y}}}$ as defined in Eq. (19).
3. $\mathsf{configDS}.\mathcal{C}$ is the (succinct) representation of the $\bar{\mathbf{y}}^{th}$ configuration of $\mathsf{Tree}(M)$.

Lines 6 and 7 compute the possible terms in the definition of σ that could change when $\bar{\mathbf{y}}$ increments, and the for-loop of Line 12 iterates over these terms. Lines 16 and 18 compute how these terms actually change (if at all), and Line 22 applies this change to σ to maintain Invariant 2. Lines 20 and 21 store these changes, and the for-loops at Lines 23 and 25 apply these changes to the entries of cache and cache' to maintain Invariant 1.

Space Usage. Other than the following variables, Algorithm 4 uses space $O(\log T' \cdot \log(|\mathbb{F}|))$:

- config, which uses space $S + O(W)$.
- cache and cache', which each use space $O\big(\tau' \cdot (\log(|\mathbb{F}|) + r)\big)$.

Running Time. The running time of Algorithm 4 excluding Lines 13 and 14 is $O(T' \cdot s \cdot T_{\mathbb{F}})$. In particular:

- The time to evaluate Line 4 is $O\big((2^r \cdot (s + 2^{s-r} \cdot r)) \cdot T_{\mathbb{F}}\big)$, which is at most $O\big(2^s \cdot s \cdot T_{\mathbb{F}}\big)$.
- $|U_{\bar{\mathbf{y}}}| \leq 2$ for all $\bar{\mathbf{y}}$, which bounds the number of iterations of the for-loops at Lines 12, 16, 18, 23 and 25.

Algorithm 4 Produces (as streaming output) $\sigma_{d,0}, \dots, \sigma_{d,2^t-2}$

```
1:  $\mathsf{configDS} \leftarrow \textsc{InitConfigDS}(M, x)$
2:  Initialize $\mathsf{cache}$ and $\mathsf{cache}'$ as new LRU caches of size $\tau' = 288 \cdot \tau \cdot W \cdot \log(S')$
3:  ▷ Below, we use the shorthands $\hat{\mathcal{C}}_0(\cdot)$ for $\mathsf{configDS}.\textsc{Eval}(\cdot)$ and $\hat{\mathcal{C}}_1(\cdot)$ for $\mathsf{configDS}.\textsc{NextEval}(\cdot)$.
4:  $\sigma \leftarrow \sum_{\bar{\mathbf{a}} \sim_d \bar{\mathbf{a}}'} \hat{\phi}_d\big((\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}')\big) \cdot \hat{V}_d\Big(\hat{\mathcal{C}}_0(\bar{\mathbf{a}} \| \bar{\mathbb{b}}), \hat{\mathcal{C}}_1(\bar{\mathbf{a}}' \| \bar{\mathbb{b}}')\Big) \cdot (\bar{\mathbb{a}} \star \bar{\mathbb{a}}')^{\bar{\mathbf{a}} \star \bar{\mathbf{a}}'}$
5:  output "$\sigma_{d,0} = \sigma$"
6:  for all $\bar{\mathbf{y}} = 0, \dots, 2^t - 3$ do
7:      ▷ Throughout this loop, we use the shorthands $\hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\cdot)$ for $\mathsf{configDS}.\textsc{Eval}(\cdot)$, $\hat{\mathcal{C}}_{\bar{\mathbf{y}}+1}(\cdot)$ for $\mathsf{configDS}.\textsc{NextEval}(\cdot)$ and $\hat{\mathcal{C}}_{\bar{\mathbf{y}}+2}(\cdot)$ for $\mathsf{configDS}.\textsc{NextNextEval}(\cdot)$
8:      $U_{\bar{\mathbf{y}}} \leftarrow \Big\{(\bar{\mathbf{u}}, \bar{\mathbf{u}}') : \bar{\mathbf{u}} \sim_d \bar{\mathbf{u}}'$ and either $\mathcal{C}_{\bar{\mathbf{y}}}(\bar{\mathbf{u}}) \neq \mathcal{C}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{u}})$ or $\mathcal{C}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{u}}') \neq \mathcal{C}_{\bar{\mathbf{y}}+2}(\bar{\mathbf{u}}')\Big\}$
9:      ▷ See Proposition 4.36 for details on the implementation of this step.
10:     $A_{\bar{\mathbf{y}}} \leftarrow \Big\{\big(\textsc{Prefix}_r(\bar{\mathbf{u}}), \textsc{Prefix}_r(\bar{\mathbf{u}}')\big) : (\bar{\mathbf{u}}, \bar{\mathbf{u}}') \in U_{\bar{\mathbf{y}}}\Big\}$
11:     $\mathsf{updates} \leftarrow \emptyset$, $\mathsf{updates}' \leftarrow \emptyset$
12:     for all $(\bar{\mathbf{a}}, \bar{\mathbf{a}}') \in A_{\bar{\mathbf{y}}}$ do
13:         $\mathsf{cur} \leftarrow \mathsf{cache}.\textsc{GetOrCompute}\big(\bar{\mathbf{a}}, \hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\cdot \| \bar{\mathbb{b}})\big)$        ▷ $\mathsf{cur} = \hat{\mathcal{C}}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}} \| \bar{\mathbb{b}})$
14:         $\mathsf{cur}' \leftarrow \mathsf{cache}'.\textsc{GetOrCompute}\big(\bar{\mathbf{a}}', \hat{\mathcal{C}}_{\bar{\mathbf{y}}+1}(\cdot \| \bar{\mathbb{b}}')\big)$        ▷ $\mathsf{cur}' = \hat{\mathcal{C}}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{a}}' \| \bar{\mathbb{b}}')$
15:         $\mathsf{new} \leftarrow \mathsf{cur}$, $\mathsf{new}' \leftarrow \mathsf{cur}'$
16:         for all $\bar{\mathbf{b}}$ such that $\mathcal{C}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}} \| \bar{\mathbf{b}}) \neq \mathcal{C}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{a}} \| \bar{\mathbf{b}})$ do
17:             $\mathsf{new} \leftarrow \mathsf{new} + \big(\mathcal{C}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{a}} \| \bar{\mathbf{b}}) - \mathcal{C}_{\bar{\mathbf{y}}}(\bar{\mathbf{a}} \| \bar{\mathbf{b}})\big) \cdot \beta_{\bar{\mathbf{b}} \to \bar{\mathbb{b}}}$
18:         for all $\bar{\mathbf{b}}'$ such that $\mathcal{C}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{a}}' \| \bar{\mathbf{b}}') \neq \mathcal{C}_{\bar{\mathbf{y}}+2}(\bar{\mathbf{a}}' \| \bar{\mathbf{b}}')$ do
19:             $\mathsf{new}' \leftarrow \mathsf{new}' + \big(\mathcal{C}_{\bar{\mathbf{y}}+2}(\bar{\mathbf{a}}' \| \bar{\mathbf{b}}') - \mathcal{C}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{a}}' \| \bar{\mathbf{b}}')\big) \cdot \beta_{\bar{\mathbf{b}}' \to \bar{\mathbb{b}}'}$
20:         $\mathsf{updates} \leftarrow \mathsf{updates} \cup \{(\bar{\mathbf{a}}, \mathsf{new})\}$
21:         $\mathsf{updates}' \leftarrow \mathsf{updates}' \cup \{(\bar{\mathbf{a}}', \mathsf{new}')\}$
22:         $\sigma \leftarrow \sigma + \hat{\phi}_d\big((\bar{\mathbf{a}} \star \bar{\mathbf{a}}') \| (\bar{\mathbb{b}} \star \bar{\mathbb{b}}')\big) \cdot \Big(\hat{V}_d(\mathsf{new}, \mathsf{new}') - \hat{V}_d(\mathsf{cur}, \mathsf{cur}')\Big) \cdot (\bar{\mathbb{a}} \star \bar{\mathbb{a}}')^{\bar{\mathbf{a}} \star \bar{\mathbf{a}}'}$
23:     for all $(k, v) \in \mathsf{updates}$ do
24:         $\mathsf{cache}.\textsc{Put}(k, v)$
25:     for all $(k, v) \in \mathsf{updates}'$ do
26:         $\mathsf{cache}'.\textsc{Put}(k, v)$
27:     output "$\sigma_{d,\bar{\mathbf{y}}+1} = \sigma$".
28:     $\mathsf{configDS}.\textsc{Advance}()$
```

Algorithm 5 Evaluates $P_i^{(\mathsf{win})}$ for $i > t$

```
1:  $\mathsf{sum} \leftarrow 0$
2:  for all $d \in \{\uparrow, \swarrow, \searrow, \otimes, \odot, \emptyset\}$ do
3:      $\bar{\mathbf{y}} \leftarrow 0$
4:      for $\sigma = \sigma_{d,0}, \dots, \sigma_{d,2^t-2}$ do        ▷ Enumerated using Algorithm 4
5:          $\mathsf{sum} \leftarrow \mathsf{sum} + \sigma \cdot (\bar{\mathbb{y}} \star \bar{\mathbb{y}}')^{\bar{\mathbf{y}} \star (\bar{\mathbf{y}}+1)}$
6:          $\bar{\mathbf{y}} \leftarrow \bar{\mathbf{y}} + 1$
7:  return $\mathsf{sum}$
```

The total running time of Line 13 (over all loop iterations) is the running time of $\mathsf{config}.\textsc{Eval}(\cdot\|\bar{\mathbb{b}})$ (which is $O(2^{s-r} \cdot T_{\mathbb{F}})$) *multiplied by* the number of missses incurred on an LRU cache of size τ' by the sequence of values taken by $\bar{\mathbf{a}}$ at Line 13 over the entire execution of Algorithm 4. Similarly, the total running time of Line 14 is $O(2^{s-r} \cdot T_{\mathbb{F}})$ multiplied by the number of missses incurred on an LRU cache of size τ' by the sequence of values taken by $\bar{\mathbf{a}}'$ at Line 14 over the entire execution of Algorithm 4.

Claim 9.8.1. *The number of misses incurred on an LRU cache of size τ' by the sequence of values for $\bar{\mathbf{a}}$ (resp., $\bar{\mathbf{a}}'$) at Line 13 (resp., Line 14) over the entire execution of Algorithm 4 is at most $288W \cdot \log(S') \cdot Q$.*

The proof of Claim 9.8.1 is similar to (but slightly more complicated than) Claim 9.6.1.

Proof. We bound the number of misses for the sequence of values for $\bar{\mathbf{a}}$. The analysis for $\bar{\mathbf{a}}'$ is similar. Recall that we denote by Q_{LRU} (resp., Q_{ideal}) the number of cache misses incurred on LRU (resp., ideal) cache (see Section 5.2).

Let $\mathfrak{a}$ denote the sequence of values taken by $\bar{\mathbf{a}}$ in the $O(T')$ executions of Line 13 in Algorithm 4. By Theorem 5.10 it holds that:

$$Q_{\mathsf{LRU}}(\mathfrak{a}, \tau') \le 2 \cdot Q_{\mathsf{ideal}}(\mathfrak{a}, \tau'/2). \tag{20}$$

Observe that $\mathfrak{a}$ is produced as follows. In the $\bar{\mathbf{y}}^{th}$ iteration of the main loop, we add values $\bar{\mathbf{a}}$ that are prefixes of all the Tree machine head positions that are either changed between the $\bar{\mathbf{y}}^{th}$ and $(\bar{\mathbf{y}}+1)^{th}$ time steps *or* are "neighbors in the d^{th} direction" of positions that are changed between the $(\bar{\mathbf{y}}+1)^{th}$ and $(\bar{\mathbf{y}}+2)^{th}$ time steps. Overall, this means that at the $\bar{\mathbf{y}}^{th}$ iteration there are at most $O(1)$ positions that could potentially be inserted into the list $\mathfrak{a}$ (here we use the fact that given the position of the Tree machine head at time $\bar{\mathbf{y}}$, there are $O(1)$ possibilities for the head position at time $\bar{\mathbf{y}}+1$. Moreover, this also holds even when considering only prefixes of the head position, see Proposition A.1). As a matter of fact, a loose accounting shows that the above constant is upper bounded by 144.[39]

Let $\mathfrak{u}$ denote r-bit prefixes of the access pattern of $\mathsf{Tree}(M)$ on input x. By the above discussion, the list $\mathfrak{a}$ is a 144-blowup of $\mathfrak{u}$ (see Definition 5.7 for the definition of an α-blowup).

