

# Proofs of Useful Work

Marshall Ball\*   Alon Rosen†   Manuel Sabin‡   Prashant Nalini Vasudevan§

May 12, 2020

## Abstract

We give Proofs of Work (PoWs) whose hardness is based on a wide array of computational problems, including Orthogonal Vectors, 3SUM, All-Pairs Shortest Path, and any problem that reduces to them (this includes deciding any graph property that is statable in first-order logic). This results in PoWs whose completion does not waste energy but instead is *useful* for the solution of computational problems of practical interest.

The PoWs that we propose are based on delegating the evaluation of low-degree polynomials originating from the study of average-case fine-grained complexity. Beyond being hard on the average (based on worst-case hardness assumptions), we prove a *direct product theorem* for the problem of evaluating our polynomials. For our context, this implies that our Proofs of Useful Work cannot be sped up through batch evaluation across multiple instances.

For applications such as Bitcoin, which use PoWs on a *massive* scale, energy is typically wasted in huge proportions. We give a framework that can utilize such otherwise wasteful work.

Note: An updated version of this paper is available at <https://eprint.iacr.org/2018/559>. The update is to accommodate the fact (pointed out by anonymous reviewers) that the definition of Proof of Useful Work in this paper is already satisfied by a generic naive construction.

**Keywords:** Proofs of Work, Fine-Grained, Delegation, Blockchain, Direct Product Theorem.

---

\*Columbia University, New York, NY, USA. Email: [marshall@cs.columbia.edu](mailto:marshall@cs.columbia.edu).

†Efi Arazi School of Computer Science, IDC Herzliya, Israel. Email: [alon.rosen@idc.ac.il](mailto:alon.rosen@idc.ac.il).

‡UC Berkeley, Berkeley, CA, USA. Email: [msabin@berkeley.edu](mailto:msabin@berkeley.edu).

§CSAIL, Massachusetts Institute of Technology, Cambridge, MA, USA. Email: [prashvas@mit.edu](mailto:prashvas@mit.edu).

# 1 Introduction

Proofs of Work (PoWs) were introduced [DN92] to enforce that a certain amount of energy was expended for doing some task in an easily verifiable way. In most applications, PoWs force malicious users to accrue a large workload, thus guarding against email spam, denial of service attacks, and, most recently, double-spending in cryptocurrencies such as Bitcoin [Nak08]. Unfortunately, existing PoW schemes are often disconnected from the task they are attached to so that the work expended is not actually useful in accomplishing that task. More importantly, the work and energy expended is generally *not useful for anything except proving that work had in fact been done*.

To this end, PoWs are wasteful of real resources and energy and, in the massive-use case of Bitcoin, have even been called an “environmental disaster” [And13]. Two early attempts to combat this are Primecoin [Kin13] and Permacoin [MJS+14]. The former suggests a Proof of Work system whose outcome enables the search for chains of prime numbers, whereas the latter repurposes Bitcoin mining resources to achieve distributed storage of archival data based on Proofs of Retrievability (thus requiring clients to invest not just computational resources, but also storage).

Another line of work studies Proofs of Space [ABFG14, DFKP15, KK14], where a user must dedicate a significant amount of disk space, instead of computing power, to perform their task. This accomplishes a similar constraint on malicious activity to PoWs, since a group of users cannot over-perform a task without having access to massive amounts of memory. However, as a typical user has free disk space, such schemes will not similarly waste a resource like energy and pollute the environment.

In this work we present an alternative to the above approaches: Proofs of Work whose work is actually useful to solving *practical computational problems*.

## 1.1 Proofs of Work

At a high level, a *Proof of Work* involves three algorithms:

- $\text{Gen}(1^n)$  is a randomized algorithm that produces a *challenge*  $\mathbf{c}$ .
- $\text{Solve}(\mathbf{c})$  is an algorithm that solves the challenge  $\mathbf{c}$ , producing a solution  $\mathbf{s}$ .
- $\text{Verify}(\mathbf{c}, \mathbf{s})$  is a (possibly randomized) algorithm that verifies the solution  $\mathbf{s}$  to  $\mathbf{c}$ .

While  $\text{Gen}$  and  $\text{Verify}$  should run very quickly, there should be a notion of *hardness* for  $\text{Solve}$ 's runtime. More specifically, for some pre-specified length of working time, say  $t(n)$ ,  $\text{Solve}$  should be able to produce solutions that  $\text{Verify}$  accepts, but any attempted solutions produced by an algorithm running in less time (e.g.  $t(n)^{1-\epsilon}$  for any  $\epsilon > 0$ ) should be rejected by  $\text{Verify}$  with high probability. Thus valid solutions ‘prove’ that  $t(n)$  work was exerted in creating them. This hardness condition should also typically be stronger, in that it is also not possible to *batch* work over a large number of challenges. That is, more instance should imply proportionately more work.

Stated as is, however, these “solutions” don’t actually solve anything. One of the most commonly used PoWs, such as in Bitcoin, is simply to find a value  $\mathbf{s}$  so that hashing it together with the given challenge (e.g. with SHA-256) maps to anything with a certain amount of leading 0’s. Not only is hardness for this based on the *heuristic* belief that SHA-256 seems to behave unpredictably, but such an arbitrarily defined value  $\mathbf{s}$  is useless.

## 1.2 The Challenge: Proofs of Useful Work

One may hope to improve over the hash-based PoW in usefulness by considering a practical problem to begin with and allowing challenges to be instances of that problem. If solutions are easily

verifiable we may believe that we have created a useful PoW scheme, but we must keep in mind that we have two goals:

1. **Hardness:** Challenges can be issued such that responding to them correctly is (conditionally) guaranteed to necessitate actual work.
2. **Usefulness:** Computational tasks can be delegated as challenges to the workers such that the solution to the delegated task can be quickly and verifiably reconstructed from the workers’ response.

Achieving a PoW scheme that simultaneously attains both desiderata for a practical problem runs into two issues then. The first is that if we delegate arbitrary instances of the practical problem to be solved, *hardness* is no longer guaranteed as easy instances may be delegated; thus, some “challenges” may not actually require work. The second issue is that if we generate the challenges randomly we not only need an average-case hardness guarantee (which do not, in general, abound) but we lose our *usefulness* in our ability to delegate chosen instances of the problem we hope to have useful solutions for.

Prior work to this has only been concerned with one of these issues at a time. Hardness is exactly captured by PoWs, which have largely forsaken usefulness except for very specific tasks, such as heuristically recycling completed PoWs to be a weak source of randomness [BCG15] or to mint coins in older cryptocurrencies [JJ99]. Usefulness, on the other hand, can be viewed through the lens of verifiable delegation of computation [Wil16, BK16, GR17] which allows useful problem instances to be delegated with quickly reconstructible solutions, yet has not been concerned with any notion of average-case hardness.

The main challenge facing a designer of a Proof of Useful Work, then, is to marry *hardness* and *usefulness* for as large as possible a class of functions  $f$ , and with as much control as possible on the hardness parameter.

### 1.3 Our Results

We construct a Proof of Useful Work (uPoW) scheme based on the Orthogonal Vectors (OV) problem, which is a well-studied problem that is conjectured to take  $n^{2-o(1)}$  time to solve in the worst-case [Wil15]. Further, the computation invested by workers in this scheme can be used to solve the OV problem itself. Roughly, we show the following.

**Informal Theorem.** *Suppose OV takes  $n^{2-o(1)}$  time to decide. Given an instance  $\mathbf{x}$ , it is possible to (randomly) generate a challenge  $\mathbf{c}_{\mathbf{x}}$  such that:*

- *A valid solution  $\mathbf{s}$  to  $\mathbf{c}_{\mathbf{x}}$  can be computed in  $\tilde{O}(n^2)$  time.*
- *The validity of a candidate solution to  $\mathbf{c}_{\mathbf{x}}$  can be verified in  $\tilde{O}(n)$  time.*
- *Any valid solution to  $\mathbf{c}_{\mathbf{x}}$  requires  $n^{2-o(1)}$  time to compute. (Hardness)*
- *Given a valid solution to  $\mathbf{c}_{\mathbf{x}}$ , OV can be decided on the instance  $\mathbf{x}$  in  $\tilde{O}(n)$  time. (Usefulness)*

This is formally stated as Theorem 2 in Section 3, and the corresponding construction is Protocol 2. Theorem 2 is actually much more general – it applies analogously to generalizations of OV, called  $k$ -OV, allowing us to set the gap between the time taken to generate/verify and the time required to solve challenges to  $n^k$  for any  $k$ , assuming the hardness of these generalized problems (which is further implied by the Strong Exponential Time Hypothesis). This gives us *fine-grained*

*control over the hardness* at the cost of having an interactive uPoW, but we show how interaction can be removed in the Random Oracle Model (Section 5).

Theorem 2 is also stronger than the above informal statement in that it further guarantees that *it is not possible to batch work* over a large number of challenges - a very important feature for PoWs to have. That is, we achieve not just that a *single* challenge is hard to solve, but also that finding valid solutions to a set of  $\ell(n)$  (for any  $\ell$  that is polynomial in  $n$ ) independently generated challenges, possibly starting from different OV (resp.  $k$ -OV) instances, takes  $(\ell(n) \cdot n^{2-o(1)})$  (resp.  $(\ell(n) \cdot n^{k-o(1)})$ ) time *on average*.

We prove this by proving a direct sum theorem (and further show that this can be strengthened to a direct product theorem, which may be of independent interest) for the problem of computing a certain low-degree polynomial  $fOV^k$ . Roughly, the uPoW scheme is structured so that the challenges are randomly generated inputs to  $fOV^k$ , and the solutions are evaluations of  $fOV^k$  at these inputs (which are easy to verify and can be used to decide  $k$ -OV). Thus, Theorem 3 is the result that  $fOV^k$ 's evaluation cannot be batch-evaluated over multiple inputs on average.

**Informal Theorem.** *Suppose  $k$ -OV takes  $n^{k-o(1)}$  time to decide. For any polynomial  $\ell$ , and inputs  $\mathbf{x}_1, \dots, \mathbf{x}_{\ell(n)}$  of size  $n$  that are selected independently and uniformly at random, any algorithm that computes all the  $fOV^k(\mathbf{x}_i)$ 's correctly with probability  $1/n^{o(1)}$  takes time  $\ell(n) \cdot n^{k-o(1)}$ .*

Note that this in particular implies a worst-case to average-case reduction from OV to  $fOV$  (similar to the one shown in [BRSV17]), though strengthens it with a direct sum theorem.

Multiple instances of  $k$ -OV can now be *delegated* concurrently to be solved, while guaranteeing that finding such solutions will be hard on average. Thus we attain a non-batchable Proof of Work that is *useful* for  $k$ -OV.

## 1.4 The Usefulness of Our PoWs

It is important to note that, besides just OV, the problems 3SUM, All-Pairs Shortest Path (APSP) [BRSV17], and many other problems [BK16, GR17, Wil16] can also be represented as low-degree polynomials whose evaluations can be verified efficiently, and so almost immediately fit into our protocol under assumptions of their hardness. One sufficient condition for a hard problem to have the above properties is presented in [GR17], which studies doubly efficient interactive proofs for these problems (and also for the related low-degree polynomials, enabling the requisite efficient verification).

Further, any problem that quickly reduces to any of these conjectured hard problems can now also use our uPoW scheme for delegating instances of that problem. Thus we achieve *hardness* (and non-batchability) matching the conjectured worst-case hardness of whichever of these problems we base our uPoW on, and *usefulness* for any problems reducible to them.

To this point, many types of graph problems already have been shown to quickly reduce to these problems [AL13, WW10] and, of particular interest, the task of deciding whether or not a graph has a certain property *for any property that can be written in first-order logic* is reducible to (moderate-dimension) OV [GI16]. Thus, we further achieve uPoWs based on the hardness of OV and its generalizations such that *any* problem that can be phrased as a first-order graph property can be delegated.

We then have a rich framework for uPoWs: If you have a problem that is believed to be worst-case hard for some time bound, then if you can express it as a low-degree polynomial you can likely achieve a uPoW that will be useful for that problem and has a matching time bound for the hardness of its challenges. Alternatively, if you have any practical problem at all (even if it's easy

or has no believed hardness assumptions) it may still fit our framework by having a fast reduction to any existing problems that already have a uPoW.

One interesting point here is that, while many reductions to these problems so far have been to bolster belief in the hardness of these problems (as was the motivation for showing OV complete for all first-order graph properties [GI16]), this now gives an *algorithmic* motivation to reduce interesting problems to them. By reducing to OV, 3SUM, or APSP, you can now delegate Proofs of Useful Work. Finding classes of problems for which the latter two are complete now has a new and particularly strong motivation.

Further, [BRSV17] gives low-degree polynomials for the Triangle-Collection and the Zero-Weight Triangle problems of which 3SUM and APSP *both* reduce to both of these problems. This allows a uPoW that is based on either of these to simultaneously be *universally useful* for any problems reducing to 3SUM or APSP while also being hard so long as *at least one of* 3SUM or APSP are hard. Note that the Triangle-Collection problem’s hardness is *also* supported by the Strong Exponential Time Hypothesis for  $k$ -SAT (and so is also useful for  $k$ -SAT).

Our Proofs of Useful Work, then, may be viewed as a delegation of computation scheme for an expandable class of practical problems while still maintaining their PoW properties that prevent activities such as spam and double-spending. Moreover, the work we require can be distributed across a community, similar to ‘mining pools’ in Bitcoin, and can be done in a manner robust to Byzantine failures and noise (cf. [BK16]) and, further, identifies where the errors of malicious community members occurred.

This lends itself nicely to applications of PoWs like for blockchains. We show what modifications need to be made to our uPoWs and give a delegation scheme to outsource problems to the massive computing organism of Bitcoin in Section 7. As a final note, we show that our uPoWs can be made *zero-knowledge* in Appendix B. While this is the opposite direction of what one may want from *usefulness*, it is not only interesting that this is possible but it can also enable interesting dynamics. For example, Bitcoin workers may ‘go on strike while continuing to work’ by doing their uPoWs in zero-knowledge, thus continuing to maintain the blockchain and receiving bitcoins for mining blocks while also withholding their answers from the delegators.

## 2 Proofs of Useful Work

Proofs of useful work aim to achieve the following two desiderata simultaneously:

1. **Hardness:** Challenges can be issued such that responding to them correctly is (conditionally) guaranteed to necessitate actual work.
2. **Usefulness:** Computational tasks can be delegated as challenges to the workers such that the solution to the delegated task can be quickly and verifiably reconstructed from the workers’ response.

A Proof of Work on its own typically achieves just the *hardness* property. As mentioned in Section 1.1, a PoW has a way to quickly generate challenges of desired difficulty such that a solver is guaranteed to expend a certain amount of (non-batchable) work in producing an easily verifiable solution.

