On the Cryptographic Hardness of Local Search[*]

Nir Bitansky[†] Idan Gerichter[‡]

January 5, 2020

Abstract

We show new hardness results for the class of Polynomial Local Search problems (PLS):

- Hardness of PLS based on a falsifiable assumption on bilinear groups introduced by Kalai, Paneth, and Yang (STOC 2019), and the Exponential Time Hypothesis for randomized algorithms. Previous standard model constructions relied on non-falsifiable and non-standard assumptions.
- Hardness of PLS relative to random oracles. The construction is essentially different than previous constructions, and in particular is unconditionally secure. The construction also demonstrates the hardness of parallelizing local search.

The core observation behind the results is that the *unique proofs* property of incrementally-verifiable computations previously used to demonstrate hardness in PLS can be traded with a simple *incremental completeness* property.

[*]This work was supported in part by ISF grant 18/484, and by Len Blavatnik and the Blavatnik Family Foundation. An extended abstract appears in the proceedings of Innovations of Theoretical Computer Science 2020.

[†]Tel Aviv University, nirbitan@tau.ac.il. Member of the Check Point Institute of Information Security. Supported also by the Alon Young Faculty Fellowship.

[‡]Tel Aviv University, idangrichter@mail.tau.ac.il.

Contents

1 Introduction

Local search is a well known approach for tackling optimization problems. Local search algorithms seek solutions that are *locally optimal* — they commonly try to improve on a given solution by considering small perturbations of it, called *neighbors*, and testing whether they are better according to some value function. Indeed, many algorithms use this approach with empirical success, for instance, the Simplex linear programming algorithm, the Lin-Kernighan TSP algorithm, and various machine learning algorithms [Nas00, LK73, IM98]. Nevertheless, the approach has its limitations and many natural local search problems are not known to admit polynomial-time algorithms.

Aiming to characterize the computational complexity of local search, Johnson, Papadimitriou, and Yannakakis [JPY88] introduced the class Polynomial Local Search (PLS). The class is defined by its canonical complete problem LOCAL-SEARCH (LS) [JPY88, HY16]. Here the input consists of two polynomial-size circuits: a *successor circuit* $\mathcal{S} : \{0,1\}^n \to \{0,1\}^n$, which given a string x outputs a string x', and a *value circuit* $\mathcal{F}$, which given a string x outputs an integer value. The goal is to find a string x which is locally optimal in the sense that $\mathcal{F}(x) \geq \mathcal{F}(\mathcal{S}(x))$. Here strings correspond to solutions, with a value assigned by $\mathcal{F}$, and the successor assigned by $\mathcal{S}$ represents "the best" neighbor. Important problems that are known to be PLS complete include finding a locally optimal max-cut [SY91] and finding pure Nash equilibrium in congestion games [FPT04].

As observed in [JPY88], PLS belongs to the class TFNP of *total search problems* in NP; namely, a local optimal solution always exists (and can be efficiently verified). As a result, it is unlikely that PLS contains NP-hard problems, as this would imply that NP = coNP [JPY88, MP91]. This barrier is shared of course by other prominent subclasses of TFNP, perhaps the most famous example being the class PPAD that captures the hardness of finding Nash equilibrium in bimatrix games [DGP09, CDT09]. Accordingly, researchers have turned to seek for alternative evidence of hardness.

One natural place to look for such evidence is cryptography. Indeed, cryptography typically relies on problems closer to the boundary of P than to NP hard problems. For some subclasses of TFNP (such as PPP, PPA, and Ramsey) evidence of hardness has been shown based on standard cryptographic assumptions [Pap94, BO06, Jer12, KNY17a, SZZ18]. Establishing the hardness of PLS (and PPAD), however, has been far more elusive. Hubáček and Yogev [HY16], building on [BPR15, GPS16], demonstrated a hard problem in PLS $\cap$ PPAD based on *indistinguishability obfuscation* [BGI$^+$12], a strong cryptographic primitive, yet to be constructed under standard assumptions.[1] Similar hardness was then shown by Choudhuri et al. [CHK$^+$19b] assuming hardness of $\#$SAT and the soundness of the Fiat-Shamir transform for the sumcheck protocol, which in turn can be based on *optimally-secure fully-homomorphic encryption against quasi-polynomial attackers* [CCR16]. More recently, the same has been shown based on the hardness of iterated squaring and soundness of Fiat-Shamir for the iterated-squaring protocol of Pietrzak [Pie18] by both Choudhuri et al. [CHK$^+$19a] and Ephraim et al. [EFKP19].

Basing the hardness of PLS (or PPAD) on standard assumptions remains an open problem.

[1]More accurately, they show a hard problem in the class *Continuous Local Search* (CLS), which is known to be in PLS $\cap$ PPAD [DP11].

1.1 Our Results

We provide new results regarding the hardness of PLS. Our first result shows worst-case hardness of PLS based on *the KPY assumption* on bilinear groups and the randomized Exponential Time Hypothesis (ETH). The KPY assumption was introduced recently by Kalai, Paneth, and Yang [KPY19] toward the construction of publicly-verifiable delegation schemes. The assumption is similar in spirit to previous standard assumptions that generalize Decisional Diffie Hellman (e.g., [BBG05, BGW05]). It is polynomially falsifiable and holds in the generic group model. Randomized ETH postulates that solving SAT in the worst case requires exponential-time even for randomized algorithms [IPZ01].

Theorem 1.1 (Informal). *Under the KPY assumption on bilinear groups and randomized* ETH, PLS *is hard in the worst case.*

The result is, in fact, more general and shows a reduction of PLS hardness to a certain type of incrementally-verifiable computation schemes (IVC) [Val08]. We then derive the required IVC from the work of [KPY19]. We can also show average-case hardness at the cost of assuming superpolynomial hardness of the KPY assumption and average-case randomized ETH. See further details in the technical overview.

Our second result is a construction of PLS search problems that are unconditionally hard in *the random oracle model.*

Theorem 1.2 (Informal). *Relative to a random oracle,* LOCAL-SEARCH *is unconditionally hard-on-average.*

Previously, Choudhuri et al. [CHK$^+$19b] showed hardness of CLS $\subseteq$ PLS $\cap$ PPAD relative to a random oracle, but their result is conditioned on hardness of $\#\mathrm{SAT}$ (the construction does not relativize with respect to this hardness and hence it cannot be derived from the random oracle itself). Indeed, our construction is quite different from theirs. Whereas their construction is based on the sum-check protocol [LFKN90] (which explains the non-black-box reliance on $\#\mathrm{SAT}$ hardness), ours comes from recent advancements in *proofs of sequential work* motivated by blockchain applications [MMV13, CP18, DLM19].

The reliance on proofs of sequential work, in fact, translates to a result on the hardness of parallelizing local search. Such hardness, in the random oracle model, was recently shown for CLS (and in particular for both PLS and PPAD) by Ephraim et al. [EFKP19] further assuming that repeated squaring modulo a composite is hard to parallelize [RSW00]. Specifically, they construct CLS instances that can be solved in some tuneable sequential polynomial time, but cannot be solved much faster by parallel algorithms. Our result gives a similar kind of hardness for PLS in the random oracle model, without relying on any unproven computational assumptions. We elaborate on this in the technical overview below.

2 Technical Overview

To motivate our approach, we start by recalling previous approaches taken toward proving PLS (or rather CLS) hardness.

2.1 Hardness via Incremental Computation

So far, the common approach toward arguing cryptographic hardness of PLS went through an intermediate problem called SINK-OF-VERIFIABLE-LINE (SVL) [AKV04, BPR15]. An instance of SVL consists of a successor circuit $\mathcal{S}$ and a verifier circuit $\mathcal{V}$. The successor $\mathcal{S}$ implicitly defines a step-by-step (or, incremental) computation $s_{t+1} := \mathcal{S}(s_t)$ starting from a canonical source s_0 and ending at a sink s_T, for some superpolynomial T:

$$s_0 \xrightarrow{\mathcal{S}} \dots \xrightarrow{\mathcal{S}} s_T$$

The verifier $\mathcal{V}(s^*, t)$ is promised to accept any s^* as the t-th state if and only if it is correct, namely $s^* = \mathcal{S}^{(t)}(s_0)$. The goal is to find the sink $s_T = \mathcal{S}^{(T)}(s_0)$, given $(\mathcal{S}, \mathcal{V})$.

It is not hard to see that solving SVL can be reduced to solving a local search problem in PLS [HY16], and thus it is sufficient to prove the hardness of SVL.

Valiant's Incrementally-Verifiable Computation and Uniqueness. A natural way to approach SVL hardness is Valiant's notion of incrementally-verifiable computation (IVC) [Val08]. According to this notion, we can take a Turing machine computation given by a machine M and input x with a configuration graph:

$$M_x^0 \to \dots \to M_x^T \quad ,$$

and associate with each intermediate configuration M_x^t a short proof π_t, attesting to its correctness. The proofs are incremental — the next proof π_{t+1} can be efficiently computed from π_t.

At first glance, this general notion seems to directly yield SVL hardness. Indeed, we can take any polynomial-space machine M solving some hard search problem R on input x and consider the corresponding incrementally-verifiable computation:

$$(M_x^0, \pi_0) \xrightarrow{\mathcal{S}} \dots \xrightarrow{\mathcal{S}} (M_x^T, \pi_T) \quad .$$

Here the successor simply advances the computation of M and incrementally computes the corresponding proofs. The SVL verifier is derived directly from the IVC verifier.

The reason that the above simple transformation does not work is that *the IVC proofs may not be unique*. In particular, a correct intermediate configuration may have many different accepting proofs attesting to its correctness, whereas SVL requires that only a single string can be accepted as the t-th node (for any t). Choudhuri et al. [CHK$^+$19b] show that the SVL uniqueness requirement can be somewhat relaxed and traded with *computational uniqueness* meaning that it is computationally hard to find more than a single proof for any given statement (even if such proofs exist).

So far, however, incrementally-verifiable computation with (even computational) uniqueness has not been achieved under standard assumptions. This is, in fact, the case even for specific (hard) computations, let alone for general ones.

2.2 Incremental Completeness

We show that the hardness of PLS does not require SVL hardness nor IVC with unique proof. Rather, we observe that IVC with a conceptually simple *incremental completeness* property suffices. Then we derive such IVC for polynomially-long computations from the delegation scheme construction of Kalai, Paneth, Yang [KPY19] and combine it with ETH to obtain PLS hardness. We next explain the notion of incremental completeness and why it suffices. In the next subsection, we explain how it is obtained.

Incremental-Complete IVC and PLS Hardness. An incremental complete IVC has the property that given *any* accepting proof π for an intermediate state M_x^t of the computation, running the prover results in an accepting proof π' for the next state M_x^{t+1}. This differs from Valiant's original formulation where completeness is only guaranteed for the prescribed proof generated by iteratively applying the prover t times.

Let us now describe how to construct hard LOCAL-SEARCH instances from incrementally-complete IVC. Once again, we start from some hard computation given by a polynomial-space machine M and instance x. Rather than considering a verifiable computation chain, we shall consider a DAG with nodes of the form (t, C, π), where t is a timestamp, C is an alleged configuration of $M(x)$ at time t, and π is an IVC proof. We call such a node *valid* if the corresponding proof is accepting. For simplicity, we assume for now that the IVC is also perfectly sound, meaning that valid nodes are always such that C is the correct configuration M_x^t. Note that for every time t, there may very well be multiple corresponding valid nodes. We can visualize this as multiple ”parallel universes”, which the verifiable computation lives in simultaneously. Incremental completeness says that in *every* such universe, as time moves forward, the computation may proceed. In particular, by induction, we always reach a *sink* of the form (T, M_x^T, π) that contains the last state of the computation.

This naturally gives an LS instance. Valid nodes (t, M_x^t, π) are given value t and are always succeeded by another valid node $(t+1, M_x^{t+1}, \pi')$ derived by advancing the computation and incrementing the IVC proof. Invalid nodes are given value -1 and are succeeded by some canonical source in the DAG $(0, M_x^0, \varepsilon)$, where ε is the empty proof, which by convention is accepted as a proof for M_x^0. The local maximums of the corresponding instance are exactly the sinks of the DAG, which by incremental completeness are of the form (T, M_x^T, π), containing the last state. Therefore, finding a local-maximum reduce to solving the underlying hard computational task given by the computation $M(x)$.

Our actual construction does not assume perfect soundness of the IVC, but only computational soundness. There may exist so called *fooling nodes* (t, C^*, π^*) that pass verification although C^* is not the correct configuration M_x^t. These nodes may be local maximums and finding them may not reduce to solving the underlying problem. Nonetheless, finding fooling local maximum corresponds to breaking the soundness of the IVC, which is computationally infeasible.

2.3 Obtaining IVC with Incremental Completeness

Valiant [Val08] put forth a general approach toward constructing IVC by *recursive proof composition*. Oversimplifying, the basic idea is that to move from the t-th configuration $C = M_x^t$ and proof π_t to the next configuration $C' = M_x^{t+1}$ and proof π_{t+1}, the new proof π_{t+1} asserts that:

1. There exists a proof π that passes IVC verification as a valid proof that $C = M_x^t$ (this part is recursive in the sense that it relies on IVC verification for smaller times stamps),
2. C' is obtained from C by applying M_x's transition function.

For this to be feasible, each "single-step proof" is computed using a succinct non-interactive NP proof system where verification time (and in particular, the size of the proof) is fixed and is not affected by the complexity of the NP relation. We observe that *any IVC construction following the above blueprint is incrementally complete* by definition as long as the succinct NP proof system used has perfect completeness.

Succinct Proof Systems. Succinct proofs for NP with unconditional soundness are unlikely to exist [GH98, GVW02]. Accordingly, Valiant suggested to instantiate the blueprint using computationally sound succinct arguments. This brings about some extra complications in following the blueprint. For once, proving that a previous proof *exists* is meaningless in the case of computational soundness. Rather we need the guarantee that a proof exists and can be efficiently found. Such systems are known as *succinct non-interactive arguments of knowledge* (SNARKs) and admit an efficient knowledge extractor that can efficiently extract a witness from a convincing prover.

Using such SNARKs enables the construction of IVC for arbitrary poly-time computations (or superpolynomial ones, if the SNARK is superpolynomially secure) [Val08, BCCT13], and the corresponding constructions are incrementally complete by the fact that the underlying SNARKs are perfectly complete. (The actual constructions are somewhat different from the described blueprint; in particular, to deal with issues such as blowup in knowledge extraction complexity, they aggregate proofs in a tree-like fashion rather than a chain, in order to reduce the depth of the recursion. Incremental completeness holds just the same.)

