2018 IEEE 59th Annual Symposium on Foundations of Computer Science

Delegating Computations with (almost) Minimal Time and Space Overhead

Extended Abstract

Justin Holmgren
Princeton University
Princeton, USA
Email: justin.holmgren@princeton.edu

Ron D. Rothblum
Technion
Haifa, Israel
Email: rothblum@cs.technion.ac.il

***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. The prover runs in time $\tilde{O}(T)$ and space $S + o(S)$, and in some 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 the main bottleneck in such schemes) $\log_2 \log(T) + O(1)$.

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 hope will be useful for future work.

I. 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, it should not take much more time or space than is required 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. A delegation scheme (with some additional properties) also forms the core component of a popular crypto currency called Zcash [1]. The line of work implementing such schemes has identified the efficiency of the prover as the major bottleneck in making such schemes practical (see, e.g., [2]). 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 [3], [4] 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?*

a) 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

This research was conducted while Justin Holmgren was affiliated with MIT, and Ron Rothblum was affiliated with MIT and Northeastern.

2575-8454/18/$31.00 ©2018 IEEE
DOI 10.1109/FOCS.2018.00021

good approximation for the efficiency of real computers.

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

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 $\zeta = 1/\text{polylog}(T)$ 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. [5]. 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 $S + \text{polylog}(T)$, 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 achieves space usage $S + \text{polylog}(T)$

Theorem 2 (Informal)**.** *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 $S + \text{polylog}(T)$.*

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. 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.

a) 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 (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., [6], [7]) 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 [8].

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). This is somewhat similar to the reason that *private information retrieval* (PIR) can be constructed from additively homomorphic encryption [9].

b) Efficient Implementations of No-Signaling MIP*s:* Our basic approach for proving Theorem 1 follows the line of work on non-interactive delegation for P based on standard assumptions [5], [10], [11], [12], [13], [14], [15]. Similarly to that line of work, our basic building block is the multi-prover interactive proof-system (MIP) of Babai et al. [16] (and its PCP variant from [17]). The above works all rely on a transformation from MIPs to non-interactive arguments, originally proposed by Biehl et al. [18], and shown to be secure for MIPs having a strong soundness property in [5]. 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 [16] MIP which is both no-signaling sound and for which the prover answers can be generated extremely efficiently.

[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 with respect to to larger blocks of memory.

[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 the full version for additional details.

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

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

Setting $\zeta = 1/\text{polylog}(T)$ 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 $S + \text{polylog}(T)$ for *cache friendly* RAM programs. For comparison, the running time of the prover in prior no-signaling MIPs [5], [10], [14] was T^c, for an unspecified constant c (which at the very least is $c \geq 3$).

In fact, Theorem 3 also improves over the most efficient known *classical* MIPs.[3] Specifically, Bitansky and Chiesa [21] and Blumberg et al. [22] show that variants of the [16] MIP can be implemented with a time $\tilde{O}(T)$ and space $S \cdot \text{polylog}(T)$ 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 $S + \text{polylog}(T)$.[4] See Section I-B for further comparison and discussion on these and other related works.

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.

a) Computing BFLS "On-The-Fly": One of our key insights is that, for (our variant of) the [17] PCP, it is possible to efficiently compute any symbol of the proof string given *streaming* access to the underlying computation transcript. 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 that operates in the following model.

The algorithm P proceeds in T steps, where T is the running time of M on x. In the i^{th} step, P is given random 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 $\text{polylog}(T)$ per step. We note that similar observations were made independently by Naor et al. [23].

As mentioned above, our variant of 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 $\text{polylog}(T)$. More generally, we count the number of *cache misses* $Q(\tau, B)$ incurred by M's memory access pattern with a cache 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 $\tau \cdot \text{polylog}(T)$.

b) Avoiding the Augmented Circuit of [10]: Building on the work of [5], [10], 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 [10].

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 [10]. See Section II-B for more details.

c) Abstractions and Computational Models: Our argument schemes (Theorems 1 and 2) have many moving pieces. One of our contributions is identifying suitable abstractions that allow us to describe the construction modularly.

For example, our informal statement of Theorem 3 does not generically imply Theorems 1 and 2. The reason is that the prover of Theorem 3 is a time-$\tilde{O}(T)$ *RAM* computation, which in Theorems 1 and 2 must be evaluated homomorphically. In contrast, homomorphic encryption schemes natively support the homomorphic evaluation of *circuits*.

The prover in our argument scheme actually invokes each MIP prover P_i on an input x given in the clear, and a query q_i given in encrypted form (that comes from the argument verifier). The reason this can be done efficiently is due to P_i's general structure: P_i first takes x as an input, and then produces an *arithmetic straight-line program* (ASLP) (a stream of arithmetic instructions). The output $P_i(x, q_i)$ is the result of executing these instructions on input q_i.

