PPAD is as Hard as LWE and Iterated Squaring

Nir Bitansky* Arka Rai Choudhuri† Justin Holmgren‡ Chethan Kamath*
Alex Lombardi§ Omer Paneth* Ron D. Rothblum¶

Abstract

One of the most fundamental results in game theory is that every finite strategic game has a Nash equilibrium, an assignment of (randomized) strategies to players with the stability property that no individual player can benefit from deviating from the assigned strategy. It is not known how to efficiently *compute* such a Nash equilibrium — the computational complexity of this task is characterized by the class **PPAD**, but the relation of **PPAD** to other problems and well-known complexity classes is not precisely understood. In recent years there has been mounting evidence, based on cryptographic tools and techniques, showing the hardness of **PPAD**.

We continue this line of research by showing that **PPAD** is as hard as *learning with errors* (LWE) and the *iterated squaring* (IS) problem, two standard problems in cryptography. Our work improves over prior hardness results that relied either on (1) sub-exponential assumptions, or (2) relied on "obfustopia," which can currently be based on a particular combination of three assumptions. Our work additionally establishes *public-coin* hardness for **PPAD** (computational hardness for a publicly sampleable distribution of instances) that seems out of reach of the obfustopia approach.

Following the work of Choudhuri et al. (STOC 2019) and subsequent works, our hardness result is obtained by constructing an *unambiguous and incrementally-updatable* succinct non-interactive argument for IS, whose soundness relies on polynomial hardness of LWE. The result also implies a verifiable delay function *with unique proofs*, which may be of independent interest.

*Tel Aviv University. Email: nirbitan@tau.ac.il, ckamath@protonmail.com, omerpa@tauex.tau.ac.il
†UC Berkeley. Email: arkarc@berkeley.edu
‡NTT Research. Email: justin.holmgren@ntt-research.com
§MIT. Email: alexlombardi@alum.mit.edu
¶Technion. Email: rothblum@cs.technion.ac.il

Contents

1 Introduction

The concept of a Nash equilibrium is fundamental to the modern understanding of *games*: given a description of payoffs as a function of k player strategies (which take value in a finite domain), what are a collection of strategy *distributions* that cannot be locally improved? It is not a priori clear that such mixed strategies should exist, but the seminal work of Nash [Nas51] shows that they do. In the language of modern computational complexity, this implies that Nash equilibrium is a *total search problem*, a search problem such that every instance of the problem is guaranteed to have a solution. It turns out that computing (arbitrarily good approximate) solutions to this problem is in fact in the complexity class **TFNP** [MP91], the class of total search problems with *efficient verification*. In fact, it is *complete* for its subclass called **PPAD** [Pap94, DGP09, CDT09], for which the existence of solution is guaranteed via "*p*olynomial *p*arity *a*rgument on *d*irected graphs". Thus, understanding the computational complexity of **PPAD** exactly corresponds to understanding the complexity of computing a Nash equilibrium.

Despite many decades of attention, we do not currently have polynomial-time algorithms for Nash (or any **PPAD**-complete problem); indeed, it is widely believed that **PPAD** is computationally intractable. Understanding to what extent this is the case, and why, has been a major line of research at the intersection of game theory, computational complexity, and (perhaps surprisingly) *cryptography*. In our work, we further explore this connection to cryptography and prove new hardness results for **PPAD** under cryptographic assumptions.

Prior work. Before describing our results, we summarize the state of affairs prior to our work. The goal of this line of work is to prove theorems of the form "if **PPAD** can be solved in polynomial-time, then standard cryptography is broken." The usual notion of "cryptography is broken" is that there is a probabilistic polynomial-time (PPT) algorithm solving a problem fundamental to cryptography with non-negligible advantage or success probability. As we will see, prior work, which fall into the two categories described below, falls somewhat short of achieving this ideal.

- **Specialized Proof Systems**: Starting from [CHK+19a], there has been a sequence of works obtaining hardness in **PPAD** by building *unambiguous, incremental, succinct non-interactive arguments* [CHK+19a, CHK+19b, EFKP20, KPY20, LV20, JKKZ21], which in turn implies the hardness of **PPAD**. These works build such proof systems (and therefby establish hardness of **PPAD**) based on (1) the hardness of breaking the Fiat-Shamir heuristic [CHK+19a, CHK+19b, EFKP20], (2) the subexponential hardness of both iterated squaring (IS) and learning with errors (LWE) [LV20], (3) the subexponential hardness only of LWE [JKKZ21], or (4) the superpolynomial hardness of a problem about bilinear groups [KPY20] along with the exponential-time hypothesis (ETH).

  Unfortunately, none of these results achieve what we required above: a polynomial-time reduction from breaking cryptography (in polynomial time) to **PPAD**. In particular, these results leave open the possibility that there is a polynomial-time algorithm for **PPAD** and yet *all* of these problems are hard in the standard cryptographic sense.

- **Obfustopia:** Another sequence of works [BPR15, GPS16, HY17] show that **PPAD** is hard in "obfustopia", which is a world where indistinguishability obfuscation [BGI+01, GGH+13] and functional encryption [BSW11, O'N10] exist. Unlike the previous approach, this line of work *is* capable of relying on polynomial hardness: in particular, [GPS16] showed that if **PPAD**

is easy, then functional encryption cannot exist. Combined with the groundbreaking results of [JLS21b, JLS21a], this in turn would imply that one of three seemingly hard problems[1] in cryptography must be easy.

While the results of [JLS21b, JLS21a] are based on well-founded assumptions, they have received less scrutiny than other cryptographic assumptions. Even more fundamentally, we do not want to base the hardness of such a central complexity class such as **PPAD** only on the conjunction of three specific hardness assumptions.

In our work, we ask whether it is possible for the *first* line of work – basing **PPAD**-hardness on unambiguous proof systems – to rely on standard, polynomial-time hardness assumptions.

1.1 Our Results

Our first result shows that (average-case) **PPAD** hardness follows from the polynomial-time hardness of iterated squaring in RSA groups and LWE. In fact, as showed in [HY17], the same techniques imply hardness in the sub-class **CLS** $\subseteq$ **PPAD** introduced in [DP11]. In Appendix B, we further strengthen these hardness results to the subclass **UEOPL** $\subseteq$ **CLS**, which is one of the lowest known sub-classes of **TFNP** [FGMS19].

Theorem 1.1 (Following Theorem 2.8 and Corollary 5.7, informally stated)**.** *If there exists a PPT algorithm that solves* **PPAD** *with non-negligible probability, then there exists a PPT algorithm that breaks either IS in RSA groups or LWE with non-negligible probability.*

Slightly more formally, for a complete problem P in **PPAD**, we construct a distribution $\mathcal{D}$ on instances of P with the following property: if there is a polynomial-time algorithm A such that $\mathsf{A}(x)$ is a solution to $P(x)$ with non-negligible probability when sampling $x \leftarrow \mathcal{D}$, then there is a polynomial-time algorithm B that solves IS or solves LWE with non-negligible probability.

Public-coin PPAD hardness. Our hardness result is actually slightly stronger than what is achieved by the obfustopia reductions. We show *public-coin* hardness of P: there is a sampling algorithm for $\mathcal{D}$ such that the existence of such a B is guaranteed *even if* A is given the random coins used in sampling x. To our knowledge, this is the first hardness result for publicly sampleable distributions in **PPAD**. Moreover, previous hardness results that were based on polynomially falsifiable assumptions [BPR15, GPS16, HY17] seem inherently limited to secret-coin hardness because their instance distributions contain obfuscated circuits (or functional encryption ciphertexts that simulate the functionality of an obfuscated circuit). We remark that our public-coin hardness result may be somewhat surprising because the IS problem in an RSA modulus does not itself have a public-coin sampler.

Unique VDFs from standard assumptions. Our techniques also yield new results for verifiable delay functions (VDFs) [BBBF18]. We construct VDFs with *unique* proofs, which we call unique VDFs, based on the standard LWE assumption and the standard sequential hardness assumption regarding IS.

[1] The problems, roughly, are to break an SXDH assumption, to break a large-field LPN assumption, and to break a low-depth PRG.

Theorem 1.2 (Theorem 6.3, informally stated)**.** *If IS in RSA group is sequentially-hard and LWE is polynomially-hard, then there exists a unique VDF.*

Ours is the first construction of unique VDFs that is based on a polynomial hardness assumption. Recently, Freitag, Pass and Sirkin [FPS22], constructed VDFs from polynomial hardness of LWE and *any* sequentially-hard function, but it does not satisfy uniqueness. We view it as an interesting question whether such VDFs have applications in cryptography.

The building block. Along the way (as in previous work) we construct an unambiguous, incremental, succinct non-interactive argument system for IS. This serves as the building block for all our results stated above. The soundness of our argument system is based on LWE, and is established by instantiating the Fiat-Shamir heuristic applied to a variant of Pietrzak's interactive proof system for IS. We also formulate an abstract protocol template (that we call "outline-and-batch" protocols) that generically implies **PPAD**-hardness and captures essentially all existing results as well as our new protocol.

1.2 Technical Overview

Toward the construction of hard **PPAD** instances, we resort to a common paradigm in the literature, that of constructing *mergeable and unambiguous proofs* [CHK+19a, CHK+19b, EFKP20, KPY20, JKKZ21, LV20]. In this paradigm, we consider some underlying computation:

$$x_1 \to x_2 \to \cdots \to x_T \ ,$$

where each step $x_t \to x_{t+1}$, $1 \le t < T$, is computable in fixed polynomial time, but computing the last state x_T cannot be done efficiently for large (super-polynomial) T. For concreteness, the reader may think of iterated squaring over the RSA group $\mathbb{Z}_N^\times$ where, for a (randomly sampled) $g \in \mathbb{Z}_N^\times$, $x_t := g^{2^t} \bmod N$; note that computing $x_t \to x_{t+1}$ can be carried out by one modular squaring, but computing x_T for a large T is believed to be infeasible [RSW96]. For $1 \le t < t' \le T$, the corresponding proof system should allow computing (non-interactive) proofs $\pi_{t\to t'}$ for statements of the form $x_t \to x_{t'}$ (i.e., the state $x_{t'}$ is reachable from state x_t) and should satisfy the following requirements:

1. **Soundness:** it should be computationally hard to prove false statements.
2. **Unambiguity:** for any (true) statement $x_t \to x_{t'}$, it should be computationally hard to find any accepting proof $\pi^*_{t\to t'}$ other than the "prescribed" proof $\pi_{t\to t'}$ computed by the efficient merging process.
3. **Recursive proof-merging:** given d proofs $\pi_{1\to t}, \pi_{t\to 2t}, \ldots, \pi_{(d-1)t\to dt}$, for statements

   $$x_1 \to x_t, x_t \to x_{2t}, \ldots, x_{(d-1)t} \to x_{dt},$$

   computing a proof $\pi_{1\to dt}$ for the statement $x_1 \to x_{dt}$, where $d \in \mathbb{N}$ is some fixed merging parameter, can be efficiently reduced to computing a single proof $\pi'_{1\to t}$ for some related statement $x'_1 \to x'_t$. In other words, the d proofs for statements of "size" t can be merged into a proof for a statement of "size" dt via a recursive call to compute an additional (related) proof of size t. In the concrete example of iterated squaring, the "size" of the statement corresponds to the number of modular squaring operations required to go from $x_t := g^{2^t}$ to $x_{t'} := g^{2^{t'}}$.

Mergeable, Unambigous Proofs from Iterated Squaring and Fiat-Shamir. As mentioned, the mergeable unambiguous proofs paradigm has by now several instantiations in the literature. Focusing on obtaining a polynomial reduction, we consider one particular instantiation, based on Pietrzak's protocol for the iterated squaring (IS) problem [Pie19]. The protocol is a public-coin interactive proof for statements of the form "g^{2^T} equals h modulo N", where N is a public modulus whose factorization is known to neither the prover nor the verifier – we denote such a statement by $g \xrightarrow{T} h$. At the heart of Pietrzak's protocol is a technique for reducing a statement $g \xrightarrow{T} h$ to a related, new statement $g' \xrightarrow{T/2} h'$ that is half the size.[2] This is done by having the (honest) prover specify an integer μ such that the intermediate statements $g \xrightarrow{T/2} \mu$ and $\mu \xrightarrow{T/2} h$ hold (i.e., μ is the "midpoint"), and then having the verifier *reduce* these two statements into one, using its random challenge r as follows:

$$g' := g^r \mu \bmod N \text{ and } h' := \mu^r h \bmod N.$$

The above, "halving sub-protocol" is repeated for $\log(T)$ rounds, at the end of which the verifier ends up with a statement of the form $g'' \xrightarrow{2} h''$, which it can, itself, check by modular squaring. To make this proof system non-interactive, previous works turn to the Fiat-Shamir paradigm [FS87] of applying an appropriate hash function to the statement to derive the verifier's challenge.

Instantiating Fiat-Shamir. Since Pietrzak's protocol has statistical soundness, the above approach already yields hard **PPAD** instances in the random oracle model [CHK+19b, EFKP20]. Our focus is of course on obtaining a construction without random oracles. Indeed, a recent surge of results has successfully instantiated Fiat-Shamir without random oracles in various scenarios [KRR17, CCRR18, HL18, CCH+19, PS19, BKM20, CPV20, CKU20, LV20, JJ21, HLR21, JKKZ21, CJJ21a, CJJ21b, HJKS22]. This has, in fact, also yielded hard **PPAD** instances, but so far none based on polynomial hardness assumptions. Especially relevant to us is the work of Lombardi and Vaikuntanathan [LV20] who instantiate the Fiat-Shamir transform for Pietrzak's protocol, based on *sub-exponential* hardness of LWE. At a high level, the sub-exponential loss in [LV20] comes from the difficulty of computing (or successfully guessing) the so called *bad verifier challenges* in the protocol — the precise quantitative complexity of this task turns out to crucially affect Fiat-Shamir instantiability. For the particular case of Pietrzak's protocol, a verifier challenge is bad if either of the intermediate statements $g \xrightarrow{T/2} \mu$ or $\mu \xrightarrow{T/2} h$ is false, but the new randomized statement $g' \xrightarrow{T/2} h'$ happens to be true. As a part of the soundness argument, it was demonstrated in [Pie19] that the set of bad verifier challenges consists of at most a few elements, but it turns out that computing them (even given the factorization of N) requires solving an intractable discrete-log problem (see, e.g., [LV20] for a discussion).

Reduced Challenge Space and Soundness Amplification. A first observation toward eliminating the subexponential loss is that bad challenges in Pietrzak's protocol are *efficiently verifiable* given the factorization of N. In particular, there is a straight-forward modification of Pietrzak's protocol that uses a polynomial size challenge space, which makes it trivial to find the bad challenges by enumerating and testing every possibility (which can be done efficiently given the above observation). However, this modification causes the protocol to have inverse polynomial soundness error, so the resulting protocol cannot be made interactive via Fiat-Shamir.

[2] Throughout this section, we assume for simplicity that the time parameter T is a power of 2.

A natural attempt to resolve this is to repeat the small-challenge protocol many times in parallel to reduce the soundness error. Indeed parallel repetition reduces the soundness error and importantly, using a recent work of Holmgren, Lombardi and Rothblum [HLR21], we can even instantiate Fiat-Shamir for such a protocol based on (polynomially secure) LWE.[3] Their instantiation essentially works for any parallel-repeated three-message proof, as long as the bad challenges in each individual copy of the protocol are efficiently verifiable. (It also works for protocols with more rounds provided a certain round-by-round soundness requirement that is satisfied by Pietrzak's protocol, further discussed below). It turns out, however, that this approach falls short of our goal. The issue is that *the resulting proofs are not unambiguous*. As noted in [RRR16], while parallel repetition amplifies soundness, it *does not* amplify unambiguity. The reason is that a cheating prover that breaks unambiguity in a *single* copy of the base protocol (out of many), can in particular obtain two accepting proofs for the same statement, breaking the unambiguity of the whole protocol.

Amplifying Unambiguity. As described above, while parallel repetition has the desired effect on soundness, unambiguity suffers from a *single point of failure*: that is, it suffices to cheat in a single copy of the base protocol without affecting the other copies. Instead we would like to start with a protocol that morally still works with many copies (as in parallel repetition) but mixes these together so that any deviations propagate across the entire protocol. Indeed such a protocol was constructed by Block et al. [BHR+21], who construct an interactive proof system for IS for a completely different purpose than considered here.[4] Specifically, for an arbitrary group $\mathbb{G}$, they consider λ (possibly-identical) statements

$$\left(g_1 \xrightarrow{T} h_1, \cdots, g_\lambda \xrightarrow{T} h_\lambda\right), \tag{1}$$

where (in the honest case) $h_i = g_i^{2^T}$ over $\mathbb{G}$ for all $i \in [1, \lambda]$. As in Pietrzak's protocol, the prover sends over a tuple of midpoints $(\mu_1, \cdots, \mu_\lambda)$, for claimed values $\mu_i = g_i^{2^{T/2}}$. This results in 2λ intermediate statements of the form

$$\left(g_1 \xrightarrow{T/2} \mu_1, \mu_1 \xrightarrow{T/2} h_1, \cdots, g_\lambda \xrightarrow{T/2} \mu_\lambda, \mu_\lambda \xrightarrow{T/2} h_\lambda\right),$$

which we rewrite as

$$\left(\widetilde{g}_1 \xrightarrow{T/2} \widetilde{h}_1, \cdots, \widetilde{g}_{2\lambda} \xrightarrow{T/2} \widetilde{h}_{2\lambda}\right). \tag{2}$$

To recurse, λ new statements are derived by a 2λ-to-λ (batch) reduction, where the i-th new statement $g'_i \xrightarrow{T/2} h'_i$ is constructed by choosing a random subset S_i of the 2λ statements as follows:

$$g'_i = \prod_{j \in S_i} \widetilde{g}_j \text{ and } \; h'_i = \prod_{j \in S_i} \widetilde{h}_j. \tag{3}$$

Even if a single original statement in Eq. (1) is false, it was shown in [BHR+21] that each new statement in Eq. (3) is true with probability at most $1/2$ (over the choice of S_i) : intuitively, in the

[3] In fact, this yields a (non-unique) VDF based on the standard hardness of IS and LWE. However, this is subsumed by the result from [FPS22] mentioned in Section 1.1. In Section 1.2.1, we will construct *unique* VDF from same assumptions.

[4] The goal in [BHR+21] (also see [HHK+22]) was to construct a statistically-sound *proof of exponentiation* that works for IS in *arbitrary* groups. In comparison, Pietrzak's protocol is statistically-sound only in groups that are guaranteed to have no low-order elements, e.g., in the group of *signed quadratic residues* [FS00, HK09].

(worst) case that the j^*-th statement is the only false statement in Eq. (1), then it is included in Eq. (4) with probability $1/2$, rendering the new statement false. Since there are λ new statements, constructed using *independent* random subsets, the soundness of the resulting protocol is $1/2^\lambda$. Unambiguity amplifies in an identical manner: a cheating prover deviating from the prescribed honest prover strategy affects each new statement with probability roughly $1/2$, "propagating" false statements and, as a result, circumventing the issue of a single point of failure we had previously discussed. By recursing, as above, $\log(T)$ times, the statement reduces to a statement which can be efficiently checked by the verifier.

Applying [HLR21]. Now that we have solved the issues with unambiguity in the interactive protocol, we would like to make it non-interactive in the common reference string (CRS) model by applying the Fiat-Shamir transform. Briefly, the Fiat-Shamir transform for any *public-coin* interactive proof is defined with respect to some hash function family $\mathcal{H}$, where a single hash function H sampled from this family is set to be the CRS. The round-collapse is due to the fact that the verifier's message for each round is simply computed to be the output of H applied to the transcript of the protocol up to that point. The security of the instantiated transform relies on correlation intractability of hash functions for *bad challenges* [CGH98]. This is, in particular, true for random oracles when the bad challenges are "sparse".

As already stated, the Fiat-Shamir transform has been successfully instantiated based on standard assumptions for several protocols. Of particular interest to our work is a recent work of Holmgren, Lombardi and Rothblum [HLR21]. We illustrate their idea directly for the [BHR+21] protocol. Consider the 2λ intermediate statements from Eq. (2) and let $j^* \in [1, 2\lambda]$ be an index such that $\widetilde{g}_{j^*}^{2^{T/2}} \neq \widetilde{h}_{j^*}$. This can occur either due to the fact that one of the initial λ statements was incorrect, or a cheating prover deviated from the prescribed prover strategy. Then, the i-th new statement $g'_i \xrightarrow{T/2} h'_i$ is true if and only if

$$\prod_{j \in S_i} (\widetilde{g}_j)^{2^{T/2}} = \prod_{j \in S_i} \widetilde{h}_j. \tag{4}$$

Recall that the above only happens with probability at most $1/2$ and, consequently, the probability that at least one of the λ new statements is false is $1 - 1/2^\lambda$. We can now define the set of bad challenges, i.e., the *bad set* $\mathcal{B}$, that results in *all* λ new statements to be true. To be precise,

$$\mathcal{B} = \mathcal{B}_{(\tilde{g}_1, \cdots, \tilde{g}_{2\lambda}), (\tilde{h}_1, \cdots, \tilde{h}_{2\lambda}), T/2} := \left\{ S_1, \cdots, S_\lambda \subseteq [1, 2\lambda] \middle| \prod_{j \in S_i} (\tilde{g}_j)^{2^{T/2}} = \prod_{j \in S_i} \tilde{h}_j \text{ for all } i \in [1, \lambda] \right\}.$$

Note that $\mathcal{B}$[5] can be represented as the product of λ sets, i.e.

$$\mathcal{B} = \mathcal{B}_1 \times \cdots \times \mathcal{B}_\lambda, \tag{5}$$

where each

$$\mathcal{B}_i := \{S_i \subseteq [1, 2\lambda] \mid \prod_{j \in S_i} (\widetilde{g}_j)^{2^{T/2}} = \prod_{j \in S_i} \widetilde{h}_j\}. \tag{6}$$

This *product structure* of $\mathcal{B}$ shown in Eq. (5) is crucial for us to invoke [HLR21] who show, assuming polynomial hardness of LWE, that there exists a hash function family $\mathcal{H}$ such that the Fiat-Shamir

[5] We drop the subscript for the $\mathcal{B}$ set for clarity when the subscript is clear from the context.

transform is sound whenever $\mathcal{B}$ is a product set such that each $\mathcal{B}_i$ is *efficiently verifiable*.[6] Here the set $\mathcal{B}_i$ is said to be efficiently verifiable if there is a polynomial-sized circuit C that on input $((\tilde{g}_1, \cdots, \tilde{g}_{2\lambda}), (\tilde{h}_1, \cdots, \tilde{h}_{2\lambda}), i, S_i)$ that decides whether $S_i \in \mathcal{B}_i$. In our setting, C needs to check whether Eq. (4) holds, which can be done efficiently if C could compute the product $\prod_{j \in S_i} (\widetilde{g}_j)^{2^{T/2}}$ in Eq. (6), even for super-polynomial T. This is possible, for instance, in any group of the form $\mathbb{Z}_N^\times$ (including RSA groups) if C has a *trapdoor*, viz., the factorization of the modulus N, hardcoded in its description: C can first compute the intermediate value $e := 2^{T/2} \bmod \phi(N)$ using the trapdoor and then compute $g^e \bmod N$ by a single modular exponentiation. Thus, as long as we work in groups where one can efficiently verify each $\mathcal{B}_i$ (with the help of a trapdoor), the Fiat-Shamir transform applied to the [BHR+21] protocol is a secure non-interactive argument in the CRS model.