Thus, by Proposition 5.8 it holds that

$$Q_{\mathsf{ideal}}(\mathfrak{a}, \tau'/2) \le 144 \cdot Q_{\mathsf{ideal}}(\mathfrak{u}, \tau'/288). \tag{21}$$

Let $\mathsf{addrs} = (\mathsf{addr}_1, \ldots, \mathsf{addr}_{O(T)})$ denote the combined access pattern of the RAM machine M on input x. Then, combining Proposition 5.8 and Proposition 4.35 we get that the right hand side of Eq. (21) is upper bounded by

$$W \cdot \log(S') \cdot Q_{\mathsf{ideal}}\left((\textsc{Prefix}_r(\mathsf{addr}_1), \ldots, \textsc{Prefix}_r(\mathsf{addr}_{O(T)})), \frac{\tau'}{288W \cdot \log(S')} \right) \tag{22}$$

which in turn is equal to $W \cdot \log(S') \cdot Q$. The claim follows by combining Eqs. (20) to (22). This concludes the proof of Claim 9.8.1. □

This concludes the proof of Proposition 9.7. □

[39] We do not try to optimize this constant here and merely specify it to make our algorithm use concrete constants.

10 No-Signaling Soundness

In this section we prove that the PCP construction from Section 7, with appropriately chosen parameters, is no-signaling sound. Recall that the PCP verifier is actually a "repeated verifier": on input x and statistical security parameter κ it runs a "base verifier" V independently κ times (each time on input x) and accepts if and only if V accepts in all κ executions. We call this repeated verifier $V^{\otimes\kappa}$, and we prove that $V^{\otimes\kappa}$ accepts false assertions with probability that is exponentially small in the security parameter κ.

Lemma 10.1. *Suppose $\mathbb{F}$, κ, δ, and ε are such that*

- $|\mathbb{F}| = \omega(t)$.
- $\delta \leq o(|\mathbb{F}|^{-25\cdot\kappa\cdot t\cdot|\mathbb{F}|^2})$.
- $\varepsilon \geq \omega\left(2^{t+s}\cdot\kappa^3\cdot|\mathbb{F}|^3\cdot t\cdot 2^{-\kappa/|\mathbb{F}|^3}\right)$.

Then, the verifier $V^{\otimes\kappa}$ has ε-soundness against $(k_{\max},\delta)$-NS provers, where $k_{\max} = O(\kappa^2\cdot|\mathbb{F}|^2)$.

Remark 10.2 (Suggested Parameter Setting)**.** *Lemma 10.1 can be used with a wide set of parameters. Still, we find it instructive to suggest a setting of parameters that the reader can keep in mind. We suggest to think of the field size $\mathbb{F}$ as being poly-logarithmic in T (since the verifier runs in time at least $\mathsf{poly}(|\mathbb{F}|)$). We think of κ as being a security parameter since, as long as $\kappa \geq (\log(T))^{\Omega(1)}$, the soundness error ε decreases exponentially in κ. Given this setting of parameters δ is roughly $2^{-\mathsf{poly}(\log(T),\kappa)}$.*

Let $r = r(\kappa) \in [\kappa]$ be a parameter. We write $V^{\geq\kappa-r}$ to denote a verifier which on input x and security parameter κ, runs κ independent copies of the base verifier V (as defined in Section 7) in parallel, and accepts if at least $\kappa - r$ of the copies of V accept.

Following [KRR14], our proof of Lemma 10.1 is composed of two main steps. Our first, and main, step is to show that for $r \ll \kappa$, the "relaxed" verifier $V^{\geq\kappa-r}$ has "weak soundness" – that is, a no-signaling strategy can only convince $V^{\geq\kappa-r}(x)$ to accept false statements with probability bounded away from 1. Note that this falls short of our goal of showing that a no-signaling strategy can only convince the verifier to accept with *negligible* probability. Indeed, our second step is an amplification step (which is taken verbatim from [KRR14]) shows that if the relaxed verifier only accepts with probability bounded away from 1, then the actual verifier $V^{\otimes k}$ accepts false statements with negligible probability.

We start with our main step which shows that the relaxed verifier accepts with probability that is bounded away from 1:

Lemma 10.3. *Suppose that $\mathbb{F}$, δ, κ, r, ε are such that:*

- $\delta \leq \varepsilon$.
- $r = o(\kappa/|\mathbb{F}|^2)$.
- $\kappa \geq \log(1/\varepsilon)$.
- $|\mathbb{F}| = \omega(t)$.

- $\varepsilon = o\left(\frac{1}{2^{t+s} \cdot \kappa^3 \cdot |\mathbb{F}|^2}\right)$.

*Then, the relaxed verifier $V^{\geq \kappa - r}$ has $(1-\varepsilon)$-soundness against $(k_{\max}, \delta)$-*NS *provers, where $k_{\max} = O(\kappa^2 \cdot |\mathbb{F}|^2)$.*

Remark 10.4. *The reader should think of δ and ε as being very small (i.e., on the order of $1/\mathsf{poly}(T)$). We emphasize that our restriction that $\delta \leq \varepsilon$ exists only to simplify the statements of our technical lemmas; in particular, this allows us to reduce the number of parameters in bounding the probabilities of various "bad" events.*

As noted above, our main step is proving Lemma 10.3, which we do in Section 10.1. The second step, establishing Lemma 10.1 based on Lemma 10.3, is then done in Section 10.2.

10.1 Weak Soundness of the Relaxed Verifier - Proving Lemma 10.3

Let $\mathbb{F}$, δ, κ, r, ε be as in the statement of Lemma 10.3. Throughout this section we fix a $(k_{\max}, \delta)$-NS prover P^* which makes $V^{\geq \kappa - r}$ accept with probability $1 - \varepsilon$.

For any set $Q \subseteq \mathbb{F}^{t+s}$ of at most $k_{\max}/(\kappa \cdot |\mathbb{F}|)$ points, let $\mathcal{Q}(Q)$ denote the following probabilistic experiment.

1. For each $q \in Q$ and each $j \in [\kappa]$, sample a uniformly random line $L_q^{(j)} : \mathbb{F} \to \mathbb{F}^{t+s}$ such that $L_q^{(j)}(0) = q$ (e.g., by sampling a random point $\bar{z} \leftarrow \mathbb{F}^{t+s}$ and taking the unique line L such that $L(0) = q$ and $L(1) = \bar{z}$).

2. Define $\bar{Q} = \{L_q^{(j)}(\alpha)\}_{q \in Q, j \in [\kappa], \alpha \in \mathbb{F} \setminus \{0\}}$.

3. Sample $A_{\bar{Q}} \leftarrow P^*(\bar{Q})$ and output $A_{\bar{Q}} : \bar{Q} \to \mathbb{F}$ (note that the distribution $P^*(\bar{Q})$ is well defined since $|\bar{Q}| \leq k_{\max}$), where by $P^*(\bar{Q})$ we refer to the part of the PCP that corresponds to $\hat{X}$.

Let $r' = O(|\mathbb{F}| \cdot r)$ be a parameter that will be fixed in Lemma 10.5. For each $q \in Q$, let v_q denote the (unique) value of $v \in \mathbb{F}$ for which a large majority of the κ lines that pass through q are assigned values that interpolate to v. More specifically, we set v_q to $v \in \mathbb{F}$ if for all but r' values of $j \in [\kappa]$ it holds that

$$\mathsf{Interp}_d\left(\left\{\alpha \mapsto A_{\bar{Q}}(L_q^{(j)}(\alpha))\right\}_{\alpha \in \mathbb{F} \setminus \{0\}}\right)(0) = v,$$

and we set v_q to $\perp$ if no such v exists (although we will soon see that this event is unlikely to happen, see Lemma 10.5).

We also denote by $\mathsf{Correct}(q)$ the event that $v_q = \hat{X}_{\mathsf{correct}}(q)$, where by $\hat{X}_{\mathsf{correct}}$ we denote the low-degree extension of the correct computation transcript for $\mathsf{Tree}(M)$ on the input x (as defined in Section 7). We extend this notation to any subset $W \subseteq Q$, writing $\mathsf{Correct}(W)$ to mean $\wedge_{q \in W} \mathsf{Correct}(q)$.

10.1.1 Lemmas from Previous Work

We recall several lemmas from [KRR14]. Recall that we have fixed a $(k_{\max}, \delta)$-NS prover P^* that makes $V^{\geq \kappa - r}(x)$ accept with probability $1 - \varepsilon$.

Lemma 10.5 (Consistency of $\hat{X}$, [KRR14, Lemma 7.27])**.** *There exists $r' = O(|\mathbb{F}| \cdot r)$ and $\varepsilon' = O(|\mathbb{F}| \cdot \varepsilon)$ such that for any $\bar{\mathbb{u}} \in \mathbb{F}^{t+s}$,*

$$\Pr_{\mathcal{Q}(\{\bar{\mathbb{u}}\})}\left[v_{\bar{\mathbb{u}}} = \perp\right] \leq \varepsilon'.$$

The following proposition follows from the definition of the polynomial $P_0^{(\mathsf{enc})}$ and its corresponding sumcheck polynomials, together with the analysis in [KRR14].

Proposition 10.6 ($\hat{X}$ is a valid symbol)**.** *There exists $\varepsilon' = O(t \cdot |\mathbb{F}| \cdot \varepsilon)$ such that for any $\bar{\mathbf{u}} \in \{0,1\}^{t+s}$, it holds that*

$$\Pr_{\mathcal{Q}(\{\bar{\mathbf{u}}\})}\left[v_{\bar{\mathbf{u}}} \notin \bar{\Gamma}\right] \leq \varepsilon',$$

where $\bar{\Gamma} \subseteq \mathbb{F}$ is the extended tape alphabet of $\mathsf{Tree}(M)$ (see Definition 4.31).

Lemma 10.7 (Local Consistency, analogous to [KRR14, Lemma 7.30])**.** *There exists $\varepsilon' = O(t \cdot |\mathbb{F}| \cdot \varepsilon)$ such that for every $\bar{\mathbf{y}} \in \{0, \dots, 2^t - 2\}$ and $\bar{\mathbf{z}} \in \{0,1\}^s$ the following holds. Let $W \subseteq \{0,1\}^s$ be a set such that $\mathcal{C}_{\bar{\mathbf{y}}+1}(\bar{\mathbf{z}})$ is determined by $\{\mathcal{C}_{\bar{\mathbf{y}}}(\bar{\mathbf{w}})\}_{\bar{\mathbf{w}} \in W}$ (as in Definition 7.3). Then,*

$$\Pr_{\mathcal{Q}((\{\bar{\mathbf{y}}\} \times W) \cup (\{(\bar{\mathbf{y}}+1, \bar{\mathbf{z}})\}))}\left[\neg \mathsf{Correct}\big((\bar{\mathbf{y}}+1, \bar{\mathbf{z}})\big) \wedge \mathsf{Correct}\big(\{\bar{\mathbf{y}}\} \times W\big)\right] \leq \varepsilon'.$$

The proof of Lemma 10.7 is analogous to that of [KRR14, Lemma 7.30] together with an application of Proposition 10.6.

10.1.2 From Average-Case to Worst-Case Correctness within a Layer

The following central proposition shows that if a small set of *random* points in some layer $\bar{\mathbf{y}}$ are "read" correctly than any *specific* point $\bar{\mathbb{z}}$ in layer $\bar{\mathbf{y}}$ will be read correctly as well.

Proposition 10.8. *There exists $\varepsilon' = O(\kappa^2 \cdot |\mathbb{F}|^2 \cdot \varepsilon)$ such that for every $\bar{\mathbf{y}} \in \{0,1\}^t$ and $\bar{\mathbb{z}} \in \{\bar{\mathbf{y}}\} \times \mathbb{F}^s$:*

$$\Pr_{\bar{\mathbb{u}}_1, \dots, \bar{\mathbb{u}}_\kappa \leftarrow \{\bar{\mathbf{y}}\} \times \mathbb{F}^s}\left[\neg \mathsf{Correct}(\bar{\mathbb{z}}) \wedge \mathsf{Correct}(\bar{\mathbb{u}}_1) \wedge \dots \wedge \mathsf{Correct}(\bar{\mathbb{u}}_\kappa)\right] \leq \varepsilon',$$

where the probability is also over $\mathcal{Q}(\{\bar{\mathbb{u}}_1, \dots, \bar{\mathbb{u}}_\kappa, \bar{\mathbb{z}}\})$ (over which the events $\mathsf{Correct}(\cdot)$ are defined).

