Proofs of Useful Work

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

February 27, 2017

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. We prove that, beyond being hard on the average (based on worst-case hardness assumptions), the task of evaluating our polynomials *cannot be amortized* 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.

Keywords: Proofs of Work, Fine-Grained, Delegation, Blockchain.

*Columbia University, New York, NY, USA. Email: marshall@cs.columbia.edu.

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

‡UC Berkeley, Berkeley, CA, USA. Email: msabin@berkeley.edu.

§CSAIL, Massachusetts Institute of Technology, Cambridge, MA, USA. Email: 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 these: Proofs of Work whose work is actually useful to solving *practical problems.*

1.1 Proofs of Work

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

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

While Gen and Verify should run very quickly, there should be a notion of *hardness* for Solve's runtime. More specifically, for some pre-specified length of working time, say $t(n)$, Solve should be able to produce solutions that 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 Verify with high probability. Thus valid solutions 'prove' that $t(n)$ work was completed in creating them. This hardness condition should also typically be extended for so that it is also not possible to amortize work over a large number of challenges.

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

- *A valid solution* $\boldsymbol{s}$ *to* $\boldsymbol{c_x}$ *can be computed in* $\widetilde{O}(n^2)$ *time.*
- *The validity of a candidate solution to* $\boldsymbol{c_x}$ *can be verified in* $\widetilde{O}(n)$ *time.*
- *Any valid solution to* $\boldsymbol{c_x}$ *requires* $n^{2-o(1)}$ *time to compute. (Hardness)*
- *Given a valid solution to* $\boldsymbol{c_x}$*,* OV *can be decided on the instance* $\boldsymbol{x}$ *in* $\widetilde{O}(n)$ *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 6).

Theorem 2 is also stronger than the above informal statement in that it says not just that any valid solution to a challenge takes $n^{2-o(1)}$ time to compute, but also that for any ℓ that is polynomial in n, finding valid solutions to a set of $\ell(n)$ 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. That is, *it is not possible to amortize work* over a large number of challenges - a very important feature for PoWs to have.

We prove this by proving the non-amortizability of computing a certain low-degree polynomial gOV (Definition 4) that simultaneously has the property that computing it in the worst-case immediately decides OV, and that there are efficiently verifiable certificates for its evaluations. Roughly, the uPoW scheme is structured so that the challenges are randomly generated inputs to gOV, and the solutions are evalutations of gOV at these inputs.

Informal Theorem 2. *Suppose k-OV takes $n^{k-o(1)}$ time to decide. For any polynomial ℓ, and inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\ell(n)}$ of size n that are selected independently and uniformly at random, any algorithm that computes all the $\mathsf{gOV}(\boldsymbol{x}_i)$'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 gOV (similar to the one shown in [BRSV17]), though it goes further than that with non-amortizability guarantees. The above is stated formally as Theorem 3, and again this theorem is more general and applies to the generalizations of OV mentioned earlier.

1.4 The Usefulness of Our PoWs

It is important to note that, besides just OV, the problems $\mathsf{3SUM}$, 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-amortizability) 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 three are complete for now has a new and particularly strong motivation.

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 PoW 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 proofs in Appendix B. While this is the opposite direction of what one may want from *usefulness*, it not only is interesting that this 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 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-amortizable) 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:

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

Taken together, these algorithms should result in an efficient proof system whose proofs are hard to find *and* useful. (Our hardness condition will actually be quite strong, including many arbitrary instances at once to account against amortizability. 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 $(\mathsf{Gen}, \mathsf{Solve}, \mathsf{Verify}, \mathsf{Recon})$. The algorithms must satisfy the following properties:*

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

- *For any $\boldsymbol{x}$, $\mathsf{Gen}(\boldsymbol{x})$ runs in time $\widetilde{O}(n)$.*
- *For any $\boldsymbol{c_x} \leftarrow \mathsf{Gen}(\boldsymbol{x})$, $\mathsf{Solve}(\boldsymbol{c_x})$ runs in time $\widetilde{O}(t(n))$.*
- *For any $\boldsymbol{c_x} \leftarrow \mathsf{Gen}(\boldsymbol{x})$ and any $\boldsymbol{s}$, $\mathsf{Verify}(\boldsymbol{c_x}, \boldsymbol{s})$ runs in time $\widetilde{O}(n)$.*
- *For any $\boldsymbol{c_x} \leftarrow \mathsf{Gen}(\boldsymbol{x})$ and $\boldsymbol{s} \leftarrow \mathsf{Solve}(\boldsymbol{c_x})$, $\mathsf{Recon}(\boldsymbol{c_x}, \boldsymbol{s})$ runs in time $\widetilde{O}(n)$.*

Completeness: *For any $\boldsymbol{c_x} \leftarrow \mathsf{Gen}(\boldsymbol{x})$ and any $\boldsymbol{s} \leftarrow \mathsf{Solve}(\boldsymbol{c_x})$,*

$$\Pr\left[\mathsf{Verify}(\boldsymbol{c}, \boldsymbol{s}) = \textit{accept}\ \right] = 1$$

Where the probability is taken over Verify's randomness.