Additionally, in order for the resulting non-interactive argument to preserve properties of the multi-round unambiguous interactive proof, the protocol needs to satisfy the stronger property [CCH+18, LV20] of *unambiguous round-by-round soundness*. In the technical section, we show that the soundness and unambiguity discussion of [BHR+21] earlier easily extend to satisfy this property.

1.2.1 Application to Unique VDFs

Having constructed an unambiguous (succinct) non-interactive argument system for IS, we essentially immediately obtain a VDF family with *unique* proofs based on (1) the polynomial hardness of LWE, and (2) the assumption that IS is an inherently sequential function. The only detail that needs to be verified is that the computational complexity of the prover is $T \cdot (1 + o(1))$ for T sequential squarings. This can be proved following an analogous argument in [Pie19]: after applying $T + 1$ sequential squaring operations

$$g_0 = g, g_1 = g^2, \ldots, g_T = g^{2^T},$$

it is possible to compute all prover messages with $\mathsf{poly}(\lambda) \cdot \sqrt{T}$ additional group operations as follows.

- Compute all prover messages from round $\frac{1}{2}\log(T)$ onwards with the naive prover algorithm, incurring an additive computational overhead of $\mathsf{poly}(\lambda) \cdot \sqrt{T}$, and
- Compute all prover messages in the first $\frac{1}{2}\log(T)$ rounds by storing $\sqrt{T}$ of the computed g_is, where each prover message is computed as product-combinations of a (pre-determined) subset of these stored values. This incurs a *total* additive overhead of $\mathsf{poly}(\lambda) \cdot \sqrt{T}$.

Remark 1 (Comparison to the [LV20] VDF). The [LV20] VDF uses complexity leveraging in a way so that the honest prover is only efficient (relative to the squaring computation) when the squaring parameter T is subexponentially large in the description of the RSA modulus. Relatedly, the protocol then only achieves a slightly superpolynomial gap between the complexity of the honest prover and the complexity of the cheating provers ruled out by soundness. In contrast, our construction does not require complexity leveraging, resulting in a VDF with far more standard efficiency parameters.

[6] We refer the reader to the technical section for full details on invoking [HLR21].

1.2.2 Applications to PPAD-Hardness

For establishing hardness of **PPAD**, we have to show that the non-interactive argument obtained above satisfies the third requirement, i.e., recursive proof-merging. The two sets of intermediate statements from Eq. (2) can be succinctly denoted as

$$(g_1, \cdots, g_\lambda) \xrightarrow{T/2} (\mu_1, \cdots, \mu_\lambda) \text{ and } (\mu_1, \cdots, \mu_\lambda) \xrightarrow{T/2} (h_1, \cdots, h_\lambda) \tag{7}$$

with corresponding [BHR+21] proofs

$$\pi((g_1, \cdots, g_\lambda) \xrightarrow{T/2} (\mu_1, \cdots, \mu_\lambda)) \text{ and } \pi((\mu_1, \cdots, \mu_\lambda) \xrightarrow{T/2} (h_1, \cdots, h_\lambda)).$$

The proof for $(g_1, \cdots, g_\lambda) \xrightarrow{T} (h_1, \cdots, h_\lambda)$ can be computed as

$$\pi((g_1, \cdots, g_\lambda) \xrightarrow{T} (h_1, \cdots, h_\lambda)) := \Big((\mu_1, \cdots, \mu_\lambda), \pi((g'_1, \cdots, g'_\lambda) \xrightarrow{T/2} (h'_1, \cdots, h'_\lambda))\Big)$$

where $(g'_1, \cdots, g'_\lambda) \xrightarrow{T/2} (h'_1, \cdots, h'_\lambda)$ is derived via the 2λ-to-λ (batch) reduction from the statements in Eq. (7). Furthermore, the proof

$$\pi((g'_1, \cdots, g'_\lambda) \xrightarrow{T/2} (h'_1, \cdots, h'_\lambda))$$

is generated by recursing on the statement $(g'_1, \cdots, g'_\lambda) \xrightarrow{T/2} (h'_1, \cdots, h'_\lambda)$ to compute its proof. Since the reduction is efficient, the non-interactive argument satisfies recursive proof merging as desired. As shown in [CHK+19a], this actually implies hardness of the sub-class **CLS** $\subseteq$ **PPAD**. We strengthen this result further in Appendix B to show hardness in **UEOPL** $\subseteq$ **CLS**, one of the lowest-lying sub-classes of **TFNP** [FGMS19].

Remark 2 (Abstract protocol). While we have limited our discussion specifically to the case of IS, in the technical sections (Sections 3 and 5) we describe an abstract protocol template that we call "outline and batch." We show that any problem family admitting a *downward self-reduction* and a *(randomized) batching reduction* (reducing k' instances of the problem to sufficiently fewer $k < k'$ instances) admits an unambiguous and incremental non-interactive argument system that suffices for our hardness results. We refer the reader to the technical sections for details.

Obtaining Public-Coin Hardness in PPAD. Finally, we discuss how to obtain hard distributions of **PPAD** instances that are *publicly samplable* under the same computational assumptions as before: the polynomial hardness of LWE and IS over RSA group. It is a priori unclear why one should expect to obtain public-coin hardness under these assumptions, since we don't know a public-coin algorithm for sampling an RSA modulus! Nevertheless, we obtain the result via the following two ideas.

First, we observe that our Fiat-Shamir hash function $\mathcal{H}$ can be sampled from a public-coin distribution. In [HLR21], the hash functions have a *computationally pseudorandom* (and private-coin) description, but they can be switched to uniformly random because even the adaptive unambiguous soundness of the protocol considered in our work is an efficiently verifiable property (given the group order as a trapdoor). Put another way, the adaptive soundness of the protocol follows from an efficiently falsifiable form of correlation intractability, which is thus preserved under computational indistinguishability.

The more serious issue is how to handle the *group* (and group element) description. We handle this by working over $\mathbb{Z}_N^\times$ for a *different* value of N (rather than an RSA modulus). A naive idea would be to work over a *uniformly random modulus* N; unfortunately, the squaring problem mod a uniformly random N is not hard, because N will be prime with inverse polynomial probability (by the prime number theorem), in which case the group order $\phi(N) = N - 1$ is efficiently computable. Our actual solution is as follows: consider $N = N_1 \cdot \ldots \cdot N_{\mathsf{poly}(\lambda)}$ for a sufficiently large $\mathsf{poly}(\lambda)$, where all integers N_i are public and uniformly random in the range $[1, 2^\lambda]$. First of all, we note that our techniques for constructing hard **PPAD** instances from iterated squaring apply to this choice of modulus as well: all that is required is that there is a way to efficiently sample (necessarily using *secret coins*) the squaring problem description along with a trapdoor containing the group order $|\mathbb{Z}_N^\times| = \phi(N)$ (this is captured by our generic construction). This is possible using efficient algorithms for generating random factored integers [Bac88, Kal03].

This tells us that public-coin hardness in **PPAD** follows from the hardness of LWE along with the polynomial hardness of IS modulo N (given the coins for sampling the IS instance). To complete the proof, we show that this follows from the polynomial hardness of (secret-coin) IS in an RSA modulus. We prove this (see Lemma 4.5) by a direct reduction that embeds an RSA modulus IS problem instance into a public-coin instance of this new IS problem; crucially, we use the fact that with all but negligible probability over $N_1, \ldots, N_{\mathsf{poly}(\lambda)}$, at least one N_i is an RSA modulus.

1.3 Organisation

We state definitions and provide background relevant to our paper in Section 2. In Section 3, we describe the abstract "outline-and-batch" interactive protocol, prove that it is an unambiguously sound proof system and then explain how existing protocols fit this abstraction. We also show that, under certain conditions, applying the Fiat-Shamir transform results in a unambiguous sound non-interactive argument system. In Section 4, we describe the unambiguous non-interactive argument for IS that forms the basis of the results in Sections 5 and 6. We show hardness of the class **PPAD** in Section 5 by constructing hard distribution of a search problem called RSVL. In Section 6, we construct unique VDFs. The definitions of **TFNP** classes relevant to the paper are given in Appendix A. Finally, in Appendix B, we extend the results in Section 5 to show hardness of the class **UEOPL** $\subseteq$ **PPAD**.

2 Preliminaries

Notation. First, we list the notation that will be used throughout this paper.

1. For $a, b \in \mathbb{N}$, $a < b$, by $[a, b]$ we denote the sequence of integers $\{a, a+1, \cdots, b\}$.
2. For an alphabet Σ and $n \in \mathbb{N}$, we write Σ^n, $\Sigma^{<n}$ and $\Sigma^{\leq n}$ to denote, respectively, strings over Σ with length equal to, less than, and less than or equal to n. We use ε to denote the empty string. For strings a and b we use ab to denote string concatenation.
3. Vectors and tuples are in bold face. We parse a vector or a tuple $\boldsymbol{x} \in \mathcal{X}^k$ as $\boldsymbol{x} =: (x_0, \cdots, x_{k-1})$; $\boldsymbol{x}$ is said to be a k-vector. A subscripted vector $\boldsymbol{x}_v \in \mathcal{X}^k$ is parsed as $\boldsymbol{x}_v =: (x_{v,0}, \cdots, x_{v,k-1})$.
4. For $x \in \mathcal{X}$, $y \in \mathcal{Y}$ and a function $f : \mathcal{X} \to \mathcal{Y}$, we write $x \xrightarrow{f} y$ to denote the (true or false) *statement* "y equals $f(x)$". Sometimes, when the context is clear, we will simplify the

notation: e.g., for $h := g^{2^T} \bmod N$, we simply write $g \xrightarrow{T} h$ to denote

$$g \xrightarrow{(\cdot)^{2^T} \bmod N} h.$$

We extend this notation to vectors: for $\boldsymbol{x} \in \mathcal{X}^k$ and $\boldsymbol{y} \in \mathcal{Y}^k$, we define $\boldsymbol{f} = f^k : \mathcal{X}^k \to \mathcal{Y}^k$ as $(f(x_0), \cdots, f(x_{k-1}))$ and therefore $\boldsymbol{x} \xrightarrow{\boldsymbol{f}} \boldsymbol{y}$ denotes statement that $x_i \xrightarrow{f} y_i$ for all $i \in [0, k-1]$.

5. For a statement x, we denote $\pi(x)$ to denote a non-interactive proof for x. For example, for x, y and f as in Item 4, we write $\pi(x \xrightarrow{f} y)$ to denote a non-interactive proof for the statement $x \xrightarrow{f} y$.

6. For $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, we write $y := \mathsf{A}(x)$ (resp., $y \leftarrow \mathsf{A}(x)$) to denote the execution of a deterministic (resp., randomised) algorithm A on input x to output y. For $k \in \mathbb{N}$, vectors $\boldsymbol{x} \in \mathcal{X}^k$ and $\boldsymbol{y} \in \mathcal{Y}^k$, we denote repeated *parallel* execution of A by $\boldsymbol{y} := \mathbf{A}(\boldsymbol{x})$, i.e., $y_i := \mathsf{A}(x_i)$ for all $i \in [0, k-1]$.

2.1 Search Problems, TFNP, and Reductions

We define below search problems, and the relevant complexity classes needed for our work. We start by defining search problems.

Definition 2.1 (Search Problems [AB09])**.** A search problem is a relation $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$. Let $\mathcal{R}(x)$ denote $\{y : (x,y) \in \mathcal{R}\}$. A function $f : \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ is said to solve $\mathcal{R}$ if for every $x \in \{0,1\}^*$ satisfying $\mathcal{R}(x) \neq \emptyset$, it holds that $f(x) \in \mathcal{R}(x)$; and for all other x, $f(x) = \perp$.

Definition 2.2 (Total Relations)**.** A relation $\mathcal{R}$ is said to be total if for all $x \in \{0,1\}^*$, there exists y such that $(x,y) \in \mathcal{R}$.

Definition 2.3 (Polynomially Balanced)**.** A relation $\mathcal{R}$ is said to be polynomially balanced if there is a polynomial p such that for any strings $x, y \in \{0,1\}^*$, if $(x,y) \in \mathcal{R}$ then $|y| \leq p(|x|)$.

Definition 2.4 (**FNP**)**.** The complexity class **FNP** consists of all polynomially balanced search problems $\mathcal{R}$ for which there is a polynomial-time algorithm that on input (x,y) outputs whether or not $(x,y) \in \mathcal{R}$.

Definition 2.5 (**TFNP**)**.** The complexity class **TFNP** consists of all total search problems in **FNP**.

The sub-classes of **TFNP** that are relevant to the paper are defined in Appendices A and B.

Definition 2.6 (Reductions)**.** If P and Q are search problems, a randomized Karp reduction from P to Q with error $\epsilon(\cdot)$ is a pair of p.p.t. machines (M, N) such that if f is a function that solves Q, then for any $x \in \{0,1\}^n$ with $P(x) \neq \emptyset$, we have

$$\Pr\left[(x,y) \in P\right] \geq 1 - \epsilon(n)$$

when sampling $x' \leftarrow \mathsf{M}(x)$, $y \leftarrow \mathsf{N}\left(f(x')\right)$.

Next, we consider the search problem RelaxedSinkOfVerifiableLine (RSVL), which is relevant to the main result of this paper. We point out that RSVL *not* a total problem since, looking ahead, there is no way to syntactically guarantee that the successor and verifier circuits are well-behaved (see Remark 3).

Definition 2.7 ([CHK+19a]). RelaxedSinkOfVerifiableLine (RSVL)

- *Instance.*
  1. Boolean circuit $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$
  2. Boolean circuit $\mathsf{V} : \{0,1\}^m \times [0, 2^m - 1] \to \{\mathsf{accept}, \mathsf{reject}\}$
  3. Integer $L \in [0, 2^m - 1]$
  4. String $v_0 \in \{0,1\}^m$
- *Promise.* For every $v \in \{0,1\}^m$ and $i \in [0, 2^m - 1]$, $\mathsf{V}(v, i) = 1$ *if* $i \leq L$ and $v = \mathsf{S}^i(v_0)$.
- *Solution.* One of the following:
  1. **The sink:** a vertex $v \in \{0,1\}^m$ such that $\mathsf{V}(v, L) = 1$; or
  2. **False positive:** a pair $(v, i) \in \{0,1\}^m \times [0, 2^m - 1]$ such that $v \neq \mathsf{S}^i(v_0)$ and $\mathsf{V}(v, i) = 1$.

Remark 3. It seems likely that RSVL is not in **FNP**, let alone in **PPAD** (Definition A.2). Specifically, checking that a pair (v, i) constitutes a false positive is difficult because i may be super-polynomial in the instance size.

Nevertheless, [CHK+19a] constructed a (randomized) reduction from RSVL to EOML (which is a search problem complete for **CLS** $\subseteq$ **PPAD**, see Appendix A) with error that is inversely polynomially bounded away from 1. This error is somewhat large, and allows for the possibility EOML is "slightly" easier than RSVL.

Still, the reduction suffices for establishing the standard cryptographic hardness of EOML (i.e. that no polynomially bounded algorithm can succeed with *any* non-negligible probability) based on analogous hardness for RSVL. In turn, we establish the latter hardness based on LWE (Assumption 2.10) and the iterated squaring assumption (Assumption 4.3).

Theorem 2.8 ([CHK+19a]). *There is a randomized Karp reduction from RSVL to EOML with error probability* $\epsilon(n) = 1 - n^{-O(1)}$.

2.2 Learning with Errors

The following standard preliminaries about the Learning with Errors (LWE) problem are based on [Pei16, LV20].

Definition 2.9 (LWE Distribution). For any $\mathbf{s} \in \mathbb{Z}_q^n$ and any distribution $\chi \subseteq \mathbb{Z}_q$, the LWE distribution $A_{\mathbf{s},\chi} \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ is sampled by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, sampling $e \leftarrow \chi$, and outputting $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$.

Assumption 2.10 (Decision LWE). Let $m = m(n) \geq 1$, $q = q(n) \geq 2$ be integers, and let $\chi(n)$ be a probability distribution on $\mathbb{Z}_{q(n)}$. The $\mathsf{LWE}_{n,m,q,\chi}$ problem, parameterized by n, is to distinguish whether $m(n)$ independent samples are drawn from $A_{\mathbf{s},\chi}$ (for $\mathbf{s}$ that is sampled uniformly at random) or are drawn from the uniform distribution. The hardness assumption is that is hard for $\mathsf{poly}(n)$-sized adversaries to decide the $\mathsf{LWE}_{n,m,q,\chi}$ problem.

2.3 Correlation-Intractable Hash Families

The following preliminaries are partially taken from [LV20, HLR21].

Definition 2.11 (Hash family)**.** For a pair of efficiently-computable functions $(n(\cdot), m(\cdot))$, a hash family with input length n and output length m is a collection $\mathcal{H} = \{H_\lambda : \{0,1\}^{s(\lambda)} \times \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}\}_{\lambda \in \mathbb{N}}$ of keyed hash functions, along with a pair of p.p.t. algorithms:

- $\mathcal{H}.\mathsf{Gen}(1^\lambda)$ outputs a hash key $k \in \{0,1\}^{s(\lambda)}$.
- $\mathcal{H}.\mathsf{Hash}(k, x)$ computes the function $H_\lambda(k, x)$. We may use the notation $H(k, x)$ to denote hash evaluation when the hash family is clear from context.

As in prior works [CCH+19, PS19] we consider the security notion of correlation intractability [CGH98] for single-input relations and its restriction to (single-input) functions.

Definition 2.12 (Correlation Intractability)**.** For a given relation ensemble $\mathcal{R} = \{\mathcal{R}_\lambda \subseteq \{0,1\}^{n(\lambda)} \times \{0,1\}^{m(\lambda)}\}$, a hash family $\mathcal{H} = \{H_\lambda : \{0,1\}^{s(\lambda)} \times \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}\}$ is said to be $\mathcal{R}$-correlation intractable with security (s, δ) if for every s-size $\mathsf{A} = \{\mathsf{A}_\lambda\}$,

$$\Pr_{\substack{k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda) \\ x \leftarrow \mathsf{A}(k)}} \Big[(x, H(k, x)) \in \mathcal{R} \Big] = O(\delta(\lambda)).$$

We say that $\mathcal{H}$ is $\mathcal{R}$-correlation intractable with security δ if it is (λ^c, δ)-correlation intractable for all $c > 1$. Finally, we say that $\mathcal{H}$ is $\mathcal{R}$-correlation intractable if it is $(\lambda^c, 1/\lambda^c)$-correlation intractable for all $c > 1$.

We will use the recent result of [HLR21] on correlation intractability for *product relations*.

Definition 2.13 (Product Relation)**.** We say that $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}^t$ is a product relation if for every $x \in \mathcal{X}$, the set $\mathcal{R}(x) = \{y : (x, y) \in \mathcal{R}\} \subseteq \mathcal{Y}^t$ has a decomposition

$$\mathcal{R}(x) := \mathcal{B}_1(x) \times \mathcal{B}_2(x) \times \ldots \mathcal{B}_t(x)$$

(where each $\mathcal{B}_i(x)$ is a subset of $\mathcal{Y}$). We say that such an $\mathcal{R}$ is efficiently product verifiable if for some such choice of $\mathcal{B}_i$, there is a poly-size circuit $\mathsf{C}(x, i, y_i)$ that decides whether $y_i \in \mathcal{B}_i(x)$.

Theorem 2.14 ([HLR21])**.** *Assume the hardness of* LWE*. Then, for every size bound $S(\lambda) = \mathsf{poly}(\lambda)$, input length $n(\lambda)$, and output length $m(\lambda) \cdot t(\lambda)$ such that $t(\lambda) \geq \lambda^{\Omega(1)}$, there exists a correlation intractable hash family $\mathcal{H}$ for product relations $\mathcal{R}$ that are (1) product verifiable by size $S(\lambda)$ circuits, and (2) sparse in the sense that for every x, i, we have that $|\mathcal{B}_i(x)| \leq \frac{1}{2} \cdot 2^{m(\lambda)}$.*

Remark 4. In [HLR21], hash function keys have a computationally pseudorandom distribution. However, for the purposes of Theorem 2.14, hash function keys may be taken to be uniformly random strings (by invoking the pseudorandomness property), because the security property in Theorem 2.14 is efficiently falsifiable.

2.4 Interactive Proofs and the Fiat-Shamir Heuristic

The following preliminaries are partially taken from [LV20, HLR21]. We begin by recalling the definitions of interactive proofs and arguments.

Definition 2.15 (Interactive proof and argument system)**.** An interactive proof (resp., interactive argument) for a promise problem $\mathcal{L} = (\mathcal{L}_{\text{YES}}, \mathcal{L}_{\text{NO}})$ is a pair (P, V) of interactive algorithms satisfying:

- ***Completeness.*** For any $x \in \mathcal{L}_{\text{YES}}$, when P and V interact on common input x, the verifier V outputs 1 with probability 1.
- ***Soundness.*** For any $x \in \mathcal{L}_{\text{NO}} \cap \{0,1\}^n$ and any *unbounded* (resp., *polynomial-time*) interactive P^*, when P^* and $\mathsf{V}(x)$ interact, the probability that V outputs 1 is a negligible function of n.

The protocol is public-coin if each of V's messages is an independent uniformly random string of some length (and the verifier's decision to accept or reject does not use any secret state). In this setting, we will denote prover messages by $(\alpha_1, \ldots, \alpha_\ell)$ and verifier messages by $(\beta_1, \ldots, \beta_{\ell-1})$ in a $2\ell - 1$-round protocol.