Proof. Fix $\bar{\mathbb{z}} \in \{\bar{\mathbf{y}}\} \times \mathbb{F}^s$, let $\mathcal{Q}'(\bar{\mathbb{z}})$ be the following probabilistic experiment.

1. For each $i \in [\kappa]$:
   (a) Sample $\bar{\mathbb{u}}_i \leftarrow \{\bar{\mathbf{y}}\} \times \mathbb{F}^s$ and an "evaluation point" $\alpha_i \leftarrow \mathbb{F} \setminus \{0\}$. We emphasize that the distribution of the points $\bar{\mathbb{u}}_i$ is uniform *within layer* $\bar{\mathbf{y}}$ (but is *not* uniform in $\mathbb{F}^{t+s}$).
   (b) For each $j \in [\kappa]$, sample i.i.d. uniformly random lines $L_{\bar{\mathbb{u}}_i}^{(j)} : \mathbb{F} \to \mathbb{F}^{t+s}$ such that $L_{\bar{\mathbb{u}}_i}^{(j)}(0) = \bar{\mathbb{u}}_i$.
2. For each $j \in [\kappa]$, sample a uniformly random line $L_{\bar{\mathbb{z}}}^{(j)} : \mathbb{F} \to \mathbb{F}^{t+s}$ such that $L_{\bar{\mathbb{z}}}^{(j)}(0) = \bar{\mathbb{z}}$.

3. For each $i, j \in [\kappa]$, let $M_i^{(j)} : \mathbb{F}^2 \to \mathbb{F}^{t+s}$ denote the (unique) bilinear mapping such that $M_i^{(j)}(0, \cdot) = L_{\bar{\mathbb{z}}}^{(j)}(\cdot)$ and $M_i^{(j)}(\alpha_i, \cdot) = L_{\bar{\mathbb{u}}_i}^{(j)}(\cdot)$.[40]

4. Define a set of queries $\bar{Q} = \left\{ M_i^{(j)}(\alpha, \beta) \right\}_{\substack{i,j \in [\kappa] \\ \alpha,\beta \in \mathbb{F}}}$, and sample $A_{\bar{Q}} \leftarrow P^*(\bar{Q})$, where $A_{\bar{Q}} : \bar{Q} \to \mathbb{F}$. Note that $P^*(\bar{Q})$ is well defined since $|\bar{Q}| \leq \kappa^2 \cdot |\mathbb{F}|^2 \leq k_{\max}$.

5. Output $A_{\bar{Q}}$.

For $q \in \{\bar{\mathbb{z}}, \bar{\mathbb{u}}_1, \ldots, \bar{\mathbb{u}}_\kappa\}$, define the random variables v'_q and the event $\mathsf{Correct}'(q)$ similarly to the definitions of v_q and $\mathsf{Correct}(q)$, but with respect to the distribution $\mathcal{Q}'$ rather than $\mathcal{Q}$, and with respect to the lines $\{L_q^{(j)}\}_{j \in [\kappa]}$.

Using the fact that P^* is non-signaling, to prove Proposition 10.8 it suffices to show that

$$\Pr_{\mathcal{Q}'(\mathbf{z})} \left[\neg \mathsf{Correct}'(\bar{\mathbb{z}}) \wedge \mathsf{Correct}'(\bar{\mathbb{u}}_1) \wedge \cdots \wedge \mathsf{Correct}'(\bar{\mathbb{u}}_\kappa) \right] \leq O(t \cdot \kappa^2 \cdot |\mathbb{F}|^2 \cdot \varepsilon). \tag{23}$$

Define the "good" event $G = G_1 \wedge \cdots \wedge G_\kappa$, where, for every $i \in [\kappa]$, the event G_i is defined as:

- $v'_{\bar{\mathbb{z}}} \neq \bot$ and $v'_{\bar{\mathbb{u}}_i} \neq \bot$ for all $i \in [\kappa]$.
- There exists a degree $t + s$ polynomial $F_i : \mathbb{F} \to \mathbb{F}$ such that for all $\alpha \in \mathbb{F}$,

$$\left| \left\{ j \in [\kappa] : \mathsf{Interp}_d \left(\left\{ \beta \mapsto A_{\bar{Q}}(M_i^{(j)}(\alpha, \beta)) \right\}_{\beta \in \mathbb{F} \setminus \{0\}} \right) (0) = F_i(\alpha) \right\} \right| \geq \kappa - r'.$$

  Informally, this says that for most j, the interpolated values for $A_{\bar{Q}}(M_i^{(j)}(\alpha, 0))$ are a low-degree polynomial F_i of α which does not depend on j (intuitively, in the honest case it should be that $F_i(0) = \hat{X}(\bar{\mathbb{z}})$ and $F_i(\alpha_i) = \hat{X}(\bar{\mathbb{u}}_i)$).

Claim 10.8.1.

$$\Pr_{\mathcal{Q}'(\bar{\mathbb{z}})} [\neg G] \leq O(\kappa^2 \cdot |\mathbb{F}|^2 \cdot \varepsilon).$$

Proof. Fix $i \in [\kappa]$. We show that $\Pr_{\mathcal{Q}'(\bar{\mathbb{z}})} [\neg G_i] \leq O(t \cdot \kappa \cdot |\mathbb{F}|^2 \cdot \varepsilon)$ and the claim will follow from a union bound over $i \in [\kappa]$.

For each $\beta \in \mathbb{F} \setminus \{0\}$, it holds that $M_i^{(j)}(\cdot, \beta)$ is a *uniformly random* line (since it is a line passing through the two independently uniform points $M_i^{(j)}(0, \beta)$ and $M_i^{(j)}(\alpha_i, \beta)$). Thus, by the verifier's low-degree test and no-signaling, with all but $|\mathbb{F}| \cdot (\varepsilon + \delta)$ probability, for each $\beta \in \mathbb{F} \setminus \{0\}$, there are at least $\kappa - r$ values of $j \in [\kappa]$ for which

$$\mathsf{Deg}\left(A_{\bar{Q}} \circ M_i^{(j)}(\cdot, \beta) \right) \leq t + s. \tag{24}$$

[40]To see that such a bilinear mapping exists, observe that $M_i^{(j)}(0, \cdot)$ and $M_i^{(j)}(\alpha_i, \cdot)$ are linear functions. For any $\alpha \in \mathbb{F}$, the function $M_i^{(j)}(\alpha, \cdot)$ is derived by interpolation (which is a linear operation) and is therefore a linear function. Lastly, for any fixed $\beta \in \mathbb{F}$, the function $M_i^{(j)}(\cdot, \beta)$ is linear since it is obtained by interpolating at two points. Thus, $M_i^{(j)}$ is bilinear.

By Lemma 10.5 and using the no-signaling condition, it holds that with all but $O(|\mathbb{F}|\cdot(|\mathbb{F}|\cdot\varepsilon+\delta)) = O(|\mathbb{F}|^2\cdot\varepsilon)$ probability, for each $\alpha\in\mathbb{F}$ there exists a value $F_i(\alpha)$ such that for at least $\kappa-r$ values of $j\in[\kappa]$,

$$\mathsf{Interp}\left(\left\{\beta\mapsto A_{\bar{Q}}(M_i^{(j)}(\alpha,\beta))\right\}_{\beta\in\mathbb{F}\setminus\{0\}}\right)(0)=F_i(\alpha). \tag{25}$$

Thus, by Eqs. (24) and (25) and a union bound, with all but $O(|\mathbb{F}|^2\cdot\varepsilon)$ probability, both of the following events happen:

1. For every $\beta\in\mathbb{F}\setminus\{0\}$, there are at least $\kappa-r'$ values of $j\in[\kappa]$ for which $\mathsf{Deg}(A_{\bar{Q}}\circ M_i^{(j)}(\cdot,\beta))\le t+s$.

2. For each $\alpha\in\mathbb{F}$ there exists a value $F_i(\alpha)$ such that for at least $\kappa-r'$ values of $j\in[\kappa]$ it holds that $\mathsf{Interp}\left(\left\{\beta\mapsto A_{\bar{Q}}(M_i^{(j)}(\alpha,\beta))\right\}_{\beta\in\mathbb{F}\setminus\{0\}}\right)(0)=F_i(\alpha)$.

In particular, this means that with the same probability, there exists a function $F_i:\mathbb{F}\to\mathbb{F}$ such that for at least $\kappa-2|\mathbb{F}|\cdot r'$ values of $j\in[\kappa]$ the following condition holds:

$$\forall(\alpha,\beta)\in\mathbb{F}\times(\mathbb{F}\setminus\{0\}):\quad \begin{array}{c}\mathsf{Deg}(A_{\bar{Q}}\circ M_i^{(j)}(\cdot,\beta))\le t+s\text{ and}\\ \mathsf{Interp}\left(\left\{\beta\mapsto A_{\bar{Q}}(M_i^{(j)}(\alpha,\beta))\right\}_{\beta\in\mathbb{F}\setminus\{0\}}\right)(0)=F_i(\alpha).\end{array} \tag{26}$$

Assuming that F_i as above exists, we argue that it must have degree $t+s$. To see this, fix any $j\in[\kappa]$ such that Eq. (26) holds. Consider a $|\mathbb{F}|\times(|\mathbb{F}|-1)$ matrix whose $(\alpha,\beta)^{th}$ entry is $A_{\bar{Q}}(M_i^{(j)}(\alpha,\beta))$ for $\beta\neq0$. Observe that all columns of this matrix have degree $t+s$. On the other hand, by Lagrange interpolation there exist coefficients c_β for $\beta\neq0$ such that $F_i(\alpha)=\sum_{\beta\neq0}c_\beta\cdot A_{\bar{Q}}(M_i^{(j)}(\alpha,\beta))$. In other words, the function F_i can be expressed as a linear combination of the columns. It follows that F_i also has degree at most $t+s$.

Thus, with all but $O(|\mathbb{F}|^2\cdot\varepsilon)$ probability, there exists a degree $t+s$ polynomial $F_i:\mathbb{F}\to\mathbb{F}$ such that for all $\alpha\in\mathbb{F}$,

$$\left|\left\{j\in[\kappa]\,:\,\mathsf{Interp}\left(\left\{\beta\mapsto A_{\bar{Q}}(M_i^{(j)}(\alpha,\beta))\right\}_{\beta\in\mathbb{F}\setminus\{0\}}\right)(0)=F_i(\alpha)\right\}\right|\ge\kappa-2|\mathbb{F}|\cdot r'. \tag{27}$$

Note that Eq. (27) is almost what we want, except that we need the right hand side to be $\kappa-r'$. To obtain this observe that by Lemma 10.5, the fact that P^* is NS and a union bound, it also holds that with all but $O(|\mathbb{F}|^2\cdot\varepsilon)$ probability, for all $\alpha\in\mathbb{F}$ there exists a value v_α such that

$$\left|\left\{j\in[\kappa]\,:\,\mathsf{Interp}\left(\left\{\beta\mapsto A_{\bar{Q}}(M_i^{(j)}(\alpha,\beta))\right\}_{\beta\in\mathbb{F}\setminus\{0\}}\right)(0)=v_\alpha\right\}\right|\ge\kappa-r'. \tag{28}$$

Using the fact[41] that $2|\mathbb{F}|\cdot r'=O(|\mathbb{F}|^2\cdot r)$ and $\kappa=\omega(|\mathbb{F}|^2\cdot r)$, Eqs. (27) and (28) and a union bound yield that with all but $O(|\mathbb{F}|^2\cdot\varepsilon)$ probability, there exists a polynomial $F_i:\mathbb{F}\to\mathbb{F}$ of degree at most $t+s$ such that for all $\alpha\in\mathbb{F}$,

$$\left|\left\{j\in[\kappa]\,:\,\mathsf{Interp}\left(\left\{\beta\mapsto A_{\bar{Q}}(M_i^{(j)}(\alpha,\beta))\right\}_{\beta\in\mathbb{F}\setminus\{0\}}\right)(0)=F_i(\alpha)\right\}\right|\ge\kappa-r'. \tag{29}$$

[41] Loosely speaking, Eq. (27) shows that a relatively large fraction of j's agree with F_i where Eq. (28) shows that an even larger fraction agrees with one another. This implies that even the larger fraction actually agrees with F_i.