Stated otherwise, each MIP prover is computable by (uniformly generated) ASLPs. This notion allows us to:

1) Avoid needlessly generic reductions from time-T RAM computations to circuits of size[5] $\Omega(T^2)$.
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 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

[3]The MIP of Theorem 3 refers to deterministic computations (which is inherent for no-signaling soundness [19], [20]). However, essentially the same MIP can be shown to achieve *classical* soundness even for non-deterministic computations.

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

[5]The frequently cited T^3 overhead of simulating a RAM by a circuit refers to a model of RAM with unbounded register size, in contrast to the common modern model of a word-RAM machine

(similar to Turing machines).[6] See Section II-C for details.

d) Low Degree Extensions of Read Once Branching Programs: We describe clean and versatile conditions which guarantee that a function's *minimal*[7] 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 [14] and [24]. In algebra-based proof-systems (e.g., [25], [5], [10], [11], [12], [14], [24]), 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.

B. Comparison to Prior Works

a) 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 [5], [10], [11], [12], [13], [14], [15]. 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. [30], [31], [32], [33]) 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 [21] (see more details below).

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. [14]:

1) First, [14] only rely on polynomial hardness assumptions (e.g., standard computational PIR) whereas we rely on sub-exponential hardness. We believe that our construction can also be proven secure under 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 [14], that we prefer to avoid.
2) Also, [14] 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 [14]).

Lastly, we mention that in a very recent work, Badrinarayanan et al. [15] 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.

b) Interactive Delegation Schemes: Kilian [34] constructed a 4-message delegation protocol based on PCPs and collision resistant hash functions. Micali [35] extended this to a non-interactive solution in the *random oracle model* (which is a somewhat controversial model [36]) and his techniques were recently further refined by [37] (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 [21] 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 [21] in several ways: (1) our delegation protocol is *non-interactive*[8], (2) our construction uses only somewhat homomorphic encryption (for $\log_2 \log(T) + O(1)$ multiplicative depth) whereas [21] 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 [21] have space usage $S \cdot \text{polylog}(T)$; and (4) [21] 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 [21] 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 [21] (but much better concrete efficiency) was given by Blumberg et al. [22].

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, Goldwasser et al. [25] and Reingold et al. [38] construct such protocols for computations that are highly

[6] 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* [19], [20], we cannot utilize non-deterministic reductions. See Section I-B for additional details.

[7] 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.

[8] [21] also give a separate *non-interactive* scheme that relies on a non-standard, and in particular *non-falsifiable*, assumption.

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 [39] transformation, obtaining heuristic soundness (see also [40], [41]). We remark that there has recently been a very successful line of work (c.f. [29], [42], [43], [44], [45]) implementing, optimizing and improving the [25] protocol.

c) Additional Delegation Schemes: A separate line of work [46], [47], [48], [49], [50], [51], [52], [21], [53] constructs non-interactive delegation schemes for *non-deterministic* computations based on non-falsifiable assumptions (which is inherent by [4]). 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 [54], but also RAM machines [55] and even Parallel RAM machines [56]. 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 [57], [58], [59], [60], 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.

II. Technical Overview

We now overview our main technical ideas. Our approach to delegation revolves around the celebrated PCP of [17]. Our presentation of this PCP departs from the textbook description (cf. [61]) 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. Throughout this section we refer to this PCP as the BFLS PCP.

We begin in Section II-A by briefly describing the BFLS PCP. In Section II-B we describe how we establish *no-signaling soundness* of the BFLS PCP without relying on the augmented circuit construct of [10]. In Section II-C we describe a new technique that allows us to efficiently handle RAM computations (rather than just Turing machine computations). Lastly, in Section II-E we describe our simple method of computing low degree extensions of simple functions.

A. 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 $s = \log_2(S)$. We construct a PCP for deciding $\mathcal{L}$ as follows.

Let $x \in \{0,1\}^n$ be an input for M. Define a function $X : \{0,1\}^t \times \{0,1\}^s \to \Gamma \times (Q \cup \bot)$, where Q is the set of internal control states of the Turing machine (and $\bot \notin Q$), 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, $q = \bot$. 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 $\Gamma \times (Q \cup \{\bot\})$ 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).

Remark II.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.*

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 $\bar{\mathbf{z}}_2 = \bar{\mathbf{z}}_1 + 1$. 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. Let $\hat{\phi}_{+1} : \mathbb{F}^t \times \mathbb{F}^t \to \mathbb{F}$ and $\hat{\phi}_{\to}, \hat{\phi}_{\leftarrow}, \hat{\phi}_{\emptyset} : \mathbb{F}^s \times \mathbb{F}^s \to \mathbb{F}$ be the respective multilinear extensions of these functions.