Definition 2.16 (Non-interactive argument system)**.** A non-interactive argument scheme (in the CRS model) for a promise problem $\mathcal{L} = (\mathcal{L}_{\text{YES}}, \mathcal{L}_{\text{NO}})$ is a triple $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ of *non-interactive* algorithms with the following properties:

- $\mathsf{Setup}(1^n)$ outputs a common reference string CRS.
- $\mathsf{P}(\mathsf{CRS}, x)$ outputs a proof π.
- $\mathsf{V}(\mathsf{CRS}, x, \pi)$ outputs a bit $b \in \{0,1\}$

It satisfies the notions of completeness and (computational) soundness as above.

We next define the notion of *unambiguous soundness* [RRR16]. For non-interactive arguments, the soundness notion we consider is *adaptive* in that we allow the prover P^* to adaptively choose the statement x after seeing the CRS.

Definition 2.17 (Unambiguous Soundness [RRR16, CHK+19a])**.** A public-coin interactive proof system Π is *unambiguously sound* if (1) it is sound, and (2) for every $x \in \mathcal{L}$ and every (complete) collection of verifier messages $(\beta_1, \ldots, \beta_{\ell-1})$, there exists a distinguished proof $\pi^*(x, \beta_1, \ldots, \beta_{\ell-1})$ such that the following soundness condition holds: For all $x \in \mathcal{L}$ and all cheating provers P^*, the probability that the transcript $\langle \mathsf{P}^*(x), \mathsf{V}(x) \rangle$ contains a proof π such that $\mathsf{V}(x, \pi) = 1$ and $\pi \neq \pi^*(x, \beta_1, \ldots, \beta_{\ell-1})$ is negligible.

Definition 2.18 (Adaptive Unambiguous Soundness)**.** A non-interactive argument system $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ is *adaptively unambiguously sound* against (uniform or non-uniform) time T adversaries if for all instances $x \in \mathcal{L}$ and all common reference strings CRS, there exists a "distinguished proof" $\pi^*(\mathsf{CRS}, x)$ such that the following soundness condition holds: For all time T cheating provers P^*, the probability that $\mathsf{P}^*(\mathsf{CRS}) = (x, \pi)$ where $\mathsf{V}(x, \pi) = 1$ and *either* $x \notin \mathcal{L}$ or $\pi \neq \pi^*(\mathsf{CRS}, x)$ is negligible.

Our results proceed by constructing (unambiguously sound) interactive proof systems and compiling them into non-interactive argument systems using the Fiat-Shamir transform, which we describe next.

Definition 2.19 (Fiat-Shamir Transform)**.** Let Π denote a public coin interactive proof (or argument) system Π that has ℓ prover messages and $\ell - 1$ verifier messages of length $m = m(\lambda)$. Then, for a hash family

$$\mathcal{H} = \left\{ \left\{ H_k : \{0,1\}^* \to \{0,1\}^{m(\lambda)} \right\}_{k \in \{0,1\}^\lambda} \right\}_\lambda,$$

we define the Fiat-Shamir non-interactive protocol $\Pi_{\mathrm{FS},\mathcal{H}} = (\mathsf{Setup}, \mathsf{P}_{\mathrm{FS}}, \mathsf{V}_{\mathrm{FS}})$ as follows:

- $\mathsf{Setup}(1^\lambda)$: sample a hash key $k \leftarrow \mathcal{H}.\mathsf{Gen}(1^\lambda)$.
- $\mathsf{P}_{\mathrm{FS}}(x)$: for $i \in \{1, \ldots, \ell\}$, recursively compute the following pairs (α_i, β_i):
  - Compute $\alpha_i = \mathsf{P}(\tau_i)$ for $\tau_i = (x, \alpha_1, \beta_1, \ldots, \alpha_{i-1}, \beta_{i-1})$.
  - Compute $\beta_i = H_k(\tau_{i-1}, \alpha_i)$.

  Then, $\mathsf{P}_{\mathrm{FS}}(x)$ outputs $\pi = (\alpha_1, \beta_1, \ldots, \alpha_\ell)$.
- $\mathsf{V}_{\mathrm{FS}}(\mathsf{CRS}, x, \pi)$ parses $\pi = (\alpha_1, \beta_1, \ldots, \alpha_\ell)$ and verifies that:
  - $\beta_i = H_k(\tau_{i-1}, \alpha_i)$ for all $1 \le i \le \ell - 1$, and
  - $\mathsf{V}(x, \pi) = 1$.

We note the following facts about $\Pi_{\mathrm{FS},\mathcal{H}}$

1. The honest prover complexity of $\Pi_{\mathrm{FS},\mathcal{H}}$ is equal to the honest prover complexity of Π with an additive overhead of computing $\ell - 1$ hash values.
2. The verifier complexity of $\Pi_{\mathrm{FS},\mathcal{H}}$ is equal to the verifier complexity of Π with the same hashing additive overhead.
3. The protocol $\Pi_{\mathrm{FS},\mathcal{H}}$ is not necessarily sound, even if Π is sound and $\mathcal{H}$ is a "strong cryptographic hash function". As we will discuss later, soundness *is* guaranteed when Π satisfies what is called "round-by-round soundness", defined next.

Round-by-Round (Unambiguous) Soundness and Fiat-Shamir. Following [CCH+18, CCH+19, CHK+19a, LV20], we consider the notion of round-by-round (unambiguous) soundness to capture a particular kind of soundness analysis for super-constant round interactive proofs. For these proof systems, it has been shown that correlation intractability for an appropriate relation suffices for a hash family to instantiate the Fiat-Shamir heuristic for unambiguously round-by-round sound interactive proofs.

Definition 2.20 (Unambiguous Round-by-Round Soundness [CCH+18, CHK+19a, LV20])**.** Let $\Pi = (\mathsf{P}, \mathsf{V})$ be a $2\ell - 1$-message public coin interactive proof system for a language $\mathcal{L}$. We say that Π has unambiguous round-by-round soundness error $\epsilon(\cdot)$ if there exist functions $(\mathsf{State}, \mathsf{NextMsg})$ with the following syntax.

- State is a deterministic (not necessarily efficiently computable) function that takes as input an instance x and a transcript prefix τ and outputs either accept or reject.
- $\mathsf{NextMsg}$ is a deterministic (not necessarily efficiently computable) function that takes as input an instance x and a transcript prefix τ and outputs a (possibly aborting) prover message $\alpha \in \{0,1\}^* \cup \{\bot\}$.

We additionally require that the following properties hold.

1. If $x \notin \mathcal{L}$, then $\mathsf{State}(x, \emptyset) = \mathsf{reject}$, where $\emptyset$ denotes the empty transcript.
2. If $\mathsf{State}(x, \tau) = \mathsf{reject}$ for a transcript prefix τ, then $\mathsf{NextMsg}(x, \tau) = \bot$. That is, $\mathsf{NextMsg}(x, \tau)$ is only defined on accepting states.
3. For every input x and partial transcript $\tau = \tau_i$, then for every potential prover message $\alpha_{i+1} \neq \mathsf{NextMsg}(x, \tau)$, it holds that
$$\Pr_{\beta_{i+1}}\left[\mathsf{State}(x, \tau|\alpha_{i+1}|\beta_{i+1}) = \mathsf{accept}\right] \leq \epsilon(n)$$
4. For any *full*[7] transcript τ, if $\mathsf{State}(x, \tau) = \mathsf{reject}$ then $\mathsf{V}(x, \tau) = 0$.

We say that Π is $\mathsf{unambiguously\ round\text{-}by\text{-}round\ sound}$ if it has unambiguous round-by-round soundness error ϵ for some $\epsilon(n) = \mathsf{negl}(n)$.

Next, we restate the result that specific forms of correlation intractability suffice to instantiate the Fiat-Shamir transform for protocols satisfying unambiguous round-by-round soundness.

Theorem 2.21 ([CCH+18, LV20])**.** *Suppose that $\Pi = (\mathsf{P}, \mathsf{V})$ is a $2\ell - 1$-message public-coin interactive proof for a language $\mathcal{L}$ with perfect completeness and unambiguous round-by-round soundness with corresponding functions $(\mathsf{State}, \mathsf{NextMsg})$. Let $\mathcal{X}_n$ denote the set of partial transcripts (including the input and all messages sent) and let $\mathcal{Y}_n$ denote the set of verifier messages when Π is executed on an input of length n.*

Finally, define the relation ensemble $\mathcal{R} = \mathcal{R}_{\mathsf{State},\mathsf{NextMsg}}$ as follows:

$$\mathcal{R}^{(n)}_{\mathsf{State},\mathsf{NextMsg}} := \left\{ \Big((x, \tau|\alpha), \beta\Big) : \begin{array}{c} x \in \{0,1\}^n, \\ \alpha \neq \mathsf{NextMsg}(x, \tau) \\ \textit{and} \\ \mathsf{State}(x, \tau|\alpha|\beta) = \mathsf{accept} \end{array} \right\}.$$

If a hash family $\mathcal{H} = \{\mathcal{H}_n : \mathcal{X}_n \to \mathcal{Y}_n\}$ is correlation intractable for $\mathcal{R}$, then the round-reduced protocol $\Pi_{\mathrm{FS},\mathcal{H}}$ is an adaptively unambiguously sound argument system for $\mathcal{L}$.

3 The Outline-and-Batch Protocol

For $\mathcal{X}, \mathcal{Y} \subseteq \{0,1\}^*$, let $f : \mathcal{X} \to \mathcal{Y}$ be a function and $\|\cdot\| : \mathcal{X} \to \mathbb{N}$ denote a "size measure" for inputs to f. Let $\mathcal{X}_n$ denote $\{x \in \mathcal{X} : \|x\| = n\}$, and let $f_n : \mathcal{X}_n \to \mathcal{Y}$ denote the restriction of f to $\mathcal{X}_n$. Recall from Section 2 that $f_n^k : \mathcal{X}_n^k \to \mathcal{Y}^k$ denotes the function mapping $(x_1, \ldots, x_k)$ to $(f_n(x_1), \ldots, f_n(x_k))$.

[7] By a full transcript, we mean a transcript for which the verifier halts.

Definition 3.1 (Downwards self-reduction)**.** A downwards self-reduction for f is a deterministic oracle algorithm D such that for any n and $x \in \mathcal{X}_n$, $\mathsf{D}^{f_{n-1}}(x) = f(x)$. If on input x, D queries $q_1, \ldots, q_d$, then we say that D is a *d-query* downwards self-reduction and we refer to $((q_1, f(q_1)), \ldots, (q_k, f(q_d)))$ as an outline of the evaluation of f on x.

Definition 3.2 (Batching reduction)**.** A k'-to-k batching reduction for f with soundness error ϵ is a probabilistic algorithm B that on input $\{(x'_i, y'_i) \in \mathcal{X}_n \times \mathcal{Y}\}_{i \in [1,k']}$ outputs $\{(x_i, y_i) \in \mathcal{X}_n \times \mathcal{Y}\}_{i \in [1,k]}$ such that:

- (Completeness) If $y'_i = f(x'_i)$ for all $i \in [1, k']$, then with probability 1, $y_i = f(x_i)$ for all $i \in [1, k]$.
- (Soundness) If $y'_i \neq f(x'_i)$ for some $i \in [1, k']$, then with all but ϵ probability over the randomness of B, $y_i \neq f(x_i)$ for some $i \in [1, k]$.

We remark that it may be useful to consider batching reductions that are *interactive*, but for our purposes, non-interactive batching reductions suffice for our instantiations. We leave discussion of abstract interactive batching reductions to future work.

Theorem 3.3. *If f has a d-query downwards self reduction D and a dk-to-k-batching reduction B with error ϵ, then there is a public-coin interactive proof for the language*

$$\mathcal{L}^k_{f_n} := \{((x_1, \cdots, x_n), (y_1, , \cdots, y_n)) \in \mathcal{X}^k \times \mathcal{Y}^k : f_n(x_i) = y_i \text{ for all } i \in [1, k]\}$$

with $n - 1$ rounds of interaction and with unambiguous round-by-round soundness error ϵ.

Remark 5. We remark that the hypotheses of Theorem 3.3 can be relaxed to only require completeness and soundness for B when applied to inputs that correspond to the queries of an evaluation of D. This relaxation captures the classical sumcheck protocol [LFKN90] as a special case.

Proof of Theorem 3.3. The prover P and verifier V both take as input a statement $((x_1, \ldots, x_k), (y_1, \ldots, y_k)) \in \mathcal{X}^k_n \times \mathcal{Y}^k_n$, and the protocol is defined recursively.

Base Case: If $n = 1$, then no messages are sent (P does nothing), and V accepts only if $f_1(x_i) = y_i$ for all $i \in [1, k]$.

Recursive Case: If $n > 1$, then:

1. P computes $\mathsf{D}^{f_{n-1}}(x_i)$ for each $i \in [1, k]$, recording the queries made by D and answering queries according to f_{n-1}. Then P sends all k corresponding d-tuples of query-answer pairs to V. Let $((\tilde{x}'_1, \tilde{y}'_1), \ldots, (\tilde{x}'_{dk}, \tilde{y}'_{dk}))$ denote the concatenation of all k d-tuples of query-answer pairs received by V.
2. When V receives k d-tuples of query-answer pairs, V checks for each i that the i-th tuple is consistent with an execution of D[8] on input x_i (if not, then V rejects). V then samples randomness r for B and sends it to P.

[8]That is, V emulates an execution of D on each x_i, checking that for every j, the jth oracle call in the sequence of k executions is to $\tilde{x}'_j$; it then uses $\tilde{y}'_j$ as the oracle's output in its emulation.

3. Let $((\tilde{x}''_1, \tilde{y}''_1), \ldots, (\tilde{x}''_k, \tilde{y}''_k))$ denote $\mathsf{B}((\tilde{x}'_1, \tilde{y}'_1), \ldots, (\tilde{x}'_{dk}, \tilde{y}'_{dk}); r)$. P and V recursively invoke the interactive proof for f^k on $((\tilde{x}'_1, \ldots, \tilde{x}'_k), (\tilde{y}'_1, \ldots, \tilde{y}'_k))$.

We next describe how to give Π the structure of an unambiguous round-by-round sound protocol.

- At any step in the recursion, we have "current inputs" $x_1, \ldots, x_k$ as well as outputs $y_1, \ldots, y_k$ claimed in the previous recursive step. At this execution point, we define State to be accept if and only if $f(x_i) = y_i$ for all i.
- After Step 1 in a recursive call, we define State to be accept iff $\mathcal{V}$ has not rejected and $f(\tilde{x}') = \tilde{y}'$ for every pair $(\tilde{x}', \tilde{y}')$ in the lists sent by the prover.

We define $\mathsf{NextMsg}(\tau)$ to be the output of the honest prover algorithm in the description of the recursion above, which means to:

- Compute B on its previous message and the verifier's challenge r, and
- Compute the downwards self-reduction on the resulting tuple of inputs.

Given this description of $(\mathsf{State}, \mathsf{NextMsg})$, unambiguous round-by-round soundness follows from the correctness of the downwards self-reduction and the soundness of the batching reduction. □

Finally, we discuss instantiating the Fiat-Shamir transform for the protocol Π in Theorem 3.3 by appealing to Theorem 2.21. Since the round-by-round State function is fairly simple for our protocol Π (in that it does not depend on the entire protocol history), we can rely on correlation intractability for relations with a fairly simple description. By invoking Theorem 2.21, we obtain the following corollary.

Corollary 3.4. *Under the hypotheses of Theorem 3.3 and additionally assuming the existence of a hash family $\mathcal{H}$ that is correlation intractable for the following relation $\mathcal{R}$:*

$$\mathcal{R}^{(n)}_{\mathsf{State},\mathsf{NextMsg}} := \left\{ \left(\alpha = (n, \tilde{x}'_1, \tilde{y}'_1, \ldots, \tilde{x}'_{dk}, \tilde{y}'_{dk}), r\right) : \begin{array}{c} \tilde{y}'_j \neq f_{n-1}(\tilde{x}'_j) \text{ for some } j \\ \text{and} \\ \tilde{y}''_i = f_{n-1}(\tilde{x}''_i) \text{ for all } i \end{array} \right\},$$

where $\{(\tilde{x}''_i, \tilde{y}''_i)\}_{i\in[1,k]}$ is the output of B on input $\{(\tilde{x}'_j, \tilde{y}'_j)\}_{j\in[1,dk]}$ and random coins r, there is a non-interactive argument system for $\mathcal{L}^k_{f_n}$ with adaptive unambiguous soundness.

3.1 Instantiations of Outline-and-Batch

Appropriate instantiations of our outline-and-batch protocol (Theorem 3.3, Corollary 3.4) can recover the interactive proof systems or non-interactive argument systems constructed in the following works:

1. *The [LFKN90] interactive proof system for #SAT and its Fiat-Shamir instantiations [CHK+19a, JKKZ21].* The sumcheck protocol can be viewed as a composition of
   (a) a $(d{+}1)$-query downward self-reduction that reduces a statement about the sum $\sum_{x_1,\ldots,x_n\in\{0,1\}} p(x_1, \ldots,$ of a d-degree, n-variate polynomial p (over some finite field) to $d+1$ statements of the form $\sum_{x_2,\ldots,x_n} p(\alpha, x_2, \ldots, x_n)$ (for hard-coded values of α); and

(b) a $(d+1)$-to-1 batching reduction reducing these $(d+1)$ statements to a single statement about $\sum_{x_2,\ldots,x_n} p(r, x_2, \ldots, x_n)$ for a *uniformly random* r.

2. *The [Pie19, CHK+19b] interactive proof system for IS over the* signed quadratic residue group $\mathbb{QR}_N^+$ *and its Fiat-Shamir instantiation in the standard model [LV20].* Let $x \xrightarrow{T} y$ (now) denote the statement "x^{2^T} equals y over $\mathbb{QR}_N^+$". These protocols consist of a 2-query downward self-reduction from a statement $x \xrightarrow{T} y$ to two statements of the form $x_i \xrightarrow{T/2} y_i$ and a 2-to-1 batching reduction that combines these two statements to a single such statement using a random linear combination.

3. *The [EFKP20] continuous VDF adapted to* $\mathbb{QR}_N^+$. This protocol consists of a d-query downward self-reduction from a statement $x \xrightarrow{T} y$ to d statements of the form $x_i \xrightarrow{T/d} y_i$ and a d-to-1 batching reduction from these d statements to a single such statement using, again, a random linear combination. The parameter d is set in their construction to $O(\lambda)$, for the security parameter λ.

4. *The [BHR+21] interactive proof system for IS.* In Section 4.3, we describe how this protocol fits the "outline-and-batch" framework and then show how to instantiate Fiat-Shamir for this protocol in the standard model.

4 Non-Interactive Argument for Iterated Squaring in a Trapdoor Group of Unknown Order

We first recall the iterated squaring (IS) problem modulo an integer N and discuss the hardness of IS. This includes a new hardness reduction showing that certain *public-coin* variants of IS are as hard as the "traditional" IS problem in the RSA group. Next, we consider a general IS problem over an arbitrary group of unknown order and construct our unambiguous "outline-and-batch" argument system in this setting.

4.1 Iterated Squaring modulo N

We first define IS and then recall the assumption of [RSW96] on its sequential hardness that our VDF is based on. Our hardness assumption on IS required for **PPAD** hardness is a relaxation of this assumption.

Definition 4.1 ([RSW96, CLSY93])**.** IteratedSquaring (IS)

- *Instance.*
  1. Integers $N, T \in \mathbb{N}$
  2. Group element $g \in \mathbb{Z}_N^*$
- *Solution.* $f(N, g, T) := g^{2^T} \bmod N$

Assumption 4.2 (Sequential hardness of IS [RSW96])**.** For a security parameter $\lambda \in \mathbb{N}$, let $_{RSA} \in \lambda^{O(1)}$ denote the size of RSA modulus that corresponds to λ bits of security. Sample $N = pq$ as the product of two random $\lambda_{RSA}/2$-bit primes and $g \leftarrow \mathbb{Z}_N^*$. Consider any time parameter

$T = 2^{o(\lambda)}$. Any A that uses $2^{o(\lambda)}$ amount of parallelism and computes $f(N, g, T)$ with a probability that is non-negligible in λ requires *sequential time* $T(1 - o(1))$ group operations.

Assumption 4.3 (Polynomial hardness of IS [CHK+19b])**.** For a security parameter $\lambda \in \mathbb{N}$, let N and g be sampled as in Assumption 4.2. There exists an efficiently computable function $T(1^\lambda)$, such that no $\lambda^{O(1)}$-time algorithm can compute $f(N, g, T)$ with a non-negligible probability.

Remark 6 (Assumption 4.3 vs. assumption in [CHK+19b, Pie19]). The hardness assumption in [CHK+19b, Pie19] is slightly different from Assumption 4.3. Firstly, the modulus N in [CHK+19b, Pie19] is sampled a product of two random $\lambda_{RSA}/2$-bit *safe primes* – the statistical soundness of Pietrzak's proof-of-exponentiation (PoE) is guaranteed only in such moduli. Secondly, to attain unambiguity, [CHK+19b, Pie19] switch to the algebraic setting of *signed quadratic residues* [FS00, HK09]. In comparison, our assumption is made on the conventional RSA modulus and this suffices since we rely the PoE from [BHR+21] which achieves statistical soundness and (as we show) unambiguity for arbitrary groups.

Sampling using public coins. Finally, for our results on public-coin **PPAD** hardness, we assume the hardness of IS over a *public-coin* modulus N. In our work, we choose a particular public-coin distribution on N and prove that the resulting IS assumption follows from Assumption 4.3.

Assumption 4.4. For a security parameter $\in \mathbb{N}$, let $N_1, N_2, \ldots, N_{\lambda^3}$ be i.i.d. uniformly random integers in the range $[1, 2^\lambda]$, let $N = \prod_{i=1}^{\lambda^3} N_i$, and let $g \leftarrow \mathbb{Z}_N^\times$ be sampled uniformly at random[9] from the unit group modulo N. There exists an efficiently computable function $T(1^\lambda)$, such that no $\lambda^{O(1)}$-time algorithm (given as input $(N_1, \ldots, N_{\lambda^3}, \mathsf{coins}(g), T)$), where $\mathsf{coins}(g)$ denotes the public coins used to sample g, can compute $f(N, g, T)$ with a non-negligible probability.

Lemma 4.5. *Assumption 4.4 follows from Assumption 4.3.*

Proof. We give a reduction from breaking Assumption 4.3 to breaking Assumption 4.4. For some efficiently-computable function $T(\lambda)$ and $N := \prod N_i$. suppose that $\mathsf{A}(N_1, \ldots, N_{\lambda^3}, \mathsf{coins}(g), T) = g^{2^T} \mod N$ with non-negligible probability over $(N_1, \ldots, N_{\lambda^3}, \mathsf{coins}(g))$. Let $N_{\text{RSA}} = pq$ be a randomly sampled RSA modulus (for $p, q \in \{0,1\}^{\lambda/2}$) and let g_{RSA} be a given uniformly random element of $\mathbb{Z}_{N_{\text{RSA}}}^\times$. We embed the $(N_{\text{RSA}}, g_{\text{RSA}}, T)$ problem into A via the following procedure.