The claim follows by applying Lemma 10.5 $\kappa+1$ times (once for $\bar{\mathbb{z}}$ and once for each $\bar{\mathbb{u}}_i$), a union bound, and the fact that P^* is no-signaling. This concludes the proof of Claim 10.8.1. $\square$

Define the event $\mathsf{Correct}'(\vec{\mathbb{u}}) = \mathsf{Correct}'(\bar{\mathbb{u}}_1) \wedge \cdots \wedge \mathsf{Correct}'(\bar{\mathbb{u}}_\kappa)$.

Claim 10.8.2.

$$\Pr_{\mathcal{Q}'(\bar{\mathbb{z}})}\left[\neg\mathsf{Correct}'(\bar{\mathbb{z}}) \wedge \mathsf{Correct}'(\vec{\mathbb{u}}) \wedge G\right] \le 2^{-\kappa}.$$

Proof. By the principle of deferred decisions, one can think of the distribution $\mathcal{Q}'(\bar{\mathbb{z}})$ as being generated as follows. For every $j \in [\kappa]$, choose a uniformly random line $L^{(j)}_{\bar{\mathbb{z}}} : \mathbb{F} \to \mathbb{F}^{t+s}$ such that $L^{(j)}_{\bar{\mathbb{z}}}(0) = \bar{\mathbb{z}}$. For every $i, j \in [\kappa]$, let $M^{(j)}_i$ be a random bilinear mapping such that (1) $M^{(j)}_i(0,\cdot) = \mathcal{L}^{(j)}_{\bar{\mathbb{z}}}(\cdot)$ and (2) the line $M^{(j)}_i(\cdot,0)$ is entirely contained within $\{\bar{\mathbf{y}}\} \times \mathbb{F}^s$ (note that the first condition refers to the line $M^{(j)}_i(0,\cdot)$ whereas the latter refers to $M^{(j)}_i(\cdot,0)$). Only then, choose $\alpha_i \in \mathbb{F} \setminus \{0\}$ at random and define $\bar{\mathbb{u}}_i = M^{(j)}_i(\alpha_i, 0)$. We shall use this view in the following analysis.

To prove the claim it suffices to bound $\Pr_{\mathcal{Q}'(\bar{\mathbb{z}})}\left[\mathsf{Correct}'(\vec{\mathbb{u}}) \;\middle|\; \neg\mathsf{Correct}'(\bar{\mathbb{z}}) \wedge G\right]$. Fix a choice of the bilinear maps $M^{(j)}_i$ and a sample from the distribution $A_{\bar{Q}}$ such that the events G and $\neg\mathsf{Correct}'(\bar{\mathbb{z}})$ occur. Observe that by the foregoing view, α_i remains uniformly random even conditioned on $M^{(j)}_i$.

Since G occurs, we know that for each $i \in [\kappa]$, there exists a degree $t+s$ polynomial $F_i : \mathbb{F} \to \mathbb{F}$ such that for all $\alpha \in \mathbb{F}$, and all but r' values of $j \in [\kappa]$:

$$\mathsf{Interp}\left(\left\{\beta \mapsto A_{\bar{Q}}(M^{(j)}_i(\alpha,\beta))\right\}_{\beta \in \mathbb{F}\setminus\{0\}}\right)(0) = F_i(\alpha). \tag{30}$$

We argue that for every $i \in [\kappa]$ it holds that $F_i(0) \neq \hat{X}_{\mathsf{correct}}(\bar{\mathbb{z}})$. This is because on the one hand, for all but r' values of $j \in [\kappa]$ Eq. (30) holds with $\alpha = 0$, but on the other hand it also holds for all but r' values of $j \in [\kappa]$ that $\mathsf{Interp}\left(\left\{\beta \mapsto A_{\bar{Q}}(M^{(j)}_i(0,\beta))\right\}_{\beta \in \mathbb{F}\setminus\{0\}}\right)(0) \neq \hat{X}_{\mathsf{correct}}(\bar{\mathbb{z}})$. Since $r' < \kappa/2$, this implies that $F_i(0) \neq \hat{X}_{\mathsf{correct}}(\bar{\mathbb{z}})$. Thus, the degree $t+s$ polynomials $F_i(\cdot)$ and $\hat{X}(M^{(j)}_i(\cdot,0))$ are distinct.

Since G occurs, the event $\mathsf{Correct}'(\bar{\mathbb{u}}_i)$ occurs if and only if $F_i(\alpha_i) = \hat{X}(\bar{\mathbb{u}}_i)$. Since $F_i(\cdot)$ and $\hat{X}(M^{(j)}_i(\cdot,0))$ are distinct degree $t+s$ polynomials, with probability at least $1 - \frac{t+s}{|\mathbb{F}|-1}$ over the choice of $\alpha_i \in \mathbb{F} \setminus \{0\}$, it holds that $F_i(\alpha_i) \neq \hat{X}(M^{(j)}_i(\alpha_i,0)) = \hat{X}(\bar{\mathbb{u}}_i)$. Thus, the event $\mathsf{Correct}'(\bar{\mathbb{u}}_i)$ occurs with probability at most $\frac{t+s}{|\mathbb{F}|-1} \le 1/2$. By the above discussion the events $\{\mathsf{Correct}'(\bar{\mathbb{u}}_i)\}_i$ are independent, conditioned on the choice of $\{M^{(j)}_i\}_{i,j}$ and $A_{\bar{Q}}$ (which fully determine the events $\neg\mathsf{Correct}'(\bar{\mathbb{z}})$ and G). Thus,

$$\begin{aligned}\Pr_{\mathcal{Q}'(\bar{\mathbb{z}})}\left[\neg\mathsf{Correct}'(\bar{\mathbb{z}})\wedge\mathsf{Correct}'(\vec{\mathbb{u}})\wedge G\right] &\leq \Pr_{\mathcal{Q}'(\bar{\mathbb{z}})}\left[\mathsf{Correct}'(\vec{\mathbb{u}})\;\middle|\;\neg\mathsf{Correct}'(\bar{\mathbb{z}})\wedge G\right]\\ &= \mathop{\mathbf{E}}_{\substack{\{M_i^{(j)}\}_{i,j\in[\kappa]},A_{\bar{Q}}\leftarrow P^*(\bar{Q})\\ \text{s.t. }\neg\mathsf{Correct}'(\bar{\mathbb{z}})\wedge G}}\left[\Pr_{\{\alpha_i\}_{i\in[\kappa]}}\left[\mathsf{Correct}'(\vec{\mathbb{u}})\right]\right]\\ &= \mathop{\mathbf{E}}_{\substack{\{M_i^{(j)}\}_{i,j\in[\kappa]},A_{\bar{Q}}\leftarrow P^*(\bar{Q})\\ \text{s.t. }\neg\mathsf{Correct}'(\bar{\mathbb{z}})\wedge G}}\left[\prod_{i\in[\kappa]}\Pr_{\alpha_i}\left[\mathsf{Correct}'(\bar{\mathbb{u}}_i)\right]\right]\\ &\leq 2^{-\kappa}.\end{aligned}$$

This concludes the proof of Claim 10.8.2. □

Combining Claim 10.8.1 and Claim 10.8.2 we have that:

$$\begin{aligned}\Pr\left[\neg\mathsf{Correct}'(\bar{\mathbb{z}})\wedge\mathsf{Correct}'(\vec{\mathbb{u}})\right] &\leq \Pr\left[\neg\mathsf{Correct}'(\bar{\mathbb{z}})\wedge\mathsf{Correct}'(\vec{\mathbb{u}})\wedge G\right]+\Pr[\neg G]\\ &\leq 2^{-\kappa}+O(\kappa^2\cdot|\mathbb{F}|^2\cdot\varepsilon)\\ &\leq O(\kappa^2\cdot|\mathbb{F}|^2\cdot\varepsilon).\end{aligned}$$

This concludes the proof of Proposition 10.8. □

10.1.3 Average-Case Correctness for Subsequent Layers

The following proposition extends Proposition 10.8 by showing that if random points in some layer $\bar{\mathbf{y}}$ are read correctly, then a given *fixed* point in layer $\bar{\mathbf{y}}+1$ will also be read correctly (whereas Proposition 10.8 considered fixed points also in layer $\bar{\mathbf{y}}$).

Proposition 10.9. *There exists* $\varepsilon' = O(2^s\cdot\kappa^2\cdot|\mathbb{F}|^2\cdot\varepsilon)$ *such that for any* $\bar{\mathbf{y}}\in\{0,\dots,2^t-2\}\subseteq\{0,1\}^t$ *and any* $\bar{\mathbb{z}}\in\{\bar{\mathbf{y}}+1\}\times\mathbb{F}^s$,

$$\Pr_{\substack{\bar{\mathbb{u}}_1,\dots,\bar{\mathbb{u}}_\kappa\leftarrow\{\bar{\mathbf{y}}\}\times\mathbb{F}^s\\ \mathcal{Q}(\{\bar{\mathbb{z}},\bar{\mathbb{u}}_1,\dots,\bar{\mathbb{u}}_\kappa\})}}\left[\neg\mathsf{Correct}(\bar{\mathbb{z}})\wedge\mathsf{Correct}(\bar{\mathbb{u}}_1)\wedge\cdots\wedge\mathsf{Correct}(\bar{\mathbb{u}}_\kappa)\right]\leq\varepsilon'. \tag{31}$$

Proof. For every $\ell\in\{0,\dots,s\}$ define

$$p_{\bar{\mathbf{y}},\ell}\overset{\mathsf{def}}{=}\max_{\bar{\mathbb{z}}\in\{\bar{\mathbf{y}}+1\}\times\mathbb{F}^\ell\times\{0,1\}^{s-\ell}}\Pr_{\substack{\vec{\mathbb{u}}\leftarrow(\{\bar{\mathbf{y}}\}\times\mathbb{F}^s)^\kappa\\ \mathcal{Q}(\{\bar{\mathbb{z}},\bar{\mathbb{u}}_1,\dots,\bar{\mathbb{u}}_\kappa\})}}\left[\neg\mathsf{Correct}(\bar{\mathbb{z}})\wedge\mathsf{Correct}(\bar{\mathbb{u}}_1)\wedge\cdots\wedge\mathsf{Correct}(\bar{\mathbb{u}}_\kappa)\right].$$

Fix $\bar{\mathbf{y}}$. We will show, by induction on $0\leq\ell\leq s$, a bound on $p_{\bar{\mathbf{y}},\ell}$ that implies the desired bound on $p_{\bar{\mathbf{y}},s}$ (which corresponds to the statement of Proposition 10.9). We start with the base case.