Following the above, we can obtain an incrementally-complete IVC, and thus PLS hardness from SNARKs. However, the existence of SNARKs is a strong non-falsifiable assumption. It is only known under non-standard *knowledge assumptions* and its construction from standard assumptions is subject to barriers [GW11, BCPR16].

Incrementally-Complete IVC from Weaker Succinct Arguments. In a recent work, Kalai, Paneth, and Yang [KPY19] considered a relaxation of SNARKs called *quasi-arguments* that instead of standard knowledge extraction only requires a certain weaker *no-signaling extraction* requirement. Unlike SNARKs, such arguments are not subject to any known lower bounds, and were in fact constructed from the learning with errors assumption in their privately-verifiable form [KRR14, PR17, BHK17, BKK$^+$18] and recently also in their publicly-verifiable form based on the so called KPY assumption in bilinear groups [KPY19].

Furthermore, Kalai, Paneth, and Yang show that similarly to SNARKs such quasi-arguments can, in fact, be recursively composed toward verifying a long computation, provided that the computation is *deterministic*. Their explicit goal was not to construct IVC but rather to bootstrap quasi-arguments with a long common reference string (CRS) to succinct arguments with a short CRS. However, this is done by implicitly constructing an IVC (this connection is also apparent in previous constructions of SNARKs with a short CRS from ones with a long CRS [BCCT13]). The resulting IVC, like other IVCs based on recursive composition, is incrementally complete.

The Class of Supported Computations and the Reliance on ETH. In its native form, the IVC derived from the KPY construction supports computations of arbitrary polynomial length $T = \lambda^{O(1)}$ in the security parameter λ, with fixed prover-verifier running time (say, λ). However, as explained earlier, to get PLS hardness, we would like to apply the IVC for a hard (and thus superpolynomial) computation.

To circumvent this difficulty, we employ a fine-grained reduction. The idea is to construct instances which are tailor-made to be hard for LS algorithms of a specific polynomial running time. Fix any constant $c > 0$ and n^c-time algorithm $\mathcal{A}$ that supposedly solve LS (here n is the size of the LS instance $(\mathcal{S}, \mathcal{F})$). Consider a search problem R and a Turing machine M that solves it in polynomial time $T(\lambda)$, while no randomized algorithm can solve it in time T^δ for constant $\delta < 1$. If we can bound the *blowup* of the IVC reduction by at most $T^{\delta/c}$, we can use $\mathcal{A}$ to construct a randomized adversary $\mathcal{A}'$ that solves R in time T^δ and get a contradiction.

The blowup of the IVC reduction is polynomially related to (a) the input size, (b) the space used by M, and (c) the efficiency of the IVC scheme. Accordingly, we require a computation where there is an arbitrarily large (polynomial) gap between the space used and the hardness of the underlying problem. To this end, we use the randomized ETH assumption. By appropriately choosing the size of the underlying SAT instance, we get computations that can be solved using fixed-space and some polynomial time via brute-force but not in much faster polynomial time, even by randomized algorithms. To bound the blowup due to the IVC efficiency, we follow the efficiency analysis by [KPY19] and adapt it to the incremental setting. (In the body, we show the above in the non-uniform setting: assuming non-uniform ETH and using standard non-uniform derandomization techniques while taking special care to bound the associated blowup.)

We note that the above only gives worst-case hardness. That is, we do not show a single distribution that is hard for all polynomial-time algorithms. Assuming slight super-polynomial security of the KPY assumption and average-case randomized ETH, we can get average-case PLS hardness by essentially the same reduction (in fact, we can slightly weaken average-case ETH to a slightly subexponential time hypothesis).

2.4 Unconditional Hardness in the Random Oracle Model

Our second result, in the random model, is based on recent advancements in *proofs of sequential work* (PoSW) [MMV13]. Roughly speaking, in a PoSW, the prover is given a statement χ and a time parameter T, and can generate a corresponding proof π by making T sequential steps. The soundness requirement is that provers that make $\ll T$ sequential steps, will fail to generate a valid proof for χ, except with negligible probability (namely, proof computation is not parallelizable). The construction we present relies on the PoSW of Cohen and Pietrzak [CP18] and its extension by Döttling, Lai, and Malavolta [DLM19] to so called *incremental proof of sequential work*; both are proven sound in the random oracle model. We now move to recalling how these systems work, and explain how we derive from them hard PLS instances relative to random oracles.

The CP PoSW. To construct a PoSW, Cohen and Pietrzak [CP18] suggest an elegant tree labeling procedure that is inherently sequential. They consider a binary tree whose edges are directed from the leaves toward the root. In addition, they add directed edges from every node ℓ having a sibling r on the right, to all the leaves in the tree rooted at r. We call this the CP graph (see

Figure 1 in Section 6 for an illustration). The nodes of the CP graph are then given random labels as follows: the label of any given node is obtained by applying a random oracle $\mathsf{H}_\chi = \mathsf{H}(\chi, \cdot)$ to the labels of its incoming nodes (the oracle is seeded with χ to guarantee an independent oracle for every statement). Intuitively, the added edges enforce that in order to compute the labels of some subtree rooted at some r, it is first necessary to compute the labels of its left sibling ℓ; in this sense, labeling must be done sequentially.

To turn this into an actual proof of sequential work π for χ, the prover publishes the label of the root of the entire tree. The verifier then responds with random challenge leaves to which the prover answers by providing all the labels in the corresponding paths toward the root along with the labels of the siblings along the path, as in standard Merkle tree authentication (indeed here the tree serves both the purpose of of a commitment and of enforcing the sequential nature). Finally, they make the proof non interactive by using the Fiat-Shamir transform. Cohen and Pietrzak prove that to successfully compute an accepting proof, a prover must sequentially call the random oracle $\approx T$ times, where T is the size of the tree.

Incremental PoSW and the DLM Construction. Cohen and Piertzak further show that the standard streaming algorithm for Merkle commitments [Mer79], can also be applied to their labeling procedure. That is, one can keep track of a small amount of nodes (about $\log T$), each a root of some tree in a gradually growing forest, and when needed, merging two nodes on the same level. In other words, the CP labeling can be done by an incremental computation with small space.

One thing that is of course missing toward getting PLS hardness is verifiability of the corresponding intermediate states. One Naïve idea toward such local verifiability is to apply the CP proof strategy for each subtree in the incremetal labeling process. Indeed, each such subtree is, in fact, CP labeled on its own, so we can use Fiat-Shamir to compute random challenges for that particular subtree. While this is a step in the right direction it is still not enough for the goal of PLS hardness, since the intermediate proofs themselves are not computed incrementally — computing each local proof may take time proportional the size of the corresponding subtree.

Döttling, Lai, and Malavolta [DLM19] suggested a neat idea to make the CP proofs incrementally computable.[2] Roughly speaking rather than sampling fresh challenges for each subtree, they suggested to derive them at random from the previously computed challenges of this subtree. In a bit more detail, whenever merging two subtrees rooted at ℓ and r into a new (taller) subtree, we assume by induction that each of the two already has a set of corresponding challenges S_ℓ and S_r, where say the size of each one is some parameter k. Then, for the new subtree, rooted at their parent v, we choose a new set S_v of k challenges by choosing k challenges at random from the $2k$ challenges in $S_\ell \cup S_r$ (this is again done non-interactive using Fiat-Shamir). Each of the challenges is again just a Merkle authentication path, which can be easily augmented to an authentication path for v. They prove that this choice of challenges guarantees, for every intermediate state, essentially the same soundness as the original CP construction guarantees for the final state.

Incremental Completeness and PLS Hardness. What we get from the DLM incremental PoSW (in the random oracle) is somewhat analogous to an IVC system in the standard model. However, while in the standard model we applied general IVC over some long (hard) computation, here

[2]They were motivated by blockhains and aimed to make PoSW a process that can be advanced in a distributed fashion by different parties.

there is no such underlying hardness. Rather, *the difficulty is only in computing the accepting proofs themselves*. Similarly to our standard model approach, here too we can address the concept of incremental completeness. Indeed, we observe that merging two accepting proofs in the DLM construction always yields an accepting proof. From here incremental completeness follows. Formally, we need to slightly augment the DLM construction to enforce consistency among different subtrees in a forest to guarantee such incremental completeness (see Section 6 for the details).

Choosing the parameters appropriately, yields problems in PLS relative to a random oracle which are exponentially hard-on-average. This translates to sub-exponential hardness of the canonical LS probalem (due to polynomial blowup of the reduction). Also, by choosing the size of the tree to be a polynomial of our choice, we get search problems in PLS which are moderately hard but ”depth-robust” in the sense that they cannot be solved much faster by parallel algorithms [EFKP19]. This follows from the sequential hardness of DLM.

2.5 More Related Work on Total Search Problems

The class TFNP was introduced by Megiddo and Papadimitriou [MP91]. They observed that TFNP is unlikely to include NP hard problems as this would imply that NP = coNP. Furthermore, it is strongly believed that TFNP does not have complete problems (see for instance discussion in [GP18]). Toward understanding of the complexity of total search problems, Papadimitriou [Pap94] introduced several “syntactic” subclasses of TFNP that cluster problems according to the mathematical argument used to establish their totality. Recently, Goldberg and Papadimitriou [GP18] presented a subclass called *provable* TFNP that contains previously defined subclasses (and can be seen as generalizing them) and admits complete problems.

A long line of works have investigated the complexity of TFNP and its subclasses, searching for efficient algorithms on one hand and evidence of hardness on the other. As explained earlier in the intro, cryptography is a natural place to look for such evidence. Indeed, basing the hardness of TFNP or its subclasses, on standard cryptographic assumptions, has been successful in several cases. For example, Papadimitriou [Pap94] observed that PPP is hard assuming one-way permutations or collision-resistant hash functions, and Jeřábek [Jer12], building on the work of Buresh-Oppenheim [BO06], demonstrated the hardness of PPA assuming factoring is hard. Hubáček, Naor and Yogev [HNY17] showed TFNP-hardness assuming any average-case hard NP language exists (in particular one-way functions) and derandomization assumptions. Komargodski, Naor, and Yogev [KNY17b] showed that hardness of the class RAMSEY is equivalent to the existence of multi-collision resistant hash functions.

As discussed earlier, demonstrating hardness for PPAD and PLS has been more challenging and so far only achieved under non-standard cryptographic assumptions. Trying to explain our failure so far to base such hardness on standard cryptographic primitives, Rosen, Segev and Shahaf [RSS17] show that TFNP hard instances constructed in a black box way from random oracles (or some variants thereof) must have a nearly exponential number of solutions (which is indeed the case in our result in the random oracle model).

Finally, we note that in the smooth complexity setting, several PLS complete problems, with natural noise distributions, have been shown to be solvable in smooth polynomial-time. This include finding locally optimal Max-Cut [ABPW16] and finding pure Nash equilibrium in network coordination games [BKM18]. These algorithms suggest an explanation of why local search may be empirically easy, while hard instances exist under reasonable assumptions and can be sampled.

3 Preliminaries

Notation. Throughout, λ will denote security parameters. When the input size is not the security parameter, we denote it by n. For $a \leq b$ integers we denote by $[a, b]$ the set $\{a, ..., b\}$ and by $[a]$ the set $[1, a]$.

3.1 Standard Computational Conventions

We recall standard computational conventions and definitions.

- By *algorithm* we mean a uniform Turing machine. An algorithm is *efficient* if it is polynomial-time.
- A polynomial-size circuit family $\mathcal{C}$ is a sequence of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, such that $\mathcal{C}_\lambda$ is of size $\text{poly}(\lambda)$ and has $\text{poly}(\lambda)$ input and outputs wires.
- We follow the standard practice of modeling efficient adversaries $\mathcal{A}$ as polynomial-size circuit families $\mathcal{A} = (\mathcal{A}_\lambda)_{\lambda \in \mathbb{N}}$. Such an adversary is called an *efficient adversary*.
- A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* if for every $c > 0$, there exists $N \in \mathbb{N}$ such that $f(n) < n^{-c}$ for all $n > N$. We will denote negligible functions by negl.
- Turing machines have separate input, work and output tape. A configuration of a Turing machine doesn't contain the content of the input tape.

3.2 Search Problems

We recall definitions of search problems and the class PLS.

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a relation. Denote by $R(x, y)$ the indicator function for $(x, y) \in R$. Denote by $R_x = \{y \in \{0,1\}^* \mid R(x, y) = 1\}$ the set of witnesses for x.

Definition 3.1 (Search Problem)**.** *A relation R is a search problem. A Turing machine M is said to solve R, if given any input x (a) if $R_x \neq \emptyset$, it returns y such that $R(x, y) = 1$, (b) otherwise, M declares no such y exists. The set of all search problems that can be solved by polynomial-time Turing machines is denoted* FP. *The set of search problems that can be solved by polynomial-size circuits is called* $\mathsf{FP/Poly}$.

Definition 3.2 (NP Search Problem)**.** *An* NP *search problem R is a search problem with the additional properties: (i) $R(x, y)$ is computed in time* $\text{poly}(|x|)$. *(ii) For every x we have $R_x \subseteq \{0,1\}^{\text{poly}(|x|))}$ for a fixed polynomial. The set of all* NP *search problems is called functional* NP *and is denoted* FNP.

Definition 3.3 (Total NP Search Problem)**.** *A total* NP *search problem is described via a pair (I, R), where $I \subseteq \{0,1\}^*$ is an efficiently recognizable set of instances and R is an* NP *search problem with the guarantee that R_x is non-empty for every instance $x \in I$. The set of all total* NP *search problems is called total* FNP *and is denoted* TFNP.

In an equivalent definition for TFNP we use, the search problem is given by R, an NP search problem and the set of instances is implicitly defined to be $I_R = \{x \mid R_x \neq \emptyset\}$ such that it is efficiently recognizable. We abuse notation and write $R \in \mathsf{TFNP}$ to denote $(R, I_R) \in \mathsf{TFNP}$.

3.2.1 Polynomial Local Search

The complexity class *polynomial local search* (PLS) consists of all total NP search problems polynomial-time reducible to the LOCAL-SEARCH problem [JPY88, HY16] we denote by LS.

Definition 3.4 (LOCAL-SEARCH)**.** *The search problem* LS *is the following: given two polynomial-size circuits $\mathcal{S} : \{0,1\}^n \to \{0,1\}^n$ and $\mathcal{F} : \{0,1\}^n \to \mathbb{N}$, find a string $v \in \{0,1\}^n$ such that $\mathcal{F}(\mathcal{S}(v)) \leq \mathcal{F}(v)$.*

We refer to $\mathcal{S}$ as the *successor circuit* and to $\mathcal{F}$ as the *value circuit*. Intuitively, the circuits $\mathcal{S}$ and $\mathcal{F}$ could be seen as representing an implicit DAG over $\{0,1\}^n$. In this graph, a node v is connected to $u = \mathcal{S}(v)$ if and only if $\mathcal{F}(u) > \mathcal{F}(w)$. This is a DAG since $\mathcal{F}$ induce a topological ordering. Notice also that every sink corresponds to a local maximum with respect to $\mathcal{F}$. With this perspective, the totality of LS follows from the fact *every finite DAG has a sink*.