Soundness: *For any $\boldsymbol{s}$ and $\boldsymbol{c_x} \leftarrow \mathsf{Gen}(\boldsymbol{x})$ such that $\mathsf{Recon}(\boldsymbol{c_x}, \boldsymbol{s}) \neq f(x)$,*

$$\Pr\left[\mathsf{Verify}(\boldsymbol{c_x}, \boldsymbol{s}) = \textit{accept}\right] < \mathrm{neg}(n)$$

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

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

$$\Pr\left[\forall i : \mathsf{Verify}(\boldsymbol{c}_i, \boldsymbol{s}_i) = \textit{accept} \;\middle|\; \begin{array}{l} (\boldsymbol{c}_i \leftarrow \mathsf{Gen}(\boldsymbol{x}_i))_{i \in [\ell(n)]} \\ (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{\ell(n)}) \leftarrow \mathsf{Solve}^*_\ell(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\ell(n)}) \end{array}\right] < \delta(n)$$

Where the probability is taken over Gen and Verify's randomness.

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

Usefulness: *For any $\boldsymbol{c_x} \leftarrow \mathsf{Gen}(\boldsymbol{x})$ and $\boldsymbol{s} \leftarrow \mathsf{Solve}(\boldsymbol{c_x})$, we have*

$$\mathsf{Recon}(\boldsymbol{c_x}, \boldsymbol{s}) = f(x).$$

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).

3 A Useful PoW for Orthogonal Vectors

In this section, we present a Proof of Useful Work (uPoW) for the Orthogonal Vectors (OV) problem and its generalization k-OV, both defined below. 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 uPoW's 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 statable in first-order logic all reduce to (moderate-dimension) OV [GI16] and so we obtain uPoWs useful for *any* problem statable 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 mainly 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 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 $\Omega(n^{2-o(1)})$ to decide, for which many conditional fine-grained hardness results are based on [Wil15], and has been 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 $\Omega(n^{k-o(1)})$ 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 $(U_1, \ldots, U_k)$ 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 $\mathbb{Z}$,*

$$\sum_{\ell \in [d(n)]} u^1_\ell \cdots u^k_\ell = 0$$

We say that such a set of vectors is k-orthogonal. *As with* OV*, 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 $\mathsf{gOV}^k_{n,d,p} : \mathbb{F}_p^{knd} \to \mathbb{F}_p$ 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^s_i represents the i^{th} vector in U_s, and for $\ell \in [d]$, $u^s_{i\ell}$ represents its ℓ^{th} coordinate.

$$\mathsf{gOV}^k_{n,d,p}(U_1, \ldots, U_k) = \sum_{i_1,\ldots,i_k \in [n]} \prod_{\ell \in [d]} \left(1 - u^1_{i_1 \ell} \cdots u^k_{i_k \ell}\right)$$

Note that the degree of this polynomial is kd. When given an instance of k-OV (from $\{0,1\}^{knd}$) as input, $\mathsf{gOV}^k_{n,d,p}$ 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 (GOV^k)**.** *Consider an integer* $k \geq 2$*. Let* $p(n)$ *be the smallest prime number larger than* $n^{\log n}$*, and* $d(n) = \lceil \log^2 n \rceil$*.* GOV^k *is the family of functions* $\left\{\mathsf{gOV}^k_{n,d(n),p(n)}\right\}$.

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

3.1 Preliminary Protocols

We describe here a protocol (Protocol 1) that proves the evaluation of polynomials in GOV on points (and can even delegate that evaluation) that 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 an (k-1)-round interactive proof that, given $U_1, \ldots, U_k \in \mathbb{F}_p^{nd}$ and $y \in \mathbb{F}_p$, proves that $\mathsf{gOV}^k_{n,d,p}(U_1, \ldots, U_k) = y$.

The special case of $k = 2$ for OV was shown as a non-interactive (MA) protocol in [Wil16] and this was used to create PoWs based on OV, $\mathsf{3SUM}$, 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 $\phi^s_1, \ldots, \phi^s_d : \mathbb{F}_p \to \mathbb{F}_p$, where ϕ^s_ℓ represents the ℓ^{th} column of U_s – that

is, for $i \in [n]$, $\phi^s_\ell(i) = u^s_{i\ell}$. Each such ϕ^s_ℓ has degree at most $(n-1)$. $\mathsf{gOV}^k_{n,d,p}$ can now be written as:

$$\begin{aligned}\mathsf{gOV}^k_{n,d,p}(U_1,\ldots,U_k) &= \sum_{i_1,\ldots,i_k\in[n]} \prod_{\ell\in[d]} \left(1 - u^1_{i_1\ell}\cdots u^k_{i_k\ell}\right) \\ &= \sum_{i_1,\ldots,i_k\in[n]} \prod_{\ell\in[d]} \left(1 - \phi^1_\ell(i_1)\cdots \phi^k_\ell(i_k)\right) \\ &= \sum_{i_1,\ldots,i_k\in[n]} q(i_1,\ldots,i_k)\end{aligned}$$

where q is defined for convenience as:

$$q(i_1,\ldots,i_k) = \prod_{\ell\in[d]} \left(1 - \phi^1_\ell(i_1)\cdots \phi^k_\ell(i_k)\right)$$

The degree of q is at most $D = k(n-1)d$. Note q can be evaluated at any point in $\mathbb{F}_p^k$ in time $\widetilde{O}(knd\log p)$, by evaluating all the $\phi^s_\ell(i_s)$'s, computing each term in the above product and then multiplying them.