1. Sample λ^3 random integers $N'_1, \ldots, N'_{\lambda^3}$ *along with* their factorizations [Bac88, Kal03].
2. Compute the first index i such that N'_i is an RSA modulus; if no such index exists, set $i = \perp$ and abort.
3. Set $N_j = N'_j$ for all $j \neq i$ and $N_i = N_{\text{RSA}}$.
4. Sample a uniformly random element $\tilde{g} \leftarrow \mathbb{Z}_{\widetilde{N}}^\times$ for $\widetilde{N} = \prod_{j \neq i} N_j$. Define $g \in \mathbb{Z}_N^\times$ to be the unique group element (efficiently computable via the Chinese Remainder Theorem[10]) such that $g \equiv g_{\text{RSA}} \pmod{N_{\text{RSA}}}$ and $g \equiv \tilde{g} \pmod{\widetilde{N}}$.

[9] To make this problem public coin, we need to specify a public-coin efficient procedure for sampling g. To do this, we repeatedly execute the following process: sample a uniformly random element a of $\mathbb{Z}_N$, and compute $\gcd(a, N)$. If the GCD is equal to 1, we are done; if not, we factor N into a product of (> 1) relatively prime integers and recursively run the procedure on all of the factors simultaneously (implicitly applying the Chinese Remainder Theorem).

[10] With negligible probability we will have that N_{RSA} is not relatively prime to $\widetilde{N}$, in which case we will abort for simplicity.

5. Simulate random coins $\mathsf{coins}(g)$ for generating g modulo N.[11]

6. Call $\mathsf{A}(N_1, \ldots, N_{\lambda^3}, \mathsf{coins}(g), T)$ and reduce the output modulo N_{RSA}. Output this group element.

To see that this reduction is valid, we note that the reduction aborts with only negligible probability; this is because Step (2) of the reduction only aborts if $N'_1, \ldots, N'_{\lambda^3}$ are all not RSA moduli. But each N'_i is an RSA modulus with probability $\Omega(1/\lambda^2)$ by the prime number theorem, and thus there is only a negligible probability that all λ^3 of them are not.

Moreover, conditioned on this non-aborting event, the distribution of $(N_1, \ldots, N_\lambda)$ is identical to the distribution of $(N'_1, \ldots, N'_\lambda)$, and thus the distribution of $(N_1, \ldots, N_\lambda)$ is statistically close to the correct input distribution for A. The group element g (and its random coins $\mathsf{coins}(g)$) is also sampled correctly by the Chinese Remainder Theorem, so the call to A will succeed with non-negligible probability, solving the IS problem modulo N (and therefore modulo N_{RSA}). □

4.2 Trapdoor Groups with Unknown Order

Definition 4.6 (Group of unknown order)**.** A group sampler for a group of unknown order consists of the following two functionalities:

- A setup algorithm $\mathsf{Setup}(1^\lambda)$ that samples the description of a group $\mathbb{G}_\lambda$ of order *at most* 2^λ. For our purposes, a group description consists of a distinguished identity element $\mathsf{id}_{\mathbb{G}}$ and efficient membership testing algorithm that takes as input an arbitrary string and decides whether the string is a valid element of $\mathbb{G}_\lambda$.
- Efficient $\mathsf{poly}(\lambda)$-time algorithms, given a description of $\mathbb{G}_\lambda$, for:
  - Sampling a uniformly random group element,
  - Computing the group law $(g, h) \mapsto gh \in \mathbb{G}_\lambda$, and
  - Computing the inverse map $g \mapsto g^{-1} \in \mathbb{G}_\lambda$.

Remark 7. These efficient group operations generically imply that one can compute exponentiations $g \mapsto g^x$ in time $\mathsf{poly}(\lambda) \cdot \log(x)$ by repeated squaring. For example, this implies that $g \mapsto g^{2^T}$ can be computed in time $T \cdot \mathsf{poly}(\lambda)$ (or T group operations).

Note that if the order of $\mathbb{G}_\lambda$ is known, then the map g^{2^T} can actually be computed in time $\mathsf{poly}(\lambda, \log T)$ by first reducing 2^T modulo the order of the group. However, when the order of the group is unknown, it is plausible that this map requires time roughly T group operations, as originally proposed by [RSW96]. We formulate two flavors of this assumption, matching Assumptions 4.2 and 4.3 in the case of RSA groups.

Assumption 4.7 ((T, p)-Sequential Hardness)**.** Given the description of $\mathbb{G}_\lambda$ and a random group element g, any algorithm running in sequential time $T(1 - o(1))$ with $p(\lambda)$ parallelism outputs g^{2^T} with only negligible probability.

[11] This is done by generating coins for a fresh uniformly random element of $\mathbb{Z}_N^\times$. Then, during the *successful step* (where units $a_1, \ldots a_k$ modulo some factors of N are generated), we replace the coins with the unique set of coins that would result in an output of g in this step.

Assumption 4.8 (Polynomial Hardness of Iterated Squaring)**.** There exists an efficiently computable function $T(1^\lambda)$ such that, given the description of $\mathbb{G}_\lambda$ and a random group element g, no algorithm running in time $\lambda^{O(1)}$ can output g^{2^T} with non-negligible probability.

In order to prove the unambiguous soundness of our non-interactive argument system for IS, we will make use of groups satisfying Assumption 4.8 that have *trapdoors* allowing for efficient iterated squaring. We formalize this by requiring that the group distribution $\mathbb{G}_\lambda$ could be sampled along with its order (using secret coins).

Definition 4.9 (Trapdoor group with unknown order)**.** A trapdoor group with unknown order is a group with unknown order (Definition 4.6) equipped with an additional setup algorithm $\mathsf{TrapSetup}(1^\lambda)$ that outputs the description of a group $\mathbb{G}_\lambda$ *along with* its order M. We require that the distribution of groups output by $\mathsf{Setup}(1^\lambda)$ is statistically indistinguishable from the distribution of groups output by $\mathsf{TrapSetup}(1^\lambda)$ (where the order information is dropped).

RSA groups $\mathbb{Z}_{pq}^\times$ are naturally equipped with the required trapdoor structure, because if $N = pq$ is sampled as the product of two known primes, then the order of $\mathbb{Z}_N^\times$ is equal to $\phi(pq) = (p-1)(q-1)$. Similarly, groups of the form $\mathbb{Z}_N^\times$ for $N = N_1 \times \ldots \times N_k$ (as in Assumption 4.4) can be given the desired trapdoor structure via the algorithm $\mathsf{TrapSetup}$ that samples each N_i along with its prime factorization [Bac88, Kal03] and uses these prime factorizations to compute $\phi(N)$.

4.3 Interactive Iterated Squaring Protocol

In this section, we recall the interactive proof system Π of [BHR+21] for IS and analyze the Fiat-Shamir heuristic applied to Π using an appropriate correlation-intractable hash family. Since the groups output by $\mathsf{Setup}(1^\lambda)$ and $\mathsf{TrapSetup}(1^\lambda)$ are statistically indistinguishable, we assume that $\mathsf{TrapSetup}(1^\lambda)$ is used for the purposes of both the construction and its analysis.

Let $\mathbb{G}_\lambda \leftarrow \mathsf{Setup}(1^\lambda)$ denote a group (distribution) with unknown order and associated generator g. For simplicity, we only consider T of the form $T = 2^t$.[12] For T of this form, we construct an interactive proof system for IS by having the prover invoke the "outline and batch" protocol (Theorem 3.3) on λ *identical computations of* g^{2^T}, i.e., $((g, \cdots, g), (g^{2^T}, \cdots, g^{2^T}))$. By Theorem 3.3, it suffices to show that the function $f : g \mapsto g^{2^T}$ has a 2-query downwards self reduction (Definition 3.1) and a 2λ-to-λ batching reduction (Definition 3.2).

- **2-Downwards Self-Reduction**: Given an instance of the T-IS problem $f(g, T)$, we can query $f(g, T/2)$ to obtain an intermediate group element μ, and then call $f(\mu, T/2)$ to obtain $\mu^{2^{T/2}} = g^{2^T}$.
- **2λ-to-λ Batching Reduction**: Given 2λ instances $g_1, \ldots, g_{2\lambda}$ for $f(\cdot, T)$ and 2λ candidate outputs $h_1, \ldots, h_{2\lambda}$, the batching reduction samples λ i.i.d. vectors $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_\lambda \leftarrow \{0,1\}^{2\lambda}$. The reduction then outputs λ statements about $f(\cdot, T)$:

$$\left(\prod_{j=1}^{2\lambda} g_j^{r_{1,j}} \xrightarrow{T} \prod_{j=1}^{2\lambda} h_j^{r_{1,j}}, \ldots, \prod_{j=1}^{2\lambda} g_j^{r_{\lambda,j}} \xrightarrow{T} \prod_{j=1}^{2\lambda} h_j^{r_{\lambda,j}} \right).$$

[12] A protocol for general T can be obtained by dividing T by computing a binary decomposition of the resulting integer, and sequentially composing squaring protocols for integers of the form 2^t.

Completeness of the batching reduction is immediate by group axioms. For soundness, suppose that $g_j^{2^T} \neq h_j$ for some j. The i-th statement output by the reduction is true if and only if

$$\prod_{j=1}^{2\lambda} (g_j^{2^T})^{r_{i,j}} = \prod_{j=1}^{2\lambda} h_j^{r_{i,j}},$$

which is equivalent to the equation

$$\prod_{j=1}^{2\lambda} (g_j^{2^T} h_j^{-1})^{r_{i,j}} = \mathsf{id}_{\mathbb{G}_\lambda}.$$

For $\boldsymbol{r}_i \leftarrow \{0,1\}^{2\lambda}$, $i \in [1,\lambda]$, this event occurs with probability at most $1/2$ (see [BHR+21, Fact 8.1]). Thus, at least one of the λ resulting statements is false except with probability $2^{-\lambda}$. In fact, this analysis gives a product set description for the "bad challenges" of the batching reduction. For a fixed $\alpha = ((g_1,h_1),\ldots,(g_{2\lambda},h_{2\lambda}))$, the bad set $\mathcal{R}_\alpha = \mathcal{B}_\alpha^{(1)} \times \ldots \times \mathcal{B}_\alpha^{(\lambda)} \subset (\{0,1\}^{2\lambda})^\lambda$, where

$$\mathcal{B}_\alpha^{(i)} = \left\{ \boldsymbol{r} \in \{0,1\}^{2\lambda} : \prod_{j=1}^{2\lambda} (g_j^{2^T})^{r_j} = \prod_{j=1}^{2\lambda} h_j^{r_j} \right\}$$

(in fact, we have that each $\mathcal{B}_\alpha^{(i)} = \mathcal{B}_\alpha$ for a fixed set $\mathcal{B}_\alpha$). As mentioned above, we have that $|\mathcal{B}_\alpha^{(i)}| \leq 2^{2\lambda}/2$ for every j and every false α. Thus, the bad-challenge relation $\mathcal{R}$ is a product relation with the appropriate sparsity, where $\mathcal{R}$ is defined as the set of pairs $(\alpha, \beta = (\boldsymbol{r}_1,\ldots,\boldsymbol{r}_\lambda))$ for which at least one of the 2λ statements defined by α is false but all of the λ statements output by the reduction are true.

Finally, we observe that for $(\mathbb{G}_\lambda, M) \leftarrow \mathsf{TrapSetup}(1^\lambda)$, the relation $\mathcal{R}$ is also *efficiently product verifiable*: to verify that $v \in \mathcal{B}_\alpha^{(i)}$, it suffices to check the equation

$$\prod_{j=1}^{2\lambda} (g_j^{2^T})^{r_j} = \prod_{j=1}^{2\lambda} h_j^{r_j}.$$

This can be checked in time $\mathsf{poly}(\lambda, \log T)$ given the order M of $\mathbb{G}_\lambda$, by first computing 2^T modulo M and then checking the equation above using the group law and repeated squaring.

Thus, by Theorem 3.3 we conclude that there is a $t = \log(T)$-round unambiguous interactive proof system for the T-IS problem with $\mathsf{poly}(\lambda)$ communication. Moreover, by Corollary 3.4 this protocol can be round-collapsed to a computationally unambiguous non-interactive argument system using a hash function family that is correlation-intractable for the relation R above (where we consider T as part of the input to the relation). Finally, by Theorem 2.14, such hash functions can be built under the learning with errors assumption. This is captured by the following corollary.

Corollary 4.10. *For a security parameter $\lambda \in \mathbb{N}$, let $\mathbb{G}_\lambda$ be a trapdoor group of unknown defined in Definition 4.9. Assuming polynomial hardness of LWE (Assumption 2.10), $\Pi_{\mathrm{FS},\mathcal{H}}$ is an adaptively unambiguously-sound non-interactive argument for the language*

$$\mathcal{L}_{\mathbb{G}_\lambda}^\lambda := \{((g_1,\cdots,g_\lambda),(h_1,\cdots,h_\lambda),T) \in \mathbb{G}_\lambda^\lambda \times \mathbb{G}_\lambda^\lambda \times \mathbb{N} : h_i = g^{2^T} \text{ for all } i \in [1,\lambda]\}.$$

5 PPAD Hardness

In this section, we construct a hard distribution of RSVL from any hard f that is downward self-reducible and batch-reducible, additionally assuming the unambiguous soundness of $\Pi_{\mathrm{FS},\mathcal{H}}$, the non-interactive "outline-and-batch" argument system for $\mathcal{L}^k_{f_n}$ (Corollary 3.4). By Theorem 2.8, this implies hardness of EOML, which is complete for **CLS** (Definition A.4); since **CLS** $\subseteq$ **PPAD**, **PPAD**-hardness follows. Furthermore, we show in Appendix B how to strengthen this result to show hardness for class **UEOPL** $\subseteq$ **CLS** by constructing a hard distribution of a (total) search problem called UniqueForwardEOML (Definition B.1). The construction of our RSVL instance is given in Section 5.1 and its hardness is then shown in Section 5.2.

5.1 Construction

We follow the blueprint from [CHK+19a, CHK+19b] of first giving an *implicit*, easy-to-understand description of the RSVL instance (Section 5.1.1) and then "simulating" it to obtain the description of successor and verifier circuits (Section 5.1.2). Since our construction works with *any* f that is downward self-reducible and batch-reducible, it generalises the constructions of RSVL instance in [CHK+19a, CHK+19b] and the continuous VDF in [EFKP20]. Indeed, as we saw in Section 3, both iterated squaring and the sumcheck problem satisfy downward self-reducibility and batch-reducibility.

5.1.1 An Implicit Description

An implicit description of our RSVL instance RSVL is given in Algorithm 1 (see Algorithm 5 for a version of Algorithm 1 with explicit subscripts). It describes a recursive procedure F that *verifiably* computes f by exploiting the structure of the outline-and-batch protocol Π from Section 3. The explicit description, given in Section 5.1.2, is obtained by simulating F's computation one step at a time, which yields an *incrementally-verifiable* procedure that computes f. Since F is non-interactive, we rely on $\Pi_{\mathrm{FS},\mathcal{H}}$, the non-interactive outline-and-batch protocol from Corollary 3.4.

Conventions on batching and downward self-reducing. Before describing F, we establish some conventions for the batching algorithm B and the downward self-reducing algorithm D that F will rely on.

- The k'-to-k batching reduction B as defined in Definition 3.2 reduces $k' = k \cdot d$ statements to k statements. Since we work with parallel repetitions and use vectors of statements, it will be convenient to define a batching reduction $\widetilde{\mathsf{B}}$ that takes d-many k-vectors of statements and outputs a k-vector of statements. That is, $\widetilde{\mathsf{B}}$ takes an input $((\boldsymbol{x}_0, \boldsymbol{y}_0), \cdots, (\boldsymbol{x}_{d-1}, \boldsymbol{y}_{d-1}))$, serialises it to

$$((x_{0,0}, y_{0,0}), \cdots, (x_{0,d-1}, y_{0,d-1}), (x_{1,0}, y_{1,0}), \cdots, (x_{k-1,d-1}, y_{k-1,d-1}))\,,$$

  invokes B on this (using same random coins it received) to obtain

$$((x_{d,0}, y_{d,0}), \cdots, (x_{d,k-1}, y_{d,k-1}))\,,$$

  which it assembles into $(\boldsymbol{x}_d, \boldsymbol{y}_d)$, its output.

- F will be required to *partially simulate* $\mathsf{D}^{f_{n-1}}$, where the output of the $(i-1)$-query partial simulation of $\mathsf{D}^{f_{n-1}}(x)$ on $(y_0, \cdots, y_{i-1})$ is q_i, with the response to query q_j, $j \in [0, i-1]$, set to be y_j. Similarly, for its parallel execution $\mathbf{D}^{f_{n-1}}(\boldsymbol{x})$, the output of the $(i-1)$-query partial simulation on $(\boldsymbol{y}_0, \cdots, \boldsymbol{y}_{i-1})$ is denoted by the k-vector $\boldsymbol{q}_i$:

$$\boldsymbol{x} \xrightarrow{\mathsf{D}} \boldsymbol{q}_0 \xrightarrow{f_{n-1}} \boldsymbol{y}_0 \xrightarrow{\mathsf{D}} \boldsymbol{q}_1 \xrightarrow{f_{n-1}} \boldsymbol{y}_1 \xrightarrow{\mathsf{D}} \cdots \xrightarrow{\mathsf{D}} \boldsymbol{q}_{i-1} \xrightarrow{f_{n-1}} \boldsymbol{y}_{i-1} \xrightarrow{\mathsf{D}} \boldsymbol{q}_i. \tag{8}$$

  Here, for convenience, $\xrightarrow{\mathsf{D}}$ denotes one step of this partial simulation, using $\boldsymbol{y}$s that have been already obtained as answers

Overview of F. It is a recursive procedure that takes as input a vector $\boldsymbol{x} \in \mathcal{X}_n^k$ (let's ignore the transcript τ for now) and outputs $\boldsymbol{y} = \boldsymbol{f}_n(\boldsymbol{x})$ and proof $\pi = \pi(\boldsymbol{x} \xrightarrow{f_n} \boldsymbol{y})$. F does this as follows.

1. *Recursive outlining.* Recursively compute (Line 3 to Line 7) the outline

   $$\boldsymbol{\mu} := ((\boldsymbol{q}_0, \boldsymbol{y}_0), \cdots, (\boldsymbol{q}_{d-1}, \boldsymbol{y}_{d-1}))$$

   for the evaluation of $\boldsymbol{f}_n(\boldsymbol{x})$ along with proofs $(\pi_0, \cdots, \pi_{d-1})$ where $\pi(\boldsymbol{q}_i \xrightarrow{f_{n-1}} \boldsymbol{y}_i)$ for all $i \in [0, d-1]$. In more details, F does the following:

   (a) Invoke $\mathbf{D}(\boldsymbol{x})$ to obtain a k-vector of queries $\boldsymbol{q}_0$.

   Assume that the partial outline $(\boldsymbol{q}_0, \boldsymbol{y}_0), \cdots, (\boldsymbol{q}_{i-1}, \boldsymbol{y}_{i-1}), \boldsymbol{q}_i$ as in Eq. (8) and proofs $(\pi_0, \cdots, \pi_{i-1})$, where

   $$\pi_j := \pi(\boldsymbol{q}_j \xrightarrow{f_{n-1}} \boldsymbol{y}_j), \ j \in [0, i-1]\,,$$

   are available at this point.

   (b) Recursively invoke $\mathsf{F}(\boldsymbol{q}_i)$ to obtain $(\boldsymbol{y}_i, \pi_i)$, where $\pi_i = \pi(\boldsymbol{q}_i \xrightarrow{f_{n-1}} \boldsymbol{y}_i)$. Then partially simulate $\mathbf{D}(\boldsymbol{x})$ using $(\boldsymbol{y}_0, \cdots, \boldsymbol{y}_i)$ to obtain the next query $\boldsymbol{q}_{i+1}$.

   (c) Repeat Item 1b d times to obtain the outline $\boldsymbol{\mu} := ((\boldsymbol{q}_0, \boldsymbol{y}_0), \cdots, (\boldsymbol{q}_{d-1}, \boldsymbol{y}_{d-1}))$ and output $\boldsymbol{y}$.

2. *Batching.* Batch-reduce $\boldsymbol{\mu}$ to obtain a k-vector of new statements (Line 8 to Line 10)

   $$\boldsymbol{x}_d \xrightarrow{f_{n-1}} \boldsymbol{y}_d. \tag{9}$$

   The hash-input required to compute the challenge r that is used as randomness in $\widetilde{\mathsf{B}}$ is obtained from the transcript τ, an auxiliary input maintained for this sole purpose.

3. *Recursive proof-merging.* Recursively invoke $\mathsf{F}(\boldsymbol{x}_d)$ to compute proof π_d for Eq. (9) (Line 11). Assemble the final proof π by appending $\boldsymbol{\mu}$ to π_d and return $(\boldsymbol{y}, \pi)$.

Note that the computation of F on inputs of size n is reduced to $d+1$ recursive calls of F on inputs of size $(n-1)$. This downward-self-reducing structure will allow – as we show in Section 5.1.2 – the computation of F to be carried out in *incremental* manner. Moreover, since every step of the recursion is accompanied by appropriate proofs, the resulting incremental procedure is also *verifiable*. Below we prove that F is a verifiable function. More specifically, we show that

- $\mathsf{F}(\boldsymbol{x}, \varepsilon)$ outputs a value $\boldsymbol{y}$ such that $\boldsymbol{x} \xrightarrow{\boldsymbol{f}_n} \boldsymbol{y}$; and
- the proof π output by F will be *identical* to the prescribed proof produced by $\Pi_{\mathrm{FS},\mathcal{H}}$.

We first recall the definition of verifiable functions from [EFKP20].