We also define validation functions $V_{\to}, V_{\leftarrow}, V_{\emptyset} : (\Gamma \times (Q \cup \{\bot\}))^2 \to \{0,1\}$ that check pairwise relations between points in X, where an output of 0 corresponds to a consistent pair and 1 indicates a violation. The function $V_{\to}$ gets as input

two alphabet symbols $\gamma_1, \gamma_2 \in \Gamma$ and two (potential) control states $q_1, q_2 \in Q \cup \{\perp\}$. It outputs 1 if and only if they are *inconsistent* in case the machine instruction corresponding to (γ_1, q_1) is a "move to the right" head instruction for the Turing machine. In more detail, if $q_1 = \perp$ (indicating that the machine head was not at the current position) then γ_2 and q_2 are unconstrained. If $q_1 \neq \perp$ then $V_{\rightarrow}$ checks the transition function of the Turing machine. If the machine is not instructed to move right given the symbol γ_1 and internal state q_1 then once again γ_2 and q_2 are unconstrained. However, if $q_1 \neq \perp$ *and* the Turing machine instruction corresponds to a "move right" then q_2 must be equal to the following control state. The validation functions $V_{\leftarrow}$ and $V_{\emptyset}$ are likewise defined to ensure pointwise relations between points in consecutive layers, where $V_{\leftarrow}$ handles "move left" instructions and $V_{\emptyset}$ corresponds to . Let $\hat{V}_{\rightarrow}, \hat{V}_{\leftarrow}, \hat{V}_{\emptyset} : \mathbb{F}^2 \to \mathbb{F}$ be the corresponding low degree extensions.

An important observation is that X is a correct 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

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

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 $\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2 \in \{0,1\}^s$:

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

This motivates our definition of the polynomial $P_0 : \mathbb{F}^t \times \mathbb{F}^s \times \mathbb{F}^t \times \mathbb{F}^s \to \mathbb{F}$, which is defined so that $P_0(\bar{\mathbf{a}}_1, \bar{\mathbf{b}}_1, \bar{\mathbf{a}}_2, \bar{\mathbf{b}}_2)$ is

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

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. This check is done by augmenting the PCP with one last component, often referred to as the "sumcheck polynomials".[9]

Let $\ell = 2(t+s)$. For every $i \in [\ell]$, define the polynomial $P_i : \mathbb{F}^\ell \to \mathbb{F}$ so that $P_i(\bar{z}_1, \ldots, \bar{z}_\ell)$ is

$$\sum_{b \in \{0,1\}} P_{i-1}(\bar{z}_1, \ldots, \bar{z}_{i-1}, b, \bar{z}_{i+1}, \ldots, \bar{z}_\ell) \cdot \bar{z}_i^b, \tag{3}$$

where we are viewing $b \in \{0,1\}$ both as a field element and as an integer, in the natural way.

The key observation is that:

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

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 $\bar{z}_i$ is the identically 0 function for any $\bar{z}_{i+1}, \ldots, \bar{z}_\ell \in \{0,1\}$. 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 $P_1, \ldots, P_\ell$. 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 $P_\ell \equiv 0$, which turns out to be much easier.

B. 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. [18] and shown to be secure in [5].

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 al. [5] establish the soundness of this approach if the the MIP satisfies a strong notion of soundness called *no-signaling soundness*.[10]

a) No-Signaling Soundness.: We first recall the definition of an MIP. An MIP consists of a verifier V and ℓ non-communicating provers $P_1, \ldots, P_\ell$. The verifier generates queries $(q_1, \ldots, q_\ell)$, 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 $P_1, \ldots, P_\ell$ to convince V to accept (say, with probability 1).

Classical soundness means that for $x \notin \mathcal{L}$, no cheating provers $P_1^*, \ldots, P_\ell^*$ *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., $\bar{q}'$).

An MIP verifier has no-signaling soundness if it accepts false statements with only negligible probability, even when

[9] Indeed, these polynomials are directly related to the classical sumcheck protocol of Lund et al. [27].

[10] The results of Dwork et al. [19] and Dodis et al. [64] show that this approach is in general *insecure* when using an MIP that is not sound against no-signaling strategies.

interacting with a no-signaling strategy. To show that our PCP has no-signaling soundness we re-visit the no-signaling soundness analysis of [5], [10].

b) Bird's Eye View of the Augmented Circuit of [10]: Kalai et al. [5] 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. [10] 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 L_i of gates in C (for $i \in [T]$), C' computes the low-degree $\hat{L}_i$ of those gates' values, and also performs the (apparently redundant) check that when $\hat{L}_i$ when restricted to any line, the result is a low-degree univariate polynomial. The BFLS PCP is now applied to the augmented circuit. In addition, the PCP is slightly modified to ensure that for every layer the restriction of the low degree extension of that layer to any particular line, is a low degree polynomial.