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

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

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 asserted by Theorem 1, which is proven in Section 4.

Interactive Proof for GOV^k:

- The prover sends the co-efficients 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 co-efficients of a polynomial $q^*_{s+1,\alpha_1,\ldots,\alpha_s}$ of degree at most $(n-1)d$.
  - The verifier checks that $\sum_{i_{s+1}\in[n]} q^*_{s+1,\alpha_1,\ldots,\alpha_s}(i_{s+1}) = q^*_{s,\alpha_1,\ldots,\alpha_{s-1}}(\alpha_s)$. If not, it rejects.
- The verifier picks $\alpha_{k-1} \leftarrow \mathbb{F}_p$ and checks that $q^*_{k-1,\alpha_1,\ldots,\alpha_{k-2}}(\alpha_{k-1}) = q_{k-1,\alpha_1,\ldots,\alpha_{k-2}}(\alpha_{k-1})$, which it computes using the fact that $q_{k-1,\alpha_1,\ldots,\alpha_{k-2}}(\alpha_{k-1}) = \sum_{i_k\in[n]} q_{k,\alpha_1,\ldots,\alpha_{k-1}}(i_k)$. If not, it rejects.
- If the verifier hasn't rejected yet, it accepts.

Protocol 1: Interactive Proof for GOV^k.

For this protocol we have the following guarantee. We will prove this in Section 4.

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 $y = \mathsf{GOV}^k(x)$. This protocol has perfect completeness and soundness error at most $\left(\frac{knd}{p}\right)$. The prover runs in time $\widetilde{O}(n^k d \log p)$, and the verifier in time $\widetilde{O}(knd^2 \log p)$.*

As observed earlier, Protocol 1 is non-interactive when $k = 2$. We then get the following corollary for GOV.

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

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 have *hardness* and *usefulness* to extend this proof system to a uPoW in some form.

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:

- $\mathsf{Gen}(\boldsymbol{x})$:
  - Given an instance $\boldsymbol{x} \in \{0,1\}^{knd}$, interpret $\boldsymbol{x}$ as an element of $\mathbb{F}_p^{knd}$ (where $p = p(n)$ is as in Definition 4).
  - Pick a random $\boldsymbol{r} \in \mathbb{F}_p^{knd}$.
  - Output the set of vectors $\boldsymbol{c_x} = \{\boldsymbol{y}_t = \boldsymbol{x} + t\boldsymbol{r} \mid t \in [kd+1]\}$.
- $(\mathsf{Solve}, \mathsf{Verify})$ work as follows given $\boldsymbol{c_x} = \{\boldsymbol{y}_t\}$:
  - Solve computes $z_t = \mathsf{gOV}^k_{n,d,p}(\boldsymbol{y}_t)$ and outputs the set $\boldsymbol{s} = \{z_t\}_{t \in [kd+1]}$.
  - For each t in parallel: Solve and Verify run Protocol 1 with input $(\boldsymbol{y}_t, z_t)$.
  - Verify accepts iff all of the above instances of Protocol 1 accept.
- $\mathsf{Recon}(\boldsymbol{c_x}, \boldsymbol{s})$:
  - Interpret $z_1, \ldots, z_{kd+1}$ as the evaluations of a univariate polynomial $h(t)$ of degree kd at $t = 1, \ldots, kd+1$.
  - 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, δ)-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 $\widetilde{O}(n^{k/2})$. To keep verification and reconstruction time at $\widetilde{O}(n)$, we instead show how to remove interaction in the Random Oracle model in Section 6.

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 amortized. We first define what it means for a function to be non-amortizable in the average-case in a manner compatible with the hardness requirement. Note that this requirement is stronger than being non-amortizable in the worst-case.

Definition 5. *Consider a function family $\mathcal{G} = \{g_n : \mathcal{X}_n \to \mathcal{Y}_n\}$, and a family of distributions $\mathcal{D} = \{D_n\}$, where D_n is over $\mathcal{X}_n$. $\mathcal{G}$ is* not (ℓ, t, δ)-amortizable on average over $\mathcal{D}$ *if, for any algorithm* Amort *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,*

$$\Pr_{x_i \leftarrow D_n}\left[\mathsf{Amort}(x_1, \ldots, x_{\ell(n)}) = (g_n(x_1), \ldots, g_n(x_{\ell(n)}))\right] < \delta(n)$$

We will be concerned with the case where the amortized time $t(n)$ is less than the time it takes to compute f_n on a single instance. Theorem 3, then, claims that we achieve this non-amortizability and will let us prove the desired hardness of Protocol 2, as GOV^k is one of the things that Solve is required to compute there. We prove this theorem in Appendix A, and prove a weaker version for illustrative purposes in Section 5.

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 $\delta < \epsilon/2$, GOV^k is* not $(n^c, n^{k-\epsilon}, 1/n^\delta)$*-amortizable 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 is argued as follows:

- $\mathsf{Gen}(\boldsymbol{x})$ simply computes $(kd+1)$ linear combinations over $\mathbb{F}^{knd}$ to create all the elements of $\boldsymbol{c_x}$ 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 Chebyshev's Theorem, this takes $\widetilde{O}(n)$ time.
- Solve computes $\mathsf{gOV}^k_{n,d,p}(\boldsymbol{y}_t)$ on $(kd+1)$ values of $\boldsymbol{y}_t$, each of which can be done in $\widetilde{O}(n^k)$ time. It then runs the prover in $(kd+1)$ instantiations of Protocol 1, each of which can be done in $\widetilde{O}(n^k)$ time by Theorem 1. So in all it takes takes $\widetilde{O}(n^k(kd+1)) = \widetilde{O}(n^k)$ time.
- Verify runs the verifier in $(kd+1)$ instantiations of Protocol 1, taking a total of $\widetilde{O}(n(kd+1)) = \widetilde{O}(n)$ 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 $\widetilde{O}((kd)^2)$, which is much less than n.