Oracle aided PLS. We denote by $\mathcal{C}^{\mathcal{O}} = \{\mathcal{C}^{\mathcal{O}}_\lambda\}$ an oracle aided circuit family. That is a circuit family with oracle gates to $\mathcal{O}_\lambda : \{0,1\}^{r(\lambda)} \to \{0,1\}^{\ell(\lambda)}$. The complexity class $\mathsf{PLS}^{\mathcal{O}}$ relative to an oracle $\mathcal{O}$, is naturally defined as all $\mathsf{TFNP}^{\mathcal{O}}$ search problem polynomial-time reducible to $\mathsf{LS}^{\mathcal{O}}$.

Definition 3.5 (Oracle Aided LOCAL-SEARCH)**.** *The search problem $\mathsf{LS}^{\mathcal{O}}$ is the following: given two polynomial-size, oracle aided, circuits $\mathcal{S}^{\mathcal{O}} : \{0,1\}^n \to \{0,1\}^n$ and $\mathcal{F}^{\mathcal{O}} : \{0,1\}^n \to \mathbb{N}$, find a string $v \in \{0,1\}^n$ such that $\mathcal{F}^{\mathcal{O}}(\mathcal{S}^{\mathcal{O}}(v)) \leq \mathcal{F}^{\mathcal{O}}(v)$.*

3.3 Average-Case Hardness of Search Problem

We recall the definition of average-case hard FNP search problem.

Definition 3.6 (Hard on average search problem)**.** *Let R be an* FNP *search problem. R is hard-on-average if there exists an efficient sampler D with the following requirements:*

1. *For every $\lambda \in \mathbb{N}$:*
$$\Pr\left[R_x \neq \emptyset \;\middle|\; x \leftarrow D(1^\lambda)\right] = 1 \ .$$
2. *For every efficient adversary $\mathcal{A}$, there exists a negligible function μ, such that for every $\lambda \in \mathbb{N}$:*
$$\Pr\left[R(x,y) = 1 \;\middle|\; \begin{array}{c} x \leftarrow D(1^\lambda) \\ y \leftarrow \mathcal{A}(x) \end{array}\right] \leq \mu(\lambda) \ .$$

4 PLS Hardness from IVC

In this section, we give a computational reduction from search problems having an IVC with incremental completeness to LS. This, in turn, establishes that any (worst/average-case) hard search problem, having an IVC with incremental completeness, implies the hardness of PLS (in the worst/average-case, respectively). We start by formally defining IVC with incremental completeness.

4.1 IVC with Incremental Completeness

Conventions. Let M be a Turing machine with $T = T(\lambda)$ and $S = S(\lambda)$ bounds on its run time and space, respectively, for input of length λ. Throughout, we assume w.l.o.g that M always makes exactly T steps and the size of a configuration is exactly S. Let $x \in \{0,1\}^\lambda$ be an input to M, we denote by $M_x^t \in \{0,1\}^S$ the configuration in the t step of the computation $M(x)$. We denote by $M_x : \{0,1\}^S \to \{0,1\}^S$ the transition circuit between configurations.

Definition 4.1 (IVC with Incremental Completeness)**.** *Let M be a Turing machine with $T = T(\lambda)$ and $S = S(\lambda)$ bounds on its run time and space, respectively, for input of length λ.*

An Incremental Verifiable Computation scheme (IVC) for M, with incremental completeness, consists of three algorithms $\mathsf{IVC} = (\mathsf{IVC.G}, \mathsf{IVC.P}, \mathsf{IVC.V})$*:*

- $\mathsf{IVC.G}(x)$ *is a randomized algorithm that given $x \in \{0,1\}^\lambda$ outputs public parameters pp.*
- $\mathsf{IVC.P}(pp, t, C, \pi)$ *is a deterministic algorithm that given public parameters pp, a natural number t, an arbitrary configuration C and an arbitrary proof π, outputs a proof π'.*
- $\mathsf{IVC.V}(pp, t, C, \pi)$ *is a deterministic algorithm that given public parameters pp, a natural number t, an arbitrary configuration C and an arbitrary proof π, outputs* ACC *or* REJ.

We make the following requirements:

1. ***Incremental Completeness:*** *For every security parameter $\lambda \in \mathbb{N}$:*

   (a) *For every input $x \in \{0,1\}^\lambda$, time $t \in [0, T-1]$ and candidate proof $\pi \in \{0,1\}^*$:*

   $$\Pr\left[\begin{array}{l} \mathsf{IVC.V}(pp, t, M_x^t, \pi) = \mathsf{ACC} \implies \\ \mathsf{IVC.V}(pp, t+1, M_x^{t+1}, \pi') = \mathsf{ACC} \end{array} \middle| \begin{array}{l} pp \leftarrow \mathsf{IVC.G}(x) \\ \pi' = \mathsf{IVC.P}(pp, t, M_x^t, \pi) \end{array}\right] = 1 \ ,$$

   where M_x^t, M_x^{t+1} are the configurations in the t and $t+1$ steps of the computation of $M(x)$, respectively.

   (b) *For every input $x \in \{0,1\}^\lambda$:*

   $$\Pr\left[\mathsf{IVC.V}(pp, 0, M_x^0, \varepsilon) = \mathsf{ACC} \;\middle|\; pp \leftarrow \mathsf{IVC.G}(x)\right] = 1 \ ,$$

   where M_x^0 is the first configuration of the computation of $M(x)$, and ε is the empty proof.

2. ***Soundness:*** *For every efficient adversary $\mathcal{A}$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^\lambda$:*

   $$\Pr\left[\begin{array}{l} C^* \neq M_x^t \\ \mathsf{IVC.V}(pp, t, C^*, \pi^*) = \mathsf{ACC} \end{array} \middle| \begin{array}{l} pp \leftarrow \mathsf{IVC.G}(x) \\ (t, C^*, \pi^*) \leftarrow \mathcal{A}(x, pp) \end{array}\right] \leq \mu(\lambda) \ ,$$

   where M_x^t is the configuration in the t step of the computation of $M(x)$.

3. ***Efficiency:*** *The efficiency of* IVC*, denoted by $\mathbb{T}_{\mathsf{IVC}}(\lambda)$, is the maximal worst-case run-time among* $\mathsf{IVC.G}, \mathsf{IVC.P}, \mathsf{IVC.V}$ *for input having security parameter λ. We require $\mathbb{T}_{\mathsf{IVC}}(\lambda) \leq p(\lambda)$ for a fixed polynomial p.*

4.2 Computationally Sound Karp Reduction

As discussed in the introduction, since we use IVC with computational soundness, the reduction we give is also computationally sound. In what follows, we define the notion of computationally sound Karp reduction.

For $R, R' \in \mathsf{FNP}$, a computationally sound reduction from R to R' consists of a pair of efficient algorithms $(\mathcal{X}, \mathcal{W})$. Where $\mathcal{X}(x)$ is a randomized algorithm that given an instance x of R, samples an instance $x' \leftarrow \mathcal{X}(x)$ of R'. $\mathcal{W}(w')$ is a deterministic algorithm that translates a witness w' for x', to a candidate witness w of x. We require that its infeasible to find w', such that $w = \mathcal{W}(w')$ is not a witness for x in R.

Definition 4.2 (Computational Karp Reduction)**.** *For $R, R' \in \mathsf{FNP}$, a computational Karp reduction from R to R' consists of a pair $(\mathcal{X}, \mathcal{W})$, where $\mathcal{X}(x)$ is a randomized efficient algorithm and $\mathcal{W}(w)$ is a deterministic efficient algorithm. We make the following requirement:*

- ***Computational Soundness:*** *For every efficient adversary $\mathcal{A}$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^\lambda$ for which R_x is non-empty:*

$$\Pr\left[\begin{array}{c} w' \in R'_{x'} \\ w \notin R_x \end{array} \middle| \begin{array}{c} x' \leftarrow \mathcal{X}(x) \\ w' \leftarrow \mathcal{A}(x') \\ w = \mathcal{W}(w') \end{array}\right] \leq \mu(\lambda) \enspace ,$$

where R_x and $R'_{x'}$ are the set of witnesses for x in R and x' in R' respectively.

Definition 4.3 (Reduction Efficiency)**.** *The efficiency of a computational Karp reduction $(\mathcal{X}, \mathcal{W})$, denoted $\mathbb{T}_{\mathsf{Red}}(\lambda)$, is the maximum of $\mathbb{T}_{\mathcal{X}}(\lambda), \mathbb{T}_{\mathcal{W}}(\lambda)$ where:*

1. *$\mathbb{T}_{\mathcal{X}}(\lambda)$ is the worst-case run-time of $\mathcal{X}$ for input $x \in \{0,1\}^\lambda$.*
2. *$\mathbb{T}_{\mathcal{W}}(\lambda)$ is the worst-case run-time of $\mathcal{W}$ for input* $w' \in \left\{ w' \middle| \begin{array}{l} x \in \{0,1\}^\lambda \\ x' \in \mathsf{Supp}(\mathcal{X}(x)) \\ w' \in R_{x'} \end{array} \right\}$.

Note that $\mathbb{T}_{\mathsf{Red}}(\lambda)$ is a bound on instance size sampled using $\mathcal{X}$.

4.3 The Hardness Reduction

Using the notion of computational Karp reduction, our hardness result is formalized in the following theorem.

Theorem 4.1 (IVC Reduction)**.** *Let $R \in \mathsf{FNP}$, solvable by a polynomial-space Turing machine M. If there exists an IVC scheme with incremental completeness for M, there exists a computationally sound Karp reduction $(\mathcal{X}, \mathcal{W})$ from R to LS. The efficiency of the reduction is*

$$\mathbb{T}_{\mathsf{Red}} = \mathrm{poly}(\mathbb{T}_{\mathsf{IVC}}, S, \lambda, |M|) \enspace ,$$

where $S(\lambda)$ is a bound on M space and $\mathbb{T}_{\mathsf{IVC}}(\lambda)$ is the efficiency of the IVC. The polynomial doesn't depend on the IVC scheme, M nor R.

Remark 4.1 (Efficiency Independent of T)**.** *Jumping ahead, the fact that $\mathbb{T}_{\mathsf{Red}}$ does not depend directly on the time of the computation T, plays an important role in the fine-grained reduction for polynomial-time computations presented in Section 5.*

Next, we describe the reduction and in the following section we analyze its security.

4.3.1 The Reduction

Let $\mathsf{IVC} = (\mathsf{IVC.G}, \mathsf{IVC.P}, \mathsf{IVC.V})$ be an incremental verifiable computation scheme with incremental completeness for M. Let $S = S(\lambda)$ be the polynomial bound on space of M when executed on $x \in \{0,1\}^\lambda$, guaranteed by the statement. Recall that we denote by M_x^t the configuration in the t step of the computation of $M(x)$. We assume w.l.o.g that the configuration size is exactly S. Let $T = T(\lambda) = 2^{S(\lambda)} \le 2^{\mathrm{poly}(\lambda)}$ be an upper bound on the running time of M. We further assume w.l.o.g that M always makes exactly T steps. With the above notation, the configuration graph of M when executing x is:

$$M_x^0 \longrightarrow \cdots \longrightarrow M_x^T \ .$$

In the following, we describe the construction of $\mathcal{X}(x)$ for an input $x \in \{0,1\}^\lambda$. Recall that we denote by $M_x : \{0,1\}^S \to \{0,1\}^S$ the transition circuit of $M(x)$, taking a configuration as input and returning the next configuration as output.

Instance translation: The algorithm $\mathcal{X}(x)$ begins by sampling $pp \leftarrow \mathsf{IVC.G}(x)$. Given pp, we apply a deterministic procedure I to generate an LS instance $(\mathcal{S}, \mathcal{F}) = I(x, pp)$. The constructed LS instance corresponds to a graph on vertices of a polynomial length $\ell = \ell(\lambda) \ge \lambda$. Each vertex is of the form $(t, C, \pi) \in \{0,1\}^\ell$ where t is parsed as an integer in $[0, T]$, C as a configuration of length S and π as a proof for the IVC padded to length ℓ if needed. We describe the successor $\mathcal{S}$ and value circuits $\mathcal{F}$ formally in the following.

Remark 4.2. *We assume w.l.o.g that M_x always outputs something. If the transition function of M fails to generate the next configuration given C, it will simply output M_x^0, the initial state.*

Remark 4.3. *We assume w.l.o.g that $\mathcal{F}$ outputs values in $[-1, T]$.*

In what follows, $v_0 := (0, M_x^0, \varepsilon)$ where M_x^0 is the initial state of $M(x)$ and ε is the empty proof.

Successor Circuit $\mathcal{S}(t, C, \pi)$

Hardwired: The public parameters pp, and the input x.
Algorithm:

```
1. If $\mathsf{IVC.V}(pp, t, C, \pi) = \mathsf{REJ}$, output $v_0$.
2. If $t = T$, output $(t, C, \pi)$.
3. Compute $C' = M_x(C)$, $\pi' = \mathsf{IVC.P}(pp, t, C, \pi)$. Output $(t+1, C', \pi')$.
```

Value Circuit $\mathcal{F}(t, C, \pi)$

Hardwired: The public parameters pp.
Algorithm:

```
1. If $\mathsf{IVC.V}(pp, t, C, \pi) = \mathsf{REJ}$, output $-1$.
2. Else, output $t$.
```

Witness translation: The algorithm $\mathcal{W}(t, C, \pi)$ simply returns the content of the output tape in the configuration C.

Efficiency. The successor circuit M_x is of size $\mathrm{poly}(S, \lambda, |M|)$. The computations related to IVC are of size $\mathrm{poly}(\mathbb{T}_{\mathsf{IVC}})$ by Turing machine simulation. All other computations are polynomial in the input length ℓ. We have that $\ell = \mathrm{poly}(S, \mathbb{T}_{\mathsf{IVC}})$. It follows that $\mathbb{T}_{\mathsf{Red}} = \mathrm{poly}(\mathbb{T}_{\mathsf{IVC}}, S, \lambda, |M|)$. All the above polynomials don't depend on M nor IVC.

By the fact M is poly-space and the efficiency requirement of IVC, that is $\mathbb{T}_{\mathsf{IVC}} \leq \mathrm{poly}(\lambda)$, both $\mathcal{X}, \mathcal{W}$ are polynomial-time algorithms, as required. In turn, this further implies that $\mathcal{S}, \mathcal{F}$ are of size $\mathrm{poly}(\lambda)$. Since the input length ℓ is at least λ, we have that $\mathcal{S}, \mathcal{F}$ are polynomial-size circuits.