Definition 5.1 (Verifiable functions [EFKP20])**.** A verifiable function is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$, where Setup is probabilistic polynomial-time that defines a domain $\mathcal{X}$ and range $\mathcal{Y}$, Eval is deterministic and Verify is deterministic polynomial-time, satisfying the following properties:

- **Completeness.** A verifiable function is complete if for every $\lambda \in \mathbb{N}$ and $x \in \mathcal{X}$

$$\Pr_{H \leftarrow \mathsf{Setup}(1^\lambda)}[\mathsf{Verify}(1^\lambda, H, x, \mathsf{Eval}(1^\lambda, H, x) = 1] = 1.$$

- **Soundness.** A verifiable function is $(s(\lambda), \epsilon(\lambda))$-sound if for every size-$s(\lambda)$ algorithm $\mathsf{A} = \{\mathsf{A}_\lambda\}_{\lambda \in \mathbb{N}}$

$$\Pr_{\substack{H \leftarrow \mathsf{Setup}(1^\lambda) \\ (x,y) \leftarrow \mathsf{A}(H)}}[\mathsf{Verify}(1^\lambda, H, x, y) = 1, \mathsf{Eval}(1^\lambda, H, x) \neq y] = O(\epsilon(\lambda).$$

Claim 5.2. *Let $\Pi_{\mathrm{FS},\mathcal{H}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be the non-interactive outline-and-batch argument for $\mathcal{L}^k_{\boldsymbol{f}_n}$ from Corollary 3.4. Then $(\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}, \mathsf{F}, \mathsf{V})$, where F is defined as in Algorithm 5 and hardwired with H sampled using $\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}$, is a verifiable function.*

Proof. For a CRS H sampled using $\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}$ and a true statement $\boldsymbol{x} \xrightarrow{\boldsymbol{f}_n} \boldsymbol{y}$, let $\pi := \mathsf{P}(H, \boldsymbol{x} \xrightarrow{\boldsymbol{f}_n} \boldsymbol{y})$ denote the prescribed proof. We argue below that $(\boldsymbol{y}, \pi) = \mathsf{F}(\boldsymbol{x}, \varepsilon)$, where F is hardcoded with H. This implies correctness; soundness then follows the (adaptive) soundness of $\Pi_{\mathrm{FS},\mathcal{H}}$ since the proofs produced by P and F are identical.

We proceed inductively in the size of the input $\boldsymbol{x} =: (x_0, \cdots, x_{k-1})$. In the base case of $|x_0| = \cdots = |x_{k-1}| = 1$, the claim is trivially true since (i) the proof returned is empty in both P and F, and (ii) $\boldsymbol{y} = (f(x_0), \cdots, f(x_{k-1}))$. Suppose that the claim is true for $|x_0| = \cdots = |x_{k-1}| = n-1$. By applying the induction hypothesis to the d recursive outlining calls (Line 5), we first infer that the outline for evaluation of $\boldsymbol{f}_n$ on $\boldsymbol{x}$ – and therefore the final output $\boldsymbol{y}$ – is correctly computed by F (recall that D is deterministic). This also implies that the challenge r in Line 9 is computed correctly since it depends only on the statement and outline, and is computed exactly as by the non-interactive prover. This, in turn, implies that the statement that the prover and verifier recurse down to is exactly Eq. (9) in both cases. Finally, we apply the induction hypothesis again to infer that the proof π_d returned by F is correct and identical, and as a result the final proof π is also correct because of the way it is assembled from π_d and the outline in Line 12. □

5.1.2 The Explicit Description

First we provide some definitions pertaining to $T_{n,d+1}$, the perfect directed $(d+1)$-ary tree of depth n, with the root ε on level n and leaves on level 0, i.e,

$$V(T_{n,d+1}) = [0,d]^{\leq n} \text{ and } E(T_{n,d+1}) = \{(vj, v)\}_{v \in [0,d]^{<n}, j \in [0,d]}.$$

Algorithm 1 Recursive description of the RSVL instance (cf. Algorithm 5 for subscripted version).

procedure $\mathsf{F}(\boldsymbol{x}, \tau)$
hardwired Descriptions of:

1. the function $f : \mathcal{X} \to \mathcal{Y}$
2. a hash $H : \{0,1\}^* \to \{0,1\}^\lambda$ sampled using $\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}(1^\lambda)$
3. the d-query downwards self-reduction algorithm D^f
4. the k'-to-k batching reduction $\widetilde{\mathsf{B}}$ where $k' = k \cdot d$

input Problem instances $\boldsymbol{x} \in \mathcal{X}_n^k$ and transcript $\tau \in \mathcal{X}^k \times \mathcal{Y}^k \times (\mathcal{X} \times \mathcal{Y})^{\leq d \cdot n}$
output $\boldsymbol{y} \in \mathcal{Y}^k$ and $\pi = \pi(\boldsymbol{x} \xrightarrow{f_n} \boldsymbol{y})$

if $|x_0| = \cdots = |x_{k-1}| = 1$ **then return** $((f_1(x_0), \cdots, f_1(x_{k-1})), \varepsilon)$

▷ **Recursive outlining**

Invoke $\mathsf{D}^{f_{n-1}}(\boldsymbol{x})$ to obtain query $\boldsymbol{q}_0$
for $j \in [1, d-1]$ **do**
  $(\boldsymbol{y}_j, \pi_j) := \mathsf{F}(\boldsymbol{q}_{j-1}, \varepsilon)$
  Partially simulate $\mathsf{D}^{f_{n-1}}$ on $\Big((\boldsymbol{q}_0, \boldsymbol{y}_0), \cdots, (\boldsymbol{q}_{j-1}, \boldsymbol{y}_{j-1})\Big)$ to obtain $\boldsymbol{q}_j$

▷ **Batching**

Simulate $\mathsf{D}^{f_{n-1}}$ on $\boldsymbol{\mu} := ((\boldsymbol{q}_0, \boldsymbol{y}_0), \cdots, (\boldsymbol{q}_{d-1}, \boldsymbol{y}_{d-1}))$ to obtain $\boldsymbol{y}$
if $\tau = \varepsilon$ **then** $\tau := \boldsymbol{x}\boldsymbol{y}\boldsymbol{\mu}$ **else** $\tau := \tau\boldsymbol{\mu}$
Compute challenge $r := H(\tau)$
Compute $(\boldsymbol{x}_d, \boldsymbol{y}_d) := \widetilde{\mathsf{B}}(\boldsymbol{\mu}; r)$

▷ **Recursive proof-merging**

$(\boldsymbol{y}_d, \pi_d) := \mathsf{F}(\boldsymbol{x}_d, \tau)$
return $\pi := (\boldsymbol{y}, (\boldsymbol{\mu}, \pi_d))$

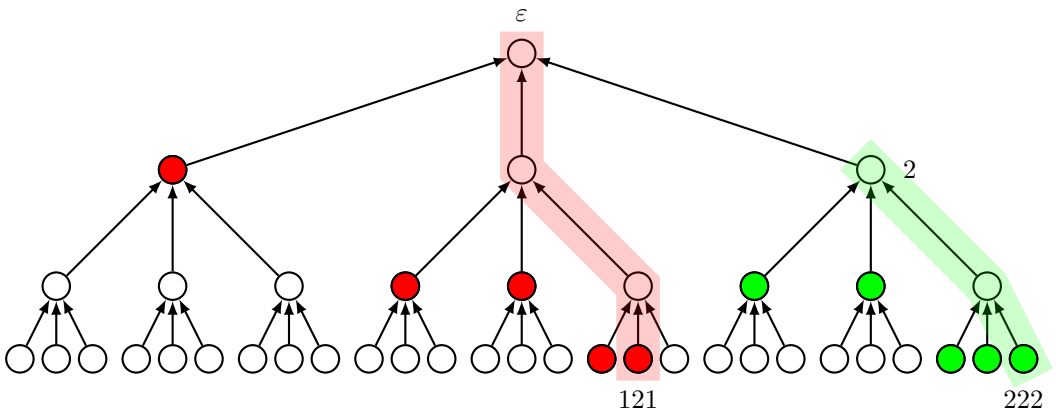


Figure 1: Inclusive ancestor, frontier and rooted frontier in a perfect 3-ary tree of depth 3. The inclusive ancestors of the leaf 121 ($\{\varepsilon, 1, 12, 121\}$) is shown in the path from 121 to root (shaded red). Its frontier is coloured red ($\mathsf{frontier}(121) = \{0, 10, 11, 120, 121\}$). The path from the leaf 222 to an ancestor 2 is shaded green; its rooted frontier till 2 is coloured green ($\mathsf{frontier}^*(222, 2) = \{222, 22, 2\}$).

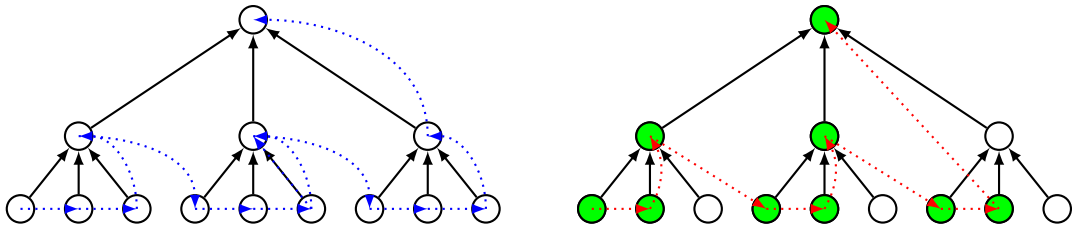
Definition 5.3 (left siblings, inclusive ancestor, frontier and rooted frontier)**.**

1. The *left siblings* of a vertex $v =: v_0 \cdots v_{\ell-1} v_\ell \in [0, d]^{\leq n}$ is the tuple of vertices
$$(v_0 \cdots v_{\ell-1} 0, \cdots, v_0 \cdots v_{\ell-1}(v_\ell - 1)) .$$
2. The *inclusive ancestors* (see Fig. 1) of a vertex $v =: v_0 \cdots v_\ell \in [0, d]^{\leq n}$ consists of a tuple of vertices with (i) the ancestors of v and (ii) v itself, i.e.:
$$(v_0, v_0 v_1, \cdots, v_0 v_1 \cdots v_{\ell-1}, v_0 \cdots v_\ell = v) .$$
3. The *frontier* of a vertex $v =: v_0 \cdots v_\ell \in [0, d]^{\leq n}$ (see Fig. 1) is the tuple of vertices consisting of (i) the left siblings of each inclusive ancestor of v and (ii) v itself (in that order).[13] We denote it by $\mathsf{frontier}(v)$.
4. More generally, the *rooted frontier* of $v =: v_0 \cdots v_\ell \in [0, d]^{\leq n}$ *till* an ancestor $v^* := v_0 \cdots v_{\ell'}$, $\ell' < \ell$, is the frontier of v restricted to the sub-tree rooted at v^* (see Fig. 1). We denote it by $\mathsf{frontier}^*(v, v^*)$. Note that for any v and its ancestor v^*, $\mathsf{frontier}^*(v, v^*) \subseteq \mathsf{frontier}(v)$.

Overview of RSVL. Recall that the explicit description of our RSVL instance RSVL is obtained by simulating F (Algorithm 1) incrementally, which yields an incrementally-verifiable computation (IVC).[14] To this end, the standard line in RSVL maintains a one-to-one correspondence with the steps of the IVC – in particular, its i-th vertex is the *state* of IVC on step i. As we will see, the successor S increments the state of IVC by one step, whereas the verifier V simply validates that

[13] We use a slightly different definition from [EFKP20]. In [EFKP20], the frontier of v is defined simply as the tuple of vertices consisting of the left siblings of each *inclusive ancestor* of v.

[14] To be precise, it is a special-purpose IVC where the underlying computation, itself, is the computation of a $\Pi_{\mathrm{FS},\mathcal{H}}$ *proof*. It does not qualify as an IVC for computation of f since the efficiency criteria from [Val08] are not met.



(a) (b)

Figure 2: A perfect 3-ary tree of depth 2 and its traversals: (a) the depth-first search (in blue) and (b) the restricted depth-first search (in red); the *left children* in the tree are highlighted in green.

the contents of a state are valid for that step. To explain this correspondence[15] more formally, it is useful to first define the *recursion tree*[16] of F, denoted $T = T_{\mathsf{F},x}$ for a parameter $x \in \mathcal{X}_n$, which has the following characteristics:

- It is a *labelled* tree with topology $T_{n,d+1}$, where $n = |x|$ and f is d-downward self-reducible.
- Each vertex $v \in V(T)$ represents a call to F on $\boldsymbol{x}_v$ that resulted in output $(\boldsymbol{y}_v, \pi_v)$ and can be therefore associated with the label $(v, \boldsymbol{x}_v, \boldsymbol{y}_v, \pi_v)$, where

  $$\pi_v = \pi(\boldsymbol{x}_v \xrightarrow{f_{n-|v|}} \boldsymbol{y}_v).$$

  In particular the root ε represents the initial call $\mathsf{F}(\boldsymbol{x}_\varepsilon)$, where $\boldsymbol{x}_\varepsilon = (x, \cdots, x) \in \mathcal{X}_n^k$. This connection is made explicit in Algorithm 5, which can be found in the appendix.
- The d left children $v0, \cdots, v(d-1)$ of an internal vertex $v \in V(T)$ represent the d-many recursive outlining calls (Line 5); its rightmost child vd represents the recursive proof-merging call (Line 11).
- Note that the proof in the label of a rightmost child v in T is *partial* in the sense that the merging of the $\Pi_{\mathrm{FS},\mathcal{H}}$ proof is only completed upstream at one of v's ancestor.[17]

The standard line now corresponds to the depth-first search (DFS) of T *restricted* to its left children $V(T) \setminus [0,d]^{<n} d$ which, recall, contain final $\Pi_{\mathrm{FS},\mathcal{H}}$ proofs (see Fig. 2.(b)):

$$0^n \to \cdots \to 0^{n-1}(d-1) \to 0^{n-2}1 \to \cdots \to d^{n-1}(d-1) \to \varepsilon. \tag{10}$$

In particular, it is the sequence of states

$$\boldsymbol{s}_{0^n} \to \cdots \to \boldsymbol{s}_{0^{n-1}(d-1)} \to \boldsymbol{s}_{0^{n-2}1} \to \cdots \to \boldsymbol{s}_{d^{n-1}(d-1)} \to \boldsymbol{s}_\varepsilon, \tag{11}$$

[15] There is a slight difference in this correspondence in [EFKP20] and [CHK+19a, CHK+19b]. In [CHK+19a, CHK+19b], the one-to-one correspondence is with (all) *vertices* of the recursion tree and therefore some of the labels will contain *partial* proofs (which, however, can be checked for consistency). In [EFKP20], the one-to-one correspondence is with the *leaves* of the recursion tree (or equivalently, with its left children as we use) and it is ensured that there are only *complete* proofs in the labels. We prefer to use the [EFKP20] approach.

[16] [EFKP20] use the term *puzzle tree* to refer to a similar object.

[17] To be precise, the merging is completed at v^*, the nearest ancestor of v that is *not* a rightmost child. That is, if $v =: v_0 \cdots v_{\ell'} d \cdots d v_\ell$, $\ell' < \ell \leq n-1$ and $v_{\ell'} \neq d$, then $v^* := v_0 \cdots v_{\ell'}$.

where $\boldsymbol{s}_v \in \{0,1\}^m$, $v \in V(T) \setminus [0,d]^{<n}\, d$, consists of label of v's frontier vertices, i.e.,

$$\boldsymbol{s}_v := \{(u, \boldsymbol{x}_u, \boldsymbol{y}_u, \pi_u)\}_{u \in \mathsf{frontier}(v)}. \tag{12}$$

Recall, here, that the label $(u, \boldsymbol{x}_u, \boldsymbol{y}_u, \pi_u)$ encodes the outlining step $(\boldsymbol{x}_u, \boldsymbol{y}_u)$ at the vertex $u \in V(T)$ and π_u certifies this step. Note that when F's computation is at $v \in V(T)$, the (recursive) calls corresponding frontier vertices of v are exactly the ones that are in its 'stack trace'. Therefore Eq. (11) captures the evolution of computation of F starting from input to output (where we assume that partial proofs are merged all the way). In particular, if i is the *index* of v in the restricted DFS over T – by which we mean that v is the i-th vertex visited in the traversal – then $\boldsymbol{s}_v$ is the state of i-th step of the IVC. Note that the length of the computation – and hence, the standard line – is

$$L := L(n,d) := \sum_{l=0}^{n} (d+1)^l - \sum_{l=0}^{n-1} (d+1)^l = (d+1)^n. \tag{13}$$

In the next two sections (Sections 5.1.3 and 5.1.4) we will explain how the verifier and successor circuits operate on such an RSVL instance. As in [CHK+19a], we resort to a description using helper circuits $\{(\mathsf{S}_l, \mathsf{V}_l)\}_{l \in [0,n]}$ (Algorithms 3 and 4). $(\mathsf{S}_l, \mathsf{V}_l)$, implements succession and verification, respectively, for inputs of size l and these are described recursively using $(\mathsf{S}_{l-1}, \mathsf{V}_{l-1})$. The description of $\mathsf{RSVL} := (\mathsf{S}, \mathsf{V}, L, \boldsymbol{s}_{0^n})$ is given in Algorithm 2.

Algorithm 2 The RSVL instance $\mathsf{RSVL} = (\mathsf{S}, \mathsf{V}, L = (d+1)^n)$. The descriptions of S_n and V_n can be found in Algorithms 3 and 4, respectively.

procedure $\mathsf{S}(\boldsymbol{s})$
hardwired Same items as in Algorithm 1 and additionally
1. an element $x \in \mathcal{X}_n$
2. description of the verifier $\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V}$

input State $\boldsymbol{s} \in \{0,1\}^m$ parsed as in Eq. (14)
output Next state $\boldsymbol{s}' \in \{0,1\}^m$
Let $i \in [0, 2^m - 1]$ denote the index of the $v_\ell \in \boldsymbol{v}$ in restricted DFS on T
if $\mathsf{V}(\boldsymbol{s}, i)$ rejects **then** return $\boldsymbol{s}$ and halt
else return $\boldsymbol{s}' := \mathsf{S}_n(\varepsilon, \boldsymbol{s}, (x, \cdots, x), \varepsilon)$ and halt

procedure $\mathsf{V}(\boldsymbol{s}, i)$
hardwired Same items as in S
input State $\boldsymbol{s} \in \{0,1\}^m$ parsed as in Eq. (14) and an index $i \in [0, 2^m - 1]$
if $i > L$ **then** reject and halt
▷ **Check well-formedness**
if the index of v_ℓ in restricted DFS on T is not i **then** reject and halt
if $\mathsf{frontier}(v_\ell) \neq \boldsymbol{v}$ **then** reject and halt
▷ **Verify recursively**
if $\mathsf{V}_n((x, \cdots, x), \varepsilon, \varepsilon)$, hardwired with $\boldsymbol{s}$, rejects **then** reject and halt
else accept and halt

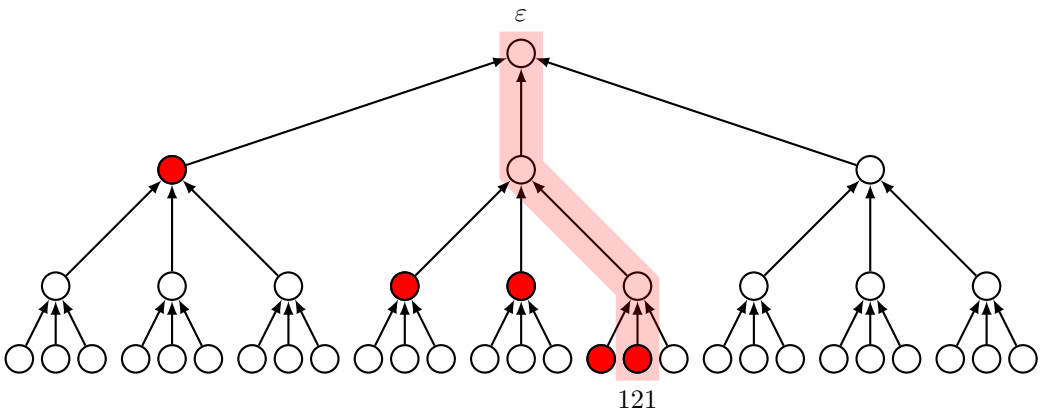


Figure 3: Checking a state's validity. The state of the IVC when F's recursion is at 121 (i.e., 17-th step) is $\boldsymbol{s}_{121} := \{(v, \boldsymbol{x}_v, \boldsymbol{y}_v, \pi_v)\}_{v\in\{0,10,11,120,121\}}$. The label $(0, \boldsymbol{x}_0, \boldsymbol{y}_0, \pi_0)$ is *final* for level 2 and is, hence, verified by V_3 using $\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V}$. The sub-state $\boldsymbol{s}_a := \boldsymbol{s}_{121} \setminus \{(0, \boldsymbol{x}_0, \boldsymbol{y}_0, \pi_0)\}$ corresponds to the *active outline* computation at 1 and its verification is hence, delegated to V_2. Further one level down, the sub-state $((120, \boldsymbol{x}_{120}, \boldsymbol{y}_{120}, \pi_{120}), (121, \boldsymbol{x}_{121}, \boldsymbol{y}_{121}, \pi_{121}))$ classifies for V_2 as *active proof-merging* at 12.

5.1.3 Checking a State's Validity

A formal description of V and $\{\mathsf{V}_l\}_{l\in[0,n]}$ is given in Algorithms 2 and 3, respectively; next, we give an informal overview (refer to Fig. 3). The verifier circuit V takes as input a state $\boldsymbol{s} \in \{0,1\}^m$ and an index $i \in [0, L]$ (since V rejects every $i > L$) and accepts only if $\boldsymbol{s}$ passes as a state in the i-th step of the IVC. It carries this out essentially by *retracing* the computation of $\mathsf{F}(\boldsymbol{x}_\varepsilon, \varepsilon)$ till step i using the certified (possibly partial) outlines present in $\boldsymbol{s}$. To this end, for $\boldsymbol{v} =: (v_0, \cdots, v_\ell)$ with $\ell < n \cdot d$, V first parses $\boldsymbol{s}$ as as a tuple of labels

$$\boldsymbol{s} =: \{(v, \boldsymbol{x}_v, \boldsymbol{y}_v, \pi_v)\}_{v\in\boldsymbol{v}}, \text{ where } (v, \boldsymbol{x}_v, \boldsymbol{y}_v, \pi_v) \in V(T) \times \mathcal{X}^k \times \mathcal{Y}^k \times (\mathcal{X} \times \mathcal{Y})^{\leq d\cdot n} \tag{14}$$

As noted earlier, for a well-formed state it must be the case – as in states belonging to Eq. (11) – that $\boldsymbol{v} = \mathsf{frontier}(v_\ell)$ (check on Line 8). Moreover, if $\boldsymbol{s}$ indeed corresponds to i-th state of the IVC then v_ℓ must have index i in the restricted DFS (check on Line 7). Next, it delegates the verification of $\boldsymbol{s}$'s contents to the recursive helper procedure V_n (Line 9).