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

Completeness follows immediately from the completeness of Protocol 1 as an interactive proof for GOV^k, as stated in Theorem 1, as this is the protocol that Solve and Verify engage in. **Soundness** follows from the soundness of Protocol 1 and the correctness of Recon as argued above.

Hardness. We proceed by contradiction. On a high level, we will assume that there is in fact a set of instances that yield easy (amortizable) challenges on average and show that, given these instances as non-uniform advice, we can break GOV^k's average-case hardness:

Suppose there is an (interactive) algorithm Solve^*, a polynomial ℓ, a set of k-OV instances $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\ell(n)} \in \{0,1\}^{knd}$, and an $\epsilon > 0$ such that Solve^* 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 gOV instances produced by $\mathsf{Gen}(\boldsymbol{x}_i)$ be $\left\{\boldsymbol{y}_t^i\right\}$, and the corresponding sets $\boldsymbol{s}_i$ produced by Solve^* be $\left\{z_t^i\right\}$. So Solve^* succeeds as a prover in Protocol 1 for *all* the instances $\left\{(\boldsymbol{y}_t^i, z_t^i)\right\}$ with probability at least $\delta(n)$.

By the negligible soundness of Protocol 1 guaranteed by Theorem 1, in order to do this, Solve^* has to use the correct values $\mathsf{gOV}^k_{n,d,p}(\boldsymbol{y}_t^i)$ 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 $\mathsf{gOV}^k_{n,d,p}$ at $\boldsymbol{y}_1^1, \ldots, \boldsymbol{y}_1^{\ell(n)}$, all of which are independent uniform points in $\mathbb{F}_p^{knd}$.

Such a Solve^* can now be used as follows to compute $\mathsf{gOV}^k_{n,d,p}$ on any independently random $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{\ell(n)} \in \mathbb{F}_p^{knd}$ as follows. Take the points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\ell(n)}$ (on which Solve^* succeeds as above) as non-uniform advice. Now for each $i \in [\ell(n)]$, take $\boldsymbol{y}_1$ to be $\hat{\boldsymbol{y}}_1^i$, and generate the other $\hat{\boldsymbol{y}}_t^i$'s as the next kd points on the line joining $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$. This set $\left\{\hat{\boldsymbol{y}}_t^i\right\}$ is distributed identically to $\left\{\boldsymbol{y}_t^i\right\}$ above, and so Solve^* computes $\mathsf{gOV}^k_{n,d,p}$ on all of these correctly with probability greater than $\delta(n)/2$.

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

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

4 Verifying GOV^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 GOV^k. We use here terminology from the theorem statement and protocol description. Recall the the input to the protocol is $U_1, \ldots, U_k \in \mathbb{F}_p^{nd}$ and $y \in \mathbb{F}_p$, and the prover wishes to prove that $y = \mathsf{gOV}^k_{n,d,p}(U_1, \ldots, U_k)$.

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

Soundness. Suppose $y \neq \mathsf{gOV}^k_{n,d,p}(U_1, \ldots, U_k)$. 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 α_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,α_1} that is different from q_{2,α_1}, which again agree on α_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 at least one of these happens 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 co-efficients of a univariate polynomial $q : \mathbb{F}_p \to \mathbb{F}_p$ of degree at most N, and N points $x_1, \ldots, x_N \in \mathbb{F}_p$, the set of evaluations $(q(x_1), \ldots, q(x_N))$ can be computed in time $O(N \log^3 N \log p)$.*

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

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

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

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

- Fix some value of $s, \alpha_1, \ldots, \alpha_{s-1}$.
- For each $\ell \in [d]$, compute the evaluation of ϕ^s_ℓ on nd points, say $\{1, \ldots, nd\}$.
  - Since its co-efficients are known, the evaluations of each ϕ^s_ℓ on these nd points can be computed in time $\widetilde{O}(nd \log p)$ by Lemma 1, for a total of $\widetilde{O}(nd^2 \log p)$ for all the ϕ^s_ℓ's.
- For each setting of $i_{s+1}, \ldots, i_k$, compute the evaluations of the polynomial $\rho_{i_{s+1},\ldots,i_k}(x) = q(\alpha_1, \ldots, \alpha_{s-1}, x, i_{s+1}, \ldots, i_k)$, on the points $\{1, \ldots, nd\}$.
  - First substitute the constants $\alpha_1, \ldots, \alpha_{s-1}, i_{s+1}, \ldots, i_k$ into the definition of q.
  - This requires computing, for each $\ell \in [d]$ and $s' \in [k] \setminus \{s\}$, either $\phi^{s'}_\ell(\alpha_s)$ or $\phi^{s'}_\ell(i_s)$. All of this can be done in time $\widetilde{O}(knd \log p)$ by direct polynomial evaluations since the co-efficients of the $\phi^{s'}_\ell$'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 $\widetilde{O}(knd \log p)$ by multiplying the constants computed in the above step with the evaluations of $\phi^{s'}_\ell$ on these points, and subtracting from 1.
  - The product of the evaluations can now be computed in time $\widetilde{O}(nd^2 \log p)$ to get what we need.
- Add up the evaluations of $\rho_{i_{s+1},\ldots,i_k}$ pointwise over all settings of $(i_{s+1}, \ldots, i_k)$.
  - There are n^{k-s} possible settings of $(i_{s+1}, \ldots, i_k)$, and for each of these we have nd evaluations. All the additions hence take $\widetilde{O}(n^{k-s+1}d \log p)$ time.