4.4 Security Analysis

Fix $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^{\lambda}$. Let $(\mathcal{S}, \mathcal{F})$ be an instance in the support of $\mathcal{X}(x)$ and let pp be the public parameters hardwired in $(\mathcal{S}, \mathcal{F})$. We prove that all local maximums of $(\mathcal{S}, \mathcal{F})$ are one of two types:

- **Honest:** Nodes (T, M_x^T, π) where $\mathsf{IVC.V}(pp, T, M_x^T, \pi) = \mathsf{ACC}$.
- **Fooling:** Nodes (t, C^*, π^*) for which $C^* \neq M_x^t$ and $\mathsf{IVC.V}(pp, t, C^*, \pi^*) = \mathsf{ACC}$.

The proof of the following claim use the incremental completeness of IVC.

Claim 4.1. *All local maximums of $(\mathcal{S}, \mathcal{F})$ are honest-type or fooling-type.*

Proof of claim: Let $v = (t, C, \pi)$ be a local maximum of $(\mathcal{S}, \mathcal{F})$, that is $\mathcal{F}(\mathcal{S}(v)) \leq \mathcal{F}(v)$. We have that $\mathsf{IVC.V}(pp, t, C, \pi) = \mathsf{ACC}$, as otherwise it is given value -1 and is connected to v_0 which is of value 0. If $C \neq M_x^t$ then v is a fooling-type local maximum and if $C = M_x^t$ and $t = T$ then v is an honest-type local maximum.

We are left with the case $C = M_x^t$ and $t < T$. By construction, $\mathcal{S}(v) = (t+1, M_x^{t+1}, \pi')$ for $\pi' = \mathsf{IVC.P}(pp, t, M_x^t, \pi)$. By incremental completeness, $\mathsf{IVC.V}(pp, t+1, M_x^{t+1}, \pi') = \mathsf{ACC}$ and thus $\mathcal{F}(\mathcal{S}(v)) = t+1$ while $\mathcal{F}(v) = t$. Therefore it is not a local maximum, proving the claim. $\square$

Note that only honest-type local maximums translate by $\mathcal{W}$ to a valid witness for R, the underlying search problem. While there exist fooling-type local maximums, by the security of the IVC, finding them efficiently is infeasible. We proceed to prove Theorem 4.1.

Proof of Theorem 4.1. Let $\mathcal{A}$ be an efficient adversary that attempts breaking the computational soundness of the reduction $(\mathcal{X}, \mathcal{W})$. Let $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^\lambda$ be such that R_x is non-empty. We denote by $\varepsilon = \varepsilon(x)$ the probability of success of $\mathcal{A}$. That is

$$\varepsilon(x) := \Pr\left[\begin{array}{l} \mathcal{F}(\mathcal{S}(w')) \le \mathcal{F}(w') \\ w \notin R_x \end{array} \;\middle|\; \begin{array}{l} (\mathcal{S},\mathcal{F}) \leftarrow \mathcal{X}(x) \\ w' \leftarrow \mathcal{A}(\mathcal{S},\mathcal{F}) \\ w = \mathcal{W}(w') \end{array}\right] .$$

It follows by Claim 4.1 that all witnesses w' for $(\mathcal{S},\mathcal{F})$ such that $\mathcal{W}(w') = w \notin R_x$ are fooling-type local maximums. Indeed, for every honest-type local maximum, W outputs the content of the output tape of M_x^T. As M solves R and R_x is non-empty, we have that the output tape of M_x^T, the last configuration, consists of w such that $w \in R_x$.

Consider $\mathcal{A}' = \{\mathcal{A}'_\lambda\}_{\lambda\in\mathbb{N}}$ that attempts breaking the soundness of the IVC defined as follows. Recall $I(x, pp)$ is the deterministic procedure used by $\mathcal{X}$ to construct the circuits $\mathcal{S}$ and $\mathcal{F}$.

$\underline{\mathcal{A}'(x, pp)}$:

1. Compute $(\mathcal{S},\mathcal{F}) = I(x, pp)$.
2. Simulate $w' \leftarrow \mathcal{A}(\mathcal{S},\mathcal{F})$.
3. Output $w' = (t, C, \pi)$.

Note that, every fooling-type local maximum corresponds to (t, C^*, π^*) breaking the IVC scheme. Further notice that the distribution $(\mathcal{S},\mathcal{F}) = I(x, pp)$ for $pp \leftarrow \mathsf{IVC.G}(x)$ is, by definition, the distribution of $\mathcal{X}(x)$. Therefore, the success probability of $\mathcal{A}'(x, pp)$ is at least $\varepsilon(x)$. Using the soundness of IVC, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^\lambda$ such that R_x is non-empty we have the following:

$$\varepsilon(x) \le \Pr\left[\begin{array}{l} C^* \neq M_x^t \\ \mathsf{IVC.V}(pp, t, C^*, \pi^*) = \mathsf{ACC} \end{array} \;\middle|\; \begin{array}{l} pp \leftarrow \mathsf{IVC.G}(x) \\ (t, C^*, \pi^*) \leftarrow \mathcal{A}'(x, pp) \end{array}\right] \le \mu(\lambda) .$$

Establishing the computational soundness of the reduction. The efficiency part of the theorem has been argued in Section 4.3. □

4.5 Applying the Reduction

In this section we present results that establish that a computational Karp reduction from an underlying hard (worst-case/average-case) problem to LS implies the hardness of LS (in worst-case/average-case respectively). The proofs of the following propositions are given in appendix B.

4.5.1 Worst-Case Hardness

The existence of a computational Karp reduction from R to LS and an efficient solver for LS implies, in a straightforward way, the existence of an efficient randomized solver for R, successful with high probability. Proposition 4.1 extends this and show the existence of a deterministic solver for R. This is by standard non-uniform derandomization techniques. We use n to denote the size of LS instances and λ the size of R instances. The proof is given in appendix B.1

Proposition 4.1 (Worst-Case Hardness)**.** *Let R be an* FNP *search problem having a computational Karp reduction to* LS*. Assume there exists a adversary $\mathcal{A} = \{\mathcal{A}_n\}_{n\in\mathbb{N}}$ of polynomial-size $s(n)$ solving* LS *in the worst-case. Then there exists an adversary $\mathcal{A}' = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ solving R in the worst-case. The size of $\mathcal{A}'$ is*

$$\mathsf{size}(\mathcal{A}') = \mathrm{poly}(s(\mathbb{T}_{\mathsf{Red}}), \mathbb{T}_R, \lambda) \ ,$$

where $\mathbb{T}_{\mathsf{Red}}(\lambda)$ is the efficiency of the reduction and $\mathbb{T}_R(\lambda)$ is the efficiency of the NP *verification $R(x, y)$.*

4.5.2 Average-Case Hardness

Proposition 4.2 (Average-Case Hardness)**.** *If there exists a hard-on-average* FNP *problem R, with a computationally sound Karp reduction from R to* LS*, then* LS *is hard-on-average.*

Using the computational Karp reduction constructed in Section 4.3, we get the following corollary:

Corollary 4.1 (IVC Average-Case Hardness)**.** *Let R be a hard-on-average* FNP *problem. If there exists an IVC scheme with incremental completeness for a polynomial-space Turing machine M solving R, then* LS *is hard-on-average.*

5 Instantiation under the KPY Assumption and ETH

In Section 4 we were able to prove that an IVC scheme with incremental completeness for some super-polynomially hard computation, suffices to show that LS is hard. It is left to instantiate the above under the KPY assumption and construct an IVC scheme for a super-polynomially hard computation. Unfortunately, under the KPY assumption we are able to construct IVC schemes only for polynomial computations. By a more careful analysis and by using non-uniform exponential time hypothesis, in this section we show the worst-case hardness of LS.

Remark 5.1. *For ease of notation, in the following section we assume w.l.o.g $\lambda > 1$.*

5.1 IVC From KPY Assumption

The following theorem is proven in appendix A.

Remark 5.2 (Dependence of Constants)**.** *In the following, we use the constant c instead of big O notation in order to emphasize that the constant is independent of α.*

Theorem A.1 (IVC from KPY Assumption)**.** *Fix any constants $c, \varepsilon > 0$. Let α be a large enough constant and let M be a Turing machine of run-time $T(\lambda) \leq c\lambda^\alpha$, configuration size $S(\lambda) \leq c\lambda$ and description size $|M| \leq c \log_2 \alpha$. Under the KPY assumption, there exists an IVC scheme with incremental completeness for M having efficiency $\mathbb{T}_{\mathsf{IVC}} = \lambda^{\varepsilon\alpha}$.*

Remark 5.3 (Non-Uniform Reduction)**.** *The security reduction of the IVC scheme given in Theorem A.1 is non-uniform and therefore we require that the IVC scheme, and accordingly the KPY assumption, to hold against non-uniform polynomial-time attackers. This is to be expected when dealing with worst-case hardness.*

5.2 Fixed Space Polynomial Hardness via ETH

In this section, assuming the non-uniform ETH assumption, we show the existence of arbitrarily polynomially hard computations for circuits that can be solved by Turing machines using a fixed amount of space and slightly larger polynomial time.

Assumption 5.1 (Non-Uniform ETH)**.** *There exists a constant $\delta > 0$ such that no circuit family of size $O(2^{\delta n})$ solves* 3SAT*, where n is the length of the input 3CNF formula.*

Proposition 5.1 (Polynomial Hardness)**.** *Under the non-uniform* ETH *assumption, there exists constants $c, \delta > 0$ such that for every constant $\alpha > 1$ there exists a search problem $R_\alpha \in \mathsf{FNP}$ satisfying the following:*

- ***Algorithm:*** *There exists a Turing machine M_α of run-time $T(\lambda) \leq c\lambda^\alpha$, configuration size $S(\lambda) \leq c\lambda$ and description size $|M_\alpha| \leq c \log \alpha$ solving R_α.*
- ***Verifer:*** *There exists an* NP *verifier for R_α of run-time $\mathbb{T}_{R_\alpha} \leq c\lambda$.*
- ***Lower Bound:*** *No circuit family of size $O(\lambda^{\delta\alpha})$ can solve R_α.*

Proof. Consider

$$R_\alpha := \left\{ (x0^{2^{|x|/\alpha} - |x|}, w) \;\middle|\; (x, w) \in \mathsf{3SAT} \right\} .$$

Algorithm: The naive brute force algorithm for R_α satisfy the requirement. In the following α is not suppressed by O notation. For $x \in \{0,1\}^n$ and input of the form $x0^{2^{n/\alpha} - n}$, brute force takes $O(2^n)$ time while the whole input is of size $\lambda = 2^{n/\alpha}$. The description size is $O(\log \alpha)$ and the memory used is $O(\lambda)$.

Verifier: Let x be a 3SAT instance of size n. A witness w for 3SAT is an assignment of size $O(n)$. Verifing an assignment satisfies x can be done time $O(n)$. A verifier of run time $c\lambda$ for R_α follows.

Lower Bound: Let $\delta > 0$ be the constant assumed to exist by non-uniform ETH. Every circuit family of size $O(\lambda^{\delta\alpha})$ solving R_α can be turned to a circuit of size $O(2^{\delta\alpha})$ solving 3SAT. Contradicting non-uniform ETH. □

5.3 Putting Things Together

In this section, we put things together to prove that under non-uniform ETH and the KPY assumption we have $\mathsf{PLS} \not\subseteq \mathsf{FP/Poly}$.

Theorem 5.1 (PLS Hardness)**.** *Assuming non-uniform* ETH *and the KPY assumption we have* $\mathsf{PLS} \not\subseteq \mathsf{FP/Poly}$.

Toward proving the theorem, we start by bounding the blowup associated with the IVC reduction when applied to the brute-force computation solving R_α defined by Proposition 5.1.

Lemma 5.1 (Bounding the Blowup)**.** *Fix a constant $\varepsilon > 0$. Let α be a large enough constant. Under the KPY assumption, there exists a computationally sound Karp reduction $(\mathcal{X}, \mathcal{W})$ from R_α to* LS *with efficiency $\mathbb{T}_{\mathsf{Red}} \leq \lambda^{\varepsilon\alpha}$.*

Proof. Fix $\varepsilon > 0$ and let $\varepsilon' > 0$ be a constant to be chosen later. Let c be the constant guaranteed by Proposition 5.1. That is, for every $\alpha > 1$ there exists a Turing machine M_α of run-time $T \leq c\lambda^\alpha$, configuration size $S \leq c\lambda$ and description size $|M_\alpha| \leq c \log_2 \alpha$ solving R_α. Let α be a large enough constant such that Theorem A.1 apply to M_α with constant ε'. This gives an IVC scheme with incremental completeness for M_α having efficiency $\mathbb{T}_{\mathsf{IVC}} \leq \lambda^{\varepsilon'\alpha}$.

Applying Theorem 4.1 using the aforementioned IVC results in a computationally sound Karp reduction $(\mathcal{X}, \mathcal{W})$ from R_α to LS with efficiency:

$$\mathbb{T}_{\mathsf{Red}} \leq (\mathbb{T}_{\mathsf{IVC}} \cdot S \cdot \lambda \cdot |M_\alpha|)^\eta \leq \lambda^{\varepsilon'\alpha\eta} \cdot (c\lambda \log_2 \alpha)^{3\eta} \leq \lambda^{4\varepsilon'\alpha\eta} \quad ,$$

where η is a constant independent of α and ε and the last inequality is true for large α. By taking $\varepsilon' < \varepsilon/4\eta$ we get $\mathbb{T}_{\mathsf{Red}} \leq \lambda^{\varepsilon\alpha}$. This completes the proof the lemma. □

We are now ready to prove Theorem 5.1.

Proof of Theorem 5.1. Assume toward contradiction that under non-uniform ETH and the KPY assumption we have that $\mathsf{PLS} \subseteq \mathsf{FP/Poly}$. Hence, there exists a constant η and adversary $\mathcal{A} = \{\mathcal{A}_n\}_{n\in\mathbb{N}}$ of size $s(n) = n^\eta$ solving LS instances of size n.