For a level $l \in [0, n]$, the labels in $\boldsymbol{s}$ can correspond to either a *final* level-$(l-1)$ outlining step (Line 6), an *active* level-$(l-1)$ outlining step (Line 9) or an *active* proof-merging step (Line 15). V_l verifies the final outlines using $\Pi_{\mathrm{FS},\mathcal{H}}$'s verifier; on the other hand, it uses V_{l-1} to recursively verify the 'sub-state' $\boldsymbol{s}_a \subseteq \boldsymbol{s}$ that corresponds to active outline or proof-merging. The reason that the verification in the latter case has to be carried out recursively is that a single proof (for level $l-1$) becomes available only upon completion of the active call.

5.1.4 Computing the Successor

A formal description of S and S_n is given in Algorithms 2 and 4, respectively; next we give an informal overview (refer to Fig. 4). The input state $\boldsymbol{s} \in \{0,1\}^m$ to the successor circuit S is parsed as in Eq. (14). Suppose that v_ℓ has index i in the restricted DFS of T. On a high level, after

Algorithm 3 Recursive description of helper verifier circuits $\{\mathsf{V}_l\}_{l\in[0,n]}$

hardwired Same items as in Algorithm 2 and a state $\boldsymbol{s} \in \{0,1\}^m$ parsed as in Eq. (14)
input

1. Vertex $v \in [0,d]^{n-l}$
2. Current input $\boldsymbol{x}_v \in \mathcal{X}_l^k$
3. Transcript $\tau_v \in \mathcal{X}^k \times \mathcal{Y}^k \times (\mathcal{X}\times\mathcal{Y})^{\leq d\cdot n}$

procedure $\mathsf{V}_l(v, \boldsymbol{x}_v, \tau_v)$

▷ **Verify final outlines**

Invoke $\mathbf{D}^{f_{l-1}}$ on input $\boldsymbol{x}_v$ to obtain $\boldsymbol{q}_{v0}$
Set $j := 0$
while $vj \in \boldsymbol{v}$ **do**
  if $\boldsymbol{q}_{vj} \neq \boldsymbol{x}_{vj}$ **then** reject and return
  if $\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V}(\boldsymbol{x}_{vj} \xrightarrow{f_{l-1}} \boldsymbol{y}_{vj}, \pi_{vj})$ rejects **then** reject and return
  Partially simulate $\mathbf{D}^{f_{l-1}}$ on $\Big((\boldsymbol{x}_{v0},\boldsymbol{y}_{v0}),\cdots,(\boldsymbol{x}_{vj},\boldsymbol{y}_{vj})\Big)$ to obtain $\boldsymbol{q}_{v(j+1)}$
  Increment $j := j+1$

▷ **Recursively verify active outline**

if $j < d$ and $\mathsf{V}_{l-1}(\boldsymbol{s}, vj, \boldsymbol{x}_{vj}, \varepsilon)$ rejects **then** reject and return

▷ **Recursively verify active proof-merging**

Simulate $\mathbf{D}^{f_{l-1}}$ on $\boldsymbol{\mu}_v := \Big((\boldsymbol{q}_{v0},\boldsymbol{y}_{v0}),\cdots,(\boldsymbol{q}_{v(d-1)},\boldsymbol{y}_{v(d-1)})\Big)$ to obtain $\boldsymbol{y}_v$
Set $\boldsymbol{\mu}_v := \Big((\boldsymbol{x}_{v0},\boldsymbol{y}_{v0}),\cdots,(\boldsymbol{x}_{v(d-1)},\boldsymbol{y}_{v(d-1)})\Big)$
if $\tau_v = \varepsilon$ **then** $\tau_{vd} := \boldsymbol{x}_v\boldsymbol{y}_v\boldsymbol{\mu}$ **else** $\tau_{vd} := \tau_v\boldsymbol{\mu}_v$
Compute challenge $r := H(\tau_{vd})$
Compute $(\boldsymbol{x}_{vd},\boldsymbol{y}_{vd}) := \widetilde{\mathsf{B}}(\boldsymbol{\mu}_v; r)$
if $\mathsf{V}_{l-1}(\boldsymbol{s}, vd, \boldsymbol{x}_{vd}, \tau_{vd})$ rejects **then** reject and return
Accept and return

procedure $\mathsf{V}_0(v, \boldsymbol{x}_v, \tau_v)$ ▷ **Base case**
  if $\boldsymbol{y}_v \neq \boldsymbol{f}(\boldsymbol{x}_v)$ **then** reject and return
  else accept and return

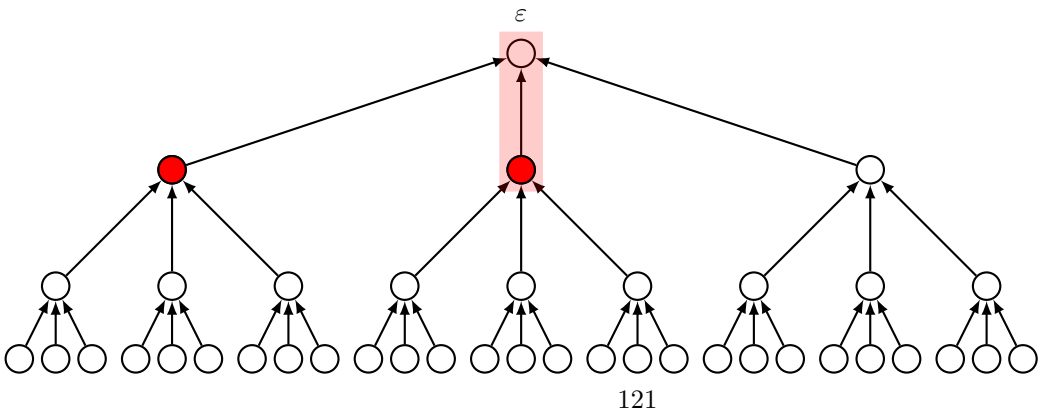


Figure 4: Computing the successor. The state of the IVC when the state $\boldsymbol{s}_{121}$ (see Fig. 3) is incremented is $\boldsymbol{s}_1 := ((0, \boldsymbol{x}_0, \boldsymbol{y}_0, \pi_0), (1, \boldsymbol{x}_1, \boldsymbol{y}_1, \pi_1))$. The labels $\boldsymbol{s}_a := \{(v, \boldsymbol{x}_v, \boldsymbol{y}_v, \pi_v)\}_{v \in \{10,11,120,121\}}$ in $\boldsymbol{s}_{121}$ correspond to the *active outline* computation at 1. Hence S_3 delegates its succession to S_2. For S_2, the labels $((120, \boldsymbol{x}_{120}, \boldsymbol{y}_{120}, \pi_{120}), (121, \boldsymbol{x}_{121}, \boldsymbol{y}_{121}, \pi_{121}))$ classify as *active proof-merging* at 12. Upon its increment by S_1, this proof merging at 12 is complete and hence S_2 merges to produce the label $\boldsymbol{s}'_a := (1, \boldsymbol{x}_1, \boldsymbol{y}_1, \pi_1)$. The new state is now obtained by replacing $\boldsymbol{s}_a$ with $\boldsymbol{s}'_a$ in $\boldsymbol{s}_{121}$, which yields $\boldsymbol{s}_1$.

validating using V that $\boldsymbol{s}$ passes as a state in the i-th step of the IVC, S increments to the state corresponding to $i+1$-th step, denoted $\boldsymbol{s}'$, using the helper circuit S_n. The circuit S_n first invokes S_{n-1} to recursively increment the sub-state $\boldsymbol{s}_a \subseteq \boldsymbol{s}$ belonging to the sole *active* recursion on step $n-1$. The sub-state $\boldsymbol{s}_a$ belongs either to the active outlining step (Line 6) or the active proof-merging step (Line 13). Finally, in case that the active proof-merging step culminates, it merges the resulting proof upstream (Line 15).

5.2 Analysis

We first claim in Claim 5.4 that S faithfully simulates F. We then prove the hardness of the RSVL instance in Theorem 5.6 assuming hardness of f (Assumption 5.5).

Claim 5.4 (Correctness of RSVL)**.** S *correctly simulates* F *using the restricted DFS on* T*: i.e.,* $\mathsf{S}^i(\boldsymbol{s}_{0^n}) = \boldsymbol{s}_v$ *for every* $i \in [1, L]$*, where* $v \in V(T) \setminus [0, d]^{<n}$ d *denotes the vertex with index* i *in the restricted DFS and* $\boldsymbol{s}_{0^n}$ *and* $\boldsymbol{s}_v$ *are as defined in Eq. (12).*

Proof. By construction of S in Algorithm 2, the claim reduces to showing $\mathsf{S}^i_n(\varepsilon, \boldsymbol{s}_{0^n}, \varepsilon) = \boldsymbol{s}_v$. We prove something stronger: for every $j \in [0, d]$ and $\tau \in \{0,1\}^*$

$$\mathsf{S}^i_n(j, \boldsymbol{s}_{0^n}, \tau) = \boldsymbol{s}_v, \tag{15}$$

where $\boldsymbol{s}_v$ is now defined as in Eq. (12) *but* with respect to the computation of $\mathsf{F}(j, \boldsymbol{x}, \tau)$ (see Algorithm 5). We proceed inductively on $\{\mathsf{S}_l\}_{l \in [0,n]}$. For induction hypothesis, let's assume that Eq. (15) holds for S_{n-1}. We now prove Eq. (15) by appealing to the induction hypothesis $d+1$ times, once for each interval

$$\left[1, \frac{L}{d+1}\right], \left[\frac{L}{d+1} + 1, \frac{2L}{d+1}\right], \cdots, \left[\frac{dL}{d+1}, L\right],$$

Algorithm 4 Recursive description of helper successor circuits $\{\mathsf{S}_l\}_{l\in[0,l]}$

procedure $\mathsf{S}_l(v, \boldsymbol{s}, \tau_v)$
hardwired Same as Algorithm 2
input Same as Algorithm 3 and a state $\boldsymbol{s} \in \{0,1\}^m$ parsed as in Eq. (14)
output Incremented state $\boldsymbol{s}' \in \{0,1\}^m$
  Set $j := 0$
  while $vj \in \boldsymbol{v}$ **do** increment $j := j + 1$
  Parse $\boldsymbol{s} =: (\boldsymbol{s}_f, \boldsymbol{s}_a)$ where $\boldsymbol{s}_a = \Big((vj, \boldsymbol{x}_{vj}, \boldsymbol{y}_{vj}, \pi_{vj}), \cdots, (v_\ell, \boldsymbol{x}_{v_\ell}, \boldsymbol{y}_{v_\ell}, \pi_{v_\ell})\Big)$
  ▷ **Increment active outline**
  if $j < d$ **then**
    Recursively compute $\boldsymbol{s}'_a := \mathsf{S}_{l-1}(vj, \boldsymbol{s}_a, \varepsilon)$
    Return $\boldsymbol{s}' := (\boldsymbol{s}_f, \boldsymbol{s}'_a)$
  ▷ **Increment active proof-merging**
  Simulate $\mathbf{D}^{f_{l-1}}$ on $\boldsymbol{\mu}_v := \Big((\boldsymbol{x}_{v0}, \boldsymbol{y}_{v0}), \cdots, (\boldsymbol{x}_{v(d-1)}, \boldsymbol{y}_{v(d-1)})\Big)$ to obtain $\boldsymbol{y}_v$
  Set $\boldsymbol{\mu}_v := \Big((\boldsymbol{x}_{v0}, \boldsymbol{y}_{v0}), \cdots, (\boldsymbol{x}_{v(d-1)}, \boldsymbol{y}_{v(d-1)})\Big)$
  if $\tau_v = \varepsilon$ **then** $\tau_{vd} := \boldsymbol{x}_v \boldsymbol{y}_v \boldsymbol{\mu}$ **else** $\tau_{vd} := \tau_v \boldsymbol{\mu}_v$
  Compute challenge $r := H(\tau_{vd})$
  Compute $(\boldsymbol{x}_{vd}, \boldsymbol{y}_{vd}) := \widetilde{\mathsf{B}}(\boldsymbol{\mu}_v; r)$
  Recursively compute $\boldsymbol{s}'_a := \mathsf{S}_{l-1}(vd, \boldsymbol{s}_a, \tau_{vd})$
  ▷ **Merge if proof complete**
  if $\boldsymbol{s}'_a$ parses as $(vd, \boldsymbol{x}_{vd}, \boldsymbol{y}_{vd}, \pi_{vd})$ **then**
    return $\boldsymbol{s}' := (v, \boldsymbol{x}_v, \boldsymbol{y}_v, \pi_v)$, where $\pi_v := \left(\left((\boldsymbol{x}_{v0}, \boldsymbol{y}_{v0}), \cdots, (\boldsymbol{x}_{vd}, \boldsymbol{y}_{vd})\right), \pi_{vd}\right)$
  else return $\boldsymbol{s}' := (\boldsymbol{s}_f, \boldsymbol{s}'_a)$

procedure $\mathsf{S}_0(v, \boldsymbol{s}, \tau_v)$ ▷ **Base case**
  Return $\boldsymbol{s}$

in that order. The claims for the first d intervals follow by applying the induction hypothesis to the computation $\mathsf{F}(j, \boldsymbol{q}_j, \varepsilon)$, where $j \in [0, d-1]$ and $\boldsymbol{q}_j$ is the j-th query made by $\mathbf{D}^{f_{l-1}}$ in its partial simulation on input $\boldsymbol{x}_\varepsilon$, and by observing that (i) the restricted DFS on T in the invocation j only affects the labels in the subtree rooted at j and (ii) the final labels are left unaffected by the S_{n-1} (Lines 7 and 16). The claim follows by a final invocation of the induction hypothesis to $\mathsf{F}(d, \boldsymbol{x}_d, \tau)$, where $\boldsymbol{x}_d$ results from the batch reduction and τ is the current transcript, and observing that the label assembled and returned by S_n (Line 15) corresponds exactly to $\boldsymbol{s}_\varepsilon$. □

Assumption 5.5 (Hardness of f)**.** Let $f : \mathcal{X} \to \mathcal{Y}$ be a function as defined in Section 3 and let X denote a sampler for $\mathcal{X}$. The function f is $(s(\lambda), \epsilon(\lambda))$-hard with respect to X if for every $s(\lambda)$-sized adversary $\mathsf{A} = \{\mathsf{A}_\lambda\}_{\lambda \in \mathbb{N}}$

$$\Pr_{\substack{x \leftarrow \mathsf{X}(1^\lambda) \\ y \leftarrow \mathsf{A}(x)}}[y = f(x)] = O(\epsilon(\lambda)).$$

Theorem 5.6 (Hardness of RSVL from f and $\Pi_{\mathrm{FS},\mathcal{H}}$)**.** *Let $k, d \in \mathbb{N}$ be parameters and $\lambda \in \mathbb{N}$ be a security parameter. Let*

- *$f : \mathcal{X} \to \mathcal{Y}$ be a d-query downwards self-reducible and dk-to-k batch-reducible function with sampler X; and*
- *$\Pi_{\mathrm{FS},\mathcal{H}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ denote the non-interactive outline-and-batch protocol for $\mathcal{L}^k_{f_n}$ from Corollary 3.4.*

Furthermore, for $H \leftarrow \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}(1^\lambda)$ and $x \leftarrow \mathsf{X}(1^\lambda)$, with $n := |x|$, define

$$m = m(d, k, |x|) \in \mathsf{poly}(d, k, |x|) \text{ and } L = L(d, k) := (d+1)^n, \tag{16}$$

and let

$$\mathsf{S} : \{0,1\}^m \to \{0,1\}^m \text{ and } \mathsf{V} : \{0,1\}^m \times [0, 2^m - 1] \to \{\mathsf{accept}, \mathsf{reject}\} \tag{17}$$

be as defined as in Algorithm 2, hardwired with $(f, H, \mathsf{D}, \widetilde{\mathsf{B}}, x, \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V})$. If f is hard with respect to X and $\Pi_{\mathrm{FS},\mathcal{H}}$ is (adaptively) unambiguously sound argument, then $\mathsf{RSVL} := (\mathsf{S}, \mathsf{V}, L, \boldsymbol{s}_{0^n})$ constitutes a hard distribution of RSVL.

On instantiating f with IS as sampled in Assumption 4.3 and $\Pi_{\mathrm{FS},\mathcal{H}}$ with non-interactive argument from Corollary 4.10, we get the following corollary to Theorem 5.6.

Corollary 5.7 (Hardness of RSVL from IS and LWE)**.** *For a security parameter $\lambda \in \mathbb{N}$, let $(\mathbb{G}_\lambda, g, T)$ be sampled as in Assumption 4.8, which defines $f_n(g, T) := g^{2^T}$ for $n := \log(T)$. Also, let $\Pi_{\mathrm{FS},\mathcal{H}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ denote the non-interactive protocol for $\mathcal{L}^k_{\mathbb{G}_\lambda}$ from Corollary 4.10, which implies $k \in \lambda^{O(1)}$ and $d = 2$. Furthermore, for $H \leftarrow \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}(1^\lambda)$, define*

$$m = m(n, k, \lambda) := n^2 k \cdot \mathsf{poly}(\lambda) \text{ and } L = L(n) = 3^n, \tag{18}$$

and let

$$\mathsf{S} : \{0,1\}^m \to \{0,1\}^m \text{ and } \mathsf{V} : \{0,1\}^m \times [0, 2^m - 1] \to \{\mathsf{accept}, \mathsf{reject}\} \tag{19}$$

be defined as in Algorithm 2, hardwired with $((\mathbb{G}_\lambda, g, T), H, \mathsf{D}, \widetilde{\mathsf{B}}, \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V})$. If Assumptions 2.10 and 4.8 hold then $\mathsf{RSVL} := (\mathsf{S}, \mathsf{V}, L, \boldsymbol{s}_{0^n})$ constitutes a hard distribution of RSVL.

Proof (of Theorem 5.6). Suppose for contradiction that there exists a $\mathsf{poly}(\lambda)$-sized algorithm $\mathsf{A} = \{\mathsf{A}_\lambda\}_{\lambda\in\mathbb{N}}$ and a polynomial $p(\cdot)$ such that A solves RSVL for infinitely-many security parameters $\lambda \in \mathbb{N}$ with probability at least $1/p(\cdot)$; fix such a λ. This implies by an averaging argument that A finds either the sink or a false positive (see Definition 2.7) for λ with probability at least $1/2p(\lambda)$ (which is also non-negligible). Given an adversary that finds the standard sink, we show in Claim 5.8 that it is possible to break the hardness of f; given an adversary that finds a false positive, we show in Claim 5.9 how to break $\Pi_{\mathrm{FS},\mathcal{H}}$'s unambiguous soundness.

Claim 5.8 (Reduction from hardness of f)**.** *Given an adversary A_1 that finds the standard sink in RSVL with probability at least $p'(\lambda)$, one can break the hardness of f with same probability.*

Proof. Given a challenge input $x \in \mathcal{X}$, the reduction samples a CRS $H \leftarrow \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}(1^\lambda)$ and sends $(\mathsf{S}, \mathsf{V}, L, \boldsymbol{s}_{0^n})$ hardwired with $(f, H, \mathsf{D}, \widetilde{\mathsf{B}}, x, \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V})$ to A_1. A_1 returns a state $\boldsymbol{s}$ such that $\mathsf{RSVL.V}(\boldsymbol{s}, L) = 1$. By completeness, this node can be parsed as

$$((\varepsilon, (x, \cdots, x), (y, \cdots, y), \pi))$$

with $y = f(x)$. The reduction simply outputs y as the solution to its challenge. □

Claim 5.9 (Reduction from unambiguous soundness of $\Pi_{\mathrm{FS},\mathcal{H}}$)**.** *Given an adversary A_2 that finds a false positive in RSVL with probability at least $p'(\lambda)$, one can break $\Pi_{\mathrm{FS},\mathcal{H}}$'s unambiguous soundness with probability at least $p'(\lambda)/(dn)$.*

Proof. Given a challenge CRS H, the reduction samples $x \leftarrow \mathsf{X}(1^\lambda)$ and sends $(\mathsf{S}, \mathsf{V}, L, \boldsymbol{s}_{0^n})$ hardwired with $(f, H, \mathsf{D}, \widetilde{\mathsf{B}}, x, \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V})$ to A_2. A_2 returns $(\boldsymbol{s}^*, i^*)$, $i^* \in [0, L]$, such that $\mathsf{RSVL.V}(\boldsymbol{s}^*, i^*) = 1$ but $\boldsymbol{s}^* \neq \mathsf{RSVL.S}^{i^*}(0^m) =: \boldsymbol{s}$, where $\boldsymbol{s}$ denotes the i-th vertex on the prescribed line. (It is not necessary for the reduction to be able to efficiently compute $\boldsymbol{s}$.) Let's parse $\boldsymbol{s}^*$ as

$$((v_0^*, \boldsymbol{x}_0^*, \boldsymbol{y}_0^*, \pi_0^*), \cdots, (v_\ell^*, \boldsymbol{x}_\ell^*, \boldsymbol{y}_\ell^*, \pi_\ell^*)), \tag{20}$$

and $\boldsymbol{s}$ as in Eq. (14). In order to prove the claim, we make a case distinction. In both cases, we first show that there *exists* an index $j \in [0, \ell]$ such that the proof and statement in the j-th label breaks unambiguity or soundness of $\Pi_{\mathrm{FS},\mathcal{H}}$ – the reduction then simply *guesses* this index at random and outputs the constituent proof and statement.

- *All the statements in $\boldsymbol{s}$ and $\boldsymbol{s}^*$ match.* In this case, since $\boldsymbol{s} \neq \boldsymbol{s}^*$, there exists at least one index $j \in [0, \ell]$ such that $\pi_j^* \neq \pi_j$ but $(\boldsymbol{x}_j^*, \boldsymbol{y}_j^*) = (\boldsymbol{x}_j, \boldsymbol{y}_j)$ is a true statement. Therefore π_j^* breaks unambiguity of $\Pi_{\mathrm{FS},\mathcal{H}}$. The reduction *guesses* this index j and returns

$$(\pi_j^*, \boldsymbol{x}_j \xrightarrow{f_{n-|v_j|}} \boldsymbol{y}_j).$$

- *Some statements are different and $j \in [0, \ell]$ is the first such index.* In this case, we claim that

$$\boldsymbol{x}_j^* \xrightarrow{f_{n-|v_j^*|}} \boldsymbol{y}_j^*. \tag{21}$$

  is false, but since $\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V}$ accepts π_j^*, soundness of $\Pi_{\mathrm{FS},\mathcal{H}}$ is broken. To see this, first note that (by assumption) the statements in labels less than j in both $\boldsymbol{s}$ and $\boldsymbol{s}^*$ are same and therefore the verifier ends up recomputing $\boldsymbol{x}_j^* = \boldsymbol{x}_j$. Therefore, if $\boldsymbol{x}_j^* \neq \boldsymbol{x}_j$ then V outright

rejects the proof π_j^*. Moreover, since $v_j = v_j^*$ (if $(v_0, \cdots, v_\ell) \neq (v_0^*, \cdots, v_\ell^*)$ then V rejects) and $\boldsymbol{y}^* \neq \boldsymbol{y}$, the statement in Eq. (21) is false but π_j^* accepts. Therefore the reduction *guesses* this index j and returns π_j^* and the statement in Eq. (21).