Claim 10.9.1.

$$p_{\bar{\mathbf{y}},0}\leq O(\kappa^2\cdot|\mathbb{F}|^2\cdot\varepsilon).$$

Proof. Let $\bar{\mathbf{z}} \in \{\bar{\mathbf{y}}+1\} \times \{0,1\}^s$. Denote by $\textsc{Suffix}_s : \{0,1\}^{t+s} \to \{0,1\}^s$ the function that outputs the last s bits of its input, and let $W \subseteq \{\bar{\mathbf{y}}\} \times \{0,1\}^s$ be such that that $\mathcal{C}_{\bar{\mathbf{y}}+1}(\textsc{Suffix}_s(\bar{\mathbf{z}}))$ is determined by $\{\mathcal{C}_{\bar{\mathbf{y}}}(\textsc{Suffix}_s(\bar{\mathbf{u}}))\}_{\bar{\mathbf{u}}\in W}$ as in Definition 7.3. Then, by Lemma 10.7 we have that:

$$\Pr_{\mathcal{Q}(W\cup\{\bar{\mathbf{z}}\})}\left[\neg\mathsf{Correct}(\bar{\mathbf{z}}) \wedge \mathsf{Correct}(W)\right] \le O(t\cdot|\mathbb{F}|\cdot\varepsilon). \tag{32}$$

Since $|W| = O(1)$, by Proposition 10.8, a union bound, and the no-signaling condition, it holds that:

$$\Pr_{\substack{\vec{\mathbb{u}}\leftarrow(\{\bar{\mathbf{y}}\}\times\mathbb{F}^s)^\kappa \\ \mathcal{Q}(W\cup\{\bar{\mathbb{u}}_1,\dots,\bar{\mathbb{u}}_\kappa\})}}\left[\neg\mathsf{Correct}(W) \wedge \mathsf{Correct}(\bar{\mathbb{u}}_1)\wedge\cdots\wedge\mathsf{Correct}(\bar{\mathbb{u}}_\kappa)\right] \le O(\kappa^2\cdot|\mathbb{F}|^2\cdot\varepsilon). \tag{33}$$

By Eqs. (32) and (33), the no-signaling condition, and elementary probability,

$$\begin{aligned}\Pr_{\substack{\vec{\mathbb{u}}\leftarrow(\{\bar{\mathbf{y}}\}\times\mathbb{F}^s)^\kappa \\ \mathcal{Q}(W\cup\{\mathbf{z},\mathbb{u}_1,\dots,\mathbb{u}_\kappa\})}}\left[\neg\mathsf{Correct}(\mathbf{z}) \wedge \mathsf{Correct}(\vec{\mathbb{u}})\right] &\le \Pr_{\mathcal{Q}(W\cup\{\mathbf{z}\})}\left[\neg\mathsf{Correct}(\mathbf{z})\wedge\mathsf{Correct}(W)\right] \\ &\quad+ \Pr_{\substack{\vec{\mathbb{u}}\leftarrow(\{\bar{\mathbf{y}}\}\times\mathbb{F}^s)^\kappa \\ \mathcal{Q}(W\cup\{\bar{\mathbb{u}}_1,\dots,\bar{\mathbb{u}}_\kappa\})}}\left[\neg\mathsf{Correct}(W)\wedge\mathsf{Correct}(\vec{\mathbf{u}})\right] + 2\delta \\ &\le O(t\cdot|\mathbb{F}|\cdot\varepsilon) + O(\kappa^2\cdot|\mathbb{F}|^2\cdot\varepsilon) + 2\delta \\ &\le O(\kappa^2\cdot|\mathbb{F}|^2\cdot\varepsilon). \end{aligned} \tag{34}$$

The claim now follows from the no-signaling condition. ☐

We proceed to the inductive argument, which is summarized in the following claim.

Claim 10.9.2. *There exists $\varepsilon'' = O(|\mathbb{F}|\cdot\varepsilon)$ such that for every $\ell \in \{1,\dots,s\}$ it holds that*

$$p_{\bar{\mathbf{y}},\ell} \le 2\cdot p_{\bar{\mathbf{y}},\ell-1} + \varepsilon''.$$

Proof. For $\ell\in[s]$, $\sigma\in\{0,1\}$ and $\bar{\mathbb{z}}\in\{\bar{\mathbf{y}}+1\}\times\mathbb{F}^\ell\times\{0,1\}^{s-\ell}$, define the function

$$\pi_{\ell,\sigma}(\bar{\mathbb{z}}) \overset{\mathsf{def}}{=} (\mathbb{z}_1,\dots,\mathbb{z}_{t+\ell-1},\sigma,\mathbb{z}_{t+\ell+1},\dots,\mathbb{z}_{t+s}) \in \{\bar{\mathbf{y}}+1\}\times\mathbb{F}^{\ell-1}\times\{0,1\}^{s-\ell+1},$$

which replaces $\mathbb{z}_\ell$, the $(t+\ell)^{th}$ component of $\bar{\mathbb{z}}$, with the value σ (which is in $\{0,1\}$). Consider the following probabilistic experiment.

1. Sample $\vec{\mathbb{u}} = (\bar{\mathbb{u}}_1,\dots,\bar{\mathbb{u}}_\kappa) \leftarrow (\{\bar{\mathbf{y}}\}\times\mathbb{F}^s)^\kappa$.
2. For each $i,j\in[\kappa]$, sample a uniformly random line $L^{(j)}_{\bar{\mathbb{u}}_i} : \mathbb{F}\to\mathbb{F}^{t+s}$ such that $L^{(j)}_{\bar{\mathbb{u}}_i}(0) = \bar{\mathbb{u}}_i$.
3. For each $j\in[\kappa]$, sample a uniformly random line $L^{(j)}_{\bar{\mathbb{z}}} : \mathbb{F}\to\mathbb{F}^{t+s}$ such that $L^{(j)}_{\bar{\mathbb{z}}}(0) = \bar{\mathbb{z}}$.
4. Define $\bar{Q} \overset{\mathsf{def}}{=} \left\{L^{(j)}_{\bar{\mathbb{u}}_i}(\alpha)\right\}_{i,j\in[\kappa],\alpha\in\mathbb{F}\setminus\{0\}} \cup \left\{L^{(j)}_{\bar{\mathbb{z}}}(\alpha)+\beta\cdot\bar{\mathbf{e}}_{t+\ell}\right\}_{j\in[\kappa],\alpha\in\mathbb{F}\setminus\{0\},\beta\in\mathbb{F}}$, where $\bar{\mathbf{e}}_{t+\ell}$ is the $(t+\ell)^{th}$ standard basis vector (i.e., it has 1 in the $(t+\ell)^{th}$ coordinate and 0 elsewhere).
5. Sample $X^*\leftarrow P^*(\bar{Q})$ (observe that $|\bar{Q}| \le \kappa^2\cdot|\mathbb{F}| + \kappa\cdot|\mathbb{F}|^2 \le k_{\max}$ and so $P^*(\bar{Q})$ is well defined).

For $q \in \{\bar{\mathbb{z}}, \bar{\mathbb{u}}_1, \ldots, \bar{\mathbb{u}}_\kappa\}$, define v_q and $\mathsf{Correct}(q)$ as in $\mathcal{Q}\big(\{\bar{\mathbb{z}}\} \cup \{\bar{\mathbb{u}}_i\}_{i\in[\kappa]}\big)$. For $q \in \{\pi_{\ell,\sigma}(\bar{\mathbb{z}})\}_{\sigma\in\{0,1\}}$, define v_q and $\mathsf{Correct}(q)$ with respect to the lines $L^{(j)}_{\pi_{\ell,\sigma}(\bar{\mathbb{z}})} \stackrel{\mathsf{def}}{=} L^{(j)}_{\bar{\mathbb{z}}} + (\sigma - \mathbb{z}_{t+\ell}) \cdot \bar{\mathbf{e}}_{t+\ell}$ (it is easy to check that $L^{(j)}_{\pi_{\ell,\sigma}(\bar{\mathbb{z}})}(\mathbb{F} \setminus \{0\})$ is contained in $\bar{Q}$).

For any fixed value of $\sigma \in \{0,1\}$, the lines $\{L^{(j)}_{\pi_{\ell,\sigma}(\bar{\mathbb{z}})}\}_{j\in[\kappa]}$ together with $\{L^{(j)}_{\bar{\mathbb{u}}_i}\}_{i,j\in[\kappa]}$ are mutually independent conditioned on $\vec{\mathbb{u}}$. Thus, by the definition of $p_{\bar{\mathbf{y}},\ell-1}$ (and the no-signaling condition) for each $\sigma \in \{0,1\}$,

$$\Pr\left[\neg\mathsf{Correct}(\pi_{\ell,\sigma}(\bar{\mathbb{z}})) \wedge \mathsf{Correct}(\vec{\mathbb{u}})\right] \leq p_{\bar{\mathbf{y}},\ell-1} + \delta,$$

where we write $\mathsf{Correct}(\vec{\mathbb{u}})$ as short-hand for $\mathsf{Correct}(\bar{\mathbb{u}}_1) \wedge \cdots \wedge \mathsf{Correct}(\bar{\mathbb{u}}_\kappa)$. Taking a union bound over both values of $\sigma \in \{0,1\}$ yields

$$\Pr\left[\mathsf{Correct}(\vec{\mathbb{u}}) \wedge \bigcup_{\sigma\in\{0,1\}} \neg\mathsf{Correct}(\pi_{\ell,\sigma}(\bar{\mathbb{z}}))\right] \leq 2p_{\bar{\mathbf{y}},\ell-1} + 2\delta. \tag{35}$$

For a fixed value of $\alpha \in \mathbb{F}\setminus\{0\}$, observe that the line $\left\{\beta \mapsto L^{(j)}_{\bar{\mathbb{z}}}(\alpha) + \beta \cdot \mathbf{e}_{t+\ell}\right\}_{\beta\in\mathbb{F}}$ is a uniformly random line that is parallel to the $(t+\ell)$-th axis. Because of our "Axis-Parallel Low-Degree Test" (see Section 7.3) and the no-signaling condition, for each $\alpha \in \mathbb{F} \setminus \{0\}$, it holds with all but $\epsilon + \delta$ probability that for all but r of the values $j \in [\kappa]$,

$$\mathsf{Deg}\left(\left\{\beta \mapsto X^*\big(L^{(j)}_{\bar{\mathbb{z}}}(\alpha) + \beta \cdot \bar{\mathbf{e}}_{t+\ell}\big)\right\}_{\beta\in\mathbb{F}}\right) \leq 1, \tag{36}$$

and in fact by union bounding over $\alpha \in \mathbb{F} \setminus \{0\}$, with all but $|\mathbb{F}| \cdot (\epsilon + \delta)$ probability, Eq. (36) holds for all but $|\mathbb{F}| \cdot r$ values of $j \in [\kappa]$ and all $\alpha \neq 0$. Let G denote this event.

Suppose that G occurs and fix $j \in [\kappa]$. Then, $X^*(\bar{\mathbb{z}})$ is a linear combination of $X^*(\pi_{\ell,0}(\bar{\mathbb{z}}))$ and $X^*(\pi_{\ell,1}(\bar{\mathbb{z}}))$. Thus, if $X^*(\pi_{\ell,\sigma}\bar{\mathbb{z}}) = \hat{X}_{\mathsf{correct}}(\pi_{\ell,\sigma}(\bar{\mathbb{z}}))$ for both $\sigma \in \{0,1\}$, then also $X^*(\bar{\mathbb{z}}) = \hat{X}_{\mathsf{correct}}(\bar{\mathbb{z}})$. Thus, if G and $\{\mathsf{Correct}(\pi_{\ell,\sigma}(\bar{\mathbb{z}})\}_{\sigma\in\{0,1\}}$ occur then:

$$\left|\left\{j \in [\kappa] \;:\; \mathsf{Interp}_d\left(\left\{\alpha \mapsto X^*(L^{(j)}_{\bar{\mathbb{z}}}(\alpha))\right\}_{\alpha\in\mathbb{F}\setminus\{0\}}\right)(0) = \hat{X}_{\mathsf{correct}}(\bar{\mathbb{z}})\right\}\right| \geq \kappa - (2r' + |\mathbb{F}| \cdot r). \tag{37}$$

Note that Eq. (37) has a $\kappa - (2r' + |\mathbb{F}| \cdot r)$ on the right hand side, whereas we would like to have $\kappa - r'$ (i.e., that $\mathsf{Correct}(\bar{\mathbb{z}})$ occurs). However, if also the event $v_{\bar{\mathbb{z}}} \neq \perp$ occurs, then these are equivalent (since all but r' of the lines interpolate to the same value). Denote by G' the event that G happens and $v_{\bar{\mathbb{z}}} \neq \perp$. Then on one hand if G' happens and $\{\mathsf{Correct}(\pi_{\ell,\sigma}(\bar{\mathbb{z}})\}_{\sigma\in\{0,1\}}$ then $\mathsf{Correct}(\bar{\mathbb{z}})$ occurs. On the other hand, by a union bound and the no-signaling condition:

$$\Pr[\neg G'] \leq |\mathbb{F}| \cdot (\epsilon + 2\delta) + O(|\mathbb{F}| \cdot \varepsilon) \leq O(|\mathbb{F}| \cdot \varepsilon). \tag{38}$$

Thus, for every $\bar{\mathbb{z}} \in \{\bar{\mathbf{y}} + 1\} \times \mathbb{F}^\ell \times \{0,1\}^{s-\ell}$

$$\begin{aligned}
\Pr[\neg\mathsf{Correct}(\bar{\mathbb{z}}) \wedge \mathsf{Correct}(\vec{\mathbb{u}})] &\leq \Pr[\neg G'] + \Pr[\neg\mathsf{Correct}(\bar{\mathbb{z}}) \wedge \mathsf{Correct}(\vec{\mathbb{u}}) \wedge G'] \\
&\leq \Pr[\neg G'] + \Pr\left[\mathsf{Correct}(\vec{\mathbb{u}}) \wedge \big(\neg\mathsf{Correct}(\pi_{\ell,0}(\bar{\mathbb{z}})) \vee \neg\mathsf{Correct}(\pi_{\ell,1}(\bar{\mathbb{z}}))\big) \wedge G'\right] \\
&\leq \Pr[\neg G'] + \Pr\left[\mathsf{Correct}(\vec{\mathbb{u}}) \wedge \big(\neg\mathsf{Correct}(\pi_{\ell,0}(\bar{\mathbb{z}})) \vee \neg\mathsf{Correct}(\pi_{\ell,1}(\bar{\mathbb{z}}))\big)\right] \\
&\leq O(|\mathbb{F}| \cdot \varepsilon) + 2 \cdot (p_{\bar{\mathbf{y}},\ell-1} + \delta) \\
&\leq 2 \cdot p_{\bar{\mathbf{y}},\ell-1} + O(|\mathbb{F}| \cdot \varepsilon),
\end{aligned}$$

where the first and third inequality follow from elementary probability theory, the second inequality follows from the discussion above, and the penultimate inequality follows from Eqs. (35) and (38).