Kalai et al. [10] use the augmented circuit to argue that for every layer i, if the verifier "reads" the correct values from a *random* point $\bar{v}$ in the low degree extension of layer i then each *worst case* point $\bar{u}$, by itself, must also be "read" correctly.[11] The argument proceeds by considering the following thought experiment. Let X_i be the values obtained by the i^{th} layer of C, and let $\hat{X}_i$ be its low degree extension. Take a random line $\ell_{\bar{u}}$ passing through the fixed point u and consider the function $f = \hat{X}_i|_{\ell_{\bar{u}}}$. By the additional checks mentioned above, the function f should be low degree (here it is crucial that the checks added in [10] ensure that the restriction to *every* line in layer i is low degree). Suppose now that the when reading the value of $\bar{u}$ we obtain an incorrect answer. Then, since distinct low degree polynomials disagree almost everywhere, with high probability over the choice of $\bar{v}$, also the value corresponding to $\bar{v}$ is incorrect. By the no-signaling condition, the same is true even if $\bar{v}$ is read by itself (rather than via the line $\ell_{\bar{u}}$).

Thus, if the worst-case point $\bar{u}$ in layer i is read incorrectly, then with high probability, also a random point $\bar{v}$ in layer i is read incorrectly. This worst-case to average-case reduction is the core component that allows [10] to bypass the limitation of [5].

c) Delegation for P *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).

[11]The meaning of what "reading a point" means is intentionally left vague. The actual statement in [10] is complex and we prefer to avoid going into the details in this overview. Also, in order to amplify the success probability [10] refers to many random points v but we also ignore this complication here.

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 [10] 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 the restriction of $\hat{X}(i,\cdot)$ to the line $\ell_{\bar{u}}$ is low degree. While the PCP verifier tests that the restriction of $\hat{X}$ to *random* lines has low degree, in the no-signaling setting it is likely not true that the restriction to a *non-random* line such as $\ell_{\bar{u}}$ is low degree. It is worth pointing our that the reason that $\ell_{\bar{u}}$ is not a random line is due to two facts (1) that it passes through the fixed, worst-case, point $\bar{u}$ and (2) that it lies entirely within the i^{th} layer of the circuit. We remark the fact that the line lies entirely within the i^{th} layer is the more concerning aspect for us. This is since, loosely speaking, the analysis of [5] shows that the restriction of the PCP 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. 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.[12] Thus, rather than taking a random line passing through $\mathbb{u}$, we take a take a random plane $M : \mathbb{F}^2 \to \mathbb{F}^{t+s}$ such that $M(0,0) = \bar{u}$ and $M(\cdot,0)$ is contained entirely within layer i (i.e., $M(\cdot,0) \subseteq \{i\} \times \mathbb{F}^s$).[13] 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. As mentioned above, the analysis in [5] shows that also the restriction of $\hat{X}$ to such lines should 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 $\hat{X} \circ M$ is a low degree (bivariate) polynomial. In particular, $\hat{X} \circ M(\cdot,0)$ is a random line that lies entirely within layer i and so we can treat it similarly to the line $\ell_{\bar{u}}$ in the above description of the [10] protocol.

C. 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 II-A), is the very fact that

[12]In the actual construction, due to some complications that arise and in particular the actual meaning of "reading" a point, we need to resort to a more complicated manifold (rather than a plane).

[13]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 $M(\alpha,\beta) = \bar{u} + \alpha \cdot \bar{v} + \beta \cdot \bar{z}$.

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, at any time step, any location in memory might potentially be affected. This was resolved in prior works, starting from [31], by utilizing a more efficient *non-deterministic* reduction from RAM programs to Turing machines (originally due to Gurevich and Shelach [65]). 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 [19], [20]) 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 memory locations 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.

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

a) 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, we show that Tree machines can emulate such a step in $O(\log(d))$ instructions as follows.

First, the RAM tape of size at length at most 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 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 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).

D. 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 II-A (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 elements of $\mathbb{F}$ take $\mathrm{polylog}(T)$ 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.)

a) 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).

b) 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.

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.

c) 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. 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,

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

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 $S + \mathrm{polylog}(T)$.

d) 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 so that $P_0(\bar{\mathbb{y}}_1, \bar{\mathbb{u}}_1, \bar{\mathbb{y}}_2, \bar{\mathbb{u}}_2)$ is

$$\hat{\phi}_{+1}(\bar{\mathbb{y}}_1, \bar{\mathbb{y}}_2) \cdot \sum_{d \in \{\leftarrow, \rightarrow, \emptyset\}} \hat{\phi}_d(\bar{\mathbb{u}}_1, \bar{\mathbb{u}}_2) \cdot \hat{V}_d\Big(\hat{X}(\bar{\mathbb{y}}_1, \bar{\mathbb{u}}_1), \hat{X}(\bar{\mathbb{y}}_2, \bar{\mathbb{u}}_2)\Big). \tag{6}$$

As was briefly discussed in Section II-A, 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 $S + \mathrm{polylog}(T)$.

e) Efficient Evaluation of P_i for $i \leq t$: Recall that we may choose an arbitrary variable ordering and grouping in the definition of P_i. We choose them so that for any $i \leq t$, 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 $\bar{\mathbb{u}}_1, \bar{\mathbb{u}}_2 \in \mathbb{F}^s$, it holds that $P_i(\bar{\mathbb{y}}_1, \bar{\mathbb{u}}_1, \bar{\mathbb{y}}_2, \bar{\mathbb{u}}_2)$ is

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

where recall that our vector exponentiation is notation for the product of the coordinate-wise exponentiations. For example, $\bar{\mathbb{a}}_1^{\bar{\mathbf{a}}_1} \stackrel{\mathsf{def}}{=} \prod_{j=1}^{i} (\bar{\mathbb{a}}_1)_j^{(\bar{\mathbf{a}}_1)_j}$.