Unfortunately, the generated challenges are typically random and detached from any fixed delegatable instance  $x$  that someone may want to learn some  $f(x)$  for. Thus - on top of generating, solving, and verifying challenges - we must further generate challenges dependent on  $x$  to define a *usefulness* property: that solutions to challenges generated according to  $x$  can allow for the

reconstruction of  $f(x)$ . We formalize this and give a definition of Proofs of Useful Work (uPoWs). Syntactically, the definition involves four algorithms:

- $\text{Gen}(\mathbf{x})$  is a randomized algorithm that takes an instance  $\mathbf{x}$  and produces a *challenge*  $\mathbf{c}_\mathbf{x}$ .
- $\text{Solve}(\mathbf{c}_\mathbf{x})$  is an algorithm that solves the challenge  $\mathbf{c}_\mathbf{x}$ , producing a solution sketch  $\mathbf{s}$ .
- $\text{Verify}(\mathbf{c}_\mathbf{x}, \mathbf{s})$  is a randomized algorithm that verifies the solution sketch  $\mathbf{s}$  to the challenge  $\mathbf{c}_\mathbf{x}$ .
- $\text{Recon}(\mathbf{c}_\mathbf{x}, \mathbf{s})$  is an algorithm that given a valid  $\mathbf{s}$  for  $\mathbf{c}_\mathbf{x}$  reconstructs  $f(\mathbf{x})$ .

Taken together, these algorithms should result in an efficient proof system whose proofs are hard to find *and* useful. (Our hardness condition will be quite strong, including many arbitrary instances at once to account against batchability. This applies to arbitrary - even easy - delegated instances).

**Definition 1** (Proof of Useful Work). *A  $(t(n), \delta(n))$ -Proof of Useful Work (uPoW) for  $f$  consists of four algorithms (Gen, Solve, Verify, Recon). The algorithms must satisfy the following properties:*

**Efficiency:** For any  $|x| = n$

- For any  $\mathbf{x}$ ,  $\text{Gen}(\mathbf{x})$  runs in time  $\tilde{O}(n)$ .
- For any  $\mathbf{c}_\mathbf{x} \leftarrow \text{Gen}(\mathbf{x})$ ,  $\text{Solve}(\mathbf{c}_\mathbf{x})$  runs in time  $\tilde{O}(t(n))$ .
- For any  $\mathbf{c}_\mathbf{x} \leftarrow \text{Gen}(\mathbf{x})$  and any  $\mathbf{s}$ ,  $\text{Verify}(\mathbf{c}_\mathbf{x}, \mathbf{s})$  runs in time  $\tilde{O}(n)$ .
- For any  $\mathbf{c}_\mathbf{x} \leftarrow \text{Gen}(\mathbf{x})$  and  $\mathbf{s} \leftarrow \text{Solve}(\mathbf{c}_\mathbf{x})$ ,  $\text{Recon}(\mathbf{c}_\mathbf{x}, \mathbf{s})$  runs in time  $\tilde{O}(n)$ .

**Completeness:** For any  $\mathbf{c}_\mathbf{x} \leftarrow \text{Gen}(\mathbf{x})$  and any  $\mathbf{s} \leftarrow \text{Solve}(\mathbf{c}_\mathbf{x})$ ,

$$\Pr [\text{Verify}(\mathbf{c}, \mathbf{s}) = \text{accept}] = 1$$

Where the probability is taken over  $\text{Verify}$ 's randomness.

**Soundness:** For any  $\mathbf{s}$  and  $\mathbf{c}_\mathbf{x} \leftarrow \text{Gen}(\mathbf{x})$  such that  $\text{Recon}(\mathbf{c}_\mathbf{x}, \mathbf{s}) \neq f(\mathbf{x})$ ,

$$\Pr [\text{Verify}(\mathbf{c}_\mathbf{x}, \mathbf{s}) = \text{accept}] < \text{neg}(n)$$

Where the probability is taken over  $\text{Verify}$ 's randomness and  $|x| = n$ .

**Hardness:** For any polynomial  $\ell$ , any  $\mathbf{x}_1, \dots, \mathbf{x}_{\ell(n)}$  each of size  $n$ , any constant  $\epsilon > 0$ , and any algorithm  $\text{Solve}_\ell^*$  that runs in time  $\ell(n) \cdot t(n)^{1-\epsilon}$  when given  $\ell(n)$  challenges of size  $n$  as input,

$$\Pr \left[ \forall i : \text{Verify}(\mathbf{c}_i, \mathbf{s}_i) = \text{accept} \mid \begin{array}{l} (\mathbf{c}_i \leftarrow \text{Gen}(\mathbf{x}_i))_{i \in [\ell(n)]} \\ (\mathbf{s}_1, \dots, \mathbf{s}_{\ell(n)}) \leftarrow \text{Solve}_\ell^*(\mathbf{c}_1, \dots, \mathbf{c}_{\ell(n)}) \end{array} \right] < \delta(n)$$

Where the probability is taken over  $\text{Gen}$  and  $\text{Verify}$ 's randomness.

**Low-Variability:** The ratio of the expected worst-case to best-case complexity of solving challenges should be small. Formally, let  $S$  be the set of all algorithms that, for every  $\mathbf{x}$ , give solutions that cause  $\text{Verify}$  to accept. Then,

$$\frac{\max_{\mathbf{x}} \min_{\text{Solve}^* \in S} \mathbb{E} [\text{TIME}(\text{Solve}^*(\mathbf{c}_\mathbf{x})) \mid \mathbf{c}_\mathbf{x} \leftarrow \text{Gen}(\mathbf{x})]}{\min_{\mathbf{x}} \min_{\text{Solve}^* \in S} \mathbb{E} [\text{TIME}(\text{Solve}^*(\mathbf{c}_\mathbf{x})) \mid \mathbf{c}_\mathbf{x} \leftarrow \text{Gen}(\mathbf{x})]} \leq 1 + \epsilon(n)$$

Where the expectation is taken over  $\text{Gen}$  and  $\text{Solve}^*$ 's randomness.

[Manuel: This definition should capture what we were talking about in Tel Aviv.  $1 + \epsilon(n)$  will be the factor more that the unlucky sap whose given a hard instance (to the height of the variability) will have to do as opposed to someone who gets an easy instance (e.g. in the trivial uPoW case, the unlucky sap does  $2^\ell + w(n)$  work while the lucky one does only  $2^\ell$  work where  $w(n)$  is the worst case complexity of the tacked on delegated part; so the ratio would be  $1 + w(n)/2^\ell$ . What  $\epsilon(n)$  is reasonable for this? How does that  $\epsilon$  affect the trivial uPoW's need to rescale to match our desired  $\epsilon$ ? Is it enough? We achieve  $\epsilon = 0$  in our uPoWs. Maybe just make it a no-variability condition and talk about why trivial solution is bad off the bat for having any variability?]

We note that this definition implies (by combining completeness and soundness) the following notion of *usefulness*:

**Usefulness:** For any  $\mathbf{c}_x \leftarrow \text{Gen}(x)$  and  $\mathbf{s} \leftarrow \text{Solve}(\mathbf{c}_x)$ , we have

$$\text{Recon}(\mathbf{c}_x, \mathbf{s}) = f(x).$$

If we ignore the condition of *low-variability* for now (we discuss it's importance in Section 2.1), a Proof of Useful Work, then, is a strengthening of the notions of both PoWs and (non-interactive) verifiable delegation of computation. Namely, a PoW is a uPoW that doesn't require soundness (and thus *usefulness*), and a (non-interactive) verifiable delegation of computation scheme is a uPoW that doesn't require *hardness*.

For further context, without the *hardness* and *usefulness* condition we simply have a proof system, similar to the one described in [Wil16] which proved evaluations of low-degree polynomials (*usefulness* was implicit in this work). In [BRSV17], *hardness* is added to [Wil16]'s proof system to obtain PoWs and, in [BK16, GR17], this proof system's *usefulness* is explicitly explored to obtain verifiable delegation of computation. We now provide a framework to add both *hardness* and *usefulness* simultaneously to achieve Proofs of Useful Work.

Note that, as with the proof system, all of these definitions can - and often are in the delegation of computation framework - be made interactive.

Further note of the definition that a PoW does not typically need a Recon algorithm (and that any fixed choice of  $x$  would yield a valid PoW scheme) and that delegation of computation does not typically need a Gen algorithm (or Gen can become a fixed mapping to any one of its otherwise random outputs). We also reiterate that soundness is not required for a PoW: we don't necessarily care that a solution is correct so long as it took work (guaranteed in *hardness*) to produce. However, soundness is now crucial for the delegation of computation as a way to tell that the value we reconstruct is what we wanted and not some garbage value produced by a fake solution (even if *hardness* guarantees that it took time to find the fake solution).

## 2.1 The Importance of Low-Variability

We now discuss why, in highly competitive and incentivized environments like that of cryptocurrencies, the low-variability criteria in the above uPoW definition becomes a necessity. For, say, just preventing email spam, this condition may not be necessary since all we would care about there is that sending an email requires *some* decided upon amount of work; amount of work varying by polylog factors isn't very important so long as we can lower and upper bound the work required reasonably. In many cryptocurrencies, however, where groups of miners compete against each other to be the first to solve PoWs, variability can lead to a complete lack of fairness and even Denial of Service type attacks in the system.

As an illustrative example, let's consider a trivial uPoW that satisfies all conditions but low-variability. Namely, consider

*Remark 2.1.* We note that variability is different than the variance of the amount of work that will be needed as a random variable. [Manuel: Talk about difference here and note that low variance is generally implicit and wanted and that SHA-256 having some variance is already an annoying problem, and that all of our uPoWs actually have *variance of 0*.]

### 3 A Useful PoW for Orthogonal Vectors

We now present a Proof of Useful Work (uPoW) for the Orthogonal Vectors (OV) problem and its generalization  $k$ -OV. The properties possessed by OV that enable this construction are also shared by other problems mentioned earlier, including 3SUM and APSP as noted in [BRSV17] and also for an array of other problems [BK16, GR17, Wil16]. Consequently, while we focus on OV, uPoWs for them can be constructed along the lines of the one here. Further, these constructions would immediately provide uPoWs for other problems that reduce to OV, 3SUM, etc. in a fine-grained manner with little, if any, degradation of security. Of particular interest, deciding graph properties that are storable in first-order logic all reduce to (moderate-dimension) OV [GI16] and so we obtain uPoWs useful for *any* problem storable as a first-order graph property.

All the algorithms we consider henceforth – reductions, adversaries, etc. – are *non-uniform Word-RAM algorithms* (with words of size  $O(\log n)$  where  $n$  will be clear from context) unless stated otherwise, both in our hardness assumptions and our constructions. Security against such adversaries is necessary for PoWs to remain hard in the presence of pre-processing, which is typical in the case of Bitcoin, for instance, where specialized hardware is often used. In the case of reductions, this non-uniformity is solely used to ensure that specific parameters determined completely by instance size (such as the prime  $p(n)$  in Definition 4) are known to the reductions and delegators do not need to compute them afresh for each problem they delegate.

**Definition 2** (Orthogonal Vectors). *The OV problem on vectors of dimension  $d$  (denoted  $\text{OV}_d$ ) is to determine, given two sets  $U, V$  of  $n$  vectors from  $\{0, 1\}^{d(n)}$  each, whether there exist  $u \in U$  and  $v \in V$  such that  $\langle u, v \rangle = 0$  (over  $\mathbb{Z}$ ). If left unspecified,  $d$  is to be taken to be  $\lceil \log^2 n \rceil$ .*

OV is commonly conjectured to require  $n^{2-o(1)}$  to decide, for which many conditional fine-grained hardness results are based on [Wil15], and has been shown to be true if the Strong Exponential Time Hypothesis (SETH) holds [Wil05]. This hardness and the hardness of its generalization to  $k$ -OV of requiring  $n^{k-o(1)}$  time (which also holds under SETH) are what we base the hardness of our uPoWs on. We now define  $k$ -OV.

**Definition 3** ( $k$ -Orthogonal Vectors). *For an integer  $k \geq 2$ , the  $k$ -OV problem on vectors of dimension  $d$  is to determine, given  $k$  sets  $(U_1, \dots, U_k)$  of  $n$  vectors from  $\{0, 1\}^{d(n)}$  each, whether there exist  $u^s \in U_s$  for each  $s \in [k]$  such that over  $\mathbb{Z}$ ,*

$$\sum_{\ell \in [d(n)]} u_\ell^1 \cdots u_\ell^k = 0$$

*We say that such a set of vectors is  $k$ -orthogonal. If left unspecified,  $d$  is to be taken to be  $\lceil \log^2 n \rceil$ .*

While these problems are conjectured worst-case hard, there are currently no wide-held beliefs for distributions that it may be average-case hard over. [BRSV17], however, defines a related problem that is shown to be average-case hard when assuming the worst-case hardness of  $k$ -OV. The average-case hard problem is that of evaluating the following polynomial:



For any prime number  $p$ , we define the polynomial  $f\text{OV}_{n,d,p}^k : \mathbb{F}_p^{knd} \rightarrow \mathbb{F}_p$  as follows. Its inputs are parsed in the manner that those of  $k$ -OV are: below, for any  $s \in [k]$  and  $i \in [n]$ ,  $u_i^s$  represents the  $i^{\text{th}}$  vector in  $U_s$ , and for  $\ell \in [d]$ ,  $u_{i\ell}^s$  represents its  $\ell^{\text{th}}$  coordinate.

$$f\text{OV}_{n,d,p}^k(U_1, \dots, U_k) = \sum_{i_1, \dots, i_k \in [n]} \prod_{\ell \in [d]} \left(1 - u_{i_1\ell}^1 \cdots u_{i_k\ell}^k\right)$$

Note that the degree of this polynomial is  $kd$ . When given an instance of  $k$ -OV (from  $\{0, 1\}^{knd}$ ) as input,  $f\text{OV}_{n,d,p}^k$  counts the number of tuples of  $k$ -orthogonal vectors (modulo  $p$ ).

For small  $d$  (e.g.  $d = \lceil \log^2 n \rceil$ ), this is a fairly low-degree polynomial. The following definition gives the family of such polynomials parameterized by input size. This family was shown to be hard to compute on uniformly random inputs if  $k$ -OV is hard in the worst-case [BRSV17].

**Definition 4** ( $\mathcal{FOV}^k$ ). Consider an integer  $k \geq 2$ . Let  $p(n)$  be the smallest prime number larger than  $n^{\log n}$ , and  $d(n) = \lceil \log^2 n \rceil$ .  $\mathcal{FOV}^k$  is the family of functions  $\left\{f\text{OV}_{n,d(n),p(n)}^k\right\}$ .

*Remark 3.1.* We note that most of our results would hold for a much smaller choice of  $p(n)$  above – anything larger than  $n^k$  would do. The reason we choose  $p$  to be this large is to achieve negligible soundness error in interactive protocols we shall be designing for this family of functions (see Protocol 1). Another way to achieve this is to use large enough extension fields of  $\mathbb{F}_p$  for smaller  $p$ 's; this is in fact preferable as the value of  $p(n)$  as defined now is much harder to compute for uniform algorithms.