This completes the proof of Claim 10.9.2. $\square$

Proposition 10.9 follows from the following fact (easily proved by induction).

Fact 10.10. *Suppose $x_0 \in \mathbb{R}$ is arbitrary and for $n \geq 1$, x_n is defined as $x_n = a \cdot x_{n-1} + b$ for $a \neq 1$. Then*

$$x_n = a^n \cdot x_0 + \frac{a^n - 1}{a - 1} \cdot b$$

for all $n \geq 1$.

$\square$

The following proposition shows that correctness of a random set of points in one layer, implies the correctness of a random set of points in the subsequent layer (with only a small additive increase in the error).

Proposition 10.11. *There exists $\varepsilon' = O(2^s \cdot \kappa^3 \cdot |\mathbb{F}|^2 \cdot \varepsilon)$ such that for every $y \in \{0, \dots, T-2\}$ it holds that:*

$$\Pr_{\substack{\vec{\mathbb{u}} \leftarrow (\{\bar{\mathbf{y}}+1\} \times \mathbb{F}^s)^\kappa \\ \mathcal{Q}(\{\bar{\mathbb{u}}_1, \dots, \bar{\mathbb{u}}_\kappa\})}} [\neg \mathsf{Correct}(\vec{\mathbb{u}})] \leq \Pr_{\substack{\vec{\mathbb{u}} \leftarrow (\{\bar{\mathbf{y}}\} \times \mathbb{F}^s)^\kappa \\ \mathcal{Q}(\{\bar{\mathbb{u}}_1, \dots, \bar{\mathbb{u}}_\kappa\})}} [\neg \mathsf{Correct}(\vec{\mathbb{u}})] + \varepsilon'.$$

Proof.

$$\begin{aligned}
\Pr_{\substack{\vec{\mathbb{u}} \leftarrow (\{\bar{\mathbf{y}}+1\} \times \mathbb{F}^s)^\kappa \\ \mathcal{Q}(\{\bar{\mathbb{u}}_1, \dots, \bar{\mathbb{u}}_\kappa\})}} [\neg \mathsf{Correct}(\vec{\mathbb{u}})] \leq & \Pr_{\substack{\vec{\mathbb{u}} \leftarrow (\{\bar{\mathbf{y}}+1\} \times \mathbb{F}^s)^\kappa \\ \vec{\mathbb{u}}' \leftarrow (\{\bar{\mathbf{y}}\} \times \mathbb{F}^s)^\kappa \\ \mathcal{Q}(\{\bar{\mathbb{u}}_1, \dots, \bar{\mathbb{u}}_\kappa, \bar{\mathbb{u}}'_1, \dots, \bar{\mathbb{u}}'_\kappa\})}} \left[\neg \mathsf{Correct}(\vec{\mathbb{u}}) \wedge \mathsf{Correct}(\vec{\mathbb{u}}')\right] \\
& + \Pr_{\substack{\vec{\mathbb{u}}' \leftarrow (\{\bar{\mathbf{y}}\} \times \mathbb{F}^s)^\kappa \\ \mathcal{Q}(\{\bar{\mathbb{u}}'_1, \dots, \bar{\mathbb{u}}'_\kappa\})}} \left[\neg \mathsf{Correct}(\vec{\mathbb{u}}')\right] + 2\delta \\
\leq & \sum_{i \in [\kappa]} \Pr_{\substack{\bar{\mathbb{u}}_i \leftarrow (\{\bar{\mathbf{y}}+1\} \times \mathbb{F}^s) \\ \vec{\mathbb{u}}' \leftarrow (\{\bar{\mathbf{y}}\} \times \mathbb{F}^s)^\kappa \\ \mathcal{Q}(\{\bar{\mathbb{u}}_i, \bar{\mathbb{u}}'_1, \dots, \bar{\mathbb{u}}'_\kappa\})}} \left[\neg \mathsf{Correct}(\bar{\mathbb{u}}_i) \wedge \mathsf{Correct}(\vec{\mathbb{u}}')\right] \\
& + \Pr_{\substack{\vec{\mathbb{u}}' \leftarrow (\{\bar{\mathbf{y}}\} \times \mathbb{F}^s)^\kappa \\ \mathcal{Q}(\{\bar{\mathbb{u}}'_1, \dots, \bar{\mathbb{u}}'_\kappa\})}} \left[\neg \mathsf{Correct}(\vec{\mathbb{u}}')\right] + 2\delta + \kappa \cdot \delta \\
\leq & \Pr_{\substack{\vec{\mathbb{u}}' \leftarrow (\{\bar{\mathbf{y}}\} \times \mathbb{F}^s)^\kappa \\ \mathcal{Q}(\{\bar{\mathbb{u}}'_1, \dots, \bar{\mathbb{u}}'_\kappa\})}} \left[\neg \mathsf{Correct}(\vec{\mathbb{u}}')\right] + O(2^s \cdot \kappa^3 \cdot |\mathbb{F}|^2 \cdot \varepsilon), \qquad (39)
\end{aligned}$$

where the first inequality follows from elementary probability and the no-signaling condition, the second inequality follows from a union bound and the non-signaling condition, and the third inequality follows from Proposition 10.9. $\square$

10.1.4 Establishing Weak Soundness

Given the above propositions, we are almost ready to prove Lemma 10.3. We first establish the following proposition, which shows that random points in the first layer of the computation are "read" correctly.

Proposition 10.12 (Correctness in Layer 0)**.** *There exists $\varepsilon' = O(|\mathbb{F}| \cdot \varepsilon)$ such that for any $\bar{\mathbb{z}} \in \{0\} \times \mathbb{F}^s$,*

$$\Pr_{\mathcal{Q}(\{\bar{\mathbb{z}}\})}[\neg \mathsf{Correct}(\bar{\mathbb{z}})] \leq \varepsilon'.$$

Proof. Fix $\bar{\mathbb{z}} \in \{0\} \times \mathbb{F}^s$.

Consider the following experiment. Sample κ planes $M^{(1)}, \ldots, M^{(\kappa)} : \mathbb{F}^2 \to \mathbb{F}$ uniformly at random conditioned on the event that for every $j \in [\kappa]$, it holds that $M^{(j)}(0,0) = \bar{\mathbb{z}}$ and the line $M^{(j)}(\cdot, 0)$ is fully contained in $\subseteq \{0\} \times \mathbb{F}^s \subseteq \mathbb{F}^t \times \mathbb{F}^s$. Define $\bar{Q} = \{M^{(j)}(\alpha, \beta)\}_{\alpha,\beta \in \mathbb{F}, j \in [\kappa]}$, and sample $X^* \leftarrow P^*(\bar{Q})$.

The lines $\{M^{(j)}(0, \cdot)\}_{j \in [\kappa]}$ are uniformly random conditioned $M^{(j)}(0,0) = \bar{\mathbb{z}}$. Thus, to prove the proposition, it suffices (using also the no-signaling condition) to show that with all but $O(|\mathbb{F}| \cdot \varepsilon)$ probability, it holds for all but r values of $j \in [\kappa]$ that

$$\mathsf{Interp}_d\left(\left\{\beta \mapsto X^*\big(M^{(j)}(0,\beta)\big)\right\}_{\beta \in \mathbb{F} \setminus \{0\}}\right)(0) = \hat{X}_{\mathsf{correct}}(\bar{\mathbb{z}}). \tag{40}$$

Observe that for every $\alpha \neq 0$, the lines $\{M^{(j)}(\alpha, \cdot)\}_{j \in [\kappa]}$ are uniformly random conditioned on the event $M^{(j)}(\alpha, 0) \in \{0\} \times \mathbb{F}^s$. Therefore, by the verifier's "Correct Initial Input Test" and the no-signaling condition, for every $\alpha \neq 0$, with all but $\varepsilon + \delta$ probability, it holds for all but r values of $j \in [\kappa]$ that:

$$\mathsf{Interp}_d\left(\left\{\beta \mapsto X^*\big(M^{(j)}(\alpha,\beta)\big)\right\}_{\beta \in \mathbb{F} \setminus \{0\}}\right)(0) = \hat{X}_{\mathsf{correct}}\big(M^{(j)}(\alpha, 0)\big). \tag{41}$$

Observe also that for every $\beta \neq 0$, the lines $\{M^{(j)}(\cdot, \beta)\}_{j \in [\kappa]}$ are uniformly random conditioned on the event that each line $M^{(j)}(\cdot, \beta)$ is contained in $\{\bar{\mathbb{u}}_j\} \times \mathbb{F}^s$ for some $\bar{\mathbb{u}}_j \in \mathbb{F}^t$ (i.e., the line is orthogonal to the first t coordinates). Therefore, by the "Layer-Parallel Low-Degree Test" and the no-signaling condition, for every $\beta \neq 0$, with all but $\varepsilon + \delta$ probability, it holds for all but r values of $j \in [\kappa]$ that:

$$\mathsf{Deg}\left(X^*\big(M^{(j)}(\cdot, \beta)\big)\right) \leq s. \tag{42}$$

By a union bound, with all but $2|\mathbb{F}| \cdot (\varepsilon + \delta)$ probability, Eqs. (41) and (42) hold simultaneously for *every* $\alpha, \beta \neq 0$ and all but $2|\mathbb{F}| \cdot r$ values of $j \in [\kappa]$. For every $j \in [\kappa]$, let $F_j : \mathbb{F} \to \mathbb{F}$ denote the function

$$F_j(\alpha) = \mathsf{Interp}_d\Big(\big\{\beta \mapsto X^*\big(M^{(j)}(\alpha,\beta)\big)\big\}_{\beta \in \mathbb{F} \setminus \{0\}}\Big)(0).$$

Let $j \in [\kappa]$ for which Eqs. (41) and (42) holds. Observe that F_j is a linear combination of the degree s polynomials $X^*\big(M^{(j)}(\cdot, \beta)\big)$ (see Eq. (42)), so F_j also has degree at most s. Furthermore, F_j agrees with the degree s polynomial $\hat{X}_{\mathsf{correct}}\big(M^{(j)}(\cdot, 0)\big)$ on all inputs $\alpha \neq 0$ (Eq. (41)). Since $|\mathbb{F}| - 1 > s$, it must also hold that $F_j(0) = \hat{X}_{\mathsf{correct}}\big(M^{(j)}(0,0)\big) = \hat{X}_{\mathsf{correct}}(\bar{\mathbb{z}})$.

Thus, we have shown that with all but $2|\mathbb{F}| \cdot (\varepsilon + \delta)$ probability, Eq. (40) holds for all but $2|\mathbb{F}| \cdot r$ values of $j \in [\kappa]$. In contrast, we want to show that it holds for all but r' values of j. But Lemma 10.5 and the no-signaling condition together imply that with all but $O(|\mathbb{F}| \cdot \varepsilon)$ probability, at least $\kappa - r'$ of the values $\{F_j(0)\}_{j \in [\kappa]}$ are equal to each other. Because $\kappa - 2|\mathbb{F}|r > r'$, by a union bound it holds with all but $O(|\mathbb{F}| \cdot \varepsilon)$ probability that Eq. (40) holds for all but r' values of κ. The proposition follows. $\square$

We now show that random points in *each* layer are "read" correctly.