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

The factors $\bar{\mathbb{a}}_1^{\bar{\mathbf{a}}_1} \cdot \bar{\mathbb{a}}_2^{\bar{\mathbf{a}}_2}$ 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

$$\Big\{P_0(\bar{\mathbf{a}}_1 \| \bar{\mathbb{b}}_1, \bar{\mathbb{u}}_1, \bar{\mathbf{a}}_2 \| \bar{\mathbb{b}}_2, \bar{\mathbb{u}}_2)\Big\}_{\bar{\mathbf{a}}_1, \bar{\mathbf{a}}_2 \in \{0,1\}^i} \tag{7}$$

in time $\tilde{O}(T)$ and space $S + \mathrm{polylog}(T)$, which will imply that P_i itself can be evaluated in time $\tilde{O}(T)$ and space $S + \mathrm{polylog}(T)$.

First, let us see that there are at most $O(T)$ such non-zero terms. This holds because by Eq. (2) $P_0(\bar{\mathbf{a}}_1 \| \bar{\mathbb{b}}_1, \bar{\mathbb{u}}_1, \bar{\mathbf{a}}_2 \| \bar{\mathbb{b}}_2, \bar{\mathbb{u}}_2)$ 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 $\hat{\phi}_{+1}(\bar{\mathbf{a}}_1 \| \bar{\mathbb{b}}_1, \bar{\mathbf{a}}_2, \bar{\mathbb{b}}_2) \neq 0$ 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. (7) 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). As in the evaluation of P_0 discussed previously, enumerating these terms reduces to enumerating

$$\Big\{(\hat{X}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}_1, \bar{\mathbb{u}}_1), \hat{X}(\bar{\mathbf{a}} \| \bar{\mathbb{b}}_2, \bar{\mathbb{u}}_2))\Big\}_{\bar{\mathbf{a}} \in \{0,1\}^i} \tag{8}$$

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 $\hat{X}$.

f) Efficient Evaluation of P_i for $i > t$: We continue our definition of the variable ordering and grouping we employ in the definition of P_i, so that for any i with $t < i \leq t + s$, any $\bar{\mathbb{y}}_1, \bar{\mathbb{y}}_2 \in \mathbb{F}^t$, any $\bar{\mathbb{u}}_1 = \bar{\mathbb{a}}_1 \| \bar{\mathbb{b}}_2 \in \mathbb{F}^{i-t} \times \mathbb{F}^{s-(i-t)}$, and any $\bar{\mathbb{u}}_2 = \bar{\mathbb{a}}_2 \| \bar{\mathbb{b}}_2 \in \mathbb{F}^{i-t} \times \mathbb{F}^{s-(i-t)}$, $P_i(\bar{\mathbb{y}}_1, \bar{\mathbb{u}}_1, \bar{\mathbb{y}}_2, \bar{\mathbb{u}}_2)$ is

$$\sum_{\substack{\bar{\mathbf{y}}_1, \bar{\mathbf{y}}_2 \in \{0,1\}^t \\ \bar{\mathbf{a}}_1, \bar{\mathbf{a}}_2 \in \{0,1\}^{i-t}}} P_0(\bar{\mathbf{y}}_1, \bar{\mathbf{a}}_1 \| \bar{\mathbb{b}}_1, \bar{\mathbf{y}}_2, \bar{\mathbf{a}}_2 \| \bar{\mathbb{b}}_2) \cdot \bar{\mathbb{y}}_1^{\bar{\mathbf{y}}_1} \cdot \bar{\mathbb{y}}_2^{\bar{\mathbf{y}}_2} \cdot \bar{\mathbb{a}}_1^{\bar{\mathbf{a}}_1} \cdot \bar{\mathbb{a}}_2^{\bar{\mathbf{a}}_2}. \tag{9}$$