While, assuming  $k$ -OV takes  $n^{k-o(1)}$  time in the worst-case implies that evaluating polynomials in  $\mathcal{FOV}$  on points is just as hard on average, it is still easily delegate the evaluation of such points.

### 3.1 Preliminary Protocols

We describe here a protocol (Protocol 1) that can prove, given valid  $x$  and  $y$ , that  $y = f\text{OV}_{n,d,p}^k(x)$  (and can even delegate that evaluation), which will be used as a sub-routine in our final uPoW protocol, and which will also find use in proving its security. This protocol is a  $(k-1)$ -round interactive proof that, specifically, given  $U_1, \dots, U_k \in \mathbb{F}_p^{nd}$  and  $y \in \mathbb{F}_p$ , proves that  $f\text{OV}_{n,d,p}^k(U_1, \dots, U_k) = y$ .

The special case of  $k = 2$  for OV was shown as a non-interactive (MA) protocol in [Wil16] and this MA scheme was used to create PoWs based on OV, 3SUM, and APSP in [BRSV17], however with randomly generated challenges that were not *useful*. The following interactive proof is essentially the sum-check protocol, but in our case we need to pay close attention to the complexity of the prover and the verifier and so use ideas from [Wil16].

We will set up the following definitions before describing the protocol. For each  $s \in [k]$ , consider the univariate polynomials  $\phi_1^s, \dots, \phi_d^s : \mathbb{F}_p \rightarrow \mathbb{F}_p$ , where  $\phi_\ell^s$  represents the  $\ell^{\text{th}}$  column of  $U_s$  – that is, for  $i \in [n]$ ,  $\phi_\ell^s(i) = u_{i\ell}^s$ . Each  $\phi_\ell^s$  has degree at most  $(n-1)$ .  $f\text{OV}_{n,d,p}^k$  can now be written as:

$$\begin{aligned} f\text{OV}_{n,d,p}^k(U_1, \dots, U_k) &= \sum_{i_1, \dots, i_k \in [n]} \prod_{\ell \in [d]} \left(1 - u_{i_1\ell}^1 \cdots u_{i_k\ell}^k\right) \\ &= \sum_{i_1, \dots, i_k \in [n]} \prod_{\ell \in [d]} \left(1 - \phi_\ell^1(i_1) \cdots \phi_\ell^k(i_k)\right) \\ &= \sum_{i_1, \dots, i_k \in [n]} q(i_1, \dots, i_k) \end{aligned}$$

where  $q$  is defined for convenience as:

$$q(i_1, \dots, i_k) = \prod_{\ell \in [d]} \left(1 - \phi_\ell^1(i_1) \cdots \phi_\ell^k(i_k)\right)$$

The degree of  $q$  is at most  $D = k(n-1)d$ . Note that  $q$  can be evaluated at any point in  $\mathbb{F}_p^k$  in time  $\tilde{O}(knd \log p)$ , by evaluating all the  $\phi_\ell^s(i_s)$ 's (these polynomials can be found using fast interpolation techniques for univariate polynomials [Hor72]), computing each term in the above product and then multiplying them.

For any  $s \in [k]$  and  $\alpha_1, \dots, \alpha_{s-1} \in \mathbb{F}_p$ , define the following univariate polynomial:

$$q_{s, \alpha_1, \dots, \alpha_{s-1}}(x) = \sum_{i_{s+1}, \dots, i_k \in [n]} q(\alpha_1, \dots, \alpha_{s-1}, x, i_{s+1}, \dots, i_k)$$

Every such  $q_s$  has degree at most  $(n-1)d$  – this can be seen by inspecting the definition of  $q$ . With these definitions, the interactive proof is described as Protocol 1 below. The completeness and soundness of this interactive proof is then asserted by Theorem 1, which is proven in Section 4.

### Interactive Proof for $\mathcal{FOV}^k$ :

- The prover sends the coefficients of a univariate polynomial  $q_1^*$  of degree at most  $(n-1)d$ .
- The verifier checks that  $\sum_{i_1 \in [n]} q_1^*(i_1) = y$ . If not, it rejects.
- For  $s$  from 1 up to  $k-2$ :
  - The verifier sends a random  $\alpha_s \leftarrow \mathbb{F}_p$ .
  - The prover sends the coefficients of a polynomial  $q_{s+1, \alpha_1, \dots, \alpha_s}^*$  of degree at most  $(n-1)d$ .
  - The verifier checks that  $\sum_{i_{s+1} \in [n]} q_{s+1, \alpha_1, \dots, \alpha_s}^*(i_{s+1}) = q_{s, \alpha_1, \dots, \alpha_{s-1}}(\alpha_s)$ . If not, it rejects.
- The verifier picks  $\alpha_{k-1} \leftarrow \mathbb{F}_p$  and checks that  $q_{k-1, \alpha_1, \dots, \alpha_{k-2}}^*(\alpha_{k-1}) = q_{k-1, \alpha_1, \dots, \alpha_{k-2}}(\alpha_{k-1})$ , computed using the fact that  $q_{k-1, \alpha_1, \dots, \alpha_{k-2}}(\alpha_{k-1}) = \sum_{i_k \in [n]} q_{k, \alpha_1, \dots, \alpha_{k-1}}(i_k)$ . If not, it rejects.
- If the verifier hasn't rejected yet, it accepts.

Protocol 1: Interactive Proof for  $\mathcal{FOV}^k$ .

**Theorem 1.** *For any  $k \geq 2$ , let  $d$  and  $p$  be as in Definition 4. Protocol 1 is a  $(k-1)$ -round interactive proof for proving that  $y = \mathcal{FOV}^k(x)$ . This protocol has perfect completeness and soundness error at most  $\left(\frac{knd}{p}\right)$ . The prover runs in time  $\tilde{O}(n^k d \log p)$ , and the verifier in time  $\tilde{O}(knd^2 \log p)$ .*

As observed earlier, Protocol 1 is non-interactive when  $k = 2$ . We then get the following corollary for  $\mathcal{FOV}$ .

**Corollary 1.** *For  $k = 2$ , let  $d$  and  $p$  be as in Definition 4. Protocol 1 is an MA proof for proving that  $y = \mathcal{FOV}(x)$ . This protocol has perfect completeness and soundness error at most  $\left(\frac{2nd}{p}\right)$ . The prover runs in time  $\tilde{O}(n^2)$ , and the verifier in time  $\tilde{O}(n)$ .*

We now currently have a proof system with completeness and soundness along with efficiency bounds on the prover and verifier. In the framework of uPoWs in Section 2, we still need a way to extend this proof system to have *hardness* and *usefulness* if we are to obtain a uPoW.

### 3.2 The uPoW Protocol

We now present Protocol 2, which we show to be a Proof of Useful Work for  $k$ -OV.

#### Proof of Useful Work for $k$ -OV:

- **Gen( $\mathbf{x}$ ):**
  - Given an instance  $\mathbf{x} \in \{0, 1\}^{knd}$ , interpret  $\mathbf{x}$  as an element of  $\mathbb{F}_p^{knd}$  (where  $p = p(n)$  is as in Definition 4).
  - Pick a random  $\mathbf{r} \in \mathbb{F}_p^{knd}$ .
  - Output the set of vectors  $\mathbf{c}_\mathbf{x} = \{\mathbf{y}_t = \mathbf{x} + t\mathbf{r} \mid t \in [kd + 1]\}$ .
- **(Solve, Verify) work as follows given  $\mathbf{c}_\mathbf{x} = \{\mathbf{y}_t\}$ :**
  - Solve computes  $z_t = fOV_{n,d,p}^k(\mathbf{y}_t)$  and outputs the set  $\mathbf{s} = \{z_t\}_{t \in [kd+1]}$ .
  - For each  $t$  in parallel: Solve and Verify run Protocol 1 with input  $(\mathbf{y}_t, z_t)$ .
  - Verify accepts iff all of the above instances of Protocol 1 accept.
- **Recon( $\mathbf{c}_\mathbf{x}, \mathbf{s}$ ):**
  - Interpret  $z_1, \dots, z_{kd+1}$  as the evaluations of a univariate polynomial  $h(t)$  of degree  $kd$  at  $t = 1, \dots, kd + 1$ .
  - Interpolate to find the coefficients of  $h$  and compute  $z = h(0)$ .
  - If  $z \neq 0$ , output 1, else 0 as the answer to the  $k$ -OV instance.

Protocol 2: Proof of Useful Work for  $k$ -OV.

**Theorem 2.** For any  $k \geq 2$ , suppose  $k$ -OV takes  $n^{k-o(1)}$  time to decide for any  $d = \omega(\log n)$ . Then, Protocol 2 is an  $(n^k, \delta)$ -Proof of Useful Work for  $k$ -OV for any function  $\delta(n) > 1/n^{o(1)}$ .

*Remark 3.2.* As is, this will be an interactive uPoW. In the special case of  $k = 2$ , Corollary 1 gives us that we have a regular non-interactive uPoW. If we want to remove interaction for general  $k$ -OV, however, we could use the MA proof in [Wil16] at cost of verification and reconstruction taking time  $\tilde{O}(n^{k/2})$ . To keep verification and reconstruction time at  $\tilde{O}(n)$ , we instead show how to remove interaction in the Random Oracle model in Section 5. This will allow us to *tune* hardness in that we can choose  $k$  and thus the amount of work,  $n^{k-o(1)}$ , that must be completed while always only needing  $\tilde{O}(n)$  time to reconstruct.

We will use Theorem 1 to argue for the completeness and soundness of Protocol 2. In order to prove the hardness, we will need lower bounds on how well the problem that Solve is required to solve can be batched. We first define what it means for a function to be non-batchable in the average-case in a manner compatible with the hardness requirement. Note that this requirement is stronger than being non-batchable in the worst-case.

**Definition 5.** Consider a function family  $\mathcal{F} = \{f_n : \mathcal{X}_n \rightarrow \mathcal{Y}_n\}$ , and a family of distributions  $\mathcal{D} = \{D_n\}$ , where  $D_n$  is over  $\mathcal{X}_n$ .  $\mathcal{F}$  is not  $(\ell, t, \delta)$ -batchable on average over  $\mathcal{D}$  if, for any algorithm Batch that runs in time  $\ell(n)t(n)$  when run on  $\ell(n)$  inputs from  $\mathcal{X}_n$ , when it is given as input  $\ell(n)$  independent samples from  $D_n$ ,

$$\Pr_{x_i \leftarrow D_n} [\text{Batch}(x_1, \dots, x_{\ell(n)}) = (f_n(x_1), \dots, f_n(x_{\ell(n)}))] < \delta(n)$$

We will be concerned with the case where the batched time  $t(n)$  is less than the time it takes to compute  $f_n$  on a single instance. This sort of statement is what a direct sum theorem for  $\mathcal{F}$ 's hardness would guarantee. Theorem 3, then, claims that we achieve this non-batchability for  $\mathcal{FOV}^k$  and, as  $\mathcal{FOV}^k$  is one of the things that **Solve** is required evaluate on, we will be able to show the desired hardness of Protocol 2. We prove Theorem 3 via a direct sum theorem in Appendix A, and prove a weaker version for illustrative purposes in Section 6 along with a direct product theorem.

**Theorem 3.** *For any  $k \geq 2$ , suppose  $k$ -OV takes  $n^{k-o(1)}$  time to decide for any  $d = \omega(\log n)$ . Then, for any constants  $c, \epsilon > 0$  and  $\delta < \epsilon/2$ ,  $\mathcal{FOV}^k$  is not  $(n^c, n^{k-\epsilon}, 1/n^\delta)$ -batchable on average over the uniform distribution over its inputs.*

We now put all the above together to prove Theorem 2 as follows.

**Proof of Theorem 2.** We prove that Protocol 2 satisfies the various requirements demanded of a Proof of Useful Work for  $k$ -OV in turn.

**Efficiency.**

- **Gen**( $\mathbf{x}$ ) simply computes  $(kd + 1)$  linear combinations over  $\mathbb{F}^{knd}$  to create all the elements of  $\mathbf{c}_\mathbf{x}$  and so takes the time of  $O(kd)$  basic operations on  $\mathbb{F}^{knd}$ . As  $d = \log^2 n$  and  $p \leq 2n^{\log n}$  by Bertrand-Chebyshev's Theorem, this takes  $\tilde{O}(n)$  time.
- **Solve** computes  $f\text{OV}_{n,d,p}^k(\mathbf{y}_t)$  on  $(kd + 1)$  values of  $\mathbf{y}_t$ , each of which can be done in  $\tilde{O}(n^k)$  time. It then runs the prover in  $(kd + 1)$  instantiations of Protocol 1, each of which can be done in  $\tilde{O}(n^k)$  time by Theorem 1. So in all it takes takes  $\tilde{O}(n^k(kd + 1)) = \tilde{O}(n^k)$  time.
- **Verify** runs the verifier in  $(kd + 1)$  instantiations of Protocol 1, taking a total of  $\tilde{O}(n(kd + 1)) = \tilde{O}(n)$  time, again by Theorem 1.
- **Recon** does interpolation and evaluation of a univariate polynomial of degree  $kd$  over  $\mathbb{F}_p$ , which can be done in time  $\tilde{O}((kd)^2)$ , which is much less than  $n$ .

**Usefulness.** Define the univariate polynomial  $h_{\mathbf{x},\mathbf{r}}$  as  $h_{\mathbf{x},\mathbf{r}}(t) = f\text{OV}_{n,d,p}^k(\mathbf{x} + t\mathbf{r})$ . Note that the degree of  $h_{\mathbf{x},\mathbf{r}}$  is at most the degree of  $f\text{OV}_{n,d,p}^k$ , which is  $kd$ . When **Solve** produces the correct evaluations  $z_1, \dots, z_{kd+1}$  of  $f\text{OV}_{n,d,p}^k$  at  $(\mathbf{x} + \mathbf{r}), \dots, (\mathbf{x} + (kd + 1)\mathbf{r})$ , these are also evaluations of  $h_{\mathbf{x},\mathbf{r}}$  at  $1, \dots, (kd + 1)$ . So when **Recon** in Protocol 2 interpolates to find coefficients of a polynomial, what it finds is  $h_{\mathbf{x},\mathbf{r}}$ . Further, its constant coefficient is  $h_{\mathbf{x},\mathbf{r}}(0) = f\text{OV}_{n,d,p}^k(\mathbf{x})$ . As  $p$  is large enough, this value counts the number of  $k$ -orthogonal vectors in the  $k$ -OV instance  $\mathbf{x} \in \{0, 1\}^{knd}$ . So  $\mathbf{x}$  has  $k$ -orthogonal vectors iff this value is non-zero.