- This gives us nd evaluations of $q_{s,\alpha_1,\ldots,\alpha_{s-1}}$, which is a univariate polynomial of degree at most $(n-1)d$. So its coefficients can be computed in time $\widetilde{O}(nd \log p)$ by Lemma 2.
- It can be verified from the intermediate complexity computations above that all these operations together take $\widetilde{O}((n^{k-s+1}d + nd^2) \log p)$ time. This proves the proposition.

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

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

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

5 Non-Amortizability of GOV

In this section we prove the following weaker version of Theorem 3, which itself is proven in Appendix A using an extension of the techniques employed here. The notion of non-amortizatbility used below is defined in Definition 5 in Section 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, for any constants $c, \epsilon > 0$, GOV^k is* not *$(n^c, n^{k-\epsilon}, 7/8)$-amortizable on average over the uniform distribution over its inputs.*

Throughout this section, $\mathcal{F}$, $\mathcal{F}'$ and $\mathcal{G}$ are families of functions $\{f_n : \mathcal{X}_n \to \mathcal{Y}_n\}$, $\{f'_n : \mathcal{X}'_n \to \mathcal{Y}'_n\}$ and $\left\{g_n : \hat{\mathcal{X}}_n \to \hat{\mathcal{Y}}_n\right\}$, and $\mathcal{D} = \{D_n\}$ is a family of distributions where D_n is over $\hat{\mathcal{X}}_n$. s, ℓ, t, by themselves or with various subscripts, are all functions from $\mathbb{N} \to \mathbb{N}$. All algorithms are to be taken to be randomised by default.

Theorem 4 is the result of two properties possessed by GOV^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, ℓ)-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 $\mathcal{X}'_{s(n)}$.*

$$\mathsf{Split}(x) = (x_1, \ldots, x_{\ell(n)})$$

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

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

- 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* ℓ-robustly reducible *to $\mathcal{G}$ in time t if there is a pair of algorithms* $(\mathsf{Split}, \mathsf{Merge})$ *satisfying:*

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

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

- *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:*

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

- 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 would have been sufficient if, in the above definition, the reconstruction above had worked for most r's (and not necessarily all r's), but we leave it as it is for simplicity of presentation.

Lemma 3. *Suppose $\mathcal{F}$, $\mathcal{F}'$ and $\mathcal{G}$ have the following properties:*

- *$\mathcal{F}$ is (s_d, ℓ_d)-downward reducible to $\mathcal{F}'$ in time t_d.*
- *$\mathcal{F}'$ is ℓ_r-robustly reducible to $\mathcal{G}$ over $\mathcal{D}$ in time t_r.*
- *$\mathcal{G}$ is $(\ell_a, t_a, 7/8)$-amortizable 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:

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

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 $\mathcal{F}$, but we leave it this way again for simplicity of presentation. We now show how to prove Theorem 4 using Lemma 3, and then prove the lemma itself.

Proof of Theorem 4. Fix any $k \geq 2$. Suppose, towards a contradiction, that for some $c, \epsilon > 0$, GOV^k is $(n^c, n^{k-\epsilon}, 7/8)$-amortizable 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 GOV^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 $\widetilde{O}(m^{c+1})$.*

Split_d, when given an instance $(U_1, \ldots, U_k) \in \{0,1\}^{k(n\times d)}$, first divides each U_i into $m^{c/k}$ partitions $U_{i1}, \ldots, U_{im^{c/k}} \in \{0,1\}^{m\times d}$. It then outputs the set of tuples $\left\{(U_{1j_1}, \ldots, U_{kj_k}) \mid j_i \in [m^{c/k}]\right\}$. 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 Merge_d simply computes the disjunction of the $\mathcal{F}'$ outputs to these instances.

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

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

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

Given an input $\boldsymbol{x} \in \mathbb{F}_p^{knd}$, Split_r picks two uniformly random $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{F}_p^{knd}$ and outputs the set of vectors $\left\{\boldsymbol{x} + t\boldsymbol{x}_1 + t^2\boldsymbol{x}_2 \mid t \in \{1, \ldots, 12kd\}\right\}$. 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 $\boldsymbol{x}$.

Define the univariate polynomial $g_{\boldsymbol{x},\boldsymbol{x}_1,\boldsymbol{x}_2}(t) = \mathsf{gOV}^k_m(\boldsymbol{x} + t\boldsymbol{x}_1 + t^2\boldsymbol{x}_2)$. Note that its degree is at most $2kd$. When Merge_r is given $(y_1, \ldots, y_{12kd})$ that are purported to be the evaluations of gOV^k_m on the points produced by Split, these can be seen as purported evaluations of $g_{\boldsymbol{x},\boldsymbol{x}_1,\boldsymbol{x}_2}$ on $\{1, \ldots, 12kd\}$. 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 Merge_r uses the Berlekamp-Welch algorithm to recover $g_{\boldsymbol{x},\boldsymbol{x}_1,\boldsymbol{x}_2}$, which can be evaluated at 0 to obtain $\mathsf{gOV}^k_n(\boldsymbol{x})$.