In both cases, the reduction incurs a loss of $1/(dn)$, which is the probability with which it correctly guesses j. □

□

Remark 8 (On loss in tightness). In case $\Pi_{\mathrm{FS},\mathcal{H}}$ satisfies the property the prescribed proofs can be *efficiently* computed given some trapdoor information –as is the case in [Pie19, BHR+21] when instantiated in trapdoor groups of unknown order (Definition 4.9)– it is possible to avoid the $1/(dn)$ loss in security since the trapdoor allows efficiently computing the index j. Thus Corollary 5.7 does not incur the $1/(dn)$ loss.

6 Unique VDF

In this section, we present our construction of unique VDFs based on the non-interactive argument for iterated squaring from Section 4.3. We first define unique VDFs; the definition is adapted from [BBBF18] to account for uniqueness of proofs, which we capture by requiring it to be hard for an adversary to come up with a output-proof pair that is different from the "prescribed" pair output by the evaluation algorithm.

Definition 6.1 (Verifiable delay functions [BBBF18])**.** A VDF is a triple of algorithms $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$, where Setup is probabilistic, whereas Eval and Verify are deterministic.

- $\mathsf{Setup}(1^\lambda, T) \to \mathsf{pp}$. On input a statistical security parameter $\lambda \in \mathbb{N}$ (in unary) and a time parameter $T \in \mathbb{N}$, the setup algorithm outputs public parameters pp. The public parameters define the domain $\mathcal{X}$ and range $\mathcal{Y}$ of the VDF.
- $\mathsf{Eval}(\mathsf{pp}, x) \to (y, \pi)$. On input $x \in \mathcal{X}$, the evaluation algorithm outputs (y, π), where π is a proof that the output $y \in \mathcal{Y}$ has been correctly computed.
- $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) \to \{\mathsf{accept}, \mathsf{reject}\}$. Given as input a tuple (x, y, π) consisting of an input, an output and a proof, the verification algorithm outputs either accept or reject.

A unique VDF must satisfy the following four properties:

- **Efficiency.** Setup and Verify run in time $\mathsf{poly}(\log(T), \lambda)$. Eval should be able to compute the output y and its proof π in (sequential) time $T(1 + o(1))$.
- **Completeness.** Correctly-generated proofs must always accept, i.e., for any $\lambda, T \in \mathbb{N}$ and $x \in \mathcal{X}$
$$\Pr_{\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T)} [\mathsf{Verify}(\mathsf{pp}, x, \mathsf{Eval}(\mathsf{pp}, x)) = \mathsf{accept}] = 1$$
- **Unambiguous soundness.** A VDF is $(s(\lambda), \epsilon(\lambda))$-unambiguosly sound if for all $T \in \mathbb{N}$ and $s(\lambda)$-sized adversaries $\mathsf{A} = \{\mathsf{A}_\lambda\}_{\lambda \in \mathbb{N}}$
$$\Pr_{\substack{\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ (x, y^*, \pi^*) \leftarrow \mathsf{A}(1^\lambda, T, \mathsf{pp})}} [\mathsf{Verify}(\mathsf{pp}, x, y^*, \pi^*) = \mathsf{accept}, (y^*, \pi^*) \neq \mathsf{Eval}(\mathsf{pp}, x)] = O(\epsilon(\lambda)).$$

- **Sequentiality.** Let's consider a two-part adversary $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$, where A_1 is a $\mathsf{poly}(T, \lambda)$-size, randomised (pre-processing) algorithm and A_2 is a $\sigma(T)$-*sequential* time, $p(T)$-parallel algorithm. A VDF is (σ, p)-sequential if for all $T \in \mathbb{N}$

$$\Pr_{\substack{\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \mathsf{state} \leftarrow \mathsf{A}_1(\lambda, T, \mathsf{pp}) \\ x \leftarrow \mathcal{X}, y^* \leftarrow \mathsf{A}_2(\mathsf{state}, x)}} [(y, \pi) := \mathsf{Eval}(\mathsf{pp}, x), y^* = y] = \mathsf{negl}(\lambda).$$

Remark 9 (Comparison with [BBBF18]). In [BBBF18] Eval can be a randomised and parallel algorithm; for us it suffices that it is deterministic and sequential. In addition, we don't have to separate public parameters into evaluation and verification keys.

6.1 Construction

Let $\Pi_{\mathrm{FS},\mathcal{H}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ denote the non-interactive protocol obtained by applying the Fiat-Shamir transform the interactive protocol from [BHR+21] (Section 4.3). The construction of our VDF $\mathsf{VDF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ follows similar template to [Pie19]. That is:

- The public parameters consist of the description of a group $\mathbb{G}_\lambda$ of unknown order sampled using $\mathsf{Setup}(1^\lambda)$ (Section 4.2) and a CRS H sampled using $\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}$.
- The evaluation algorithm, on input a group element $g \in \mathbb{G}_\lambda$, returns (i) $h = g^{2^T}$ by repeated squaring over $\mathbb{G}_\lambda$ and (ii) a proof π for the statement (g, h, T) computed as in $\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{P}$. We will explain in Claim 6.2 how h and π can be computed jointly in $T(1+o(1))$ group operations.
- The verification algorithm simply invokes $\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V}$.

6.2 Analysis

Completeness, unambiguous soundness, and sequentiality of this construction follow either by direct assumption or invoking Corollary 4.10. The efficiency of Setup and Verify are also immediate from the definition of the protocol. To prove that our protocol is a unique VDF, what remains is to analyze its *prover efficiency*.

First, we address a technical point: in order to obtain a VDF with prover efficiency $T(1+o(1))$ group operations, the prover must generate the outputs g^{2^T} and proof π *together*, rather than first computing g^{2^T} and then computing π. We show that this is indeed possible by analyzing the Section 4 prover algorithm when it is given access to the $T+1$ group elements $g, g^2, \ldots, g^{2^T}$ (or possibly some subset of them). In this setting, we show that the prover can generate π with $o(T)$ additional group operations.

Note that the naive prover algorithm, even given $g, g^2, \ldots, g^{2^T}$, requires roughly T additional group operations (because it must recursively run the 2-round reduction $\log(T) - 1$ additional times), but (following [Pie19]) a more efficient proving algorithm exists.

Claim 6.2. *The* Eval *algorithm in* VDF *can be implemented in* $T(1+o(1))$ *time and* $O(\sqrt{T} \cdot \mathsf{poly}(\lambda))$ *space.*

Proof. This can be proved following an analogous argument in [Pie19]: given all $T+1$ sequential squaring group elements $g_0 = g, g_1 = g^2, \ldots, g_T = g^{2^T}$, it is possible to compute all prover messages with $\mathsf{poly}(\lambda)\sqrt{T}$ additional group operations by:

1. Computing all prover messages from round $\frac{1}{2}\log(T)$ onwards with the naive prover algorithm, incurring an additive computational overhead of $\mathsf{poly}(\lambda)\sqrt{T}$, and
2. Computing all prover messages in the first $\frac{1}{2}\log(T)$ rounds as product-combinations of $\lambda \cdot \sqrt{T/\lambda}$ of the g_i, also incurring an additive overhead of $\mathsf{poly}(\lambda)\sqrt{T}$.

Property 1 follows directly from the description of the protocol: for these messages, the prover simply has to compute the "current" statements $\tilde{g}_i \xrightarrow{T'} \tilde{h}_i$ (for $T' = T/2^c$ after c recursive steps) and then compute $\tilde{g}_i^{2^{T'/2}}$ for each i. The statements themselves can be computed in time $\mathsf{poly}(\lambda, \log T)$, so the overall prover runtime for these steps is $\mathsf{poly}(\lambda, \log T) + \lambda \cdot T'/2 \leq \mathsf{poly}(\lambda) \cdot \sqrt{T}$.

To prove Property 2, we have to understand the structure of the (honest) prover messages in round $i \in [0, \log(T)/2]$ of $\Pi_{\mathrm{FS},\mathcal{H}}$. Note that each prover message in the first round is of the form

$$\left(g^{2^{T/2}}, \cdots, g^{2^{T/2}}\right).$$

Now recall how the batching reduction is carried out: given 2λ instances $\tilde{g}_1, \ldots, \tilde{g}_{2\lambda}$ for $f(\cdot, T)$ and 2λ candidate outputs $\tilde{h}_1, \ldots, \tilde{h}_{2\lambda}$, the batching reduction samples λ i.i.d. vectors $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_\lambda \leftarrow \{0,1\}^{2\lambda}$. The reduction then outputs λ statements about $f(\cdot, T)$:

$$\left(\prod_{j=1}^{2\lambda} \tilde{g}_j^{r_{1,j}} \xrightarrow{T} \prod_{j=1}^{2\lambda} \tilde{h}_j^{r_{1,j}}, \ldots, \prod_{j=1}^{2\lambda} \tilde{g}_j^{r_{\lambda,j}} \xrightarrow{T} \prod_{j=1}^{2\lambda} \tilde{h}_j^{r_{\lambda,j}}\right).$$

As a result, the prover message in the second round is of the form

$$\left(g^{a_{1,1}+a_{1,2}2^{T/2}}, \cdots, g^{a_{1,\lambda}+a_{2,\lambda}2^{T/2}}\right),$$

where $a_{j,\ell}$, $j \in [1,2]$ and $\ell \in [1,\lambda]$, can be efficiently computed given the vectors $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_\lambda$. Now, as in [Pie19], it can be inductively argued that each element in the prover message in round i can be computed using (at most) 2^i elements $\{g^{2^{T\cdot j/2^i}}\}_{j\in[1,2^i]}$. That is, the ℓ-th element is of the form

$$g^{\sum_{j\in[1,2^i]} a_{j,\ell} 2^{T\cdot j/2^s}},$$

where the $a_{j,\ell}$, $j \in [1, 2^i]$ and $\ell \in [1,\lambda]$, can be efficiently computed given $a_{j,\ell}$ and $\boldsymbol{r}_\ell$ from the previous round. The overall computational cost of generating these group elements is then (up to $\mathsf{poly}(\lambda)$ factors) that of generating the coefficients $\{a_{j,\ell}\}$. By inspection, these coefficients can be generated in time linear (up to $\mathsf{poly}(\lambda)$ factors) in the number of such coefficients, which is $\sum_{i'\leq i} 2^{i'}$. Property (2) then follows by setting $i = \log(T)/2$. □

Theorem 6.3 (VDF from IS and LWE)**.** *Assuming that IS is (T,p)-sequentially hard (Assumption 4.7) and LWE is polynomially-hard (Assumption 2.10),* VDF *is a $(T(1-o(1)),p)$-sequential, unambiguosly-sound VDF.*

Proof. Since evaluating the VDF requires solving IS, its sequentiality follows from that of IS. Unambiguous soundness follows unambiguous soundness of $\Pi_{\mathrm{FS},\mathcal{H}}$, which itself is based on Assumption 2.10, the polynomial hardness of LWE (Corollary 4.10). □

Remark 10. Note that it is possible to apply the ideas in [EFKP20] – i.e., (i) use $O()$-query downwards self-reducibility (instead of 2-query downwards self-reducibility) and (ii) prune the recursion tree for a few levels – to obtain continuous VDFs.

7 Conclusion and Open Problems

In this work, we demonstrated hardness in the class **PPAD** assuming the polynomial hardness of iterated squaring and LWE. Moreover, we (1) strengthened this result to show hardness in **UEOPL** $\subseteq$ **PPAD** (which is first cryptographic hardness shown for that class) and (2) constructed a unique VDF based on similar assumptions.

We briefly mention two interesting open questions that are closely related to this work:

- Can the iterated squaring hardness assumption be replaced by a weaker assumption such as the hardness of factoring? This seems plausible since to achieve **PPAD** hardness, it suffices for iterated squaring to be polynomially hard for *some* efficiently computable iteration parameter. This question was also posed in [CHK+19b].
- Can we show **PPAD**-hardness solely from polynomial hardness of LWE, and thus establish a (polynomially) tight hardness result for quantum algorithms? Currently, only [JKKZ21] demonstrates post-quantum hardness of **PPAD** (under sub-exponential LWE).

8 Acknowledgements

Nir Bitansky is a member of the checkpoint institute of information security and is supported by the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (grant agreement No. 101042417, acronym SPP), and by Len Blavatnik and the Blavatnik Family Foundation.

Arka Rai Choudhuri is supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by the Sloan Foundation, and Visa Inc. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

Chethan Kamath is supported by Azrieli International Postdoctoral Fellowship and ISF grants 484/18 and 1789/19. He thanks Alexandros Hollender and Ninad Rajagopal for discussions on the class **UEOPL** and Krzysztof Pietrzak for clarifications about unique VDFs.

Alex Lombardi is supported in part by DARPA under Agreement No. HR00112020023, a grant from MIT-IBM Watson AI, a grant from Analog Devices, a Microsoft Trustworthy AI grant, the Thornton Family Faculty Research Innovation Fellowship and a Charles M. Vest fellowship. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

Omer Paneth is a member of the Checkpoint Institute of Information Security and is supported by an Azrieli Faculty Fellowship, Len Blavatnik and the Blavatnik Foundation, the Blavatnik Interdisciplinary Cyber Research Center at Tel Aviv University, and ISF grant 1789/19.

Ron Rothblum was funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. 12

[Bac88] Eric Bach. How to generate factored random numbers. *SIAM Journal on Computing*, 17(2):179–193, 1988. 11, 21, 23

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018. 4, 38, 39

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001. 3

[BHR+21] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 123–152, Virtual Event, August 2021. Springer, Heidelberg. 7, 8, 9, 10, 20, 21, 23, 24, 38, 39

[BKM20] Zvika Brakerski, Venkata Koppula, and Tamer Mour. NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 738–767. Springer, Heidelberg, August 2020. 6

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In Venkatesan Guruswami, editor, *56th FOCS*, pages 1480–1498. IEEE Computer Society Press, October 2015. 3, 4

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011. 3

[CCH+18] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, and Ron D. Rothblum. Fiat-Shamir from simpler assumptions. Cryptology ePrint Archive, Report 2018/1004, 2018. https://eprint.iacr.org/2018/1004. 9, 16, 17

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019. 6, 14, 16

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-Shamir and correlation intractability from strong KDM-secure encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 91–122. Springer, Heidelberg, April / May 2018. 6

[CDT09] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM (JACM)*, 56(3):1–57, 2009. 3

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998. 8, 14

[CHK+19a] Arka Rai Choudhuri, Pavel Hubácek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a nash equilibrium is no easier than breaking Fiat-Shamir. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1103–1114. ACM Press, June 2019. 3, 5, 10, 13, 15, 16, 19, 25, 30, 31

[CHK+19b] Arka Rai Choudhuri, Pavel Hubacek, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. PPAD-hardness via iterated squaring modulo a composite. Cryptology ePrint Archive, Report 2019/667, 2019. https://eprint.iacr.org/2019/667. 3, 5, 6, 20, 21, 25, 30, 41

[CJJ21a] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 394–423, Virtual Event, August 2021. Springer, Heidelberg. 6

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for **P** from LWE. In *FOCS*, pages 68–79. IEEE, 2021. 6

[CKU20] Geoffroy Couteau, Shuichi Katsumata, and Bogdan Ursu. Non-interactive zero-knowledge in pairing-free groups from weaker assumptions. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 442–471. Springer, Heidelberg, May 2020. 6

[CLSY93] J. Y. Cai, R. J. Lipton, R. Sedgewick, and A. C. Yao. Towards uncheatable benchmarks. In *[1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 2–11, May 1993. 20

[CPV20] Michele Ciampi, Roberto Parisella, and Daniele Venturi. On adaptive security of delayed-input sigma protocols and fiat-shamir NIZKs. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 670–690. Springer, Heidelberg, September 2020. 6

[DGP09] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009. 3

[DP11] Constantinos Daskalakis and Christos H. Papadimitriou. Continuous local search. In Dana Randall, editor, *22nd SODA*, pages 790–804. ACM-SIAM, January 2011. 4, 48

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 125–154. Springer, Heidelberg, May 2020. 3, 5, 6, 20, 25, 27, 29, 30, 40

[FGHS21] John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: CLS = PPAD ∩ PLS. In *STOC*, pages 46–59. ACM, 2021. 48

[FGMS19] John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *ICALP 2019*, volume 132 of *LIPIcs*, pages 56:1–56:15. Schloss Dagstuhl, July 2019. 4, 10, 48, 49, 50

[FPS22] Cody Freitag, Rafael Pass, and Naomi Sirkin. Parallelizable delegation from LWE. Cryptology ePrint Archive, Report 2022/1025, 2022. https://eprint.iacr.org/2022/1025. 5, 7

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. 6

[FS00] Roger Fischlin and Claus-Peter Schnorr. Stronger security proofs for RSA and Rabin bits. *Journal of Cryptology*, 13(2):221–244, March 2000. 7, 21

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013. 3

[GHJ+22] Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Further collapses in TFNP. *CoRR*, abs/2202.07761, 2022. 48

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 579–604. Springer, Heidelberg, August 2016. 3, 4

[HHK+22] Charlotte Hoffmann, Pavel Hubáček, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Practical statistically-sound proofs of exponentiation in any group. Cryptology ePrint Archive, Report 2022/1021, 2022. https://eprint.iacr.org/2022/1021. 7

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In *EUROCRYPT 2022, Part II*, LNCS, pages 520–549. Springer, Heidelberg, June 2022. 6

[HK09] Dennis Hofheinz and Eike Kiltz. The group of signed quadratic residues and applications. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 637–653. Springer, Heidelberg, August 2009. 7, 21

[HL18] Justin Holmgren and Alex Lombardi. Cryptographic hashing from strong one-way functions (or: One-way product functions and their applications). In Mikkel Thorup, editor, *59th FOCS*, pages 850–858. IEEE Computer Society Press, October 2018. 6

[HLR21] Justin Holmgren, Alex Lombardi, and Ron D. Rothblum. Fiat-shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge). In *STOC*, pages 750–760. ACM, 2021. 6, 7, 8, 9, 10, 14, 15

[HY17] Pavel Hubácek and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. In Philip N. Klein, editor, *28th SODA*, pages 1352–1371. ACM-SIAM, January 2017. 3, 4, 48

[JJ21] Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from sub-exponential DDH. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 3–32. Springer, Heidelberg, October 2021. 6

[JKKZ21] Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. Snargs for bounded depth computations and PPAD hardness from sub-exponential LWE. In *STOC*, pages 708–721. ACM, 2021. 3, 5, 6, 19, 41

[JLS21a] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over F_p, DLIN, and PRGs in NC^0. Cryptology ePrint Archive, Report 2021/1334, 2021. https://eprint.iacr.org/2021/1334. 4

[JLS21b] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021. 4

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79 – 100, 1988. 48

[Kal03] Adam Kalai. Generating random factored numbers, easily. *Journal of Cryptology*, 16(4):287–289, September 2003. 11, 21, 23

[KPY20] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. Delegation with updatable unambiguous proofs and PPAD-hardness. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 652–673. Springer, Heidelberg, August 2020. 3, 5

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of Fiat-Shamir for proofs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 224–251. Springer, Heidelberg, August 2017. 6

[LFKN90] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st FOCS*, pages 2–10. IEEE Computer Society Press, October 1990. 18, 19

[LV20] Alex Lombardi and Vinod Vaikuntanathan. Fiat-shamir for repeated squaring with applications to PPAD-hardness and VDFs. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 632–651. Springer, Heidelberg, August 2020. 3, 5, 6, 9, 13, 14, 15, 16, 17, 20

[MP91] Nimrod Megiddo and Christos H Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991. 3

[Nas51] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951. 3

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. https://eprint.iacr.org/2010/556. 3

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994. 3, 46, 48

[Pei16] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424, 2016. 13

[Pie19] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019. 6, 9, 20, 21, 38, 39, 40

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Heidelberg, August 2019. 6, 14

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 49–62. ACM Press, June 2016. 7, 15

[RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996. 5, 20, 22

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008. 29

A TFNP Classes

Here we recall the definition of **TFNP** classes that are relevant to our paper. In Fig. 5 the relationship between these classes is illustrated.

Definition A.1 ([Pap94])**.** EndOfLine (EOL)

- *Instance.*
  1. Boolean circuits $\mathsf{S}, \mathsf{P} : \{0,1\}^m \to \{0,1\}^m$
  2. String $v_0 \in \{0,1\}^m$
- *Guarantee.*[18] v_0 is unbalanced: $\mathsf{P}(v_0) = v_0$ and $\mathsf{S}(v_0) \neq v_0$

[18] Unlike a promise, e.g., as in RSVL (Definition 2.7), a *guarantee* is locally checkable and, therefore, in case it does not hold the convention is to output this violation itself as a solution in order to maintain totality of the problem.