**Completeness and Soundness.** This follows immediately from the completeness and soundness of Protocol 1 as an interactive proof for  $\mathcal{FOV}^k$ , as stated in Theorem 1, as this is the protocol that **Solve** and **Verify** engage in, where soundness further uses the correctness of **Recon** as argued above.

**Hardness.** We proceed by contradiction. On a high level, we will assume that there is a set of instances for which the hardness constraint of our protocol *breaks* on average and show that, given these instances as non-uniform advice, we can then break  $\mathcal{FOV}^k$ 's average-case non-batchability:

Suppose there is an (interactive) algorithm **Solve**<sup>\*</sup>, a polynomial  $\ell$ , a set of  $k$ -OV instances  $\mathbf{x}_1, \dots, \mathbf{x}_{\ell(n)} \in \{0, 1\}^{knd}$ , and an  $\epsilon > 0$  such that **Solve**<sup>\*</sup> runs in time  $\ell(n)n^{k-\epsilon}$  and makes **Verify** accept on all these instances with probability at least  $\delta(n)$  that is  $1/n^{o(1)}$ . Let the  $f\text{OV}$  instances

produced by  $\text{Gen}(\mathbf{x}_i)$  be  $\{\mathbf{y}_t^i\}$ , and the corresponding sets  $\mathbf{s}_i$  produced by  $\text{Solve}^*$  be  $\{z_t^i\}$ . So  $\text{Solve}^*$  succeeds as a prover in Protocol 1 for *all* the instances  $\{(\mathbf{y}_t^i, z_t^i)\}$  with probability at least  $\delta(n)$ .

By the negligible soundness of Protocol 1 guaranteed by Theorem 1, in order to do this,  $\text{Solve}^*$  has to use the correct values  $f\text{OV}_{n,d,p}^k(\mathbf{y}_t^i)$  for all  $z_t^i$ 's with probability negligibly close to  $\delta(n)$  and definitely more than, say,  $\delta(n)/2$ . In particular, with this probability, it has to explicitly compute  $f\text{OV}_{n,d,p}^k$  at  $\mathbf{y}_1^1, \dots, \mathbf{y}_1^{\ell(n)}$ , all of which are independent uniform points in  $\mathbb{F}_p^{knd}$ .

Such a  $\text{Solve}^*$  can now be used as follows to compute  $f\text{OV}_{n,d,p}^k$  on any independently random  $\mathbf{y}_1, \dots, \mathbf{y}_{\ell(n)} \in \mathbb{F}_p^{knd}$ : Take the points  $\mathbf{x}_1, \dots, \mathbf{x}_{\ell(n)}$  (on which  $\text{Solve}^*$  succeeds as above) as non-uniform advice. Now for each  $i \in [\ell(n)]$ , instead of choosing a random “slope”  $\mathbf{r}$  for a line passing through  $\mathbf{y}_i$  (as  $\text{Gen}(\mathbf{y}_i)$  would do), we will draw the line connecting  $\mathbf{y}_i$  and easy instance  $\mathbf{x}_i$  and have the challenge be the set of points,  $\{\mathbf{y}_t^i\}$ , on *that* line. Since  $\mathbf{y}_i$  is uniformly random, this is simply a random line passing through  $\mathbf{x}_i$ . That is, a random challenge based on instance  $\mathbf{x}_i$  which was assumed *easy* on average for  $\text{Solve}^*$ . Thus,  $\text{Solve}^*$  computes  $f\text{OV}_{n,d,p}^k$  on all of these correctly with probability greater than  $\delta(n)/2$ .

Hence  $\text{Solve}^*$  can be used to construct a non-uniform algorithm that runs in time  $\ell(n)n^{k-\epsilon}$  and computes  $f\text{OV}_{n,d,p}^k$  correctly on all of  $\mathbf{y}_1, \dots, \mathbf{y}_{\ell(n)}$  with probability at least  $\delta(n)/2$  ( $\gg 1/n^{\epsilon/2}$ ) when all of these are distributed independently and uniformly. But this is exactly what Theorem 3 says is impossible. So such a  $\text{Solve}^*$  cannot exist, and this proves the hardness of Protocol 2.

**Low-Variability.** [Manuel: needs to be proved]

We have thus proven all the properties necessary and hence Protocol 2 is indeed an  $(n^k, \delta)$ -Proof of Useful Work for  $k$ -OV for any  $\delta(n) > 1/n^{o(1)}$ .  $\square$

## 4 Verifying $\mathcal{FOV}^k$

In this section, we prove Theorem 1 (stated in Section 3), which is about Protocol 1 being a valid interactive proof for proving evaluations of  $\mathcal{FOV}^k$ . We use here terminology from the theorem statement and protocol description. Recall the the input to the protocol is  $U_1, \dots, U_k \in \mathbb{F}_p^{nd}$  and  $y \in \mathbb{F}_p$ , and the prover wishes to prove that  $y = f\text{OV}_{n,d,p}^k(U_1, \dots, U_k)$ .

**Completeness.** If indeed  $y = f\text{OV}_{n,d,p}^k(U_1, \dots, U_k)$ , the prover can make the verifier in the protocol accept by using the polynomials  $(q_1, q_{2,\alpha_1}, \dots, q_{k,\alpha_1, \dots, \alpha_k})$  in place of  $(q_1^*, q_{2,\alpha_1}^*, \dots, q_{k,\alpha_1, \dots, \alpha_k}^*)$ . Perfect completeness is then seen to follow from the definitions of these polynomials and their relation to  $q$  and hence  $f\text{OV}_{n,d,p}^k$ .

**Soundness.** Suppose  $y \neq f\text{OV}_{n,d,p}^k(U_1, \dots, U_k)$ . We now analyze the probability with which a cheating prover could make the verifier accept.

To start with, note that the prover's  $q_1^*$  has to be different from  $q_1$ , as otherwise the check in the second step would fail. Further, as the degree of these polynomials is less than  $nd$ , the probability that the verifier will then choose an  $\alpha_1$  such that  $q_1^*(\alpha_1) = q_1(\alpha_1)$  is less than  $\frac{nd}{p}$ .

If this event does not happen, then the prover has to again send a  $q_{2,\alpha_1}^*$  that is different from  $q_{2,\alpha_1}$ , which again agree on  $\alpha_2$  with probability less than  $\frac{nd}{p}$ . This goes on for  $(k-1)$  rounds, at the end of which the verifier checks whether  $q_{k-1}^*(\alpha_{k-1})$  is equal to  $q_{k-1}(\alpha_{k-1})$ , which it computes by itself. If at least one of these accidental equalities at a random point has not occurred throughout

the protocol, the verifier will reject. The probability that no violations occur over the  $(k-1)$  rounds is, by the union bound, less than  $\frac{knd}{p}$ .

**Efficiency.** Next we discuss details of how the honest prover and the verifier are implemented, and analyze their complexities. To this end, we will need the following algorithmic results about computations involving *univariate* polynomials over finite fields.

**Lemma 1** (Fast Multi-point Evaluation [Fid72]). *Given the coefficients of a univariate polynomial  $q : \mathbb{F}_p \rightarrow \mathbb{F}_p$  of degree at most  $N$ , and  $N$  points  $x_1, \dots, x_N \in \mathbb{F}_p$ , the set of evaluations  $(q(x_1), \dots, q(x_N))$  can be computed in time  $O(N \log^3 N \log p)$ .*

**Lemma 2** (Fast Interpolation [Hor72]). *Given  $N+1$  evaluations of a univariate polynomial  $q : \mathbb{F}_p \rightarrow \mathbb{F}_p$  of degree at most  $N$ , the coefficients of  $q$  can be computed in time  $O(N \log^3 N \log p)$ .*

To start with, both the prover and verifier compute the coefficients of all the  $\phi_\ell^s$ 's. Note that, by definition, they know the evaluation of each  $\phi_\ell^s$  on  $n$  points, given by  $\{(i, u_{i\ell}^s)\}_{i \in [n]}$ . This can be used to compute the coefficients of each  $\phi_\ell^s$  in time  $\tilde{O}(n \log p)$  by Lemma 2. The total time taken is hence  $\tilde{O}(knd \log p)$ .

The proof of the following proposition specifies further details of the prover's workings.

**Proposition 1.** *The coefficients of the polynomial  $q_{s, \alpha_1, \dots, \alpha_{s-1}}$  can be computed in time  $\tilde{O}((n^{k-s+1}d + nd^2) \log p)$  given the above preprocessing.*

- Fix some value of  $s, \alpha_1, \dots, \alpha_{s-1}$ .
- For each  $\ell \in [d]$ , compute the evaluation of  $\phi_\ell^s$  on  $nd$  points, say  $\{1, \dots, nd\}$ .
  - Since its coefficients are known, the evaluations of each  $\phi_\ell^s$  on these  $nd$  points can be computed in time  $\tilde{O}(nd \log p)$  by Lemma 1, for a total of  $\tilde{O}(nd^2 \log p)$  for all the  $\phi_\ell^s$ 's.
- For each setting of  $i_{s+1}, \dots, i_k$ , compute the evaluations of the polynomial  $\rho_{i_{s+1}, \dots, i_k}(x) = q(\alpha_1, \dots, \alpha_{s-1}, x, i_{s+1}, \dots, i_k)$ , on the points  $\{1, \dots, nd\}$ .
  - First substitute the constants  $\alpha_1, \dots, \alpha_{s-1}, i_{s+1}, \dots, i_k$  into the definition of  $q$ .
  - This requires computing, for each  $\ell \in [d]$  and  $s' \in [k] \setminus \{s\}$ , either  $\phi_\ell^{s'}(\alpha_s)$  or  $\phi_\ell^{s'}(i_s)$ . All of this can be done in time  $\tilde{O}(knd \log p)$  by direct polynomial evaluations since the coefficients of the  $\phi_\ell^{s'}$ 's are known.
  - This reduces  $q$  to a product of  $d$  univariate polynomials of degree less than  $n$ , whose evaluations on the  $nd$  points can now be computed in time  $\tilde{O}(knd \log p)$  by multiplying the constants computed in the above step with the evaluations of  $\phi_\ell^{s'}$  on these points, and subtracting from 1.
  - The product of the evaluations can now be computed in time  $\tilde{O}(nd^2 \log p)$  to get what we need.
- Add up the evaluations of  $\rho_{i_{s+1}, \dots, i_k}$  pointwise over all settings of  $(i_{s+1}, \dots, i_k)$ .
  - There are  $n^{k-s}$  possible settings of  $(i_{s+1}, \dots, i_k)$ , and for each of these we have  $nd$  evaluations. All the additions hence take  $\tilde{O}(n^{k-s+1}d \log p)$  time.
- This gives us  $nd$  evaluations of  $q_{s, \alpha_1, \dots, \alpha_{s-1}}$ , which is a univariate polynomial of degree at most  $(n-1)d$ . So its coefficients can be computed in time  $\tilde{O}(nd \log p)$  by Lemma 2.
- It can be verified from the intermediate complexity computations above that all these operations together take  $\tilde{O}((n^{k-s+1}d + nd^2) \log p)$  time. This proves the proposition.

Recall that what the honest prover has to do is compute  $q_1, q_{2,\alpha_1}, \dots, q_{k,\alpha_1, \dots, \alpha_{k-1}}$  for the  $\alpha_s$ 's specified by the verifier. By the above proposition, along with the preprocessing, the total time the prover takes is:

$$\tilde{O}(knd \log p + (n^k d + nd^2) \log p) = \tilde{O}(n^k d \log p)$$

The verifier's checks in steps (2) and (3) can each be done in  $\tilde{O}(n \log p)$  time using Lemma 1. Step (4), finally, can be done by using the above proposition with  $s = k$  in time  $\tilde{O}(nd^2 \log p)$ . Even along with the preprocessing, this leads to a total time of  $\tilde{O}(knd^2 \log p)$ .

## 5 Removing Interaction

In this section we show how to remove the interaction in Protocol 2 via the Fiat-Shamir heuristic and thus prove security in the Random Oracle model. In what follows, we take  $H$  to be a random oracle that outputs an element of  $\mathbb{F}_p$ , where  $p$  is as in Definition 4 and  $n$  will be clear from context.

Recall the definition of the polynomials  $q(i_1, \dots, i_k)$  and  $q_{s,\alpha_1, \dots, \alpha_{s-1}}(x)$  from Section 3. The non-interactive Proof of Useful Work for  $k$ -OV is described as Protocol 3.

**Theorem 4.** *For any  $k \geq 2$ , suppose  $k$ -OV takes  $n^{k-o(1)}$  time to decide for any  $d = \omega(\log n)$ . Then, Protocol 3 is a non-interactive  $(n^k, \delta)$ -Proof of Useful Work for  $k$ -OV in the Random Oracle model for any function  $\delta(n) > 1/n^{o(1)}$ .*

Efficiency, correctness, usefulness (given soundness), and hardness of Protocol 3 follow from the corresponding arguments about Protocol 2 in the proof of Theorem 2 in Section 3. The following lemma implies that the protocol is sound as well, completing the proof of Theorem 4.

**Lemma 3.** *For any  $k \geq 2$ , if Protocol 2 is sound as a Proof of Useful Work for  $\mathcal{FOV}^k$ , then Protocol 3 is sound as a non-interactive Proof of Useful Work for  $\mathcal{FOV}^k$  in the Random Oracle model.*

Note that in the above uPoW, we can cluster the  $s^{\text{th}}$  oracle calls for each instance together, so that only  $k$  oracle calls need to be made.

*Proof.* Let  $P$  be a cheating prover for the non-interactive uPoW that breaks soundness with non-negligible probability  $\varepsilon(n)$ . We will construct a prover,  $P'$ , that then also breaks the interactive uPoW soundness with non-negligible probability.

Notice that if  $P$  outputs a proof with a non-queried "challenge," then by the Schwartz-Zippel Lemma the probability the transcript is accepted is negligible. Thus, any cheating prover must query for all "challenges."

Suppose  $P$  makes at most  $m = \text{poly}(n)$  queries to the random oracle,  $H$ . We select  $k$  of the  $m$  query indices,  $i_1, \dots, i_k$ . Let the verifier  $V$  (recall that our protocol is public coin) output the  $k$  independently drawn uniform challenges  $\alpha_1, \dots, \alpha_k$  on randomness  $r$ . We then program a random oracle  $H_r$  to output  $\alpha_j$  on the  $i_j$  query. Now, we define  $P'$  to be the interactive prover that is consistent with the transcript of  $P$ . Notice that  $P'$  will fool  $V(r)$  with probability  $\varepsilon(n)$  (when given access to  $H_r$ ), conditioned on the fact that the  $i_j$ 's are chosen correctly (which happens with probability  $(1/m)^k$ ). So,  $P'$  breaks soundness with probability  $\varepsilon'(n) = \frac{\varepsilon(n)}{m^k}$ , which is still non-negligible given  $k$  is constant.