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

By our assumption at the beginning, $\mathcal{G}$ is $(n^c, n^{k-\epsilon}, 7/8)$-amortizable on average over the uniform distribution. Together with the above propositions, this satisfies all the requirements in the hypothesis of Lemma 3, which now tells us that $\mathcal{F}$ can be computed in the worst-case in time:

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

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

Proof of Lemma 3. Given the hypothesised downward reduction $(\mathsf{Split}_d, \mathsf{Merge}_d)$, robust reduction $(\mathsf{Split}_r, \mathsf{Merge}_r)$ and amortization algorithm Amort for $\mathcal{F}$, f_n can be computed as follows (for large enough n) on an input $x \in \mathcal{X}_n$:

- Run $\mathsf{Split}_d(x)$ to get $x_1, \ldots, x_{\ell_d(n)} \in \mathcal{X}'_{s_d(n)}$.
- For each $i \in [\ell_d(n)]$, run $\mathsf{Split}_r(x_i; r_i)$ to get $x_{i1}, \ldots, x_{i\ell_r(s_d(n))} \in \hat{\mathcal{X}}_{s_d(n)}$.

- For each $j \in [\ell_r(s_d(n))]$, run $\mathsf{Amort}(x_{1j}, \ldots, x_{\ell_d(n)j})$ to get the outputs $y_{1j}, \ldots, y_{\ell_d(n)j} \in \hat{\mathcal{Y}}_{s_d(n)}$.
- For each $i \in [\ell_d(n)]$, run $\mathsf{Merge}_r(x_i, r_i, y_{i1}, \ldots, y_{i\ell_r(s_d(n))})$ to get $y_i \in \mathcal{Y}'_{s_d(n)}$.
- Run $\mathsf{Merge}_d(x, y_1, \ldots, y_{\ell_d(n)})$ 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 Amort, 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 $\mathsf{Amort}(x_{1j}, \ldots, x_{\ell_d(n)j})$ 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 Amort, we have the following:

$$\mathrm{E}[I] \geq \frac{7}{8}m$$
$$\mathrm{Var}[I] \leq \frac{7}{64}m$$

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

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

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 Merge_r directly in the above algorithm, we would use Merge'_r that runs 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 Amort on sets of $\ell_d(n) = \ell_a(s_d(n))$ instances, summing up to the total time stated in the lemma. □

6 Removing Interaction

In this section we show how to remove the interaction in Protocol 2 via the Fiat-Shamir Heuristic, and 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 will be as in Definition 4 of GOV^k, where the instance size n will be clear from context.

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

A Non-interactive uPoW for k-OV

$\mathsf{Gen}(\boldsymbol{x})$**:**

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

$\mathsf{Solve}(\boldsymbol{c}_x)$**:**

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

- Compute $z_y = \mathsf{gOV}^k_{n,d,p}(\boldsymbol{y})$.
- Compute coefficients of q_1^y Let $\tau_1^y = (z^y, q_1^y)$
- For s from 1 to $k-2$:
  - Compute $\alpha_s^y = H(\boldsymbol{c}_x, \tau_s^y)$.
  - Compute coefficients of $q_{s+1}^y = q_{s+1,\alpha_1,\ldots,\alpha_s}$, with respect to $\boldsymbol{y}$.
  - Set $\tau_{s+1}^y = (\tau_s^y, q_{s+1}^y)$.

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

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

Given $T^* = \{\tau^{y*} = (z^{y*}, q_1^{y*}, q_2^{y*}, \ldots, q_{k-1}^{y*})\}$, 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 $\alpha_1^{y*} = H(\boldsymbol{c}_x, z^{y*}, q_1^{y*})$.
- For s from 1 up to $k-2$:
  - Compute $\alpha_s^{y*} = H(\boldsymbol{c}_x, z^{y*}, q_1^{y*}, \ldots, q_s^{y*})$.
  - 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 $\alpha_{k-1}^{y^*} = H(\boldsymbol{c}_x, z^{y*}, q_1^{y*}, \ldots, q_{k-2}^{y*})$.
- 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.

$\mathsf{Recon}(\boldsymbol{c_x}, \{z_{y_t}\})$:

- Interpret $z_{y_1}, \ldots, z_{y_{kd+1}}$ as the evaluations of a univariate polynomial $h(t)$ of degree kd at $t = 1, \ldots, kd+1$.
- 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

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, Protocol 3 is a* non-interactive *(n^k, δ)-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 5.