Proposition 10.13. *There exists $\varepsilon' = O(2^s \cdot \kappa^3 \cdot |\mathbb{F}|^2 \cdot \varepsilon)$ such that for every $\bar{\mathbf{y}} \in \{0, \dots, T-1\}$ it holds that:*

$$\Pr_{\substack{\vec{\mathrm{u}} \leftarrow (\{\bar{\mathbf{y}}\} \times \mathbb{F}^s)^\kappa \\ \mathcal{Q}(\{\bar{\mathrm{u}}_1, \dots, \bar{\mathrm{u}}_\kappa\})}} [\neg \mathsf{Correct}(\vec{\mathrm{u}})] \leq (\bar{\mathbf{y}} + 1) \cdot \varepsilon'.$$

Proof. We prove by induction on $\bar{\mathbf{y}}$, where the base case $\bar{\mathbf{y}} = 0$ follows from Proposition 10.12, the no-signaling condition and a union bound. The inductive step follows from Proposition 10.11. $\square$

Recall that the point $(2^t - 1, 0) \in \{0,1\}^{t+s}$ refers to the output bit (i.e., result) of the computation.

$$\begin{aligned}
\Pr_{\mathcal{Q}(\{(2^t-1,0)\})} \left[\neg \mathsf{Correct}((2^t - 1, 0))\right] &\leq \Pr_{\substack{\vec{\mathrm{u}} \leftarrow \left(\{2^t-1\} \times \mathbb{F}^s\right)^\kappa \\ \mathcal{Q}(\{\bar{\mathrm{u}}_1, \dots, \bar{\mathrm{u}}_\kappa, (2^t-1,0)\})}} \left[\neg \mathsf{Correct}((2^t - 1, 0))\right] + \delta \\
&\leq \Pr_{\substack{\vec{\mathrm{u}} \leftarrow \left(\{2^t-1\} \times \mathbb{F}^s\right)^\kappa \\ \mathcal{Q}(\{\bar{\mathrm{u}}_1, \dots, \bar{\mathrm{u}}_\kappa, (2^t-1,0)\})}} \left[\neg \mathsf{Correct}((2^t - 1, 0)) \wedge \mathsf{Correct}(\vec{\mathrm{u}})\right] \\
&\quad + \Pr_{\substack{\vec{\mathrm{u}} \leftarrow \left(\{2^t-1\} \times \mathbb{F}^s\right)^\kappa \\ \mathcal{Q}(\{\bar{\mathrm{u}}_1, \dots, \bar{\mathrm{u}}_\kappa\})}} \left[\neg \mathsf{Correct}(\vec{\mathrm{u}})\right] + 2\delta \\
&\leq O(2^{t+s} \cdot \kappa^3 \cdot |\mathbb{F}|^2 \cdot \varepsilon),
\end{aligned} \tag{43}$$

where the first inequality follows from the no-signaling condition, the second inequality follows from elementary probability and the no-signaling condition, and the third inequality follows from Propositions 10.8 and 10.11.

On the other hand, by the verifier V's "Accepting Final State Test" and the no-signaling condition, it holds that

$$\Pr_{\mathcal{Q}((2^t-1,0))} [v_{\bar{\mathbf{z}}} = (q_{\mathsf{acc}}, \gamma, e) \text{ for some } \gamma,\, e] \geq 1 - \varepsilon - \delta. \tag{44}$$

Since $x \notin \mathcal{L}$, and the right hand side of Eq. (43) is less than $1 - \varepsilon - \delta$, we obtain a contradiction between Eqs. (43) and (44). This concludes the proof of Lemma 10.3.

10.2 Soundness of $V^{\otimes \kappa}$ – Proving Lemma 10.1

To prove Lemma 10.1, we invoke a generic transformation from the literature, due to [KRR14, Lemma 6.1] and [BHK16, Lemma 1],which transforms weak soundness (i.e., $1 - 1/\mathsf{poly}(\kappa)$ soundness error) for the relaxed verifier $V^{\geq \kappa - r}$ into strong soundness (i.e., negligible soundness error).

Lemma 10.14. *There exists $\delta^* = \Omega\left(\frac{\varepsilon}{|\mathbb{F}|^{24\kappa\cdot t\cdot|\mathbb{F}|^2}}\right)$ such that the following holds. Suppose that there exists a $(k_{\max}, \delta)$-no-signaling strategies that makes $V^{\otimes\kappa}$ accept $x \notin \mathcal{L}$ with probability ε, where $\delta \leq \delta^*$. Then, there exists a $(k'_{\max}, \delta')$-no-signaling strategy that convinces $V^{\geq k-r}$ to accept $x \notin \mathcal{L}$ with probability $1-\varepsilon'$, where $k'_{\max} = k_{\max}/2$, $\varepsilon' = O\left(\frac{t\cdot|\mathbb{F}|\cdot 2^{-r}+\delta}{\varepsilon}\right)$ and $\delta' = O\left(\delta \cdot |\mathbb{F}|^{24\cdot\kappa\cdot t\cdot|\mathbb{F}|^2}/\varepsilon\right)$.*

Lemma 10.1 now follows by combining Lemma 10.14 and Lemma 10.3.

10.2.1 Proof of Lemma 10.1

Suppose that $\mathbb{F}, k_{\max}, \delta, \varepsilon, \kappa$ are such that:

- $k_{\max} = O(\kappa^2 \cdot |\mathbb{F}|^2)$.
- $|\mathbb{F}| = \omega(t)$.
- $\delta \leq o(|\mathbb{F}|^{-25\cdot\kappa\cdot t\cdot|\mathbb{F}|^2})$.
- $\varepsilon \geq \omega\left(2^{t+s}\cdot\kappa^3\cdot|\mathbb{F}|^3\cdot t\cdot 2^{-\kappa/|\mathbb{F}|^3}\right)$.

Suppose that there exists a $(k_{\max}, \delta)$-no-signaling strategy that makes $V^{\otimes\kappa}$ accept $x \notin \mathcal{L}$ with probability ε, where $\delta \leq \delta^*$. Let $r = \kappa/|\mathbb{F}|^3$ (note that $r = o(\kappa/|\mathbb{F}|^2)$).

Then, by Lemma 10.14 there exists a $(k'_{\max}, \delta')$-no-signaling strategy that convinces $V^{\geq k-r}$ to accept $x \notin \mathcal{L}$ with probability $1-\varepsilon'$, where

- $k'_{\max} = k_{\max}/2$,
- $\varepsilon' = \Theta\left(\frac{t\cdot|\mathbb{F}|\cdot 2^{-r}+\delta}{\varepsilon}\right)$, and
- $\delta' = O\left(\delta\cdot|\mathbb{F}|^{24\cdot\kappa\cdot t\cdot|\mathbb{F}|^2}/\varepsilon\right)$.

Observe that:

- $\delta' \leq \varepsilon'$.
- $r = o(\kappa/|\mathbb{F}|^2)$.
- $\kappa \geq \log(1/\varepsilon')$.
- $|\mathbb{F}| = \omega(t)$.
- $\varepsilon' = o\left(\frac{1}{2^{t+s}\cdot\kappa^3\cdot|\mathbb{F}|^2}\right)$.

Thus, we obtain a contradiction to Lemma 10.3. This concludes the proof of Lemma 10.1.

Remark 10.15. *[Classical Soundness for* Non-Deterministic *Computations] Lemma 10.1 shows that our* PCP *has soundness against no-signaling strategies for any* deterministic *computation. By results of [DLN+01, IKM09] the restriction to deterministic computations is inherent. However, our* PCP *can easily be adapted to provide* classical *soundness even for* non-deterministic *computations as follows.*

Given an input x and witness w, the PCP *proof is the same as that in Section 7 applied to the input (x, w). The verifier's input check is applied only to x. In the classical setting, one only needs to consider fixed (rather than randomized)* PCP *strings and Lemma 10.1 implies that the computation was done correctly wrt x and* some *witness w.*

11 Putting it All Together: Proving Theorems 6.1 and 6.4

11.1 Proof of Theorem 6.1

Let $W = \mathsf{poly}(w)$ be the maximal complexity of implementing any (standard) word operation by a single-tape Turing machine on w-bit inputs, let $S' = \max(n, S)$, and let $\mathsf{Tree}(M)$ be the tree machine from Construction 4.32 that simulates M in time $T' = \Theta(T \cdot W^2 \cdot \log S')$. Let $s \overset{\mathsf{def}}{=} \lceil \log S' \rceil + \lceil \log W \rceil$ and $t \overset{\mathsf{def}}{=} \lceil \log T' \rceil$.

Let $\mathbb{F}$ be an explicit field ensemble of size $|\mathbb{F}| = \Theta(t)$ and let κ be such that $\kappa = \Theta(\lambda \cdot |\mathbb{F}|^3 + t + s + \log(\kappa) + \log(|\mathbb{F}|))$. Let $(P, V^{\otimes \kappa})$ be the PCP defined in Section 7 with respect to the RAM machine M and security parameter κ.

Then, by Lemma 10.1 the verifier $V^{\otimes \kappa}$ has ε-soundness against $(k_{\max}, \delta)$-NS provers, where:

- $\varepsilon \geq \omega\left(2^{t+s} \cdot \kappa^3 \cdot |\mathbb{F}|^3 \cdot t \cdot 2^{-\kappa/|\mathbb{F}|^3}\right)$,
- $\delta \leq o(|\mathbb{F}|^{-25 \cdot \kappa \cdot t \cdot |\mathbb{F}|^2})$, and
- $k_{\max} = O(\kappa^2 \cdot |\mathbb{F}|^2)$.

By our setting of κ it holds that $\varepsilon = 2^{-\lambda}$, $\delta = 2^{-\lambda \cdot \mathsf{poly}(t))}$ and $k_{\max} = \lambda \cdot \mathsf{poly}(t)$.

The efficiency of $V^{\otimes \kappa}$ follows from Lemma 8.1 and the efficiency of P follows from Lemma 9.1.

11.2 Proof of Theorem 6.4

To prove Theorem 6.4 we rely on the following lemma that shows that PCPs with soundness against no-signaling strategies can be converted into argument schemes. The proof follows directly from the analysis in [KRR14].

Lemma 11.1 ([KRR14]). *Assume the existence of $(s_{\mathsf{FHE}}, \delta_{\mathsf{FHE}})$-secure homomorphic encryption that supports evaluation of arithmetic circuits of size T and multiplicative depth D. Suppose that the language $\mathcal{L}$ is accepted by a* PCP *with soundness error $\varepsilon_{\mathsf{PCP}}$ against $(k_{\max}, \delta_{\mathsf{PCP}})$-*NS *strategies, where the alphabet is a binary finite field $\mathbb{F}$ of size 2^f and the length of the* PCP *is 2^ℓ. Suppose further that the verifier runs in time T_V and space S_V and that each symbol from the* PCP *can be generated by a* $\mathsf{TISP}(T, S)$*-uniform* ASLP *over $\mathbb{F}$ with K registers and multiplicative depth D.*

Then, if $\delta_{\mathsf{FHE}} \leq \frac{\delta_{\mathsf{PCP}}}{\ell \cdot k_{\max}}$, there exists a 2-message argument scheme for $\mathcal{L}$ with $(s_{\mathsf{argument}}, \varepsilon_{\mathsf{argument}})$-soundness, where $s_{\mathsf{argument}} = s_{\mathsf{FHE}} - 2^{\sigma \cdot k_{\max}}$ and $\varepsilon_{\mathsf{argument}} = \varepsilon_{\mathsf{PCP}}$.

The verifier of the argument-system runs in time $T_V + k_{max} \cdot (\sigma + \ell) \cdot \mathsf{poly}(\lambda)$ and space $S_V + k_{max} \cdot (\sigma + \ell) \cdot \mathsf{poly}(\lambda)$. The prover runs in time $t \cdot \ell \cdot \mathsf{poly}(\lambda)$ and space $s + K \cdot \mathsf{poly}(\lambda)$. The communication complexity is $k_{\max} \cdot (\ell + \sigma) \cdot \mathsf{poly}(\lambda)$.