Moreover, the distribution of random oracles  $\{H\}$  is identical to the distribution of  $\{H_r\}_r$ . Therefore,  $P'$  cannot distinguish between the two cases. Thus, we can define  $P''$  that simply flips the coins for output themselves and breaks soundness with probability  $\varepsilon'(n)$ .  $\square$

### A Non-interactive uPoW for $k$ -OV

Gen( $\mathbf{x}$ ):

- Given an instance  $\mathbf{x} \in \{0, 1\}^{knd}$ , interpret  $\mathbf{x}$  as an element of  $\mathbb{F}_p^{knd}$  (where  $p = p(n)$  is as in Definition 4).
- Pick a random  $\mathbf{r} \in \mathbb{F}_p^{knd}$ .
- Output the set of vectors  $\mathbf{c}_x = \{\mathbf{y}_t = \mathbf{x} + t\mathbf{r} \mid t \in [kd + 1]\}$ .

Solve( $\mathbf{c}_x$ ):

Given input  $\mathbf{c}_x = \{\mathbf{y}_t\}$ , for each  $\mathbf{y} \in \mathbf{c}_x$  do the following:

- Compute  $z_y = f\text{OV}_{n,d,p}^k(\mathbf{y})$ .
- Compute coefficients of  $q_1^y$ . Let  $\tau_1^y = (z_y, q_1^y)$ .
- For  $s$  from 1 to  $k - 2$ :
  - Compute  $\alpha_s^y = H(\mathbf{c}_x, \tau_s^y)$ .
  - Compute coefficients of  $q_{s+1}^y = q_{s+1, \alpha_1, \dots, \alpha_s}$ , with respect to  $\mathbf{y}$ .
  - Set  $\tau_{s+1}^y = (\tau_s^y, q_{s+1}^y)$ .

Output  $T = \{\tau_{s+1}^y\}_{\mathbf{y} \in \mathbf{c}_x}$

Verify( $\mathbf{c}_x, T^*$ ):

Given  $T^* = \{\tau^{y^*} = (z^{y^*}, q_1^{y^*}, q_2^{y^*}, \dots, q_{k-1}^{y^*})\}$ , for each  $\tau^{y^*} \in T^*$  do the following:

- Check  $\sum_{i_1 \in [n]} q_1^{y^*}(i_1) = z^{y^*}$ . If check fails, reject.
- Compute  $\alpha_1^{y^*} = H(\mathbf{c}_x, z^{y^*}, q_1^{y^*})$ .
- For  $s$  from 1 up to  $k - 2$ :
  - Compute  $\alpha_s^{y^*} = H(\mathbf{c}_x, z^{y^*}, q_1^{y^*}, \dots, q_s^{y^*})$ .
  - Check that  $\sum_{i_{s+1} \in [n]} q_{s+1}^{y^*}(i_{s+1}) = q_s^{y^*}(\alpha_s^{y^*})$ . If check fails, reject.
- Compute  $\alpha_{k-1}^{y^*} = H(\mathbf{c}_x, z^{y^*}, q_1^{y^*}, \dots, q_{k-2}^{y^*})$ .
- Check that  $q_{k-1}^{y^*}(i_{s+1}) = \sum_{i_k \in [n]} q_k^y(i_k)$ . If check fails, reject.

If verifier has yet to reject, accept.

Recon( $\mathbf{c}_x, \{z_{y_t}\}$ ):

- Interpret  $z_{y_1}, \dots, z_{y_{kd+1}}$  as the evaluations of a univariate polynomial  $h(t)$  of degree  $kd$  at  $t = 1, \dots, kd + 1$ .
- Interpolate to find the coefficients of  $h$  and compute  $z = h(0)$ .
- If  $z \neq 0$ , output 1, else 0 as the answer to the  $k$ -OV instance.

Protocol 3: A Non-interactive uPoW for  $k$ -OV

## 6 A Direct Product Theorem for $\mathcal{FOV}$

A direct sum theorem for a problem roughly states that solving  $m$  independent instances of a problem takes  $m$  times as long as a single instance. The converse of this is attaining a non-trivial speed-up when given a *batch* of instances. In this section we prove a direct sum theorem for the problem of evaluating  $\mathcal{FOV}$  and thus its non-batchability.

Direct product theorems are stronger results that state that, beyond being a direct sum theorem, the probability of solving the  $m$  instances drops exponentially in  $m$  when only given the time it would typically take for a *single* instance. We prove both a direct sum theorem and a direct product theorem here.



Direct sum and, especially, direct product theorems are typically elusive in complexity theory and so our results, which we prove for generic problems with a certain set of properties, may be of independent interest to the study of hardness amplification. That our results show that batch-evaluating our multivariate low-degree polynomials is *hard* may be particularly surprising since batch-evaluation for *univariate* low-degree polynomials is known to be *easy* [Fid72, Hor72] and, further, [BK16, GR17, Wil16] show that batch-evaluating multivariate low-degree polynomials (including our own) is *easy to delegate*. For more rigorous definitions of direct sum and direct product theorems, see [She12].

## 6.1 A Direct Sum Theorem

In this section we prove the following weaker version of Theorem 3 on  $\mathcal{FOV}$ 's non-batchability (Theorem 3 is proven in Appendix A using an extension of the techniques employed here). The notion of non-batchability used below is defined in Definition 5 in Section 3.

**Theorem 5.** *For any  $k \geq 2$ , suppose  $k$ -OV takes  $n^{k-o(1)}$  time to decide for any  $d = \omega(\log n)$ . Then, for any constants  $c, \epsilon > 0$ ,  $\mathcal{FOV}^k$  is not  $(n^c, n^{k-\epsilon}, 7/8)$ -batchable on average over the uniform distribution over its inputs.*

Throughout this section,  $\mathcal{F}$ ,  $\mathcal{F}'$  and  $\mathcal{G}$  are families of functions  $\{f_n : \mathcal{X}_n \rightarrow \mathcal{Y}_n\}$ ,  $\{f'_n : \mathcal{X}'_n \rightarrow \mathcal{Y}'_n\}$  and  $\{g_n : \hat{\mathcal{X}}_n \rightarrow \hat{\mathcal{Y}}_n\}$ , and  $\mathcal{D} = \{D_n\}$  is a family of distributions where  $D_n$  is over  $\hat{\mathcal{X}}_n$ .

Theorem 5 is the result of two properties possessed by  $\mathcal{FOV}^k$ . We define these properties below, prove a more general lemma about functions that have these properties, and use it to prove this theorem.

**Definition 6.**  $\mathcal{F}$  is said to be  $(s, \ell)$ -downward reducible to  $\mathcal{F}'$  in time  $t$  if there is a pair of algorithms (Split, Merge) satisfying:

- For all large enough  $n$ ,  $s(n) < n$ .
- Split on input an  $x \in \mathcal{X}_n$  outputs  $\ell(n)$  instances from  $\mathcal{X}'_{s(n)}$ .

$$\text{Split}(x) = (x_1, \dots, x_{\ell(n)})$$

- Given the value of  $\mathcal{F}'$  at these  $\ell(n)$  instances, Merge can reconstruct the value of  $\mathcal{F}$  at  $x$ .

$$\text{Merge}(x, f'_{s(n)}(x_1), \dots, f'_{s(n)}(x_{\ell(n)})) = f_n(x)$$

- Split and Merge together run in time at most  $t(n)$ .

If  $\mathcal{F}'$  is the same as  $\mathcal{F}$ , then  $\mathcal{F}$  is said to be downward self-reducible.

**Definition 7.**  $\mathcal{F}$  is said to be  $\ell$ -robustly reducible to  $\mathcal{G}$  in time  $t$  if there is a pair of algorithms (Split, Merge) satisfying:

- Split on input an  $x \in \mathcal{X}_n$  (and randomness  $r$ ) outputs  $\ell(n)$  instances from  $\hat{\mathcal{X}}_n$ .

$$\text{Split}(x; r) = (x_1, \dots, x_{\ell(n)})$$

- For such a tuple  $(x_i)_{i \in [\ell(n)]}$  and any function  $g^*$  such that  $g^*(x_i) = g_n(x_i)$  for at least  $2/3$  of the  $x_i$ 's, Merge can reconstruct the function value at  $x$  as:

$$\text{Merge}(x, r, g^*(x_1), \dots, g^*(x_{\ell(n)})) = f_n(x)$$

- Split and Merge together run in time at most  $t(n)$ .
- Each  $x_i$  is distributed according to  $D_n$ , and the  $x_i$ 's are pairwise independent.

The above is a more stringent notion than the related non-adaptive random self-reducibility as defined in [FF93]. We remark that to prove what we need, it can be shown that it would have been sufficient if the reconstruction above had only worked for *most*  $r$ 's.

**Lemma 4.** *Suppose  $\mathcal{F}$ ,  $\mathcal{F}'$  and  $\mathcal{G}$  have the following properties:*

- $\mathcal{F}$  is  $(s_d, \ell_d)$ -downward reducible to  $\mathcal{F}'$  in time  $t_d$ .
- $\mathcal{F}'$  is  $\ell_r$ -robustly reducible to  $\mathcal{G}$  over  $\mathcal{D}$  in time  $t_r$ .
- $\mathcal{G}$  is  $(\ell_a, t_a, 7/8)$ -batchable on average over  $\mathcal{D}$ , and  $\ell_a(s_d(n)) = \ell_d(n)$ .

Then  $\mathcal{F}$  can be computed in the worst-case in time:

$$t_d(n) + \ell_d(n)t_r(s_d(n)) + \ell_r(s_d(n))\ell_d(n)t_a(s_d(n))$$

We note, that the condition  $\ell_a(s_d(n)) = \ell_d(n)$  above can be relaxed to  $\ell_a(s_d(n)) \leq \ell_d(n)$  at the expense of a factor of 2 in the worst-case running time obtained for  $\mathcal{F}$ . We now show how to prove Theorem 5 using Lemma 4, and then prove the lemma itself.

**Proof of Theorem 5.** Fix any  $k \geq 2$ . Suppose, towards a contradiction, that for some  $c, \epsilon > 0$ ,  $\mathcal{FOV}^k$  is  $(n^c, n^{k-\epsilon}, 7/8)$ -batchable on average over the uniform distribution. In our arguments we will refer to the following function families:

- $\mathcal{F}$  is  $k$ -OV with vectors of dimension  $d = \left(\frac{k}{k+c}\right)^2 \log^2 n$ .
- $\mathcal{F}'$  is  $k$ -OV with vectors of dimension  $\log^2 n$ .
- $\mathcal{G}$  is  $\mathcal{FOV}^k$  (over  $\mathbb{F}_p^{knd}$  for some  $p$  that definitely satisfies  $p > n$ ).

Let  $m = n^{k/(k+c)}$ . Note the following two properties :

- $\frac{n}{m^{c/k}} = m$
- $d = \left(\frac{k}{k+c}\right)^2 \log^2 n = \log^2 m$

We now establish the following relationships among the above function families.

**Proposition 2.**  $\mathcal{F}$  is  $(m, m^c)$ -downward reducible to  $\mathcal{F}'$  in time  $\tilde{O}(m^{c+1})$ .

$\text{Split}_d$ , when given an instance  $(U_1, \dots, U_k) \in \{0, 1\}^{k(n \times d)}$ , first divides each  $U_i$  into  $m^{c/k}$  partitions  $U_{i1}, \dots, U_{im^{c/k}} \in \{0, 1\}^{m \times d}$ . It then outputs the set of tuples  $\{(U_{1j_1}, \dots, U_{kj_k}) \mid j_i \in [m^{c/k}]\}$ . Each  $U_{ij}$  is in  $\{0, 1\}^{m \times d}$  and, as noted earlier,  $d = \log^2 m$ . So each tuple in the set is indeed an instance of  $\mathcal{F}'$  of size  $m$ . Further, there are  $(m^{c/k})^c = m^c$  of these.

Note that the original instance has a set of  $k$ -orthogonal vectors if and only if at least one of the  $m^c$  smaller instances produced does. So  $\text{Merge}_d$  simply computes the disjunction of the  $\mathcal{F}'$  outputs to these instances.

Both of these can be done in time  $O(m^c \cdot k \cdot md + m^c) = \tilde{O}(m^{c+1})$ .

**Proposition 3.**  $\mathcal{F}'$  is  $12kd$ -robustly reducible to  $\mathcal{G}$  over the uniform distribution in time  $\tilde{O}(m)$ .

Notice that for any  $U_1, \dots, U_k \in \{0, 1\}^{m \times d}$ , we have that  $k\text{-OV}(U_1, \dots, U_k) = f\text{OV}_m^k(U_1, \dots, U_k)$ . So it is sufficient to show such a robust reduction from  $\mathcal{G}$  to itself. We do this now.

Given input  $\mathbf{x} \in \mathbb{F}_p^{knd}$ ,  $\text{Split}_r$  picks two uniformly random  $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{F}_p^{knd}$  and outputs the set of vectors  $\{\mathbf{x} + t\mathbf{x}_1 + t^2\mathbf{x}_2 \mid t \in \{1, \dots, 12kd\}\}$ . Recall that our choice of  $p$  is much larger than  $12kd$  and hence this is possible. The distribution of each of these vectors is uniform over  $\mathbb{F}_p^{knd}$ , and they are also pairwise independent as they are points on a random quadratic curve through  $\mathbf{x}$ .

Define the univariate polynomial  $g_{\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2}(t) = f\text{OV}_m^k(\mathbf{x} + t\mathbf{x}_1 + t^2\mathbf{x}_2)$ . Note that its degree is at most  $2kd$ . When  $\text{Merge}_r$  is given  $(y_1, \dots, y_{12kd})$  that are purported to be the evaluations of  $f\text{OV}_m^k$  on the points produced by  $\text{Split}$ , these can be seen as purported evaluations of  $g_{\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2}$  on  $\{1, \dots, 12kd\}$ . This can, in turn, be treated as a corrupt codeword of a Reed-Solomon code, which under these parameters has distance  $10kd$ .

The Berlekamp-Welch algorithm can be used to decode any codeword that has at most  $5kd$  corruptions, and if at least  $2/3$  of the evaluations are correct, then at most  $4kd$  evaluations are wrong. Hence  $\text{Merge}_r$  uses the Berlekamp-Welch algorithm to recover  $g_{\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2}$ , which can be evaluated at 0 to obtain  $f\text{OV}_n^k(\mathbf{x})$ .

Thus,  $\text{Split}_r$  takes  $\tilde{O}(12kd \cdot kmd) = \tilde{O}(m)$  time to compute all the vectors it outputs.  $\text{Merge}_r$  takes  $\tilde{O}((12kd)^3)$  time to run Berlekamp-Welch, and  $\tilde{O}(12kd)$  time to evaluate the resulting polynomial at 0. So in all both algorithms take  $\tilde{O}(m)$  time.