Lemma 4. *For any $k \geq 2$, if Protocol 2 is sound as a Proof of Useful Work for GOV^k, then Protocol 3 is sound as a non-interactive Proof of Useful Work for GOV^k in the Random Oracle model.*

Note that in the above uPoW, we can cluster the s^{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", 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 $m\mathsf{poly}(n)$ queries to the random oracle, H. We select k of the m query indices, $i_1, \ldots, i_k$. Let the verifier V (recall that our protocol is public coin) output the k indendently drawn uniform challenges $\alpha_1, \ldots, \alpha_k$ on randomness r. We then program a random oracle H_r to output α_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 H_r 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 himself and breaks soundness with probability $\varepsilon'(n)$. □

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 existing 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 $(f, \boldsymbol{x})$ 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 $\boldsymbol{x}$ is the instance being delegated to find $f(\boldsymbol{x})$ for (we assume for now that our practical problems are already in the

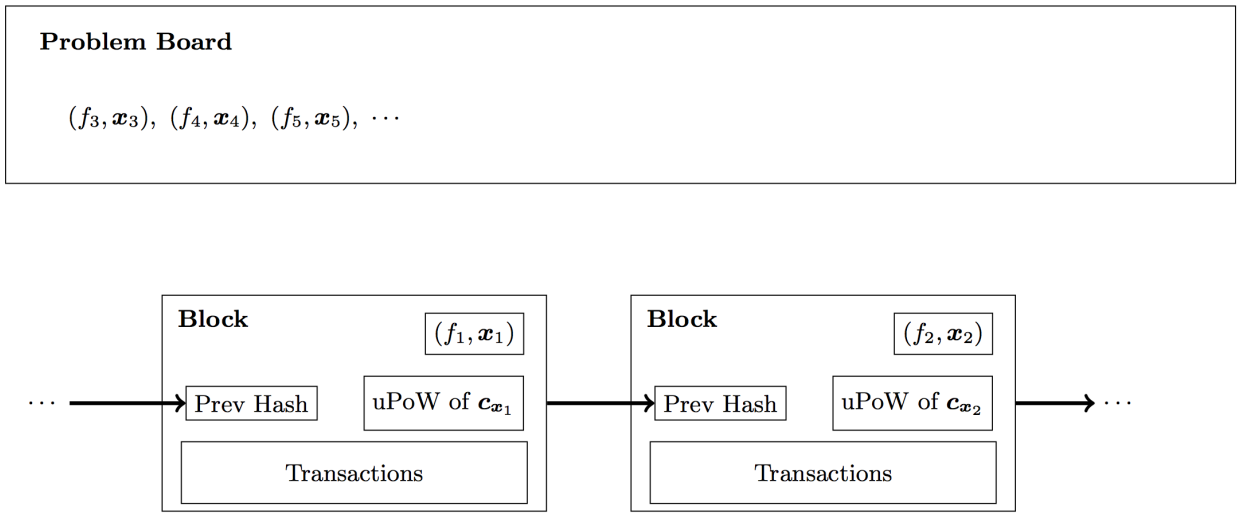

Figure 1: We present a framework for Blockchain to use uPoWs. Delegators post to a public Problem Board from which workers grab problems to mine a block with by producing a Proof of Useful Work that they attach to the block.

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 $\boldsymbol{c}_{\boldsymbol{x}}$ themselves as usual except substituting $r = H(\text{current block})$ for the randomness usually used to generate the challenge: $\boldsymbol{c}_{\boldsymbol{x}} = \{\boldsymbol{x} + rt \mid t \in [D+1]\}$, where $r = H(\text{current block})$. For a truly Random Oracle H, 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 $(f, \boldsymbol{x})$ 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 Bitcion 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

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

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, 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 $\boldsymbol{c_x}$ challenge with H and then checking the uPoW for it. Further, the delegator upon seeing the uPoW can quickly reconstruct $f(x)$ 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 Guy Rothblum for his 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 Vaikuntanathan 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).

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+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.

[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 Even Further Non-Amortizability of GOV

In this section, we show that a slightly stronger, but natural form of amortization is impossible for GOV^k. In particular, it is sufficient to define a notion of amortizability 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, $\mathsf{GOV}^k_{n,d,p}$ is $(n, t, \delta) - amortizable$.

Instead, we simply generalize amortizability in a parametrized fashion for $\mathsf{GOV}^k_{n,d,p}$.

Definition 8. *A parametrized class, $\mathcal{G}_\rho$, is* not (ℓ, t, δ)-amortizable on average over $\mathcal{D}_\rho$, a parametrized family of distributions *if, for any fixed parameters N, ρ, and algorithm Amort_ρ that runs in time $\ell(\rho)t(\rho)$, when run on $\ell(\rho)$ inputs from $\hat{\mathcal{X}}_\rho$, when it is given as input $\ell(\rho)$ independent samples from D_ρ,*

$$\Pr_{x_i \leftarrow D_\rho}\left[\mathsf{Amort}(x_1, \ldots, x_{\ell(\rho)}) = (g_\rho(x_1), \ldots, g_\rho(x_{\ell(\rho)}))\right] < \delta(\rho)$$

We now show how a generalization of the list decoding reduction from [BRSV17] yields strong amortization 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, $(x_1, y_1), \ldots, (x_m, y_m)$. We include a laconic proof here with the given notation for convenience.