Let $\varepsilon > 0$ be a constant to be chosen later. By bounding the blowup lemma, Lemma 5.1, there exists a large enough constant α such that there is a computationally sound Karp reduction $(\mathcal{X}, \mathcal{W})$ from R_α to LS having efficiency $\mathbb{T}_{\mathsf{Red}} \leq \lambda^{\varepsilon\alpha}$. Note that the efficiency of the NP verifier for R_α is $\mathbb{T}_{\mathsf{R}_\alpha} = O(\lambda)$ with a constant independent of α. By Proposition 4.1, there exists an adversary $\mathcal{A}' = \{\mathcal{A}'_\lambda\}_{\lambda\in\mathbb{N}}$, solving R_α instances of size λ of circuit size:

$$\mathsf{size}(\mathcal{A}'_\lambda) \leq \mathrm{poly}\left(s(\mathbb{T}_{\mathsf{Red}}), \mathbb{T}_{\mathsf{R}_\alpha}, \lambda\right) \leq \mathrm{poly}\left(\lambda^{\varepsilon\alpha\eta}, \lambda\right) \underbrace{\leq}_{\alpha > 1/\varepsilon\eta} \mathrm{poly}\left(\lambda^{\varepsilon\alpha\eta}\right) = O(\lambda^{\varepsilon\alpha\xi}) \quad ,$$

where ξ is a constant independent of ε and α. Let $\delta > 0$ be the constant associated with the lower bound of R_α. Recall δ is fixed and independent of α and ε. By choosing $\varepsilon < \delta/\xi$ we have

$$\mathsf{size}(\mathcal{A}'_\lambda) = O(\lambda^{\varepsilon\alpha\xi}) = O(\lambda^{\delta\alpha}) \quad .$$

A contradiction to the lower bound. We conclude that $\mathsf{PLS} \not\subseteq \mathsf{FP/Poly}$. This completes the proof. □

Remark 5.4 (Average-Case Hardness)**.** *In this remark we sketch how one can establish the average-case hardness of* LS *assuming superpolynomial security of the KPY assumption and average-case exponential time hypothesis. Let $T = T(\lambda)$ be a superpolynomial such that the KPY assumption holds for adversaries of size $T^{O(1)}$ and with distinguish advantage of $T^{-\omega(1)}$. Using the same IVC construction as the one used in the above, we are able to get a secure IVC scheme with incremental completeness having time parameter T. By appropriate padding for* SAT*, assuming average-case* ETH *yields search problems that are hard-on-average yet can be solved with polynomial-space Turing machines with less than T time. Using Corollary 4.1 we have that indeed* LS *is hard-on-average as desired.*

6 Unconditional PLS Hardness in the ROM

In this section we show that relative to a random oracle, there exist average-case exponentially hard problems in PLS. We further show, relative to a random oracle, the existence of depth-robust moderately hard problems in PLS. Our result is based on the incremental proofs of sequential work (iPoSW) by Döttling, Lai, and Malavolta [DLM19]. We modify the DLM construction such that an incremental completeness analog is achieved.

6.1 Graph Related Preliminaries

We recall some graph related definitions and lemmas used in the DLM scheme.

Notation. Throughout, we consider directed acyclic graphs (DAGs) $G = (V, E)$ where $V \subseteq \{0,1\}^{\leq d}$ is the vertex set and we refer to the parameter d as the depth of G. A vertex $v \in V$ is a parent of $u \in V$ if $(v, u) \in E$. The set of parents of u is denoted by $\mathsf{parents}(u)$. A vertex v is an ancestor of u if there is a directed path from v to u in G. The set of ancestors of u is denoted by $\mathsf{ancestors}(u)$. Any vertex $v \in V \cap \{0,1\}^d$ is called a leaf. Denote by $\mathsf{leaves}(u)$ the set of all leaf nodes that are ancestors of u.

6.1.1 CP Graphs

We recall the definition of CP graphs (see Figure 1), a family of directed acyclic graphs defined by Cohen and Pietrzak [CP18]. We denote the aforementioned graph family by $CP = \{CP_d\}_{d \in \mathbb{N}}$.

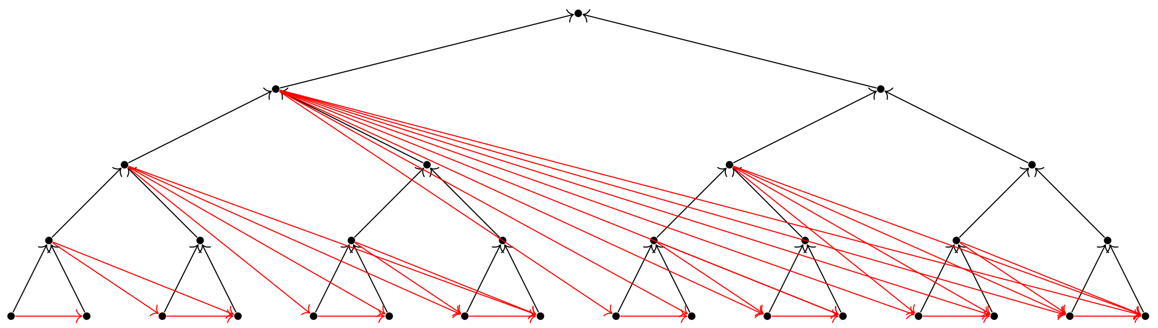

Figure 1: The CP_4 graph; red edges corresponds to E''.

Definition 6.1 (CP Graphs, [CP18]). *Let $B_d = (V, E')$ be the complete binary tree of depth d with edges pointing from the leaves to the root. The graph $CP_d = (V, E)$ is constructed from B_d with some added edges E''. Where E'' contains (v, u) for any leaf $u \in \{0,1\}^d$ and any v which is a left sibling to a node on the path form u to the root. Formally, $E = E' \cup E''$ where:*

$$E'' := \{(v, u) \mid u \in \{0,1\}^d, u = a||1||a', v = a||0, \text{ for some } a, a' \in \{0,1\}^{\leq d}\} \ .$$

The following fact on non-leaves in CP graphs is used in the construction and analysis.

Fact 6.1 (Left and Right Parents). *Let $CP_d = (V, E)$ and let $v \in V$ be a non-leaf node which is not the root. The node v has exactly two parents ℓ and r. We have the following:*

$$\mathsf{leaves}(v) = \mathsf{leaves}(\ell) \cup \mathsf{leaves}(r) \ .$$

6.1.2 Graph Labeling

Notation. For two sets A and B, we denote by $A \uplus B$ the disjoint union.

Definition 6.2 (Partial Vertex Labeling)**.** *For a graph $G = (V, E)$, a partial vertex labeling is a function $\mathcal{L} : V \to \{0,1\}^* \cup \{\perp\}$. The label for node $u \in V$ is denoted by $\mathcal{L}[u]$.*

Definition 6.3 (Valid DAG Labeling)**.** *Let $G = (V, E)$ be a DAG with $V \subseteq \{0,1\}^*$ and let $f : \{0,1\}^* \to \{0,1\}^w$ be a labeling function. The labeling of G with respect to f is defined recursively, for every $v \in V$:*

$$\mathcal{L}[v] := f(v, \mathcal{L}[p_1], ..., \mathcal{L}[p_r]) \ ,$$

where $(p_1, ..., p_r) = \mathsf{parents}(v)$, ordered lexicographically.

Note that the above labeling is well defined since G is a acyclic and can be computed with $|V|$ queries to f in a topological order.

In the following lemma we recall the labeling procedure from [CP18]. See Figure 2 for an illustration of the procedure.

Lemma 6.1 (CP Labeling Procedure, Lemma 3 in [CP18])**.** *The labeling of CP_d with respect to $f : \{0,1\}^* \to \{0,1\}^w$ can be computed in a topologicial ordering using $O(w \cdot d)$ bits of memory.*

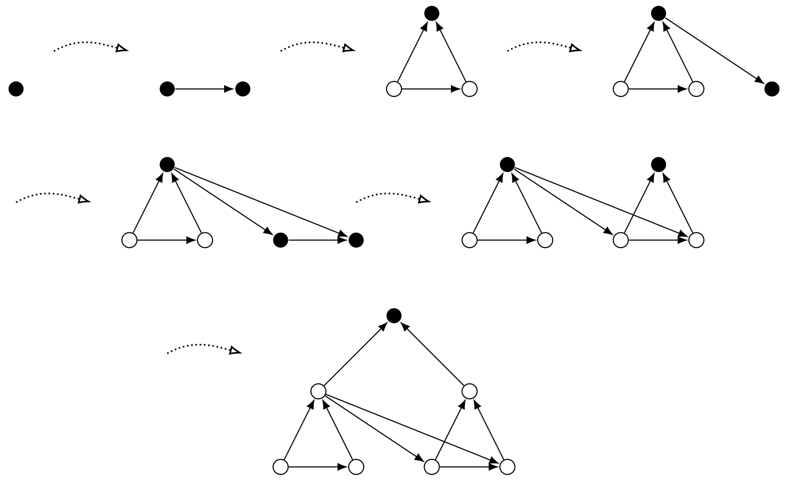

Figure 2: The CP labeling procedure for CP_2; black nodes represent labels the algorithm keeps track of; denoted $\mathcal{U}_t$ when computing the t label.

We require some additional properties from the labeling procedure for the modified DLM scheme. We describe the procedure bellow and prove several corresponding lemmas.

The Labeling Procedure. The algorithm is recursive. The base case CP_0 consist of one node and labeling is trivial. For $d \geq 1$, let v be the root of CP_d and let ℓ, r be the left and right parents of v, respectively. The subtree rooted by ℓ is isomorphic to CP_{d-1} and hence its labels can be computed recursively. Keep track of the last label $\mathcal{L}[\ell]$ and proceed to the subtree rooted by r. Apart from edges coming from ℓ to the leaves, it is isomorphic to CP_{d-1}. One can use $\mathcal{L}[\ell]$, which is in memory, and recursively compute the right subtree. Keep track of the last label $\mathcal{L}[r]$. Compute the root label $\mathcal{L}[v] = f(v, \mathcal{L}[\ell], \mathcal{L}[r])$ and forget both $\mathcal{L}[\ell]$ and $\mathcal{L}[r]$. Output the root label $\mathcal{L}[v]$.

Incremental Labeling. We consider an incremental version of the above procedure. Fix $d \in \mathbb{N}$ and let $v_1, v_2, \ldots, v_T$ be the topological ordering in which labels are being outputted by the CP labeling procedure for depth d. Let $\mathcal{U}_t \subseteq V$ be the set of nodes the procedure keeps track of after outputting v_t. For the incremental version, we are interested in algorithms that efficiently compute $\mathcal{U}_t$ and v_t given t. We show that $\mathcal{U}_t$ and v_t can be computed efficiently given t.

Lemma 6.2 (Computing $\mathcal{U}_t$ Efficiently)**.** *For CP_d, computing $\mathcal{U}_t$ can be done in* $\mathrm{poly}(d)$ *time.*

Proof. The algorithm is recursive and follows the same post-order traversal as the labeling procedure. For CP_d, let v be the root node and let ℓ, r be its left and right parents respectively. If $t = 2^{d+1} - 1$ return the root node, $\{v\}$. If $t \leq 2^d - 1$, proceed to recursively compute $\mathcal{U}_t$ on the left subtree rooted by ℓ, which is isomorphic to CP_{d-1}. Output the resulting set. Else $t > 2^d - 1$, recursively compute $\mathcal{U}_{t-(2^d-1)}$ on the subtree rooted by r while ignoring the extra edges from ℓ. This graph is also isomorphic to CP_{d-1}. Append ℓ to the resulting set and output.

The algorithm's correctness follows by induction. For efficiency, notice that the depth of the recursion is d and every level takes $\mathrm{poly}(d)$ time. □

Lemma 6.3 (Computing v_t Efficiently)**.** *For CP_d, computing v_t can be done in* $\mathrm{poly}(d)$ *time.*

Proof. Notice that by construction of the CP labeling procedure, $v_1, \ldots, v_T$ is a post-order traversal on the binary tree B_d which is the base of CP_d. Finding the t-th node in a post-order traversal can be done by a simple recursion with efficiency $\mathrm{poly}(d)$. □

Lemma 6.4 (One Vertex at a Time)**.** *For every $t \in [2, T]$ we have $\mathcal{U}_t = \{v_t\} \uplus (\mathcal{U}_t \cap \mathcal{U}_{t-1})$.*

Proof. The theorem statement is equivalent to proving $\mathcal{U}_t \setminus \mathcal{U}_{t-1} = \{v_t\}$. By definition we have that $v_t \in \mathcal{U}_t$. Apart from v_t, the procedure only forget nodes, hence $\mathcal{U}_t \subseteq \{v_t\} \cup \mathcal{U}_{t-1}$. It is left to show that $v_t \notin \mathcal{U}_{t-1}$. Indeed, using the above, by induction $\mathcal{U}_{t-1} \subseteq \{v_1, \ldots, v_{t-1}\}$, in particular $v_t \notin \mathcal{U}_{t-1}$. □

6.2 Modified DLM

In this section, we present a modified version of the incremental proof of sequential work (iPoSW) due to [DLM19]. The main difference, apart from syntax, is that we require stricter conditions when verifying nodes.

Parameters. Let λ be a computational security parameter. Let $T(\lambda) = T = 2^{\lambda+1} - 1$ and let $s(\lambda) = s = \Theta(\lambda^3)$ that is a power of 2. Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ and $\mathsf{H}' : \{0,1\}^* \to \{0,1\}^{3s}$ be independently chosen random oracle ensembles depending on λ.

Remark 6.1 (Random Coins Count)**.** *The use of $3s$ random coins allows to sample a statistically close to uniform subset. See [DLM19] for details.*

Syntax. The scheme consists of two algorithms, $\mathsf{DLM.P}^{\mathsf{H},\mathsf{H}'}(\chi, t, \pi)$, $\mathsf{DLM.V}^{\mathsf{H},\mathsf{H}'}(\chi, t, \pi)$ with the following syntax.

1. $\mathsf{DLM.P}^{\mathsf{H},\mathsf{H}'}(\chi, t, \pi)$ is a deterministic algorithm with oracle access to H, H'. Given a statement χ, a natural number t and proof π, it outputs a proof π'.

2. $\mathsf{DLM.V}^{\mathsf{H},\mathsf{H}'}(\chi, t, \pi)$ is a deterministic algorithm with oracle access to H, H'. Given a statement χ, a natural number t and proof π, it outputs ACC or REJ.

Proof Structure. The proofs π are of the form $\pi = (\mathcal{L}, (u_i, S_{u_i}, \mathcal{P}_{u_i})_{i=1}^m)$ where:

- $\mathcal{L}$ is a partial vertex labeling for CP_λ.
- u_i is a node of CP_λ.
- S_{u_i} is a set of challenge leaves, indexed by the node u_i.
- $\mathcal{P}_{u_i}$ is a set of candidate authentication paths, indexed by the node u_i.
- $\mathsf{Path} \in \mathcal{P}_{u_i}$ is an ordered list of the form $\mathsf{Path} = (w_j)_j$ where each w_j is a node of CP_λ.