By our assumption at the beginning,  $\mathcal{G}$  is  $(n^c, n^{k-\epsilon}, 7/8)$ -batchable on average over the uniform distribution. Together with the above propositions, this satisfies all the requirements in the hypothesis of Lemma 4, which now tells us that  $\mathcal{F}$  can be computed in the worst-case in time:

$$\begin{aligned} \tilde{O}(m^{c+1} + m^c \cdot m + 12kd \cdot m^c \cdot m^{k-\epsilon}) &= \tilde{O}(m^{c+1} + m^{c+k-\epsilon}) \\ &= \tilde{O}(n^{k(c+1)/(k+c)} + n^{k(k+c-\epsilon)/(k+c)}) \\ &= \tilde{O}(n^{k-\epsilon'}) \end{aligned}$$

for some  $\epsilon' > 0$ . But this is what the hypothesis of the theorem says is not possible. So  $\mathcal{FOV}^k$  cannot be  $(n^c, n^{k-\epsilon}, 7/8)$ -batchable on average, and this argument applies for any  $c, \epsilon > 0$ .  $\square$

**Proof of Lemma 4.** Given the hypothesised downward reduction  $(\text{Split}_d, \text{Merge}_d)$ , robust reduction  $(\text{Split}_r, \text{Merge}_r)$  and batch-evaluation algorithm  $\text{Batch}$  for  $\mathcal{F}$ ,  $f_n$  can be computed as follows (for large enough  $n$ ) on an input  $x \in \mathcal{X}_n$ :

- Run  $\text{Split}_d(x)$  to get  $x_1, \dots, x_{\ell_d(n)} \in \mathcal{X}'_{s_d(n)}$ .
- For each  $i \in [\ell_d(n)]$ , run  $\text{Split}_r(x_i; r_i)$  to get  $x_{i1}, \dots, x_{i\ell_r(s_d(n))} \in \hat{\mathcal{X}}_{s_d(n)}$ .
- For each  $j \in [\ell_r(s_d(n))]$ , run  $\text{Batch}(x_{1j}, \dots, x_{\ell_d(n)j})$  to get the outputs  $y_{1j}, \dots, y_{\ell_d(n)j} \in \hat{\mathcal{Y}}_{s_d(n)}$ .
- For each  $i \in [\ell_d(n)]$ , run  $\text{Merge}_r(x_i, r_i, y_{i1}, \dots, y_{i\ell_r(s_d(n))})$  to get  $y_i \in \mathcal{Y}'_{s_d(n)}$ .
- Run  $\text{Merge}_d(x, y_1, \dots, y_{\ell_d(n)})$  to get  $y \in \mathcal{Y}_n$ , and output  $y$  as the alleged  $f_n(x)$ .

We will prove that with high probability, after the calls to  $\text{Batch}$ , enough of the  $y_{ij}$ 's produced will be equal to the respective  $g_{s_d(n)}(x_{ij})$ 's to be able to correctly recover all the  $f'_{s_d(n)}(x_i)$ 's and hence  $f_n(x)$ .

For each  $j \in [\ell_r(s_d(n))]$ , define  $I_j$  to be the indicator variable that is 1 if  $\text{Batch}(x_{1j}, \dots, x_{\ell_d(n)j})$  is correct and 0 otherwise. Note that by the properties of the robust reduction of  $\mathcal{F}'$  to  $\mathcal{G}$ , for a fixed  $j$  each of the  $x_{ij}$ 's is independently distributed according to  $D_{s_d(n)}$  and further, for any two distinct  $j, j'$ , the tuples  $(x_{ij})$  and  $(x_{ij'})$  are independent.

Let  $I = \sum_j I_j$  and  $m = \ell_r(s_d(n))$ . By the aforementioned properties and the correctness of  $\text{Batch}$ , we have the following:

$$\begin{aligned} \mathbb{E}[I] &\geq \frac{7}{8}m \\ \text{Var}[I] &\leq \frac{7}{64}m \end{aligned}$$

Note that as long as  $\text{Batch}$  is correct on more than a  $2/3$  fraction of the  $j$ 's,  $\text{Merge}_r$  will get all of the  $y_i$ 's correct, and hence  $\text{Merge}_d$  will correctly compute  $f_n(x)$ . The probability that this does not happen is bounded using Chebyshev's inequality as:

$$\begin{aligned} \Pr \left[ I \leq \frac{2}{3}m \right] &\leq \Pr \left[ |I - \mathbb{E}[I]| \geq \left( \frac{7}{8} - \frac{2}{3} \right) m \right] \\ &\leq \frac{\text{Var}[I]}{(5m/24)^2} \\ &\leq \frac{63}{25 \cdot m} < \frac{3}{m} \end{aligned}$$

As long as  $m > 9$ , this probability of failure is less than  $1/3$ , and hence  $f_n(x)$  is computed correctly in the worst-case with probability at least  $2/3$ . If it is the case that  $\ell_r(s_d(n)) = m$  happens to be less than 9, then instead of using  $\text{Merge}_r$  directly in the above algorithm, we would use  $\text{Merge}'_r$  that runs  $\text{Merge}_r$  several times so as to get more than 9 samples in total and takes the majority answer from all these runs.

The time taken is  $t_d(n)$  for the downward reduction,  $t_r(s_d(n))$  for each of the  $\ell_d(n)$  robust reductions on instances of size  $s_d(n)$ , and  $\ell_d(n)t_a(s_d(n))$  for each of the  $\ell_r(s_d(n))$  calls to  $\text{Batch}$  on sets of  $\ell_d(n) = \ell_a(s_d(n))$  instances, summing up to the total time stated in the lemma.  $\square$

## 6.2 A Direct Product Theorem

**Definition 8.**  $\text{cOV}$  denotes the problem of outputting the coefficients of the polynomial  $R_{U,V}(\cdot)$ , given  $(U, V)$ .

Futhermore,  $\text{cOV}^m$  denotes the  $m$ -wise cartesian product of the  $\text{cOV}$  relation.

From previous work, we have the following lemmas about  $\text{cOV}$ .

**Lemma 5.** *There exists an algorithm  $V$  that runs in time  $\tilde{O}(n)$  such that*

$$\Pr[V(U, V, R^*) = 1 | R^* \neq R_{U,V}] < 2^{-\text{polylog}(n)}.$$

Recall the following definition:

**Definition 9.** *A problem  $F = \{f_n\}$  is  $(t(n), \delta(n))$ -(io)-hard on average over  $D = \{D_n\}$  if  $A$  is an algorithm that runs in time  $t(n)$  on  $\text{Supp}(D_n)$ , then for large enough (infinitely many)  $n$ :*

$$\Pr_{x \leftarrow D_n} [A(x) = f_n(x)] < \delta(n).$$

**Lemma 6.** *Assuming the orthogonal vectors conjecture,  $\text{cOV}$  is  $(n^{2-\varepsilon}, 2/3)$ -hard on average over the uniform distribution, for all constant  $\varepsilon > 0$ .*

Armed with the above definitions, we observe the following parametrized direct product Lemma. Coarse variants of the Lemma are common knowledge, implying for example that weak one-way functions imply strong ones.

**Lemma 7.** *Suppose a relation  $R$  is  $(T, \delta) = (T(n), \delta(n))$ -hard on  $D$ , samplable in time  $T_D = T_D(n)$  and can be verified in time  $T_V = T_V(n)$  by an algorithm  $V$  with soundness error probability  $\gamma(n)$  (and perfect completeness).*

*If for  $m(n) = m$  it is the case that  $\frac{16\gamma}{1-\delta} < \frac{(\frac{1+\delta}{2})^m}{m^2} \leq 1$ , then  $R^m$  is  $(T'(n), \varepsilon(n))$ -hard on  $D^m$ , where  $\varepsilon \geq 2(\frac{1+\delta}{2})^m$  and  $T \leq \frac{\varepsilon T}{-4\ln(\frac{1-\delta}{4})} - \log m - mT_D - T_V$ .*

*(Note that instances in  $R^m$  are of size  $mn$ .)*

Before proving the Lemma, notice that in conjunction with Lemmas 5 and 6 we immediately arrive that the following Theorem which allows us to trade time for hardness:

**Theorem 6.** *If  $m^3(n) = O(n^{1-\alpha})$ ,  $(5/6)^{m(n)} \leq \varepsilon/2$ , and  $\frac{m^2 t(n)}{\varepsilon(n)} = O(n^{2-\beta})$  for  $\alpha, \beta > 0$ ; then  $\text{cOV}^m$  is  $(t(n), \varepsilon(n))$ -hard on average over the uniform distribution.*

*In particular for  $m = \log^2 n$ ,  $\text{cOV}^m$  is  $(n^{2-\alpha}, n^{-\beta})$ -hard on average over the uniform distribution for any  $\alpha, \beta > 0$  such that  $\beta - \alpha < 0$ .*

Put differently, any subquadratic algorithm for  $\text{cOV}^{\log^2 n}$  has success probability negligible in  $\log n$ .

*Proof.* Let  $\varepsilon(n) \geq 2(\frac{1+\delta(n)}{2})^{m(n)}$  and  $\frac{16\gamma}{1-\delta} < \frac{(\frac{1+\delta}{2})^m}{m^2} \leq 1$ . Suppose for the sake of contradiction that  $A$  runs in time  $T_A(n) \leq \frac{\varepsilon(n)T(n)}{-4\ln(\frac{1-\delta(n)}{4})} - \log m(n) - mT_D(n) - T_V(n)$ , and

$$\Pr_{\mathbf{X} \leftarrow D_n^m} [(X, A(X)) \in R^m] \geq \varepsilon.$$

Consider  $A'$  that does the following on input  $X$ :

1. Draw random  $i \in [m]$ .
2. Call  $A$  on  $(X_1, \dots, X_{i-1}, X, X_{i+1}, \dots, X_m)$  to get  $(Y_1, \dots, Y_m)$ , where the  $m-1$   $X_j$ s are drawn independently from  $D_n$ .
3. Output  $Y_i$ .

Let  $S \subset \text{Supp}(D_n)$  such that

$$S = \{X \in \text{Supp}(D_n) \mid \Pr[(X, A'(X)) \in R] \geq \zeta(n)\},$$

where  $\zeta = \frac{\beta^m}{4m^2}$  and  $\beta = \frac{1+\delta}{2}$ . Notice that  $0 \leq \frac{4\gamma}{1-\delta} < \zeta < \frac{\beta^m}{2m^2} \leq 1/2$ .

Now we want to show that  $\Pr_{X \leftarrow D_n}[X \in S] > \beta$ . Suppose this is not the case:  $\Pr_{D_n}[X \in S] \leq \beta$ . Then,

$$\begin{aligned}
\Pr_D[(\mathbf{X}, A(\mathbf{X})) \in R] &\leq \Pr[(\mathbf{X}, A(\mathbf{X})) \in R^m \wedge \exists i(\mathbf{X}_i \notin S)] + \Pr[(\mathbf{X}, A(\mathbf{X})) \in R^m \wedge \forall i(\mathbf{X}_i \in S)] \\
&\leq \sum_{i=1}^m \Pr[(\mathbf{X}, A(\mathbf{X})) \in R^m | \mathbf{X}_i \notin S] + \Pr[\forall i(\mathbf{X}_i \in S)] \\
&\leq \sum_{i=1}^m m \Pr[(\mathbf{X}_i, A'(\mathbf{X}_i)) \in R | \mathbf{X}_i \notin S] + \prod_{i=1}^m \Pr[\mathbf{X}_i \in S] \\
&\leq m^2 \zeta + \beta^m,
\end{aligned}$$

where the second to last inequality follows from the fact that  $A'$  puts  $\mathbf{X}_i$  in the  $i$ th location with probability  $1/m$ .

By our assumption on and definition of parameters,

$$\begin{aligned}
m^2 \zeta + \beta^m &< m^2 \frac{\varepsilon}{2m^2} + \frac{\varepsilon}{2} \\
&= \varepsilon.
\end{aligned}$$

This contradicts the assumption on  $A$ . Thus,  $\Pr_{X \leftarrow D_n}[X \in S] > \beta$ .

Using this fact we show how to use  $A$  (and  $A'$ ) to violate the hardness of  $R$ . Consider  $A''$  that does the following on input  $X$ :

- For  $i = 1$  to  $\ell = \frac{-\ln(\frac{1-\delta}{4})}{\zeta}$ :
  - Call  $A'(X)$ .
  - If  $V(X, A'(X)) = 1$ , return  $A'(X)$  and halt. Otherwise, continue.
- If all trials fail, output  $\perp$ .

Now we analyze the success probability of  $A''$ . Notice that  $A''$  fails if  $V$  makes a mistake or  $A'$  never succeeds.

$$\begin{aligned}
\Pr_D[(X, A''(X)) \notin R] &\leq \Pr[(X, A''(X)) \notin R | X \in S] + \Pr[X \notin S] \\
&\leq \Pr[\forall i \in [\ell]((X, A'(X)) \notin R | X \in S)] + \Pr[\exists i \in [\ell](V \text{ fails } i\text{th trial}) | X \in S] \\
&\quad + \Pr[X \notin S] \\
&\leq (1 - \zeta)^\ell + \ell\gamma + 1 - \beta.
\end{aligned}$$

Again by our choice of parameters,

$$\begin{aligned}
(1 - \zeta)^\ell + \ell\gamma + (1 - \beta) &\leq e^{\ln(\frac{1-\delta}{4})} + \frac{-\ln(\frac{1-\delta}{4})}{\zeta} \gamma + \frac{1-\delta}{2} \\
&< \frac{1-\delta}{4} + \frac{1-\delta}{4} + \frac{1-\delta}{2} \\
&= 1 - \delta.
\end{aligned}$$

Now, we observe that the reduction is efficient enough,  $T_{A''} < T$ , to contradict the hardness of  $R$ :

$$T_{A''} \leq \ell(T_{A'} + T_V) = \ell(\log m + mT_D + T_A + T_V) \leq T,$$

by the assumption on  $A$ . □

## 7 A Blockchain Scheme

Bitcoin uses PoWs on a massive scale. In this light, having Proofs of Useful Work is much desired from the perspective of decreasing environmental costs and also from the perspective of having an enormous, incentivized computing community already in existence that can be fed problems to solve.

Unfortunately, a generic uPoW does not immediately fit into the framework in which Bitcoin uses them. Namely, PoWs or uPoWs used in Bitcoin should be rendered invalid if Bitcoin transactions are altered. We give a brief overview of how PoWs are currently used and then describe how our uPoWs can be incorporated to a Blockchain-like mechanism.

Bitcoin’s main innovation is a system implementing a public ledger on which transactions are recorded: the blockchain. The blockchain is a discretized timeline of transactions in which each discrete group of transactions is called a block and the blocks are then chained together by each block containing a hash of its previous block (enforcing its temporal structure). The role of PoWs in this framework is that for a malicious user to ‘rewrite history’ or change a block, they must produce a PoW that is *sensitive to changes in the block*. Thus, creating dishonest blocks requires work and, since the blockchain is always decided by majority, PoWs ensure that any adversarial community cannot reliably succeed without having the majority of computing power in the entire system.