Proof Sketch. Following the original proposal of Biehl et al. [BMW98], the protocol proceeds as follows. The verifier generates the PCP queries Q. Each query q_i is encrypted under the encryption scheme, using a fresh public key pk_i, to produce $\hat{q}_i = E_{\mathsf{pk}_i}(q_i)$. The encrypted queries $(\hat{q}_1, \ldots, \hat{q}_\ell)$ are all sent to the prover, who in turn generates $\hat{a}_i = E_{\mathsf{pk}_i}(a_i)$ by running the homomorphic evaluation algorithm on the ciphertext $\hat{q}_i$ wrt the circuit C_x that has the main input x hardcoded and given as input a query q_i for the PCP, produces the q_i^{th} symbol of the PCP. The verifier is then able to decrypt these answers, interpret them as a sequence $(a_1, \ldots, a_\ell)$ of answers for the PCP and decide whether to accept or reject based on whether the PCP verifier's decision.

The analysis in [KRR13] shows that this argument-system is (computationally) sound, as long as the PCP is sound against δ-no-signaling strategies (for non-negligible δ).[42]

For the prover complexity, we observe that if C_x can be generated by a $(\mathsf{TISP}(T,S)$-uniform ASLP with K registers and multiplicative depth D, and the homomorphic encryption scheme works in a gate-by-gate manner (as is the case for all such schemes, see Section 3.2), then we can do the homomorphic operation as follows. We generate the ASLP on-the-fly and for every instruction generated, we perform the corresponding homomorphic operation on the input ciphertext. This allows us to only use space $S + K \cdot \mathsf{poly}(\lambda)$ and bounds the multiplicative depth of the homomorphic operation by D.

Note that the above description assumes that the FHE scheme supports addition and multiplication over $\mathbb{F}$. However, current schemes only natively support $\mathbb{GF}(2)$ addition and multiplication over $\mathbb{GF}(2)$. As noted in Remark 3.5, we can use the fact that for a *binary* field $\mathbb{F}$, emulating addition and multiplication over $\mathbb{F}$ by the corresponding operations over $\mathbb{GF}(2)$ does not increase the multiplicative degree, and only increases the size of the arithmetic circuit by an $O(\log(|\mathbb{F}|)$ factor. $\square$

Theorem 6.4 now follows by combining Lemma 11.1 and Theorem 6.1, instantiated with a binary finite field.

Remark 11.2 (Improved Usage of Homomorphic Encryption)**.** *If our scheme were to be implemented, the following ideas may yield fruitful optimizations:*

1. *Utilizing homomorphic encryption that works over larger fields (rather than just* $\mathbb{GF}(2)$*). Note that we only require fields of size logarithmic in* T *(i.e., fields elements are represented by* $\log\log(T)$ *bits).*

2. *By utilizing ciphertext packing and SIMD operations [SV10, GHS12, BGH13, SV14].*

We leave studying these and other optimizations to future work.

Acknowledgments

We thank Yael Kalai for collaborating with us on the initial stages of this project and for multiple useful discussions. We also thank the anonymous reviewers for valuable comments on a previous version of this manuscript that led us to improve both the results and the presentation.

References

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP (1)*, pages 152–163, 2010.

[AV88] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988.

[42] To be more precise, for the PCP needs to be converted into a no-signaling MIP. [KRR13] give a simple procedure for doing so which does not affect the efficiency of neither the prover nor verifier.

[BBC$^+$17] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational integrity with a public random string from quasi-linear pcps. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, pages 551–579, 2017.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive*, 2018:46, 2018.

[BC12] Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 255–272, 2012.

[BCCT12a] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.

[BCCT12b] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. *IACR Cryptology ePrint Archive*, 2012:95, 2012.

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474, 2014.

[BCGT13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 585–594, 2013.

[BCI$^+$13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 31–60, 2016.

[BF91] László Babai and Lance Fortnow. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1:41–66, 1991.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium* *on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 21–31, 1991.

[BGH13] Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in lwe-based homomorphic encryption. In *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*, pages 1–13, 2013.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *TOCT*, 6(3):13:1–13:36, 2014.

[BHK16] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive RAM and batch NP delegation from any PIR. *IACR Cryptology ePrint Archive*, 2016:459, 2016.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 474–482, 2017.

[BKK$^+$18] Saikrishna Badrinarayanan, Yael Kalai, Dakshita Khurana, Amit Sahai, and Daniel Wichs. Non-interactive delegation for low-space non-deterministic computation. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:9, 2018.

[BMW98] Ingrid Biehl, Bernd Meyer, and Susanne Wetzel. Ensuring the integrity of agent-based computations by short proofs. In *Mobile Agents*, volume 1477 of *Lecture Notes in Computer Science*, pages 183–194. Springer, 1998.

[BS08] Eli Ben-Sasson and Madhu Sudan. Short pcps with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008.

[BTVW14] Andrew J. Blumberg, Justin Thaler, Victor Vu, and Michael Walfish. Verifiable computation using multiple provers. *IACR Cryptology ePrint Archive*, 2014:846, 2014.

[CCC$^+$16] Yu-Chi Chen, Sherman S. M. Chow, Kai-Min Chung, Russell W. F. Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Cryptography for parallel RAM from indistinguishability obfuscation. In *ITCS*, pages 179–190. ACM, 2016.

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-shamir and correlation intractability from strong kdm-secure encryption. *IACR Cryptology ePrint Archive*, 2018:131, 2018.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. In *ITCS*, pages 169–178. ACM, 2016.

[CKV10] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 468–497, 2015.

[CMS18] Alessandro Chiesa, Peter Manohar, and Igor Shinkar. Probabilistic checking against non-signaling strategies from linearity testing. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:123, 2018.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *ITCS*, pages 90–112. ACM, 2012.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, pages 54–74, 2012.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, pages 93–122, 2016.

[DLN$^+$01] Cynthia Dwork, Michael Langberg, Moni Naor, Kobbi Nissim, and Omer Reingold. Succinct proofs for np and spooky interactions. Unpublished manuscript, 2001.

[FLPR99] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *FOCS*, pages 285–298. IEEE Computer Society, 1999.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GGPR12] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. *IACR Cryptology ePrint Archive*, 2012:215, 2012.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 465–482, 2012.

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015.

[GLR$^+$91] Peter Gemmell, Richard J. Lipton, Ronitt Rubinfeld, Madhu Sudan, and Avi Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 32–42, 1991.

[GLR11] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier cs-proofs. *IACR Cryptology ePrint Archive*, 2011:456, 2011.

[GR17] Tom Gur and Ron D. Rothblum. A hierarchy theorem for interactive proofs of proximity. In *Proceedings of the 2017 Conference on Innovations in Theoretical Computer Science, ITCS 2017*, 2017.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, pages 321–340, 2010.

[GS89] Yuri Gurevich and Saharon Shelah. Nearly linear time. In *Logic at Botik '89, Symposium on Logical Foundations of Computer Science, Pereslav-Zalessky, USSR, July 3-8, 1989, Proceedings*, pages 108–118, 1989.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 75–92, 2013.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108. ACM, 2011.

[IKM09] Tsuyoshi Ito, Hirotada Kobayashi, and Keiji Matsumoto. Oracularization and two-prover one-round interactive proofs against nonlocal strategies. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 217–228, 2009.

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

[Kiy18] Susumu Kiyoshima. No-signaling linear pcps. *IACR Cryptology ePrint Archive*, 2018:649, 2018.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *STOC*, pages 419–428. ACM, 2015.

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

[KR15] Yael Tauman Kalai and Ron D. Rothblum. Arguments of proximity - [extended abstract]. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 422–442, 2015.

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In *STOC*, pages 565–574. ACM, 2013.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *STOC*, pages 485–494. ACM, 2014.

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of fiat-shamir for proofs. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 224–251, 2017.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, pages 169–189, 2012.

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

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, pages 96–109, 2003.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[PR17] Omer Paneth and Guy N. Rothblum. On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, pages 283–315, 2017.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 49–62, 2016.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[Sud00] Madhu Sudan. Probabilistically checkable proofs - lecture notes, 2000. Available at http://people.csail.mit.edu/madhu/pcp/pcp.ps.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, pages 420–443, 2010.

[SV14] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1):57–81, 2014.

[SVP$^+$12] Srinath T. V. Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 253–268, 2012.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 71–89, 2013.

[Vit06] Jeffrey Scott Vitter. Algorithms and data structures for external memory. *Foundations and Trends in Theoretical Computer Science*, 2(4):305–474, 2006.

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

[WJB$^+$17] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. Full accounting for verifiable outsourcing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2071–2086, 2017.

[ZGK$^+$17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vsql: Verifying arbitrary SQL queries over dynamic outsourced databases. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 863–880, 2017.

A Deferred Proofs

Proposition A.1. *For any d and fixed $\bar{\mathbf{a}} \in \{0,1\}^i$, the set $\{\bar{\mathbf{a}}' : \bar{\mathbf{a}} \sim_d \bar{\mathbf{a}}'\}$ has size at most 2 and is computable in time $\mathsf{poly}(i)$ given $\bar{\mathbf{a}}$. Similarly, for any $\bar{\mathbf{a}}'$, the set $\{\bar{\mathbf{a}} : \bar{\mathbf{a}} \sim_d \bar{\mathbf{a}}'\}$ has size at most 2 and is computable in time $\mathsf{poly}(i)$ given $\bar{\mathbf{a}}'$.*

Proof. We explicitly describe the $\sim_d$ relation for each d. Let $\bar{\mathbf{a}}$ and $\bar{\mathbf{a}}'$ be i-bit strings. If $i > \lceil \log S' \rceil$, write $\bar{\mathbf{a}} = \bar{\mathbf{v}} \| \bar{\mathbf{z}}$ and $\bar{\mathbf{a}}' = \bar{\mathbf{v}}' \| \bar{\mathbf{z}}'$ for $\bar{\mathbf{v}}, \bar{\mathbf{v}}' \in \{0,1\}^{\lceil \log S' \rceil}$.

- If $d = \uparrow$, then $\bar{\mathbf{a}} \sim_d \bar{\mathbf{a}}'$ iff either
  - $i \le \lceil \log S' \rceil$ and $\bar{\mathbf{a}}' = \lfloor \bar{\mathbf{a}}/2 \rfloor$ *or*
  - $i > \lceil \log S' \rceil$ and $\bar{\mathbf{v}}' = \lfloor \bar{\mathbf{v}}/2 \rfloor$ and $\bar{\mathbf{z}}' = \bar{\mathbf{z}}$.
- If $d = \swarrow$ (resp., $\searrow$), $\bar{\mathbf{a}} \sim_d \bar{\mathbf{a}}'$ iff either

  - $i \leq \lceil \log S' \rceil$ and $\bar{\mathbf{a}}' = 2 \cdot \bar{\mathbf{a}}$ (resp., $\bar{\mathbf{a}}' = 2 \cdot \bar{\mathbf{a}} + 1$) *or*
  - $i > \lceil \log S' \rceil$ and $\bar{\mathbf{v}}' = 2 \cdot \bar{\mathbf{v}}$ (resp., $\bar{\mathbf{v}}' = 2 \cdot \bar{\mathbf{v}} + 1$) and $\bar{\mathbf{z}}' = \bar{\mathbf{z}}$.

- If $d = \otimes$, then $\bar{\mathbf{a}} \sim_d \bar{\mathbf{a}}'$ iff $\bar{\mathbf{a}}$ and $\bar{\mathbf{a}}'$ agree on the first $\lceil \log S' \rceil$ bits, and $\bar{\mathbf{a}}' \in \{\bar{\mathbf{a}}, \bar{\mathbf{a}} + 1\}$.
- If $d = \odot$, then $\bar{\mathbf{a}} \sim_d \bar{\mathbf{a}}'$ iff $\bar{\mathbf{a}}$ and $\bar{\mathbf{a}}'$ agree on the first $\lceil \log S' \rceil$ bits, and $\bar{\mathbf{a}}' \in \{\bar{\mathbf{a}}, \bar{\mathbf{a}} - 1\}$.
- If $d = \emptyset$, then $\bar{\mathbf{a}} \sim_d \bar{\mathbf{a}}'$ iff $\bar{\mathbf{a}} = \bar{\mathbf{a}}'$. $\square$