Remark 6.2 (Labeling Data Structure)**.** *The partial vertex labeling $\mathcal{L}$ is given by a dictionary data structure. Where keys are nodes v and the value associated is the corresponding label $\mathcal{L}[v]$. A node which is not in the dictionary is given the default value $\perp$. Let n be the amount of nodes v for which $\mathcal{L}[v] \neq \perp$. We use a dictionary implementation where size and retrieval time are* $\mathrm{poly}(n)$*.*

6.2.1 Algorithms Description

Conventions. A random oracle salted with the statement $\chi \in \{0,1\}^*$ is denoted by $\mathsf{H}_\chi := \mathsf{H}(\chi, \cdot)$. Let $v_1, v_2, \ldots, v_T$ be the topological order given by Lemma 6.1 for nodes of CP_λ. Recall that $\mathcal{U}_t$ is the list of nodes the CP labeling procedure keeps track of when computing the label for v_t. We extend the definition such that $\mathcal{U}_0 = \emptyset$. For a set of candidate authentication paths $\mathcal{P}$ we define the set

$$A_{\mathcal{P}} := \left\{ v \;\middle|\; \begin{array}{l} \mathsf{Path} \in \mathcal{P} \\ w \in \mathsf{Path} \\ v = w \text{ or } v \in \mathsf{parents}(w) \end{array} \right\},$$

the set of nodes whose labels are used in Merkle authentication verification.

```
DLM.P^{H,H'}(χ, t, π)
Input:
1. String $\chi \in \{0,1\}^\lambda$.
2. Time $t \in [T]$.
3. Candidate proof $\pi = (\mathcal{L}, (u_i, S_{u_i}, \mathcal{P}_{u_i})_{i=1}^m)$ .
Algorithm:
1. If $\{u_1, \ldots, u_m\} \neq \mathcal{U}_{t-1}$ output $\varepsilon$.
2. Set $\mathcal{L}[v_t] = \mathsf{H}_\chi(v_t, \mathcal{L}[p_1], ..., \mathcal{L}[p_r])$ where $(p_1, ..., p_r) = \mathsf{parents}(v_t)$ in lexicographic order.
3. If $v_t$ is a leaf, let $S_{v_t} = \{v_t\}$ and $\mathcal{P}_{v_t} = \{(v_t)\}$.
4. Otherwise, $v_t$ has two parents, $\ell, r \in \mathcal{U}_{t-1}$.
    (a) Sample a random subset $S_{v_t}$ of size $\min(s, |\mathsf{leaves}(v_t)|)$ from $S_\ell \cup S_r$ using $\mathsf{rand} \leftarrow \mathsf{H}'_\chi(v_t, \mathcal{L}[v_t])$ as random coins.
    (b) Let $\mathcal{P}_{v_t} = \emptyset$. For every $\mathsf{Path} \in \mathcal{P}_\ell \cup \mathcal{P}_r$ starting from a leaf in $S_{v_t}$, extend with $v_t$ and append to $\mathcal{P}_{v_t}$.
5. Remove labels from $\mathcal{L}$ for all nodes but $\bigcup_{u \in \mathcal{U}_t} A_{\mathcal{P}_u}$.
6. Output $\pi' = (\mathcal{L}, (u, S_u, \mathcal{P}_u)_{u \in \mathcal{U}_t})$.
```

Note that in the above we use the efficient algorithms due to computing $\mathcal{U}_t$ efficiently Lemma 6.2 and computing v_t efficiently Lemma 6.3 to compute $\mathcal{U}_{t-1}, \mathcal{U}_t$ and v_t. We also use one vertex at a time Lemma 6.4 in Line 5 and Line 6.

In Line 5 we remove labels that are not used by the verifier. By doing so, the proof size remains short. Jumping forward, the verifier checks that indeed the partial vertex labeling $\mathcal{L}$ only contains the relevant labels and reject otherwise. This allows us to immediately reject proofs that are too long — important for efficiency of the scheme.

```
DLM.V^{H,H'}(χ, t, π)
Input:
1. String $\chi \in \{0,1\}^\lambda$.
2. Time $t \in [T]$.
3. Candidate proof $\pi = (\mathcal{L}, (u_i, S_{u_i}, \mathcal{P}_{u_i})_{i=1}^m)$ .
Algorithm:
1. Verify that $u_1, \ldots, u_m$ are distinct and $\{u_1, \ldots, u_m\} = \mathcal{U}_{t-1}$. Otherwise, output REJ.
2. Short Proof Verification:
   Verify the nodes having label in $\mathcal{L}$ are exactly $\bigcup_{u \in \mathcal{U}_{t-1}} A_{\mathcal{P}_u}$. Otherwise, output REJ.
3. Authentication Path Verification:
   For $i = 1, \ldots, m$:
    (a) Validate $S_{u_i} \subseteq \mathsf{leaves}(u_i)$ and is of size $\min(s, |\mathsf{leaves}(u_i)|)$. Otherwise output REJ.
    (b) Check that every path in $\mathcal{P}_{u_i}$ is a valid path from a leaf in $S_{u_i}$ to $u_i$. Check that
        every leaf in $S_{u_i}$ has exactly one corresponding path in $\mathcal{P}_{u_i}$. Otherwise output REJ.
    (c) Let $(\mathsf{Path}_1, \ldots, \mathsf{Path}_k) = \mathcal{P}_{u_i}$ ordered in lexicographic order of corresponding
        source leaves.
    (d) For $j = 1, \ldots, k$:
         i. If $\mathsf{VerifyMerklePath}^{\mathsf{H}}(\chi, \mathsf{Path}_j, \mathcal{L})$ rejects, output REJ.
        ii. If $\mathsf{VerifyRandChoice}^{\mathsf{H}'}(\chi, \mathsf{Path}_j, \mathcal{L}, j)$ rejects, output REJ.
4. Output ACC.
```

In the above description we used the subroutines $\mathsf{VerifyAuthPath}^{\mathsf{H}}$ and $\mathsf{VerifyRandChoice}^{\mathsf{H}'}$ that are described in the following.

Merkle Authentication Path Verification $\mathsf{VerifyAuthPath}^{\mathsf{H}}(\chi, \mathsf{Path}, \mathcal{L})$**:** Let $\mathsf{Path} = (w_j)_j$ be a candidate authentication path. For every node w_j with $(p_1, \ldots, p_r) = \mathsf{parents}(w_j)$ ordered in lexicographic order, verify

$$\mathcal{L}[w_j] = \mathsf{H}_\chi(w_j, \mathcal{L}[p_1], \ldots, \mathcal{L}[p_r]) \ .$$

If all checks pass, output ACC, otherwise output REJ.

Random Choice Verification $\mathsf{VerifyRandChoice}^{\mathsf{H}'}(\chi, \mathsf{Path}, \mathcal{L}, \mathsf{ind})$**:** We give a sketch of the verification, for further details see [DLM19]. Let $\mathsf{Path} = w_1 \to \cdots \to w_r$ be a path where $w_1 \in \mathsf{leaves}(w_r)$. In order to verify the random choice done by H'_χ, we recreate the choice going from the top down. We are given the index of Path in $\mathcal{P}$ ordered by source leaves, let us denote it by ind. This is also the index of the leaf that was picked in the last random choice.

Note that since s is a power of 2, the random choice procedure used by the prover either chooses an s-size subset from a $2s$-size set or takes the whole set. Starting from w_r, use H'_χ to generate the random coins rand. Using rand pick an s-size subset from the set $[2s]$. Let i be the ind-th element in the resulting subset. If $i \leq s$, we know that w_{r-1} should be a left parent to w_r. Otherwise $i > s$ which tells us that w_{r-1} should be a right parent.

We continue to verify w_{r-1}, changing ind appropriately according to w_{r-1} being a left or right parent. More concretely, if $i \leq s$ then $\mathsf{ind} := i$, else $i > s$ then $\mathsf{ind} := i - s$. We continue with this procedure until we get to a node with less than s leaves. If all verifications of the path passed, output ACC, otherwise output REJ.

Efficiency. All the computations done above are polynomial in the input length that is of size $\text{poly}(\lambda, |\pi|)$. Notice that due to the short proof verification (Line 2 of $\mathsf{DLM.V}$), the size of all accepted proofs is $\text{poly}(\lambda)$. Therefore, for all accepted proofs the runtime is $\text{poly}(\lambda)$ for a fixed polynomial. Hence, we may modify $\mathsf{DLM.V}$ to reject and $\mathsf{DLM.P}$ to output ε once this $\text{poly}(\lambda)$ time limit exceeds. This way, the efficiency of $\mathsf{DLM.V}$ and $\mathsf{DLM.P}$ is $\text{poly}(\lambda)$ for every input.

Remark 6.3 (Proofs are Not Computationally Unique)**.** *Proofs in the DLM scheme are not computationally unique. To see why, consider the following adversary that attempts to find a collision for accepting proofs of time $t = \Theta(s^2) = \Theta(\lambda^6)$ that is of the form $2^{k+1} - 1$. The first proof is generated honestly. For the second proof, select uniformly at random $u \leftarrow \mathsf{leaves}(v_t)$. Compute the proof honestly up until the label for u is computed. Alter the label for u. Continue as the honestly generated proof. With overwhelming probability, the two proofs generated are different. The second proof is accepting if there is no authentication path starting with u. Since t is of the form $2^{k+1} - 1$, there are s authentication paths starting with leaves in $\mathsf{leaves}(v_t)$. Also, all the leaves have the same probability to be chosen. There are 2^k leaves, therefore the probability u is selected is $s/2^k = O(\lambda^{-3})$. Thus, the probability of accepting is $1 - \Omega(\lambda^{-3})$. The described adversary is efficient and successful with noticeable probability.*

6.2.2 Incremental Completeness

In this section, we prove that the modified construction we propose achieves incremental completeness.

Proposition 6.1 (Incremental Completeness of Modified DLM)**.** *For every security parameter $\lambda \in \mathbb{N}$, time $t \in [T-1]$, candidate proof $\pi \in \{0,1\}^*$ and statement $\chi \in \{0,1\}^\lambda$:*

$$\textit{If } \mathsf{DLM.V}^{\mathsf{H},\mathsf{H}'}(\chi, t, \pi) = \mathsf{ACC} \textit{ then } \mathsf{DLM.V}^{\mathsf{H},\mathsf{H}'}(\chi, t+1, \pi') = \mathsf{ACC} \ ,$$

where $\pi' = \mathsf{DLM.P}^{\mathsf{H},\mathsf{H}'}(\chi, t, \pi)$.

Proof. Fix $\lambda \in \mathbb{N}, t \in [T-1], \chi \in \{0,1\}^\lambda$ and $\pi \in \{0,1\}^*$ such that $\mathsf{DLM.V}^{\mathsf{H},\mathsf{H}'}(\chi, t, \pi) = \mathsf{ACC}$. Let $\pi' = \mathsf{DLM.P}^{\mathsf{H},\mathsf{H}'}(\chi, t, \pi)$. Parse $\pi = (\mathcal{L}, (u_i, S_{u_i}, \mathcal{P}_{u_i})_{i=1}^m)$ and $\pi' = (\mathcal{L}', (u'_i, S'_{u'_i}, \mathcal{P}'_{u'_i})_{i=1}^k)$.

Since π passes verification we have that $\{u_1, \ldots, u_m\} = \mathcal{U}_{t-1}$[3] and $u_1, \ldots, u_m$ are distinct. By construction of $\mathsf{DLM.P}$ this implies that $\{u'_1, \ldots, u'_k\} = \mathcal{U}_t$ and $u'_1, \ldots, u'_k$ are distinct. We also have by construction that the nodes having labels in $\mathcal{L}'$ are exactly the ones verified by short proof verification (Line 2 of $\mathsf{DLM.V}$).

Next, in the authentication path verification (Line 3 of $\mathsf{DLM.V}$) the checks are performed on every vertex in $\{u'_i\} = \mathcal{U}_t$ separately. By the one vertex at a time Lemma 6.4 we have $\mathcal{U}_t = \{v_t\} \uplus (\mathcal{U}_t \cap \mathcal{U}_{t-1})$. We consider two cases: 1) $u \in \mathcal{U}_t \cap \mathcal{U}_{t-1}$ and 2) $u = v_t$.

[3]For $t = 1$, recall that we defined $\mathcal{U}_0 = \emptyset$.

Case 1. Fix $u \in \mathcal{U}_t \cap \mathcal{U}_{t-1}$. Fix $u \in \mathcal{U}_t \cap \mathcal{U}_{t-1}$. The checks on S'_u and $\mathcal{P}'_u$ in Line 3.a and Line 3.b pass since $S_u = S'_u$ and $\mathcal{P}'_u = \mathcal{P}_u$ by the construction. To see why the checks in Line 3.d involving $\mathsf{VerifyMerklePath}$ and $\mathsf{VerifyRandChoice}$ pass, we observe that both subroutines use the labels of nodes contained in $A_{\mathcal{P}'_u}$, where recall $A_{\mathcal{P}'_u}$ is the set of all nodes in $\mathcal{P}'_u$ and their parents. It suffices to prove that for every node $v \in A_{\mathcal{P}'_u}$ we have $\mathcal{L}'[v] = \mathcal{L}[v]$ as this implies that the checks due to $\mathsf{VerifyMerklePath}$ and $\mathsf{VerifyRandChoice}$ pass verification. Indeed, note that since $A_{\mathcal{P}'_u} \subseteq \{v_1, \ldots, v_{t-1}\}$ we have $v_t \notin A_{\mathcal{P}'_u}$. We further have by construction (Line 5 of $\mathsf{DLM.P}$) that labels of $A_{\mathcal{P}'_u}$ are not discarded. Hence the labels of $A_{\mathcal{P}'_u}$ remain unchanged. This concludes the proof of this case.

Case 2. If v_t is a leaf, all checks pass by construction. Otherwise, by the structure of the CP graph, v_t has two parents $\ell, r \in \mathcal{U}_{t-1}$.

Line 3.a. We have that $S'_{v_t} \subseteq S_\ell \cup S_r$. By the verification on S_ℓ and S_r, we have $S_\ell \subseteq \mathsf{leaves}(\ell)$ and $S_r \subseteq \mathsf{leaves}(r)$. By properties of the CP graph, $\mathsf{leaves}(v_t) = \mathsf{leaves}(\ell) \cup \mathsf{leaves}(r)$ hence $S'_{v_t} \subseteq \mathsf{leaves}(v_t)$. By construction we have $|S_{v_t}| = \min(s, |\mathsf{leaves}(v_t)|)$. Therefore the check in Line 3.a of $\mathsf{DLM.V}$ pass.