A main point to notice here is that the PoW must be sensitive to changes in the block that the proof is made for. To account for this we show how our uPoWs can be made to be sensitive to such changes and give a scheme for which a blockchain based system such as Bitcoin can use our uPoWs to operate while dually serving as a source of computation for delegators.

As seen in Figure 1, there is a public board that delegators post problems to. We currently write each problem in the form  $(f, \mathbf{x})$  where  $f$  is either an arithmetic circuit or simply a label of a commonly requested problem, such as OV, that the workers are familiar with and  $\mathbf{x}$  is the instance being delegated to find  $f(\mathbf{x})$  for (we assume for now that our practical problems are already in the form of evaluating a low-degree polynomial). When a worker needs to perform a PoW, they grab a problem from the board according to any type of priority scheduling and keep it to mine their next block.

Notice that the worker currently has the actual delegated instance  $x$ . Using a Random Oracle  $H$ , the worker will generate the challenge  $\mathbf{c}_x$  themselves as usual except substituting  $r = H(\text{current block})$  for the randomness usually used to generate the challenge:  $\mathbf{c}_x = \{\mathbf{x} + \mathbf{r}t \mid t \in [D + 1]\}$ , where  $\mathbf{r} = H(\text{current block})$ . For a truly Random Oracle  $H$ ,  $\mathbf{r}$  will be random and this becomes a standard challenge for a uPoW. Further, any altercations to the current block being mined will produce an entirely new random challenge and so a new uPoW will have to be made for changed blocks. Thus we attain uPoWs that are sensitive to changes in the block.

Note that this also means that if a party fails to mine a block before another party successfully does, they can still use the same problem  $(f, \mathbf{x})$  on a new block they attempt to mine as the challenge will be ‘re-randomized’ with respect to  $H$  and so parties will hold on to their problems until they complete a uPoW with it.

Substituting a standard cryptographic hash function such as SHA-256 for  $H$ , this falls very much in line with what Bitcoin currently does. Current Bitcoin PoWs are essentially to find a nonce so that, when hashed along with the current block, the hash value has a prescribed number of leading 0’s. Thus these current (useless) PoWs also rely on SHA-256 behaving as a Random Oracle (and this is also used to chain blocks together by hashing the previous block). We follow this approach in using  $H$  to generate our randomness for generation.

A block then, as seen in Figure 1, is composed of

### Problem Board

$(f_3, \mathbf{x}_3), (f_4, \mathbf{x}_4), (f_5, \mathbf{x}_5), \dots$

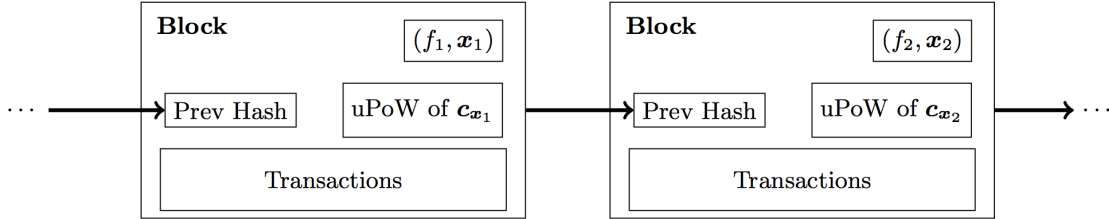


Figure 1: We present a framework for Blockchain to use uPoWs. Delegators post to a public Problem Board from which workers grab problems to mine a block with by producing a Proof of Useful Work that they attach to the block.

1. The hash of the previous block,
2. The transactions that the block is recording,
3. The problem  $(f, \mathbf{x})$  the block claims to have a proof for, and
4. The uPoW for  $f$  on challenge  $\mathbf{c}_{\mathbf{x}} = \{\mathbf{x} + H(\text{current block})t \mid t \in [D + 1]\}$ .

To verify a block as a valid addition to the blockchain, a user simply checks that the hash of the previous block is correct, that the problem  $(f, \mathbf{x})$  hadn't been previously solved (this is to ensure that each PoW is useful in that no two people redundantly have the same problem and that miners constantly pull new problems from the problem board), and that the PoW is valid by deterministically computing the  $\mathbf{c}_{\mathbf{x}}$  challenge with  $H$  and then checking the uPoW for it. Further, the delegator upon seeing the uPoW can quickly reconstruct  $f(\mathbf{x})$  by uPoW's usefulness property.

We then have uPoWs for Bitcoin. As is common now, workers can still create 'mining pools' and parallelize the work amongst their pool and, in fact, our framework naturally enhances this joint effort to be robust to errors and Byzantine failures, even identifying non-cooperative members of the mining pool (this uses the fact that the solution sketch to our uPoWs are recovered by means of decoding a Reed-Muller code for which there is good error-correction for [BK16]). Further, while the total *combined* work done by a mining pool is still guaranteed to be of a certain amount by uPoW's Hardness condition, the time a delegator has to wait may be significantly shorter as they parallelize their work amongst themselves.

## Acknowledgements

We are grateful to Oded Goldreich and Guy Rothblum for clarifying definitions of direct sum and direct product theorems, and for the suggestion of using interaction to increase the gap between solution and verification in our uPoWs. We would also like to thank Tal Moran and Vinod Vaikunathan for several useful discussions.



The bulk of this work was performed while the authors were at IDC Herzliya’s FACT center and supported by NSF-BSF Cyber Security and Privacy grant #2014/632, ISF grant #1255/12, and by the ERC under the EU’s Seventh Framework Programme (FP/2007-2013) ERC Grant Agreement #07952. Marshall Ball is supported in part by the Defense Advanced Research Project Agency (DARPA) and Army Research Office (ARO) under Contract #W911NF-15-C-0236, and NSF grants #CNS-1445424 and #CCF-1423306. Manuel Sabin is also supported by the National Science Foundation Graduate Research Fellowship under Grant #DGE-1106400. Prashant Nalini Vasudevan is also supported by the IBM Thomas J. Watson Research Center (Agreement #4915012803), by NSF Grants CNS-1350619 and CNS-1414119, and by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under contracts W911NF-15-C-0226 and W911NF-15-C-0236..

## References

- [ABFG14] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. In *International Conference on Security and Cryptography for Networks*, pages 538–557. Springer, 2014.
- [AL13] Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-sum conjecture. In *International Colloquium on Automata, Languages, and Programming*, pages 1–12. Springer, 2013.
- [And13] Nate Anderson. Mining bitcoins takes power, but is it an “environmental disaster?”. <http://tinyurl.com/cdh95at>, April 2013.
- [BCG15] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*, 2015:1015, 2015.
- [BK16] Andreas Björklund and Petteri Kaski. How proofs are prepared at camelot. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 391–400. ACM, 2016.
- [BRSV17] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Average-case fine-grained hardness. to appear in Symposium on Theory of Computing (STOC’17), 2017.
- [CPS99] Jin-yi Cai, Aduri Pavan, and D. Sivakumar. On the hardness of permanent. In Christoph Meinel and Sophie Tison, editors, *STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science, Trier, Germany, March 4-6, 1999, Proceedings*, volume 1563 of *Lecture Notes in Computer Science*, pages 90–99. Springer, 1999.
- [DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *Annual Cryptology Conference*, pages 585–605. Springer, 2015.
- [DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO ’92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer, 1992.

- [FF93] Joan Feigenbaum and Lance Fortnow. Random-self-reducibility of complete sets. *SIAM J. Comput.*, 22(5):994–1005, 1993.
- [Fid72] Charles M. Fiduccia. Polynomial evaluation via the division algorithm: The fast fourier transform revisited. In Patrick C. Fischer, H. Paul Zeiger, Jeffrey D. Ullman, and Arnold L. Rosenberg, editors, *Proceedings of the 4th Annual ACM Symposium on Theory of Computing, May 1-3, 1972, Denver, Colorado, USA*, pages 88–93. ACM, 1972.
- [GI16] Jiawei Gao and Russell Impagliazzo. Orthogonal vectors is hard for first-order properties on sparse graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:53, 2016.
- [GR17] Oded Goldreich and Guy Rothblum. Simple doubly-efficient interactive proof systems for locally-characterizable sets. Electronic Colloquium on Computational Complexity Report TR17-018, February 2017.
- [Hor72] Ellis Horowitz. A fast method for interpolation using preconditioning. *Inf. Process. Lett.*, 1(4):157–163, 1972.
- [JJ99] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In Bart Preneel, editor, *Secure Information Networks: Communications and Multimedia Security, IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99), September 20-21, 1999, Leuven, Belgium*, volume 152 of *IFIP Conference Proceedings*, pages 258–272. Kluwer, 1999.
- [Kin13] Sunny King. Primecoin: Cryptocurrency with prime number proof-of-work. *July 7th*, 2013.
- [KK14] Nikolaos P Karvelas and Aggelos Kiayias. Efficient proofs of secure erasure. In *International Conference on Security and Cryptography for Networks*, pages 520–537. Springer, 2014.
- [MJS<sup>+</sup>14] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 475–490. IEEE, 2014.
- [Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
- [RR00] Ron M. Roth and Gitit Ruckenstein. Efficient decoding of reed-solomon codes beyond half the minimum distance. *IEEE Trans. Information Theory*, 46(1):246–257, 2000.
- [She12] Alexander A Sherstov. Strong direct product theorems for quantum communication and query complexity. *SIAM Journal on Computing*, 41(5):1122–1165, 2012.
- [Wil05] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.
- [Wil15] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis. In *Proc. International Symposium on Parameterized and Exact Computation*, pages 16–28, 2015.
- [Wil16] Richard Ryan Williams. Strong ETH breaks with merlin and arthur: Short non-interactive proofs of batch evaluation. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 2:1–2:17, 2016.

[WW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 645–654. IEEE, 2010.

## A A Stronger Direct Sum Theorem for $\mathcal{FOV}$

In this section, we prove a stronger direct sum theorem (and, thus, non-batchability evaluation) for  $\mathcal{FOV}^k$ . That is, we prove Theorem 3.

In particular, it is sufficient to define a notion of batchability for parametrized families of functions with a monotonicity constraint. In our case, monotonicity will essentially say “adding more vectors of the same dimension and field size does not make the problem easier.” This is a natural property of most algorithms. Namely, it is the case if for any fixed  $d, p$ ,  $\mathcal{FOV}_{n,d,p}^k$  is  $(n, t, \delta)$  – batchable.

Instead, we simply generalize batchability in a parametrized fashion for  $\mathcal{FOV}_{n,d,p}^k$ .

**Definition 10.** A parametrized class,  $\mathcal{F}_\rho$ , is not  $(\ell, t, \delta)$ -batchable on average over  $\mathcal{D}_\rho$ , a parametrized family of distributions if, for any fixed parameter  $\rho$  and algorithm  $\text{Batch}_\rho$  that runs in time  $\ell(\rho)t(\rho)$  when it is given as input  $\ell(\rho)$  independent samples from  $D_\rho$ ,

$$\Pr_{x_i \leftarrow D_\rho} [\text{Batch}(x_1, \dots, x_{\ell(\rho)}) = (f_\rho(x_1), \dots, f_\rho(x_{\ell(\rho)}))] < \delta(\rho).$$

*Remark A.1.* We use a more generic parameterization of  $\mathcal{F}_\rho$  by  $\rho$  rather than just  $n$  since we need the batch evaluation procedure to have the property that it should still run quickly as  $n$  shrinks, as we use downward self-reducibility of  $\mathcal{FOV}_{n,d,p}^k$ , even when  $p$  and  $d$  remain the same.

We now show how a generalization of the list decoding reduction from [BRSV17] yields strong batch evaluation bounds. Before we begin, we will present a few Lemmas from the literature to make certain bounds explicit.

First, we present an inclusion-exclusion bound from [CPS99] on the polynomials consistent with a fraction of  $m$  input-output pairs,  $(x_1, y_1), \dots, (x_m, y_m)$ . We include a laconic proof here with the given notation for convenience.

**Lemma 8** ([CPS99]). Let  $q$  be a polynomial over  $\mathbb{F}_p$ , and define  $\text{Graph}(q) := \{(i, q(i)) \mid i \in [p]\}$ . Let  $c > 2$ ,  $\delta/2 \in (0, 1)$ , and  $m \leq p$  such that  $m > \frac{c^2(d-1)}{\delta^2(c-2)}$  for some  $d$ . Finally, let  $I \subseteq [p]$  such that  $|I| = m$ . Then, for any set  $S = \{(i, y_i) \mid i \in I\}$ , there are less than  $\lceil c/\delta \rceil$  polynomials  $q$  of degree at most  $d$  that satisfy  $|\text{Graph}(q) \cap S| \geq m\delta/2$ .

**Corollary 2.** Let  $S$  be as in Lemma 8 with  $I = \{m+1, \dots, p\}$ , for any  $m < p$ . Then for  $m > 9d/\delta^2$ , there are at most  $3/\delta$  polynomials,  $q$ , of degree at most  $d$  such that  $|\text{Graph}(q) \cap S| \geq m\delta/2$ .

*Proof.* Reproduced from [CPS99] for convenience; see original for exposition.

Suppose there exist at least  $\lceil c/\delta \rceil$  such polynomials. Consider a subset of exactly  $N = \lceil c/\delta \rceil$

such polynomials,  $\mathcal{F}$ . Define  $S_f := \{(i, f(i)) \in \text{Graph}(f) \cap S\}$ , for each  $f \in \mathcal{F}$ .

$$\begin{aligned}
m &\geq \left| \bigcup_{f \in \mathcal{F}} S_f \right| \geq \sum_{f \in \mathcal{F}} |S_f| - \sum_{f, f' \in \mathcal{F}: f \neq f'} |S_f \cap S_{f'}| \\
&\geq N \frac{m\delta}{2} - \frac{N(N-1)(d-1)}{2} \\
&> \frac{N}{2} \left( m\delta - \frac{c(d-1)}{\delta} \right) \\
&\geq \frac{c}{2\delta} \left( m\delta - \frac{c(d-1)}{\delta} \right) \\
&= \frac{cm}{2} - \frac{c^2(d-1)}{2\delta^2} \\
&= m + \frac{1}{2} \left( (c-2)m - \frac{c^2(d-1)}{\delta^2} \right) > m.
\end{aligned}$$

□

Now, we give a theorem based on an efficient list-decoding algorithm, related to Sudan's, from Roth and Ruckenstein. [RR00]

**Lemma 9** ([RR00]). *List decoding for  $[n, k]$  Reed-Solomon (RS) codes over  $\mathbb{F}_p$  given a code word with almost  $n - \sqrt{2kn}$  errors (for  $k > 5$ ), can be performed in*

$$O\left(n^{3/2}k^{-1/2} \log^2 n + (n-k)^2 \sqrt{n/k} + (\sqrt{nk} + \log q)n \log^2(n/k)\right)$$

operations over  $\mathbb{F}_q$ .