Unlike in the case $i \leq t$, we cannot now say that Eq. (9) has $O(T)$, or even $\tilde{O}(T)$ non-zero terms. However, we can make a similar observation that $P_0(\bar{\mathbf{y}}_1, \bar{\mathbf{a}}_1 \| \bar{\mathbb{b}}_1, \bar{\mathbf{y}}_2, \bar{\mathbf{a}}_2 \| \bar{\mathbb{b}}_2) \neq 0$ 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 subset of the above terms in which $\bar{\mathbf{a}}_2 = \bar{\mathbf{a}}_1$. The summation of the other cases can be evaluated similarly, but is slightly more complex. Note that $\bar{\mathbb{y}}_1^{\bar{\mathbf{y}}_1}$ and $\bar{\mathbb{y}}_2^{\bar{\mathbf{y}}_2}$ are computable in time $\mathrm{polylog}(T)$. We will show that the $O(T)$ terms

$$\left\{\sum_{\bar{\mathbf{a}} \in \{0,1\}^{i-t}} P_0(\bar{\mathbf{y}}, \bar{\mathbf{a}} \| \bar{\mathbb{b}}_1, \bar{\mathbf{y}} + 1, \bar{\mathbf{a}} \| \bar{\mathbb{b}}_2) \cdot \bar{\mathbb{a}}_1^{\bar{\mathbf{a}}} \cdot \bar{\mathbb{a}}_2^{\bar{\mathbf{a}}}\right\}_{\bar{\mathbf{y}} \in \{0,1\}^t} \tag{10}$$

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

Let $\sigma_{\bar{\mathbf{y}}}$ denote the $\bar{\mathbf{y}}^{th}$ term of Eq. (10). 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 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 $2^{s-(i-t)} \cdot \text{polylog}(T)$ given access to $X(\bar{\mathbf{y}}, \cdot)$ and $X(\bar{\mathbf{y}}+1, \cdot)$ (which is simulatable with constant overhead given $X(\bar{\mathbf{y}}, \cdot)$).

Now suppose we have computed $\sigma_{\bar{\mathbf{y}}}$, have access to $X(\bar{\mathbf{y}}, \cdot)$, and want to compute $\sigma_{\bar{\mathbf{y}}+1}$. For simplicity, suppose that $X(\bar{\mathbf{y}}+1, \cdot)$ differs from $X(\bar{\mathbf{y}}, \cdot)$ 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

$$P_0(\bar{\mathbf{y}}, \bar{\mathbf{a}} \| \bar{\mathbb{b}}_1, \bar{\mathbf{y}}+1, \bar{\mathbf{a}} \| \bar{\mathbb{b}}_2) \cdot \bar{\mathbb{a}}_1^{\bar{\mathbf{a}}} \cdot \bar{\mathbb{a}}_2^{\bar{\mathbf{a}}}. \tag{11}$$

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. (11) 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 $\text{polylog}(T)$ – *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 $\text{polylog}(T)$). 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 $2^{s-(t-i)} \cdot \text{polylog}(T)$ 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 $2^{i-t} \cdot \text{polylog}(T)$, which can be as large as $S \cdot \text{polylog}(T)$. 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 $\text{polylog}(T)$ 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. This concludes the overview of the efficient implementation of the PCP provers.

E. 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 [14] and [24]. 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.[14] 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 $i+1$. Then, 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

[14] 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 $\hat{P}(\bar{\mathbb{x}}) = \hat{P}'(\mathbb{x}_{\pi(1)}, \ldots, \mathbb{x}_{\pi(n)})$.

simple. For any $\bar{\mathbb{x}} \in \mathbb{F}^n$, we simply have

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

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 the full version.

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

[1] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, 2014, pp. 459–474. [Online]. Available: https://doi.org/10.1109/SP.2014.36

[2] M. Walfish and A. J. Blumberg, "Verifying computations without reexecuting them," *Commun. ACM*, vol. 58, no. 2, pp. 74–84, 2015. [Online]. Available: http://doi.acm.org/10.1145/2641562

[3] M. Naor, "On cryptographic assumptions and challenges," in *CRYPTO*, 2003, pp. 96–109.

[4] C. Gentry and D. Wichs, "Separating succinct non-interactive arguments from all falsifiable assumptions," in *STOC*. ACM, 2011, pp. 99–108.

[5] Y. T. Kalai, R. Raz, and R. D. Rothblum, "Delegation for bounded space," in *STOC*. ACM, 2013, pp. 565–574.

[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *TOCT*, vol. 6, no. 3, pp. 13:1–13:36, 2014. [Online]. Available: http://doi.acm.org/10.1145/2633600

[7] C. Gentry, A. Sahai, and B. 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*, 2013, pp. 75–92. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40041-4_5

[8] R. Canetti, H. Lin, S. Tessaro, and V. 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*, 2015, pp. 468–497. [Online]. Available: https://doi.org/10.1007/978-3-662-46497-7_19

[9] E. Kushilevitz and R. Ostrovsky, "Replication is NOT needed: SINGLE database, computationally-private information retrieval," in *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, 1997, pp. 364–373. [Online]. Available: https://doi.org/10.1109/SFCS.1997.646125

[10] Y. T. Kalai, R. Raz, and R. D. Rothblum, "How to delegate computations: the power of no-signaling proofs," in *STOC*. ACM, 2014, pp. 485–494.