Line 3.b. Consider $\mathsf{Path}' \in \mathcal{P}'_{v_t}$. It is constructed by taking $\mathsf{Path} \in \mathcal{P}_\ell \cup \mathcal{P}_r$, that starts from a leaf in S'_{v_t} and extending it with v_t. Since Path is a valid path to ℓ or r, parents of v_t, the resulting path is a valid path from a leaf in S'_{v_t} to v_t. Since $\mathcal{P}_\ell$ and $\mathcal{P}_r$ contain all paths corresponding with leaves from S_ℓ and S_r, respectively, in $\mathcal{P}'_{v_t}$ there are all paths corresponding to leaves in $S'_{v_t} \subseteq S_\ell \cup S_r$. Therefore the check in Line 3.b of $\mathsf{DLM.V}$ pass.

Line 3.d. We continue to prove that $\mathsf{VerifyMerklePath}$ passes on paths in $\mathcal{P}'_{v_t}$. By construction of the prover (Line 5 in $\mathsf{DLM.P}$), we have that for every label that corresponds to a node in $A_{\mathcal{P}_{v_t}}$ is not discarded. Note that the only other label that can be changed is the label corresponding to v_t. Hence, for every node $v \in A_{\mathcal{P}_{v_t}} \setminus \{v_t\}$ that $\mathcal{L}'[v] = \mathcal{L}[v]$. For the verification on v_t, in Line 2 of $\mathsf{DLM.P}$, we construct its label to be

$$\mathcal{L}'[v_t] = \mathsf{H}_\chi(v_t, \mathcal{L}[\ell], \mathcal{L}[r]) \ .$$

We are left to show that $\mathcal{L}'[\ell] = \mathcal{L}[\ell]$ and $\mathcal{L}'[r] = \mathcal{L}[r]$. Indeed, ℓ, r are parents of v_t. Since $\mathcal{P}'_{v_t}$ is non-empty and $v_t \in \mathsf{Path}'$ for every $\mathsf{Path}' \in \mathcal{P}'_{v_t}$ also $\ell, r \in A_{\mathcal{P}_{v_t}}$.

The random choice verification pass by construction, for more details see the Random Choice Verification paragraph.

We conclude that $\mathsf{DLM.V}^{\mathsf{H},\mathsf{H}'}(\chi, t+1, \pi') = \mathsf{ACC}$, proving the proposition. $\square$

6.2.3 Soundness

In this section, we state the soundness property of the modified DLM system. While we altered the DLM construction, we did so by enforcing stricter verification conditions. Accordingly, we inherit the soundness of the original scheme — every valid proof in the modified scheme corresponds to a valid proof in the original scheme without the use of the random oracle in the correspondence.

Theorem 6.1 (DLM Soundness, Follows From [DLM19])**.** *Let $\mathcal{A}^{\mathsf{H},\mathsf{H}'} = \{\mathcal{A}_\lambda^{\mathsf{H},\mathsf{H}'}\}$ be an oracle aided adversary that makes at most $q = q(\lambda)$ total queries to both H and H'. For every $\lambda, d \in \mathbb{N}$ with $d \leq \lambda$, such that $\mathcal{A}$ makes less than 2^d sequential queries to H we have:*

$$\Pr\left[\mathsf{DLM.V}^{\mathsf{H},\mathsf{H}'}(\chi, T_d, \pi) = \mathsf{ACC} \;\middle|\; \begin{array}{c} \chi \leftarrow \{0,1\}^\lambda \\ \pi \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{H}'}(\chi) \end{array}\right] \leq O(q^2)2^{-\Omega(\lambda)} \;\;,$$

where $T_d = 2^{d+1} - 1$.

Corollary 6.1 (Sequential Hardness)**.** *Let $d = d(\lambda) \leq \lambda$ be a depth parameter. For every oracle aided adversary $\mathcal{A}^{\mathsf{H},\mathsf{H}'} = \{\mathcal{A}_\lambda^{\mathsf{H},\mathsf{H}'}\}_{\lambda \in \mathbb{N}}$ of depth 2^d and size $2^{o(\lambda)}$, we have for every $\lambda \in \mathbb{N}$:*

$$\Pr\left[\mathsf{DLM.V}^{\mathsf{H},\mathsf{H}'}(\chi, T_d, \pi) = \mathsf{ACC} \;\middle|\; \begin{array}{c} \chi \leftarrow \{0,1\}^\lambda \\ \pi \leftarrow \mathcal{A}^{\mathsf{H},\mathsf{H}'}(\chi) \end{array}\right] \leq 2^{-\Omega(\lambda)} \;\;,$$

where $T_d = 2^{d+1} - 1$.

Single Oracle. The DLM construction is described in terms of two random oracles H, H' (which is done mostly to simplify the analysis). From hereon, it will be simpler to think of a single random oracle $\mathcal{O} : \{0,1\}^* \to \{0,1\}$. The algorithms $\mathsf{DLM.P}^{\mathcal{O}}, \mathsf{DLM.V}^{\mathcal{O}}$ simulate oracle calls to H, H' in the above construction by setting an appropriate prefix for every output bit in H and H'. The security reduction incurs a multiplicative polynomial loss in efficiency and a multiplicative logarithmic loss in depth (both in terms of λ).

Indeed, consider an adversary $\mathcal{A}$ attempting to break the scheme with $\mathcal{O}$. In order reduce this adversary to $\mathcal{A}'$ attacking the scheme using H, H' we modify the oracle gates to $\mathcal{O}$ in $\mathcal{A}$ by examining the prefix and simulating the output of the associate output bit in H or H'. In this reduction, comparing the prefix, incurs a logarithmic loss in depth and polynomial loss in size of the new adversary $\mathcal{A}'$. We conclude this in the following corollary.

Corollary 6.2 (Single Oracle Sequential Hardness)**.** *Let $d = d(\lambda) \leq \lambda$ be a depth parameter. For every oracle aided adversary $\mathcal{A}^{\mathcal{O}} = \{\mathcal{A}_\lambda^{\mathcal{O}}\}_{\lambda \in \mathbb{N}}$ of depth $o(2^d / \log \lambda)$ and size $2^{o(\lambda)}$, we have for every $\lambda \in \mathbb{N}$:*

$$\Pr\left[\mathsf{DLM.V}^{\mathcal{O}}(\chi, T_d, \pi) = \mathsf{ACC} \;\middle|\; \begin{array}{c} \chi \leftarrow \{0,1\}^\lambda \\ \pi \leftarrow \mathcal{A}^{\mathcal{O}}(\chi) \end{array}\right] \leq 2^{-\Omega(\lambda)} \;\;,$$

where $T_d = 2^{d+1} - 1$.

By considering $d = \lambda$, we derive the following corollary:

Corollary 6.3 (Exponential Hardness)**.** *For every oracle aided adversary $\mathcal{A}^{\mathcal{O}} = \{\mathcal{A}_\lambda^{\mathcal{O}}\}_{\lambda \in \mathbb{N}}$ of size $2^{o(\lambda)}$ and $\lambda \in \mathbb{N}$:*

$$\Pr\left[\mathsf{DLM.V}^{\mathcal{O}}(\chi, T_\lambda, \pi) = \mathsf{ACC} \;\middle|\; \begin{array}{c} \chi \leftarrow \{0,1\}^\lambda \\ \pi \leftarrow \mathcal{A}^{\mathcal{O}}(\chi) \end{array}\right] \leq 2^{-\Omega(\lambda)} \;\;,$$

where $T_\lambda = 2^{\lambda+1} - 1$.

6.3 Hard instances.

In this section, we consider the problem $R_d^{\mathcal{O}}$, where instances are strings χ and a witness for χ is a corresponding proof of sequential work in the modified DLM scheme for time $T_d = 2^{d+1} - 1$. We show that $R_d^{\mathcal{O}}$ lies in $\mathsf{PLS}^{\mathcal{O}}$.

In the following, a depth parameter $d(\lambda)$ is a polynomial-time computable function. We require this for the efficiency of the reduction.

Definition 6.4. *Let $d = d(\lambda) \leq \lambda$ be a depth parameter. The search problem $R_d^{\mathcal{O}}$ is defined as follows:*

$$R_d^{\mathcal{O}} = \left\{(\chi, \pi) \;\middle|\; \mathsf{DLM.V}^{\mathcal{O}}(\chi, T_d(|\chi|), \pi) = \mathsf{ACC}\right\} \quad ,$$

where $T_d = 2^{d+1} - 1$.

Note that $R_d^{\mathcal{O}}$ is in $\mathsf{TFNP}^{\mathcal{O}}$. It is in $\mathsf{FNP}^{\mathcal{O}}$ by the efficiency of DLM. It is total by the completeness of DLM, since for every χ, there exists a proof π that passes verification – the honestly generated proof.

6.3.1 The Reduction to Local Search

Fix a depth parameter $d = d(\lambda) \leq \lambda$. We construct a polynomial-time search reduction (f_d, g_d) from $R_d^{\mathcal{O}}$ to $\mathsf{LS}^{\mathcal{O}}$.

Instance Translation: Let $\chi \in \{0,1\}^\lambda$ be an instance for the problem $R_d^{\mathcal{O}}$. We construct an $\mathsf{LS}^{\mathcal{O}}$ instance $f_d(\chi) = (\mathcal{S}^{\mathcal{O}}, \mathcal{F}^{\mathcal{O}})$ in the following. Intuitively, we embed the computation of the modified DLM PoSW in an $\mathsf{LS}^{\mathcal{O}}$ instance such that local maximums correspond to valid proofs for time $T_d = 2^{d+1} - 1$. Concretely, every node will be of a fixed polynomial length $\ell = \ell(\lambda) \geq \lambda$. Every vertex is of the form $(t, \pi) \in \{0,1\}^\ell$ where t is parsed as an integer in $[T_d]$ and π as a proof of the modified DLM scheme padded to length ℓ if needed. The size ℓ is determined by the size of valid proofs which is polynomial by the efficiency of modified DLM.

Let π_1 be a valid proof for time $t = 1$ in the modified DLM scheme. For the node (t, π), if the proof π is verified with respect to time t, the node will be given value t and be connected to $(t+1, \pi')$ where π' is the proof the prover generates given π. If π is rejected, it is given value -1 and is connected to $(1, \pi_1)$. A formal definition of the circuits is given bellow.

Successor Circuit $\mathcal{S}^{\mathcal{O}}(t, \pi)$

Hardwired: The instance χ.
Algorithm:

```
1. If $\mathsf{DLM.V}^{\mathcal{O}}(\chi, t, \pi) = \mathsf{REJ}$, output $(1, \pi_1)$.
2. If $t = T_d$, output $(t, \pi)$.
3. Compute $\pi' = \mathsf{DLM.P}^{\mathcal{O}}(\chi, t, \pi)$ and output $(t+1, \pi')$.
```

Value Circuit $\mathcal{F}^{\mathcal{O}}(t, \pi)$

Hardwired: The instance χ.
Algorithm:

```
1. If $\mathsf{DLM.V}^{\mathcal{O}}(\chi, t, \pi) = \mathsf{REJ}$, output $-1$.
2. Else, output $t$.
```

Witness Translation: The function $g_d(t, \pi)$ outputs π.

Efficiency. By the efficiency of the modified DLM scheme and the efficiency of the depth parameter $d(\lambda)$, the instance translation f_d is done in $\mathrm{poly}(\lambda)$ time. The witness translation g_d is also done in $\mathrm{poly}(\lambda)$ time since the length of nodes ℓ is of size $\mathrm{poly}(\lambda)$. This further implies that $\mathcal{S}^{\mathcal{O}}, \mathcal{F}^{\mathcal{O}}$ are of size $\mathrm{poly}(\lambda)$. Since the input length ℓ is at least λ we have that $\mathcal{S}^{\mathcal{O}}, \mathcal{F}^{\mathcal{O}}$ are polynomial-size circuits.

6.3.2 Security Analysis

In the following claim we rely on the incremental completeness proven in Section 6.2.2.

Claim 6.1. *Let $d = d(\lambda) \leq \lambda$ be a depth parameter. Fix $\lambda \in \mathbb{N}$ and $\chi \in \{0,1\}^{\lambda}$. Let $w = (t, \pi)$ be a local maximum of $(\mathcal{S}^{\mathcal{O}}, \mathcal{F}^{\mathcal{O}}) = f_d(\chi)$. We have $\mathsf{DLM.V}^{\mathcal{O}}(\chi, t, \pi) = \mathsf{ACC}$ and $t = T_d$.*

Proof. We have $\mathsf{DLM.V}^{\mathcal{O}}(\chi, t, \pi) = \mathsf{ACC}$ as otherwise w is given value -1 and his successor is $(1, \pi_1)$ of value 1. If $t < T_d$ then w is given value t and is connected to $(t+1, \pi')$ for $\pi' = \mathsf{DLM.P}^{\mathcal{O}}(\chi, t, \pi)$. By incremental completeness Proposition 6.1, we have that $\mathsf{DLM.V}^{\mathcal{O}}(\chi, t+1, \pi') = \mathsf{ACC}$. Therefore, the value of $(t+1, \pi')$ is $t+1$, and w is not a local maximum. We are left with $t = T_d$. This concludes the claim. $\square$

Theorem 6.2 (Hard Problems in $\mathsf{PLS}^{\mathcal{O}}$). *For every depth parameter $d = d(\lambda) \leq \lambda$, the search problem $R_d^{\mathcal{O}}$ lies in $\mathsf{PLS}^{\mathcal{O}}$.*

Proof. The class $\mathsf{PLS}^{\mathcal{O}}$ consists of all $\mathsf{TFNP}^{\mathcal{O}}$ search problems that are polynomial-time reducible to $\mathsf{LS}^{\mathcal{O}}$. The search problem $R_d^{\mathcal{O}} \in \mathsf{TFNP}^{\mathcal{O}}$ and by Claim 6.1, the tuple (f_d, g_d) is a valid polynomial-time reduction to $\mathsf{PLS}^{\mathcal{O}}$. Indeed, for every string χ, we have that all witnesses w' for the instance $(\mathcal{S}^{\mathcal{O}}, \mathcal{F}^{\mathcal{O}}) = f_d(\chi)$ are mapped under g_d to $w = \pi$ such that $\mathsf{DLM.V}^{\mathcal{O}}(\chi, T_d, \pi) = \mathsf{ACC}$ – a valid witness to $R_d^{\mathcal{O}}$. Hence $R_d^{\mathcal{O}} \in \mathsf{PLS}^{\mathcal{O}}$. $\square$

Corollary 6.4 (Exponential Hardness in $\mathsf{PLS}^{\mathcal{O}}$). *There exists search problems in $\mathsf{PLS}^{\mathcal{O}}$ that are average-case exponentially hard.*

Proof. By Theorem 6.2, for every depth parameter $d = d(\lambda) \leq \lambda$, the search problem $R_d^{\mathcal{O}}$ is in $\mathsf{PLS}^{\mathcal{O}}$. By Corollary 6.3, for $d = \lambda$ the search problem $R_\lambda^{\mathcal{O}}$ is exponentially average-case hard. $\square$

This implies that $\mathsf{LS}^{\mathcal{O}}$ is sub-exponentially hard due to the polynomial blowup the reduction incurs. We formulate it in the following corollary. In the following, n stands for the size of the $\mathsf{LS}^{\mathcal{O}}$ instance $(\mathcal{S}^{\mathcal{O}}, \mathcal{F}^{\mathcal{O}})$ and λ stands for the security parameter that is given as input to the sampler.