Plugging in specific parameters and using efficient list decoding, we get the following corollary which will be useful below.

**Corollary 3.** *For parameters  $n \in \mathbb{N}$  and  $\delta \in (0, 1)$ , list decoding for  $[m, k]$  RS over  $\mathbb{F}_p$  where  $m = \Theta(d \log n / \delta^2)$ ,  $k = \Theta(d)$ ,  $p = O(n^2)$ , and  $d = \Omega(\log n)$  can be performed in time*

$$O\left(\frac{d^2 \log^{5/2} n \text{Arith}(n)}{\delta^5}\right),$$

where  $\text{Arith}(n)$  is a time bound on arithmetic operations over prime fields size  $O(n)$ .

**Theorem 7.** *Suppose  $k$ -OV takes  $n^{k-o(1)}$  time to decide for any  $d = \omega(\log n)$ , for any  $k \geq 2$ . Then, for any positive constants  $c, \epsilon > 0$  and  $0 < \delta < \epsilon/2$ ,  $\mathcal{FOV}^k$  is not*

$$(n^c \text{poly}(d, \log(p)), n^{k-\epsilon} \text{poly}(d, \log(p)), n^{-\delta} \text{poly}(d, \log(p)))$$

-batchable on average over the uniform distribution over its inputs.

*Proof.* Let  $k = 2c' + c$  and  $p > n^k$ . Suppose for the sake of contradiction that  $\mathcal{FOV}_{n,d,p}$  is  $(n^c \text{poly}(d, \log(p)), n^{2c'+c-\epsilon} \text{poly}(d, \log(p)), n^{-c'} \text{poly}(d, \log(p)))$ -batchable on average over the uniform distribution.

Let  $m = n^{k/(k+c)}$ , as before. By Proposition 2,  $k$ -OV with vectors of dimension  $d = \left(\frac{k}{k+c}\right)^2 \log^2 n$  is  $(m, m^c)$ -downward reducible to  $k$ -OV with vectors of dimension  $\log^2(n)$ , in time  $\tilde{O}(m^{c+1})$ .

For each  $j \in [m^c]$   $X_j = (U^{j1}, \dots, U^{jk}) \in \{0, 1\}^{kmd}$  is the instance of boolean-valued orthogonal vectors from the above reduction. Now, consider splitting these lists in half,  $U^{ji} = (U_0^{ji}, U_1^{ji})$  ( $i \in [k]$ ), such that  $(U_{a_1}^{j1}, \dots, U_{a_k}^{jk}) \in \{0, 1\}^{kmd/2}$  for  $\mathbf{a} \in \{0, 1\}^k$ . Interpret  $\mathbf{a}$  as binary number in  $\{0, \dots, 2^k - 1\}$ . Then, define the following  $2^k$  sub-problems:

$$A^{\mathbf{a}} = ((U_{a_1}^{j1}, \dots, U_{a_k}^{jk})), \forall \mathbf{a} \in \{0, \dots, 2^k - 1\}$$

Notice that given solutions to  $f\text{OV}_d^k$  on  $\{A^{\mathbf{a}}\}_{\mathbf{a} \in \{0, 1\}^k}$  we can trivially construct a solution to  $\text{OV}_d^k$  on  $X_j$ .

Now, draw random  $B_j, C_j \in \mathbb{F}_p^{kmd/2}$  and consider the following degree  $2^k$  polynomial in  $x$ :

$$D_j(x) = \sum_{i=1}^{2^k} \delta_i(x) A^{i-1} + (B_j + xC_j) \prod_{i=1}^{2^k} (x - i),$$

where  $\delta_i$  is the unique degree  $2^k - 1$  polynomial over  $\mathbb{F}_p$  that takes value 1 at  $i \in [2^k]$  and 0 on all other values in  $[2^k]$ . Notice that  $D_j(i) = A^{i-1}$  for  $i \in [2^k]$ .

Let  $r > 2^{k+1}d/\delta^2 \log m$ .  $D_j(2^k + 1), D_j(6), \dots, D_j(r + 2^k)$ . By the properties of Batch and because the  $D_j(\cdot)$ 's are independent,  $D_1(i), \dots, D_{m^c}(i)$  are independent for any fixed  $i$ . Thus,

$$\text{Batch}(D_1(i), \dots, D_{m^c}(i)) = f\text{OV}^k(D_1(i)), \dots, f\text{OV}^k(D_{m^c}(i))$$

for  $\delta r/2$   $i$ 's with probability at least  $1 - \frac{4}{\delta r} = 1 - 1/\text{polylog}(m)$ , by Chebyshev.

Now, because  $\delta r/2 > \sqrt{16dr}$ , we can run the list decoding algorithm of Roth and Ruckenstein, [RR00], to get a list of all polynomials with degree  $\leq 2^{k+1}d$  that agree with at least  $\delta r/2$  of the values. By Corollary 2, there are at most  $L = 3/\delta$  such polynomials.

By a counting argument, there can be at most  $2^k d \binom{L}{2} = O(dL^2)$  points in  $\mathbb{F}_p$  on which any two of the  $L$  polynomials agree. Because  $p > n^k > 2^k d \binom{L}{2}$ , we can find such a point,  $\ell$ , by brute-force in  $O(L \cdot dL^2 \log^3(dL^2) \log p)$  time, via batch *univariate* evaluation [Fid72]. Now, to identify the correct polynomials  $f\text{OV}^k(D_j(\cdot))$ , one only needs to determine the value  $f\text{OV}^k(D_j(\ell))$ . To do so, we can recursively apply the above reduction to all the  $D_j(\ell)$ s until the number of vectors,  $m$ , is constant and  $f\text{OV}^k$  can be evaluated in time  $O(d \log p)$ .

Because each recursive iteration cuts  $m$  in half, the depth of recursion is  $\log(m)$ . Additionally, because each iteration has error probability  $< 4/(\delta r)$ , taking a union bound over the  $\log(m)$  recursive steps yields an error probability that is  $\varepsilon < 4 \log m / (\delta r)$ .

We can find the prime  $p$  via  $O(\log m)$  random guesses in  $\{m^k + 1, \dots, 2m^k\}$  with overwhelming probability. By Corollary 3, taking  $r = 8d \log m / \delta^2$ , Roth and Ruckenstein's algorithm takes time  $O(d^2 / \delta^5 \log^{5/2} m \text{Arith}(m^k))$  in each recursive call. The brute force procedure takes time  $O(d / \delta^3 \log^3(d / \delta^2) \log m)$ , which is dominated by list decoding time. Reconstruction takes time  $O(\log m)$  in each round, and is also dominated. Thus the total run time is

$$T = O(m^c(m^{k-\varepsilon} d \log^2 m / \delta^2 + d^2 / \delta^5 \log^{7/2} m \text{Arith}(m^k))),$$

with error probability  $\varepsilon < 4 \log m \delta / d$ . □

## B Zero-Knowledge Proofs of Work

We combine our PoW with ElGamal encryption and a zero-knowledge proof of discrete logarithm equality to get a non-repudiable, non-transferable proof of work from the Decisional Diffie-Hellman assumption on Schnorr groups.

**Protocol.** Let  $\mathbb{Z}_p$  be a Schnorr group such of size  $p = qm + 1 \leq 2^{\text{polylog}(n)}$  such that DDH holds with generator  $g$ . (Assuming the DDH problem is hard for  $o(|G|^{1/2})$ -time probabilistic algorithms on a group  $G$ , we can take  $|G| \approx n^4$ .) Let  $(E, D)$  denote an ElGamal encryption system on  $G$ .

- Challenge is issued as before:  $(U, V) \leftarrow \mathbb{Z}_q^{2nd}$ .
- Prover generates a secret key  $x \leftarrow \mathbb{Z}_{p-1}$ , and sends encryptions of the coefficients of the challenge response over the subgroup size  $q$  to Verifier with the public key  $(g, h = g^x)$ :

$$\begin{aligned} E(R^*(\cdot); S(\cdot)) &= E(mr_0^*; s_0), \dots, E(mr_{nd-1}^*; s_{nd-1}) \\ &= (g^{s_0}, g^{r_0^*} h^{xs_0}), \dots, (g^{s_{nd-1}}, g^{mr_{nd-1}^*} h^{xs_{nd-1}}). \end{aligned}$$

Prover additionally draws  $t \leftarrow \mathbb{Z}_{p-1}$  and sends  $a_1 = g^t, a_2 = h^t$ .

- Verifier draws random  $z \leftarrow \mathbb{Z}_q$  and challenge  $c \leftarrow \mathbb{Z}_p^*$  and sends to Prover.
- Prover sends  $w = t + cS(z)$  to verifier.
- Verifier evaluates  $y = f\text{OV}_V(\phi_1(z), \dots, \phi_d(z))$  to get  $g^{my}$ . Then, homomorphically evaluates  $E(R^*; S)$  on  $z$  so that  $E(R^*(z); S(z))$  equals

$$\begin{aligned} & \left( (g^{s_0})(g^{s_1})^z \dots (g^{s_{nd-1}})^z, (g^{r_0^*} h^{s_0})(g^{mr_1^*} h^{s_1})^z \dots (g^{mr_{nd-1}^*} h^{s_{nd-1}})^z \right) \\ &= (u_1, u_2) \end{aligned}$$

Then, Verifier accepts if and only if

$$g^w = a_1(u_1)^c \quad \& \quad h^w = a_2(u_2/g^{my})^c.$$

Recall that the success probability of a subquadratic prover (in the non-zero-knowledge case) does not have negligible success probability. Thus the above should be performed on  $k = \text{polylog}(n)$  instances simultaneously and the verifier should accept iff an only if all instances accept.

*Remark B.1.* Note that the above protocol is public coin. Therefore, we can apply the Fiat-Shamir heuristic, and use a random oracle on partial transcripts to make the protocol non-interactive.

More explicitly, let  $H$  be a random oracle. Then:

- Prover computes

$$\begin{aligned} & (g, h), \\ & E(R^*; S), \\ & a_1 = g^t, a_2 = h^t, \\ & z = H(U, V, g, h, E(R^*; S), a_1, a_2), \\ & c = H(U, V, g, h, E(R^*; S), a_1, a_2, z), \\ & w = t + cS(z) \end{aligned}$$

and sends  $(g, h, E(R^*; S), a_1, a_2, w)$ .

- Verifier calls random oracle twice to get

$$z = H(U, V, g, h, E(R^*; S), a_1, a_2), c = H(U, V, g, h, E(R^*; S), a_1, a_2, z).$$

Then, the verifier homomorphically evaluates  $E(R^*; S)(z) = (u_1, u_2)$ , it then computes the value  $y = f\text{OV}_V(\phi_1(z), \dots, \phi_d(z))$ . Finally, accepts if and only if

$$g^w = a_1(u_1)^c \quad \& \quad h^w = a_2(u_2/g^{my})^c.$$

**Correctness.** From before, if  $R^* \equiv R_{U,V}$  as is the case for an honest prover, then for any  $z \in \mathbb{Z}_q$  we have  $R^*(z) = R_{U,V}(z) = f\text{OV}_V(\phi_1(z), \dots, \phi_d(z))$ . Moreover

$$g^w = g^{t+cS(z)} = g^t(g^{S(z)})^c = a_1 \left( (g^{s_0})(g^{s_1})^z \dots (g^{s_{nd-1}})^{z^d} \right)^c,$$

and

$$\begin{aligned} h^w &= h^{t+cS(z)} \\ &= h^t(g^0 h^{S(z)})^c \\ &= a_2 \left( (g^{s_0} h^{s_0})(g^{mr_1} h^{s_1})^z \dots (g^{mr_{nd-1}} h^{s_{nd-1}})^{z^d} g^{-f\text{OV}_V(\phi_1(z), \dots, \phi_d(z))} \right)^c. \end{aligned}$$

**Soundness.** Suppose Prover runs in subquadratic time, then with high probability  $R^* \neq R_{U,V}$ , and so for random  $z$ ,  $R^*(z) \neq f\text{OV}_V(\phi_1(z), \dots, \phi_d(z))$  with overwhelming probability. Suppose this is the case in what follows, namely:  $R^*(z) = y^* \neq y = f\text{OV}_V(\phi_1(z), \dots, \phi_d(z))$ . In particular,

$$\log_g u_1 \neq \log_h u_2 / g^{f\text{OV}_V(\phi_1(z), \dots, \phi_d(z))}.$$

Note that  $u_1, u_2 / g^{f\text{OV}_V(\phi_1(z), \dots, \phi_d(z))}$  can be calculated from the Prover's first message.

As is standard, we will fix the prover's first message and (assuming  $y \neq y^*$ ) rewind any two accepting transcripts with distinct challenges to show that  $\log_g u_1 = \log_h u_2 / g^y$ . Fix  $a_1, a_2$  as above and let  $(c, w), (c', w')$  be the two transcripts. Recall that if a transcript is accepted,  $g^w = a_1 u_1^c$  and  $h^w = a_2 (u_2 / g^y)^c$ . Then,

$$g^{w-w'} = u_1^{c-c'} \Rightarrow \log_g u_1 = \frac{w-w'}{c-c'} = \log_h u_2 / g^y \Leftarrow h^{w-w'} = (u_2 / g^y)^{c-c'}.$$

Therefore, because  $u_1 \neq u_2 / g^y$  there can be at most one  $c$  for which a Prover can convince the verifier. Such a  $c$  is chosen with negligible probability.

**Honest Verifier Zero Knowledge.** Given the verifier's challenge  $z, c$ , we can simulate the transcript of an honest prover as follows:

- Draw public key  $(g, h)$ .
- Compute the ElGamal Encryption  $E_{g,h}(R'; S)$  where  $R'$  is the polynomial with constant term  $f\text{OV}_V(\phi_1(z), \dots, \phi_d(z))$  and zeros elsewhere.
- Draw random  $w$ .
- Compute  $a_1 = \frac{g^w}{g^{cS(z)}}$  and  $a_w = \frac{h^w}{h^{cS(z)}}$ .
- Output  $((g, h), a_1, a_2, z, c, w)$ .

Notice that do to the semantic security of ElGamal, the transcript output is computationally indistinguishable from that of an honest Prover. Moreover, the simulator runs in  $\tilde{O}(nd)$  time, the time to compute  $R'$ , encrypt, evaluate  $S$  and exponentiate. Thus, the protocol is (honest verifier) zero-knowledge.

As mentioned, the Fiat-Shamir heuristic can remove interaction in the Random Oracle model and thus remove the condition that the verifier must be honest via the forking lemma.

**Efficiency.** The honest prover runs in time  $\tilde{O}(n^2)$ , because the  $nd$  encryptions can be performed in time  $\text{polylog}(n)$  each. The verifier takes  $\tilde{O}(nd)$  time as well. Note that the homomorphic evaluation requires  $O(d \log z^d) = O(d^2 \log z) = \text{polylog}(d)$  exponentiations and  $d = \text{polylog}(n)$  multiplications.