[11] Y. T. Kalai and R. 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*, 2015, pp. 422–442. [Online]. Available: https://doi.org/10.1007/978-3-662-48000-7_21

[12] Y. T. Kalai and O. Paneth, "Delegating RAM computations," in *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, 2016, pp. 91–118. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-53644-5_4

[13] O. Paneth and G. 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*, 2017, pp. 283–315. [Online]. Available: https://doi.org/10.1007/978-3-319-70503-3_9

[14] Z. Brakerski, J. Holmgren, and Y. T. 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*, 2017, pp. 474–482. [Online]. Available: http://doi.acm.org/10.1145/3055399.3055497

[15] S. Badrinarayanan, Y. Kalai, D. Khurana, A. Sahai, and D. Wichs, "Non-interactive delegation for low-space non-deterministic computation," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 25, p. 9, 2018. [Online]. Available: https://eccc.weizmann.ac.il/report/2018/009

[16] L. Babai, L. Fortnow, and C. Lund, "Non-deterministic exponential time has two-prover interactive protocols," *Computational Complexity*, vol. 1, pp. 3–40, 1991. [Online]. Available: https://doi.org/10.1007/BF01200056

[17] L. Babai, L. Fortnow, L. A. Levin, and M. 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*, 1991, pp. 21–31. [Online]. Available: http://doi.acm.org/10.1145/103418.103428

[18] I. Biehl, B. Meyer, and S. Wetzel, "Ensuring the integrity of agent-based computations by short proofs," in *Mobile Agents*, ser. Lecture Notes in Computer Science, vol. 1477. Springer, 1998, pp. 183–194.

[19] C. Dwork, M. Langberg, M. Naor, K. Nissim, and O. Reingold, "Succinct proofs for np and spooky interactions," Unpublished manuscript, 2001.

[20] T. Ito, H. Kobayashi, and K. 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*, 2009, pp. 217–228. [Online]. Available: https://doi.org/10.1109/CCC.2009.22

[21] N. Bitansky and A. 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*, 2012, pp. 255–272. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32009-5_16

[22] A. J. Blumberg, J. Thaler, V. Vu, and M. Walfish, "Verifiable computation using multiple provers," *IACR Cryptology ePrint Archive*, vol. 2014, p. 846, 2014. [Online]. Available: http://eprint.iacr.org/2014/846

[23] M. Naor, O. Paneth, and G. N. Rothblum, "Personal communication."

[24] T. Gur and R. 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.

[25] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: Interactive proofs for muggles," *J. ACM*, vol. 62, no. 4, pp. 27:1–27:64, 2015. [Online]. Available: http://doi.acm.org/10.1145/2699436

[26] L. Babai and L. Fortnow, "Arithmetization: A new method in structural complexity theory," *Computational Complexity*, vol. 1, pp. 41–66, 1991. [Online]. Available: https://doi.org/10.1007/BF01200057

[27] C. Lund, L. Fortnow, H. J. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," *J. ACM*, vol. 39, no. 4, pp. 859–868, 1992. [Online]. Available: http://doi.acm.org/10.1145/146585.146605

[28] A. Shamir, "IP = PSPACE," *J. ACM*, vol. 39, no. 4, pp. 869–877, 1992. [Online]. Available: http://doi.acm.org/10.1145/146585.146609

[29] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs," in *ITCS*. ACM, 2012, pp. 90–112.

[30] E. Ben-Sasson and M. Sudan, "Short pcps with polylog query complexity," *SIAM J. Comput.*, vol. 38, no. 2, pp. 551–607, 2008.

[31] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. 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, 2013, pp. 585–594. [Online]. Available: http://doi.acm.org/10.1145/2488608.2488681

[32] E. Ben-Sasson, I. Bentov, A. Chiesa, A. Gabizon, D. Genkin, M. Hamilis, E. Pergament, M. Riabzev, M. Silberstein, E. Tromer, and M. 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*, 2017, pp. 551–579. [Online]. Available: https://doi.org/10.1007/978-3-319-56617-7_19

[33] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," *IACR Cryptology ePrint Archive*, vol. 2018, p. 46, 2018. [Online]. Available: http://eprint.iacr.org/2018/046

[34] J. Kilian, "A note on efficient zero-knowledge proofs and arguments (extended abstract)," in *STOC*, 1992, pp. 723–732.

[35] S. Micali, "Computationally sound proofs," *SIAM J. Comput.*, vol. 30, no. 4, pp. 1253–1298, 2000. [Online]. Available: http://dx.doi.org/10.1137/S0097539795284959

[36] R. Canetti, O. Goldreich, and S. Halevi, "The random oracle methodology, revisited," *J. ACM*, vol. 51, no. 4, pp. 557–594, 2004. [Online]. Available: http://doi.acm.org/10.1145/1008731.1008734

[37] E. Ben-Sasson, A. Chiesa, and N. Spooner, "Interactive oracle proofs," in *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, 2016, pp. 31–60. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-53644-5_2

[38] O. Reingold, G. N. Rothblum, and R. 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*, 2016, pp. 49–62. [Online]. Available: http://doi.acm.org/10.1145/2897518.2897652

[39] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986, pp. 186–194.