Lemma 5 ([CPS99])**.** *Let q be a polynomial over $\mathbb{F}_p$, and define* $\mathrm{Graph}(q) := \{(i, p(i)) \mid i \in [p]\}$. *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 $|\mathrm{Graph}(q) \cap S| \geq m\delta/2$.*

Corollary 2. *Let S be as in Lemma 5 with $I = \{m+1, \ldots, p\}$, 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 $|\mathrm{Graph}(q) \cap S| \geq m\delta/2$.*

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 $\mathsf{S}_f := \{(i, f(i)) \in \mathrm{Graph}(f) \cap S\}$, 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 6 ([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, 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*

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

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

Theorem 6. *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 $0 < \delta < \varepsilon/2$, GOV^k is* not

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

-amortizable 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 $\mathsf{GOV}_{n,d,p}$ is $(n^c\mathsf{poly}(d, \log(p)), n^{2c'+c-\epsilon}\mathsf{poly}(d, \log(p)), n^{-c'}\mathsf{poly}(d, \log(p)))$-amortizable on average over the uniform distribution.

Let $m = n^{k/(k+c)}$, as before. By Proposition 2, k-OV with vectors of dimension $d = (\frac{k}{k+c}))^2 \log^2 n$ 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 $j \in [m^c]$ $X_j = (U^{j1}, \ldots, U^{jk}) \in \{0,1\}^{kmd}$ 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 $(U_{a_1}^{j1}, \ldots, U_{a_k}^{jk}) \in \{0,1\}^{kmd/2}$ for $\boldsymbol{a} \in \{0,1\}^k$. Interpret $\boldsymbol{a}$ as binary number in $\{0, \ldots, 2^k - 1\}$. Then, define the following 2^k sub-problems:

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

Notice that given solutions to gOV_d^k on $\{A^{\boldsymbol{a}}\}_{\boldsymbol{a}\in\{0,1\}^k}$ we can trivially construct a solution to 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 δ_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 $r > 2^{k+1}d/\delta^2 \log m$. $D_j(2^k + 1), D_j(6), \ldots, D_j(r + 2^k)$. By the properties of Amort and because the $D_j(\cdot)$'s are independent, $D_1(i), \ldots, D_{m^c}(i)$ are independent for any fixed i. Thus,

$$\mathsf{Amort}(D_1(i), \ldots, D_{m^c}(i)) = \mathsf{gOV}^k(D_1(i)), \ldots, \mathsf{gOV}^k(D_{m^c}(i))$$

for $\delta r/2$ i's with probability at least $1 - \frac{4}{\delta r} = 1 - 1/\operatorname{polylog}(m)$, 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, ℓ, 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 $\mathsf{gOV}^k(D_j(\cdot))$, one only needs to determine the value $\mathsf{gOV}^k(D_j(\ell))$. 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 gOV^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 $\{m^k + 1, \ldots, 2m^k\}$ with overwhelming probability. By Corollary 3, taking $r = 8d \log m/\delta^2$, Roth and Ruckenstein's algorithm takes time $O(d^2/\delta^5 \log^{5/2} m \operatorname{Arith}(m^k))$ 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

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

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 an non-repudiatable, 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 $p = qm + 1 \leq 2^{\mathrm{polylog}(n)}$ 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 coeffecients of the challenge response over the subgroup size q to Verifier with the public key $(g, h = g^x)$:

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

  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 $y = \mathsf{gOV}_V(\phi_1(z), \ldots, \phi_d(z))$ to get g^{my}. Then, homomorphically evaluates $\mathsf{E}(R^*; S)$ on z so that $\mathsf{E}(R^*(z); S(z))$ equals

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

  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 $k = \mathrm{polylog}(n)$ 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

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

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

- Verifier calls random oracle twice to get

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

  Then, the verifier homomorphically evaluates $\mathsf{E}(R^*; S)(z) = (u_1, u_2)$, it then computes the value $y = gOV_V(\phi_1(z), \ldots, \phi_d(z))$. 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 $R^*(z) = R_{U,V}(z) = \mathsf{gOV}_V(\phi_1(z), \ldots, \phi_d(z))$. Moreover

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

and

$$\begin{aligned} h^w &= h^{t+cS(z)} \\ &= h^t(g^0 h^{S(z)})^c \\ &= a_2 \left((g^{r_0^*} h^{s_0})(g^{mr_1^*} h^{s_1})^z \cdots (g^{mr_{nd-1}^*} h^{s_{nd-1}})^{z^d} g^{-\mathsf{gOV}_V(\phi_1(z),\ldots,\phi_d(z))}\right)^c. \end{aligned}$$

Soundness. Suppose Prover runs in subquadratic time, then with high probability $R^* \not\equiv R_{U,V}$, and so for random z, $R^*(z) \neq \mathsf{gOV}_V(\phi_1(z), \ldots, \phi_d(z))$ with overwhelming probability. Suppose this is the case in what follows, namely: $R^*(z) = y^* \neq y = \mathsf{gOV}_V(\phi_1(z), \ldots, \phi_d(z))$. In particular,

$$\log_g u_1 \neq \log_h u_2/g^{\mathsf{gOV}_V(\phi_1(z),\ldots,\phi_d(z))}.$$

Note that $u_1, u_2/g^{\mathsf{gOV}_V(\phi_1(z),\ldots,\phi_d(z))}$ 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 $\mathsf{E}_{g,h}(R'; S)$ where R' is the polynomial with constant term $\mathsf{gOV}_V(\phi_1(z), \ldots, \phi_d(z))$ 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.

The usual trick allows us to remove the honest verifier condition.

Efficiency. The honest prover runs in time $\tilde{O}(n^2)$, because the nd encryptions can be performed in time polylog(n) each. The verfier 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.