Algorithm 5 Recursive description of the RSVL instance. This is the version of Algorithm 1 with explicit subscripts.

procedure $\mathsf{F}(v, \boldsymbol{x}_v, \tau_v)$

hardwired Descriptions of:

1. the function $f : \mathcal{X} \to \mathcal{Y}$
2. a hash $H : \{0,1\}^* \to \{0,1\}^\lambda$ sampled using $\Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}(1^)$
3. the d-query downwards self-reduction algorithm D^f
4. the (k', k) batching reduction $\widetilde{\mathsf{B}}$ where $k' = k \cdot d$

input

1. Vertex $v \in V(T)$
2. Problem instances $\boldsymbol{x}_v := (x_{v,0}, \cdots, x_{v,k-1}) \in \mathcal{X}_n^k$
3. Transcript $\tau_v \in \mathcal{X}^k \times \mathcal{Y}^k \times (\mathcal{X} \times \mathcal{Y})^{\leq d \cdot n}$

output $\boldsymbol{y}_v \in \mathcal{Y}^k$ and π_v, a proof for $\boldsymbol{x}_v \xrightarrow{f_n} \boldsymbol{y}_v$

if $|x_{v,0}| = \cdots = |x_{v,k-1}| = 1$ **then return** $((f_1(x_{v,0}), \cdots, f_1(x_{v,k-1})), \perp)$

▷ **Outlining**

Invoke $\mathbf{D}^{f_{n-1}}(\boldsymbol{x}_v)$ to obtain query $\boldsymbol{q}_{v0}$

for $j \in [1, d-1]$ **do**

  $(\boldsymbol{y}_{vj}, \pi_{vj}) := \mathsf{F}(v(j-1), \boldsymbol{q}_{v(j-1)}, \varepsilon)$

  Simulate $\mathbf{D}^{f_{n-1}}$ on $\Big((\boldsymbol{q}_{v0}, \boldsymbol{y}_{v0}), \cdots, (\boldsymbol{q}_{v(j-1)}, \boldsymbol{y}_{v(j-1)})\Big)$ to obtain $\boldsymbol{q}_{vj}$

▷ **Batching**

Simulate $\mathbf{D}^{f_{n-1}}$ on $\boldsymbol{\mu}_v := \Big((\boldsymbol{q}_{v0}, \boldsymbol{y}_{v0}), \cdots, (\boldsymbol{q}_{v(d-1)}, \boldsymbol{y}_{v(d-1)})\Big)$ to obtain $\boldsymbol{y}_v$

if $\tau_v = \varepsilon$ **then** $\tau_{vd} := \boldsymbol{x}_v \boldsymbol{y}_v \boldsymbol{\mu}_v$ **else** $\tau_{vd} := \tau_v \boldsymbol{\mu}_{vd}$

Compute challenge $r_{vd} := H(\tau_{vd})$

Compute $(\boldsymbol{x}_{vd}, \boldsymbol{y}_{vd}) := \widetilde{\mathsf{B}}(\boldsymbol{\mu}_{vd}; r_{vd})$

▷ **Recursive proof-merging**

$(\boldsymbol{y}_{vd}, \pi_{vd}) := \mathsf{F}(vd, \boldsymbol{x}_{vd}, \tau)$

return $(\boldsymbol{y}_{v(d-1)}, \boldsymbol{\mu}_{vd} \pi_{vd})$

- *Solution.* An unbalanced vertex $v \in \{0,1\}^m$: $\mathsf{P}(\mathsf{S}(v)) \neq v$ or $\mathsf{S}(\mathsf{P}(v)) \neq v \neq v_0$

The successor and predecessor circuits implicitly define a *directed graph* of degree at most one on $\{0,1\}^m$, in which an edge $(u,v) \in \{0,1\}^{2m}$ exists if and only if $\mathsf{S}(u) = v$ and $\mathsf{P}(v) = u$. Since v_0 is guaranteed to unbalanced (a source) in this implicit graph *handshaking lemma on directed graphs* guarantees the existence of another unbalanced vertex (a sink), which is the (locally-checkable) solution to the instance.

Definition A.2 ([Pap94])**. PPAD** is defined as the set of all total problems that are Karp-reducible to EOL.

Definition A.3 ([HY17])**.** EndOfMeteredLine (EOML)

- *Instance.*
  1. A successor circuit $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$
  2. A predecessor circuit $\mathsf{P} : \{0,1\}^m \to \{0,1\}^m$
  3. A meter circuit $\mathsf{M} : \{0,1\}^m \to [0, 2^m]$
  4. String $v_0 \in \{0,1\}^m$
- *Guarantee.*
  1. v_0 is unbalanced: $\mathsf{P}(v_0) = v_0$ and $\mathsf{S}(v_0) \neq v_0$
  2. v_0 is the first vertex: $\mathsf{M}(v_0) = 1$
- *Solution.* A vertex $v \in \{0,1\}^m$ satisfying one of the following:
  1. **End of line:** either $\mathsf{P}(\mathsf{S}(v)) \neq v$ or $\mathsf{S}(\mathsf{P}(v)) \neq v \neq v_0$,
  2. **False source:** $v \neq v_0$ and $\mathsf{M}(v) = 1$,
  3. **Miscount:** either $\mathsf{M}(v) > 0$ and $\mathsf{M}(\mathsf{S}(v)) - \mathsf{M}(v) \neq 1$ or $\mathsf{M}(v) > 1$ and $\mathsf{M}(v) - \mathsf{M}(\mathsf{P}(v)) \neq 1$.

The successor, predecessor and meter circuits implicitly define a *directed acyclic graph* (DAG) of degree at most one on $\{0,1\}^m$, in which an edge $(u,v) \in \{0,1\}^{2m}$ exists if and only if $\mathsf{S}(u) = v$, $\mathsf{P}(v) = u$ *and* the meter is consistent with this, i.e., $\mathsf{M}(v) = \mathsf{M}(u) + 1$. Any deviation from valid meter behaviour is accepted as solutions of types 2 and 3. Since v_0 is guaranteed to be a source, *handshaking lemma on directed graphs* guarantees the existence of a solution of type 1, i.e., a sink.

Definition A.4 ([DP11])**. CLS** is defined as the set of all total problems that are Karp-reducible to EOML.

Remark 11. **CLS** was originally defined by Daskalakis and Papadimitrou [DP11] using a total problem called ContinuousLocalOptimum. Subsequent works have improved our understanding of this class. Firstly, Fearnley et al. [FGHS21] showed that **CLS** = **PPAD** ∩ **PLS**, where **PLS** is a subclass of **TFNP** [JPY88] that captures local search problems. Secondly, in a series of works [FGMS19, GHJ+22] it was shown that EOML, known to lie in **CLS** [HY17], is **CLS**-complete. Definition A.4 is as a result of these sequence of works.

FP ⟶ **UEOPL** ⟶ **CLS** ⟶ **PPAD** ⟶ **TFNP** ⟶ **FNP**

Figure 5: The **TFNP** landscape relevant to this paper (arrows indicate containment). **FP** and **FNP** are the function equivalent of the decision classes **P** and **NP**, respectively. The class **UEOPL** is defined in Appendix B.

B UEOPL Hardness

In this section, we show hardness of **UEOPL** $\subseteq$ **CLS** [FGMS19], which is one of the lowest-lying class in **TFNP**. Since it is a sub-class of **CLS**, the hardness results in Section 5 will be subsumed by ones presented in this section. We begin with the definition of the class.

B.1 Class UEOPL

Definition B.1 ([FGMS19][19]). UniqueForwardEOML (UFEOML)

- *Instance.*
  1. Boolean circuit $\mathsf{S} : \{0,1\}^m \to \{0,1\}^m$
  2. Boolean circuit $\mathsf{M} : \{0,1\}^m \to [0, 2^m - 1]$
  3. String $v_0 \in \{0,1\}^m$
- *Guarantee.*
  1. v_0 is unbalanced: $\mathsf{S}(v_0) \neq v_0$
  2. $\mathsf{M}(v_0) = 0$
- *Solution.* One of the following:
  1. **Sink:** A vertex $v \in \{0,1\}^m$ such that $\mathsf{S}(v) \neq v$ and $\mathsf{S}(\mathsf{S}(v)) = \mathsf{S}(v)$
  2. **Violation of meter:** A vertex $v \in \{0,1\}^m$ such that $\mathsf{S}(v) \neq v$ and $\mathsf{M}(\mathsf{S}(v)) \neq \mathsf{M}(v) + 1$
  3. **Collision of meter:** Two vertices $u \neq v \in \{0,1\}^m$ such that $u \neq \mathsf{S}(u), v \neq \mathsf{S}(v)$ and $\mathsf{M}(u) = \mathsf{M}(v)$.

Intuitively, S and M implicitly define a DAG on $\{0,1\}^m$, where an edge $(u, v) \in \{0,1\}^{2m}$ exists if and only if $\mathsf{S}(u) = v$ and the meter is consistent with this, i.e., $\mathsf{M}(v) = \mathsf{M}(u) + 1$. Any violation of meter counts as a type 2 solution. A type 1 solution is now guaranteed by the principle that *any non-empty DAG has a sink* and to ensure that the DAG is non-empty, an outgoing edge is guaranteed from v_0. Moreover, any witness to the *non-uniqueness* of the path from v_0 also qualifies as additional solutions of type 3 (these solutions are not necessary for totality).

Remark 12. Note that in the type 1 solution, we ask for a *predecessor* of the sink and not the sink itself. This is to distinguish actual sinks from *isolated vertices*, i.e., v such that $\mathsf{S}(v) = v$, but $\nexists u \in \{0,1\}^m : \mathsf{S}(u) = v$, since these are *not* the vertices guaranteed by the existence principle. Moreover, since there is no predecessor circuit in UFEOML, there is no way to *locally* check whether or not a vertex is isolated.

[19] In [FGMS19], this problem is called UniqueForwardEOPL + 1; we prefer the name UniqueForwardEOML.

Definition B.2 ([FGMS19])**. UEOPL** is defined as the set of all total problems that are Karp-reducible to UFEOML.

Remark 13. Instead of UFEOML, Fearnley et al. [FGMS19] define **UEOPL** using another total problem called UniqueEndOfPotentialLine. These two problems are equivalent with respect to Karp reductions [FGMS19]. The advantage of working with UFEOML is that we do not have to deal with predecessor circuit (this is taken care of in the reduction from UFEOML to UEOPL).

B.2 Construction

The construction of our UFEOML instance $\mathsf{UFEOML} = (\mathsf{S}, \mathsf{M}, \boldsymbol{s}_{0^n})$ is given in Algorithm 6. It is similar to the RSVL construction $\mathsf{RSVL} = (\mathsf{S}, \mathsf{V}, L, \boldsymbol{s}_{0^n})$ from Section 5.1. In particular, $\mathsf{UFEOML.S}$ the same as $\mathsf{RSVL.S}$ (except that it is hardwired with the length parameter $L = (d+1)^n$). The meter circuit $\mathsf{UFEOML.M}$ simply returns the index of vertex v_ℓ in the restricted DFS of T.

Algorithm 6 The UFEOML instance $\mathsf{UFEOML} = (\mathsf{S}, \mathsf{M}, \boldsymbol{s}_{0^n})$. The descriptions of S_n and V can be found in Algorithms 3 and 4, respectively.

procedure $\mathsf{S}(\boldsymbol{s})$
hardwired Same items as in Algorithm 2 and additionally $L := (d+1)^n$
input State $\boldsymbol{s} \in \{0,1\}^m$ parsed as in Eq. (14)
output Next state $\boldsymbol{s}' \in \{0,1\}^m$
    Let $i \in [0, 2^m - 1]$ denote the index of the $v_\ell \in \boldsymbol{v}$ in restricted DFS on T
    if $\mathsf{V}(\boldsymbol{s}, i)$ rejects **then** return $\boldsymbol{s}$ and halt
    else return $\boldsymbol{s}' := \mathsf{S}_n(\varepsilon, \boldsymbol{s}, (x, \cdots, x), \varepsilon)$ and halt

procedure $\mathsf{M}(\boldsymbol{s})$
hardwired Same items as in S
input State $\boldsymbol{s} \in \{0,1\}^m$ parsed as in Eq. (14)
    Let $i \in [0, 2^m - 1]$ denote the index of the $v_\ell \in \boldsymbol{v}$ in restricted DFS on T
    Return i and halt

B.3 Analysis

Theorem B.3 (Hardness of UFEOML from f and $\Pi_{\mathrm{FS},\mathcal{H}}$)**.** *Let $k, d \in \mathbb{N}$ be parameters and $\lambda \in \mathbb{N}$ be a security parameter. Suppose that*

- *$f : \mathcal{X} \to \mathcal{Y}$ is a d-query downwards self-reducible and dk-to-k batch-reducible function with sampler X; and*
- *$\Pi_{\mathrm{FS},\mathcal{H}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ is the non-interactive protocol for $\mathcal{L}^k_{f_n}$ from Corollary 3.4.*

Furthermore, for $H \leftarrow \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}(1^\lambda)$ and $x \leftarrow \mathsf{X}(1^\lambda)$, define

$$m = m(d, k, |x|) \in \mathsf{poly}(d, k, |x|) \text{ and } L = L(d, k) := (d+1)^n, \tag{22}$$

and let

$$\mathsf{S} : \{0,1\}^m \to \{0,1\}^m \text{ and } \mathsf{M} : \{0,1\}^m \to [0, d^n - 1] \tag{23}$$

be as defined as in Algorithm 6, hardwired with $(f, H, \mathsf{D}, \widetilde{\mathsf{B}}, x, \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V}, L)$*. If* f *is hard with respect to* X *and* $\Pi_{\mathrm{FS},\mathcal{H}}$ *is (adaptively) unambiguously sound argument, 1then* $\mathsf{UFEOML} = (\mathsf{S}, \mathsf{M}, \boldsymbol{s}_{0^n})$*, constitutes a hard distribution of* UFEOML*.*

On instantiating f with IS as sampled in Assumption 4.8 and $\Pi_{\mathrm{FS},\mathcal{H}}$ with non-interactive protocol from Corollary 4.10, we get the following corollary to Theorem B.3.

Corollary B.4 (Hardness of UFEOML from IS and LWE)**.** *For a security parameter* $\lambda \in \mathbb{N}$*, let* $(\mathbb{G}_\lambda, g, T)$ *be sampled as in Assumption 4.8, which defines* $f_n(g, T) := g^{2^T}$ *for* $n := \log(T)$*. Also, let* $\Pi_{\mathrm{FS},\mathcal{H}} = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ *denote the non-interactive protocol for* $\mathcal{L}^k_{\mathbb{G}}$ *from Corollary 4.10, which implies* $k \in \lambda^{O(1)}$ *and* $d = 2$*. Furthermore, for* $H \leftarrow \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}(1^\lambda)$*, define*

$$m = m(n, k, \lambda) := n^2 k \cdot \mathsf{poly}(\lambda) \text{ and } L = L(n) = 3^n, \tag{24}$$

and let

$$\mathsf{S} : \{0,1\}^m \to \{0,1\}^m \text{ and } \mathsf{M} : \{0,1\}^m \to [0, 2^m - 1] \tag{25}$$

be defined as in Algorithm 6, hardwired with $((\mathbb{G}_\lambda, g, T), H, \mathsf{D}, \widetilde{\mathsf{B}}, \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V}, L)$*. If Assumptions 2.10 and 4.8 hold then* $\mathsf{UFEOML} = (\mathsf{S}, \mathsf{M}, \boldsymbol{s}_{0^n})$ *constitutes a hard distribution of UFEOML.*

Proof (of Theorem B.3). First, we claim that UFEOML does not (by construction) contain violations of meter by showing that for any valid state $\boldsymbol{s} \in \{0,1\}^m$, $\mathsf{M}(\mathsf{S}(\boldsymbol{s})) = \mathsf{M}(\boldsymbol{s}) + 1$ holds. To see this, first recall that

- the meter of a valid state $\boldsymbol{s}$ parsed as in Eq. (14) depends solely on $(v_0, \cdots, v_\ell)$ and is, in particular, defined as the index of v_ℓ in the restricted DFS; and
- S is defined in Algorithm 6 (Line 3) to *self-loop* at an invalid state $\boldsymbol{s}$, i.e., $\mathsf{S}(\boldsymbol{s}) = \boldsymbol{s}$ if $\mathsf{V}(\boldsymbol{s}) = 0$.

Since $\boldsymbol{s}$ is valid, we have $\boldsymbol{s} \neq \boldsymbol{s}' := \mathsf{S}(\boldsymbol{s})$ (unless it is the sink of the standard line). By construction of S, the tuple $(v'_0, \cdots, v'_\ell)$ corresponding to $\boldsymbol{s}'$ (when parsed as in Eq. (14)) is such that v'_ℓ is the vertex visited immediately after v_ℓ in the restricted DFS and therefore has index one more than v_ℓ's.

Now, suppose for contradiction that there exists a $\mathsf{poly}(\lambda)$-sized algorithm $\mathsf{A} = \{\mathsf{A}_\lambda\}_{\lambda \in \mathbb{N}}$ and a polynomial $p(\cdot)$ such that A solves UFEOML for infinitely-many security parameters $\lambda \in \mathbb{N}$ with probability at least $1/p(\cdot)$; fix one such λ. This implies by an averaging argument that A finds either the standard sink, a non-standard sink or a collision of meter for λ with probability at least $1/3p(\lambda)$. Given an adversary that finds the standard sink, we show in Claim B.5 that it is possible to break hardness of f (Assumption 5.5); given an adversary that finds a non-standard sink or a false positive, we show in Claims B.6 and B.7, respectively, how to break unambiguous soundness.

Claim B.5 (Reduction from hardness of f)**.** *Given an adversary* A_1 *that finds the standard sink in* UFEOML *with probability at least* $p'(\lambda)$*, one can break the hardness of* f *with same probability.*

Proof. Given a challenge input $x \in \mathcal{X}$, the reduction samples a CRS $H \leftarrow \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{Setup}(1^\lambda)$ and sends $(\mathsf{S}, \mathsf{M}, \boldsymbol{s}_{0^n})$ hardwired with $(f, H, \mathsf{D}, \widetilde{\mathsf{B}}, x, \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V}, L)$ to A_1. A_1 returns the state $\boldsymbol{s}_{1^n} := \mathsf{S}^{L-1}(\boldsymbol{s}_{0^n})$ such that $\mathsf{S}(\boldsymbol{s}_{1^n}) = \boldsymbol{s}_\varepsilon$ and therefore $\mathsf{S}(\mathsf{S}(\boldsymbol{s}_{1^n})) = \mathsf{S}(\boldsymbol{s}_\varepsilon) = \boldsymbol{s}_\varepsilon$. By applying S once to this node the reduction can compute $\boldsymbol{s}_\varepsilon =: ((\varepsilon, (x, \cdots, x), (y, \cdots, y), \pi))$ with $y = f_n(x)$. The reduction simply returns y. □

Claim B.6 (Reduction from unambiguous soundness of $\Pi_{\mathrm{FS},\mathcal{H}}$)**.** *Given an adversary* A_2 *that finds a non-standard sink in* UFEOML *with probability at least* $p'(\lambda)$*, one can break* $\Pi_{\mathrm{FS},\mathcal{H}}$*'s unambiguous soundness with same probability.*

Proof. Given a challenge CRS H, the reduction samples $x \leftarrow \mathsf{X}(1^\lambda)$ and sends $(\mathsf{S}, \mathsf{M}, \boldsymbol{s}_{0^n})$ hardwired with $(f, H, \mathsf{D}, \widetilde{\mathsf{B}}, x, \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V}, L)$ to A_2. A_2 returns $\boldsymbol{s}^* \in \{0,1\}^m$ such that $\mathsf{S}(\mathsf{S}(\boldsymbol{s}^*)) = \mathsf{S}(\boldsymbol{s}^*)$. Suppose that $\mathsf{M}(\boldsymbol{s}^*) = i$ and let v be the vertex in T with index i. Moreover, let $\boldsymbol{s}_v$ denote the i-th state on the standard line. (As in Claim 5.9, it is not necessary for the reduction to be able to efficiently compute $\boldsymbol{s}_v$.) The proof now is similar to Claim 5.9. Let's parse $\boldsymbol{s}^*$ as

$$((v_0^*, \boldsymbol{x}_0^*, \boldsymbol{y}_0^*, \pi_0^*), \cdots, (v_\ell^*, \boldsymbol{x}_\ell^*, \boldsymbol{y}_\ell^*, \pi_\ell^*)), \tag{26}$$

and $\boldsymbol{s}_v$ as in Eq. (14). In order to prove the claim, we make a case distinction.

- *All the statements in* $\boldsymbol{s}_v$ *and* $\boldsymbol{s}^*$ *match.* In this case, since $\boldsymbol{s}_v \neq \boldsymbol{s}^*$, there exists at least one index $j \in [0, \ell]$ such that $\pi_j^* \neq \pi_j$ but $(\boldsymbol{x}_j^*, \boldsymbol{y}_j^*) = (\boldsymbol{x}_j, \boldsymbol{y}_j)$ is a true statement. Therefore π_j^* breaks unambiguity of $\Pi_{\mathrm{FS},\mathcal{H}}$ and the reduction returns

$$(\pi_j^*, \boldsymbol{x}_j \xrightarrow{f_{n-|v_j|}} \boldsymbol{y}_j).$$

- *Some statements are different and* $j \in [0, \ell]$ *is the first such index.* In this case, we argue that

$$\boldsymbol{x}_j^* \xrightarrow{f_{n-|v_j^*|}} \boldsymbol{y}_j^*. \tag{27}$$

  is false, but since π_j^* is accepted it breaks soundness of $\Pi_{\mathrm{FS},\mathcal{H}}$. To see this, first note that (by assumption) the statements in labels less than j in both $\boldsymbol{s}_v$ and $\boldsymbol{s}^*$ are same and therefore the verifier ends up recomputing $\boldsymbol{x}_j^* = \boldsymbol{x}_j$ as in Claim 5.9. Therefore, if $\boldsymbol{x}_j^* \neq \boldsymbol{x}_j$ then the state is invalid and the verifier would have rejected (which results in $\mathsf{S}(\boldsymbol{s}^*) = \boldsymbol{s}^*$). Moreover, since V rejects if $(v_0, \cdots, v_\ell) \neq (v_0^*, \cdots, v_\ell^*)$, we have $v_j = v_j^*$ but $\boldsymbol{y}^* \neq \boldsymbol{y}$ and therefore the statement in Eq. (27) is false (since f is a function) but π_j^* accepts. Therefore the reduction can return π_j^* and the statement in Eq. (27). □

Claim B.7 (Reduction from unambiguous soundness of $\Pi_{\mathrm{FS},\mathcal{H}}$)**.** *Given an adversary* A_3 *that finds a collision of meter in* UFEOML *with probability at least* $p'(\lambda)$*, one can break* $\Pi_{\mathrm{FS},\mathcal{H}}$*'s unambiguous soundness with same probability.*

Proof. Given a challenge CRS H, the reduction samples $x \leftarrow \mathsf{X}(1^\lambda)$ and sends $(\mathsf{S}, \mathsf{M}, \boldsymbol{s}_{0^n})$ hardwired with $(f, H, \mathsf{D}, \widetilde{\mathsf{B}}, x, \Pi_{\mathrm{FS},\mathcal{H}}.\mathsf{V}, L)$ to A_3. A_3 returns $\boldsymbol{s}, \boldsymbol{s}' \in \{0,1\}^m$ such that $\boldsymbol{s} \neq \boldsymbol{s}'$, $\mathsf{S}(\boldsymbol{s}) \neq \boldsymbol{s}$, $\boldsymbol{s}' \neq \mathsf{S}(\boldsymbol{s}')$ and $\mathsf{M}(\boldsymbol{s}) = \mathsf{M}(\boldsymbol{s}') =: i$. Suppose that v is the vertex in T with index i and $\boldsymbol{s}_v$ be the i-th state on the standard line, which can be efficiently computed using (p, q). Fix $\boldsymbol{s}^* \in \{\boldsymbol{s}, \boldsymbol{s}'\}$ such that $\boldsymbol{s}^* \neq \boldsymbol{s}_v$ – this must exist since $\boldsymbol{s} \neq \boldsymbol{s}'$. From here the proof proceeds exactly as in Claim B.6. □

□