[40] Y. T. Kalai, G. N. Rothblum, and R. 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*, 2017, pp. 224–251. [Online]. Available: https://doi.org/10.1007/978-3-319-63715-0_8

[41] R. Canetti, Y. Chen, L. Reyzin, and R. D. Rothblum, "Fiat-shamir and correlation intractability from strong kdm-secure encryption," *IACR Cryptology ePrint Archive*, vol. 2018, p. 131, 2018. [Online]. Available: http://eprint.iacr.org/2018/131

[42] S. T. V. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. 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*, 2012, pp. 253–268. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/setty

[43] J. 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*, 2013, pp. 71–89. [Online]. Available: https://doi.org/10.1007/978-3-642-40084-1_5

[44] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. 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*, 2017, pp. 863–880. [Online]. Available: https://doi.org/10.1109/SP.2017.43

[45] R. S. Wahby, Y. Ji, A. J. Blumberg, A. Shelat, J. Thaler, M. Walfish, and T. 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*

[48] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *ITCS*, 2012, pp. 326–349.

30 - November 03, 2017, 2017, pp. 2071–2086. [Online]. Available: http://doi.acm.org/10.1145/3133956.3133984

[46] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *ASIACRYPT*, 2010, pp. 321–340.

[47] H. Lipmaa, "Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments," in *TCC*, 2012, pp. 169–189.

[49] I. Damgård, S. Faust, and C. Hazay, "Secure two-party computation with low communication," in *TCC*, 2012, pp. 54–74.

[50] S. Goldwasser, H. Lin, and A. Rubinstein, "Delegation of computation without rejection problem from designated verifier cs-proofs," *IACR Cryptology ePrint Archive*, vol. 2011, p. 456, 2011.

[51] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "Recursive composition and bootstrapping for snarks and proof-carrying data," *IACR Cryptology ePrint Archive*, vol. 2012, p. 95, 2012.

[52] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," *IACR Cryptology ePrint Archive*, vol. 2012, p. 215, 2012.

[53] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth, "Succinct non-interactive arguments via linear interactive proofs," in *TCC*, 2013, pp. 315–333. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36594-2_18

[54] V. Koppula, A. B. Lewko, and B. Waters, "Indistinguishability obfuscation for turing machines with unbounded memory," in *STOC*. ACM, 2015, pp. 419–428.

[55] R. Canetti and J. Holmgren, "Fully succinct garbled RAM," in *ITCS*. ACM, 2016, pp. 169–178.

[56] Y. Chen, S. S. M. Chow, K. Chung, R. W. F. Lai, W. Lin, and H. Zhou, "Cryptography for parallel RAM from indistinguishability obfuscation," in *ITCS*. ACM, 2016, pp. 179–190.

[57] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *CRYPTO*, 2010, pp. 465–482.

[58] K.-M. Chung, Y. T. Kalai, and S. P. Vadhan, "Improved delegation of computation using fully homomorphic encryption," in *CRYPTO*, 2010, pp. 483–501.

[59] B. Applebaum, Y. Ishai, and E. Kushilevitz, "From secrecy to soundness: Efficient verification via secure computation," in *ICALP (1)*, 2010, pp. 152–163.

[60] B. Parno, M. Raykova, and V. Vaikuntanathan, "How to delegate and verify in public: Verifiable computation from attribute-based encryption," in *TCC*, 2012, pp. 422–439.

[61] M. Sudan, "Probabilistically checkable proofs - lecture notes," 2000, available at http://people.csail.mit.edu/madhu/pcp/pcp.ps.

[62] R. Rubinfeld and M. Sudan, "Robust characterizations of polynomials with applications to program testing," *SIAM J. Comput.*, vol. 25, no. 2, pp. 252–271, 1996. [Online]. Available: https://doi.org/10.1137/S0097539793255151

[63] P. Gemmell, R. J. Lipton, R. Rubinfeld, M. Sudan, and A. 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*, 1991, pp. 32–42. [Online]. Available: http://doi.acm.org/10.1145/103418.103429

[64] Y. Dodis, S. Halevi, R. D. Rothblum, and D. 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*, 2016, pp. 93–122. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-53015-3_4

[65] Y. Gurevich and S. Shelah, "Nearly linear time," in *Logic at Botik '89, Symposium on Logical Foundations of Computer Science, Pereslavl-Zalessky, USSR, July 3-8, 1989, Proceedings*, 1989, pp. 108–118. [Online]. Available: https://doi.org/10.1007/3-540-51237-3_10

[66] C. 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*, 2009, pp. 169–178. [Online]. Available: http://doi.acm.org/10.1145/1536414.1536440