Corollary 6.5. *There exists an efficient sampler* HARD *of* $\mathsf{LS}^{\mathcal{O}}$ *instances and a constant* $\delta > 0$ *such that for every oracle aided adversary* $\mathcal{A}^{\mathcal{O}} = \{\mathcal{A}_n^{\mathcal{O}}\}_{n\in\mathbb{N}}$ *of size at most* 2^{n^δ} *and for every* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\mathcal{F}^{\mathcal{O}}(\mathcal{S}^{\mathcal{O}}(w)) \leq \mathcal{F}^{\mathcal{O}}(w) \;\middle|\; \begin{array}{c}(\mathcal{S}^{\mathcal{O}}, \mathcal{F}^{\mathcal{O}}) \leftarrow \mathsf{HARD}(1^\lambda) \\ w \leftarrow \mathcal{A}(\mathcal{S}^{\mathcal{O}}, \mathcal{F}^{\mathcal{O}})\end{array}\right] \leq 2^{-\Omega(\lambda)} \ ,$$

where $\mathcal{O} : \{0,1\}^* \to \{0,1\}$ *is a random oracle.*

Proof. The algorithm $\mathsf{HARD}(1^\lambda)$ samples $\chi \leftarrow \{0,1\}^\lambda$ and outputs $f_\lambda(\chi)$. Let $\delta > 0$ be a constant to be chosen later. Let $\mathcal{A} = \mathcal{A}^{\mathcal{O}} = \{\mathcal{A}_n^{\mathcal{O}}\}_{n\in\mathbb{N}}$ be an adversary of size at most 2^{n^δ} that succeeds in finding a local maximum for $(\mathcal{S}^{\mathcal{O}}, \mathcal{F}^{\mathcal{O}}) \leftarrow \mathsf{HARD}(1^\lambda)$ with probability $\varepsilon(\lambda)$. We construct an oracle aided adversary $\mathcal{A}' = \mathcal{A}'^{\mathcal{O}} = \{\mathcal{A}'^{\mathcal{O}}_\lambda\}_{\lambda\in\mathbb{N}}$ that attempts breaking the soundness of the modified DLM scheme as follows:

$\underline{\mathcal{A}'(\chi)}$:

1. Compute $(\mathcal{S}^{\mathcal{O}}, \mathcal{F}^{\mathcal{O}}) = f_\lambda(\chi)$.
2. Simulate $w' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathcal{S}^{\mathcal{O}}, \mathcal{F}^{\mathcal{O}})$
3. Compute $w = g_\lambda(w')$ and output w.

By the correctness of the reduction (f_λ, g_λ), the adversary $\mathcal{A}'$ succeeds with probability at least $\varepsilon(\lambda)$. Let $p(\lambda)$ be a polynomial bound on the runtime of f_λ, g_λ. If we choose δ such that $p(\lambda)^\delta = o(\lambda)$, the size of $\mathcal{A}'$ is at most $\mathrm{poly}(\lambda) + 2^{p(\lambda)^\delta} = 2^{o(\lambda)}$. Therefore, we can apply the soundness of the DLM scheme (Corollary 6.3). Thus

$$\varepsilon(\lambda) \leq \Pr\left[\mathsf{DLM.V}^{\mathcal{O}}(\chi, T, \pi) = \mathsf{ACC} \;\middle|\; \begin{array}{c}\chi \leftarrow \{0,1\}^\lambda \\ \pi \leftarrow \mathcal{A}'(\chi)\end{array}\right] \leq 2^{-\Omega(\lambda)} \ .$$

This concludes the proof. □

6.4 Depth Robust Instances

In this section, we deduce the existence of depth-robust, moderately hard, search problems in $\mathsf{PLS}^{\mathcal{O}}$. That is, search problems with instances that can be solved in polynomial time yet also require high sequential time.

Proposition 6.2 (Depth Robust Problems in $\mathsf{PLS}^{\mathcal{O}}$)**.** *Fix any constant* $\varepsilon > 0$*. For any large enough constant* α*, consider* $R_d^{\mathcal{O}}$ *for* $d(\lambda) = \alpha \log_2 \lambda$ *a depth parameter. The following properties hold for* $\lambda > 1$*:*

- ***Moderately Hard:*** *The search problem* $R_d^{\mathcal{O}}$ *can be solved in time* $\lambda^{(1+\varepsilon)\alpha}$*.*
- ***Depth Robust:*** *If* $\mathcal{A}$ *is an adversary of depth* $\lambda^{(1-\varepsilon)\alpha}$ *and size* $2^{o(\lambda)}$ *then:*

$$\Pr\left[(\chi, \pi) \in R_d^{\mathcal{O}} \;\middle|\; \begin{array}{c}\chi \leftarrow \{0,1\}^\lambda \\ \pi \leftarrow \mathcal{A}^{\mathcal{O}}(\chi)\end{array}\right] \leq 2^{-\Omega(\lambda)} \ .$$

Proof. Fix $\varepsilon > 0$ throughout the proof. We start by proving that $R_d^{\mathcal{O}}$ is moderately hard.

Moderately Hard: Fix $\lambda > 1$ and let $\chi \in \{0,1\}^\lambda$ be an instance of $R_d^{\mathcal{O}}$. We have $T_d = 2^{\alpha \log_2 \lambda + 1} - 1 \leq 2\lambda^\alpha \leq \lambda^{\alpha+1}$. By efficiency of DLM, the run-time of $\mathsf{DLM.P}^{\mathcal{O}}$ is at most λ^c for some fixed constant c, independent of ε and α. The proof π_1 for time $t = 1$ can also be generated in $\mathrm{poly}(\lambda)$ time. We assume w.l.o.g that it can be generated in time λ^c. By completeness of DLM, generating π_1 and applying the prover $T_d - 1$ times on it, results in a valid proof for time $t = T_d$. This is a valid witness for the instance χ in $R_d^{\mathcal{O}}$. The run-time of the above algorithm is at most

$$T_d \cdot \lambda^c \leq \lambda^{\alpha+1} \cdot \lambda^c \leq \lambda^{(1+\varepsilon)\alpha} \ ,$$

where the last inequality holds for large enough α, concretely $\alpha > (c+1)/\varepsilon$.

Depth Robust: For every $\alpha > 1$, we have $T_d = 2^{\alpha \log_2 \lambda + 1} - 1 \geq \lambda^\alpha$. Hence, $\lambda^{(1-\varepsilon)\alpha} = o(T_d / \log \alpha)$. By direct application of single oracle sequential hardness Corollary 6.2, the claim follows. □

References

[ABPW16] Omer Angel, Sébastien Bubeck, Yuval Peres, and Fan Wei. Local max-cut in smoothed polynomial time. *CoRR*, abs/1610.04807, 2016.

[AKV04] Tim Abbot, Daniel Kane, and Paul Valiant. On algorithms for nash equilibria. *Unpublished*, 2004.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 440–456, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, pages 111–120, New York, NY, USA, 2013. ACM.

[BCPR16] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. *SIAM J. Comput.*, 45(5):1910–1952, 2016.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48, May 2012.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, pages 258–275, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 474–482, 2017.

[BKK$^+$18] Saikrishna Badrinarayanan, Yael Tauman Kalai, Dakshita Khurana, Amit Sahai, and Daniel Wichs. Succinct delegation for low-space non-deterministic computation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 709–721, 2018.

[BKM18] Shant Boodaghians, Rucha Kulkarni, and Ruta Mehta. Nash equilibrium in smoothed polynomial time for network coordination games, 09 2018.

[BO06] Buresh-Oppenheim. On the tfnp complexity of factoring. *Unpublished*, 2006.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a nash equilibrium. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1480–1498, Oct 2015.

[CCR16] Ran Canetti, Yilei Chen, and Leonid Reyzin. On the correlation intractability of obfuscated pseudorandom functions. In *Proceedings, Part I, of the 13th International Conference on Theory of Cryptography - Volume 9562*, TCC 2016-A, pages 389–415, Berlin, Heidelberg, 2016. Springer-Verlag.

[CDT09] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *J. ACM*, 56(3):14:1–14:57, May 2009.

[CHK$^+$19a] 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`.

[CHK$^+$19b] Arka Rai Choudhuri, Pavel Hubáček, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a nash equilibrium is no easier than breaking fiat-shamir. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 1103–1114, New York, NY, USA, 2019. ACM.

[CP18] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 451–467, Cham, 2018. Springer International Publishing.

[DGP09] Constantinos Daskalakis, Paul Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. *SIAM J. Comput.*, 39:195–259, 02 2009.

[DLM19] Nico Döttling, Russell W. F. Lai, and Giulio Malavolta. Incremental proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 292–323, Cham, 2019. Springer International Publishing.

[DP11] Constantinos Daskalakis and Christos Papadimitriou. Continuous local search. In *Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 790–804, Philadelphia, PA, USA, 2011. Society for Industrial and Applied Mathematics.

[EFKP19] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. Cryptology ePrint Archive, Report 2019/619, 2019. `https://eprint.iacr.org/2019/619`.

[FPT04] Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 604–612, New York, NY, USA, 2004. ACM.

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.*, 67(4):205–214, 1998.

[GP18] Paul W. Goldberg and Christos H. Papadimitriou. Towards a unified complexity theory of total functions. *J. Comput. Syst. Sci.*, 94:167–192, 2018.

[GPS16] Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In *Advances in Cryptology – CRYPTO 2016*, pages 579–604, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[GVW02] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, 11(1-2):1–53, 2002.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 99–108, New York, NY, USA, 2011. ACM.

[HNY17] Pavel Hubácek, Moni Naor, and Eylon Yogev. The Journey from NP to TFNP Hardness. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 60:1–60:21, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[HY16] Pavel Hubek and Eylon Yogev. *Hardness of Continuous Local Search: Query Complexity and Cryptographic Lower Bounds*, pages 1352–1371. ACM, 2016.

[IM98] Hisao Ishibuchi and Tadahiko Murata. A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 28(3):392–403, Aug 1998.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512 – 530, 2001.

[Jer12] Emil Jerábek. Integer factoring and modular square roots. *CoRR*, abs/1207.5220, 2012.

[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.

[KNY17a] Ilan Komargodski, Moni Naor, and Eylon Yogev. White-box vs. black-box complexity of search problems: Ramsey and graph property testing. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 622–632, Oct 2017.

[KNY17b] Ilan Komargodski, Moni Naor, and Eylon Yogev. White-box vs. black-box complexity of search problems: Ramsey and graph property testing. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 622–632, 2017.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 1115–1124, New York, NY, USA, 2019. ACM.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: The power of no-signaling proofs. In *In Proceedings of the 46th annual ACM symposium on Theory of computing (STOC)*, pages 485–494. ACM, January 2014.

[LFKN90] Carsten Lund, Lance Fortnow, H Karloff, and Noam Nisan. Algebraic methods for interactive proof systems, 11 1990.

[LK73] S. Lin and Brain W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.*, 21(2):498–516, April 1973.

[Mer79] Ralph Merkle. *Secrecy, Authentication and Public Key Systems*. PhD thesis, Stanford University, Department of Electrical Engineering, June 1979.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Publicly verifiable proofs of sequential work. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 373–388, New York, NY, USA, 2013. ACM.

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

[Nas00] John C. Nash. The (dantzig) simplex method for linear programming. *Computing in Science Engineering*, 2(1):29–31, Jan 2000.

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498 – 532, 1994.

[Pie18] Krzysztof Pietrzak. Simple verifiable delay functions. Cryptology ePrint Archive, Report 2018/627, 2018. `https://eprint.iacr.org/2018/627`.

[PR17] Omer Paneth and Guy N. Rothblum. On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 283–315, Cham, 2017. Springer International Publishing.

[RSS17] Alon Rosen, Gil Segev, and Ido Shahaf. Can ppad hardness be based on standard cryptographic assumptions? In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 747–776, Cham, 2017. Springer International Publishing.

[RSW00] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, MIT, February 2000.

[SY91] Alejandro Schaffer and Mihalis Yannakakis. Simple local search problems that are hard to solve. *SIAM J. Comput.*, 20:56–87, 02 1991.

[SZZ18] Katerina Sotiraki, Manolis Zampetakis, and Giorgos Zirdelis. PPP-completeness with connections to cryptography. *CoRR*, abs/1808.06407, 2018.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *Theory of Cryptography*, pages 1–18, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

A IVC under KPY Assumption

In this section, we present the IVC scheme derived from the bootstrapping theorem by Kalai et al. [KPY19]. Soundness of the scheme is established assuming the KPY assumption (See appendix A.6). The analysis done in this section for IVC is very similar to the analysis in [KPY19] for delegation schemes. The main theorem we prove is Theorem A.1.

Remark A.1. *For ease of notation, in the following section we assume w.l.o.g $\lambda > 1$.*

Theorem A.1 (IVC from KPY Assumption)**.** *Fix any constants $c, \varepsilon > 0$. Let α be a large enough constant and let M be a Turing machine of run-time $T(\lambda) \leq c\lambda^{\alpha}$, configuration size $S(\lambda) \leq c\lambda$ and description size $|M| \leq c \log_2 \alpha$. Under the KPY assumption, there exists an IVC scheme with incremental completeness for M having efficiency $\mathbb{T}_{\mathsf{IVC}} = \lambda^{\varepsilon\alpha}$.*

A roadmap to this section. In appendix A.1, we recall the definition of quasi-arguments, the main ingredient in the IVC construction. In appendix A.2 we define GIVC, a generalization of IVC suitable for the recursive construction. Next, in appendix A.3 we give the recursive construction of GIVC. We proceed in appendix A.4.3 and appendix A.5 to analyze the GIVC construction and show it implies Theorem A.1. In appendix A.6 we recall the KPY assumption.

A.1 Quasi-Arguments

In this subsection we define quasi-arguments – non-interactive NP proof system with no-signaling extraction property. The definition given is a slightly altered version of the original by Kalai et al. [KPY19]. Namely, we don't consider adversaries that pass auxiliary information along with the attempted proof. Nonetheless, this weaker definition suffices for our needs and simplifies the construction.

Convention. Let $\sigma : I \subseteq [M] \to \{0,1\}$ be a partial assignment for an M-variate 3CNF formula φ. We say σ locally satisfies φ if every clause in φ that only contains variables in I is satisfied by σ. We denote by $[\![\varphi]\!]_\sigma$ the bit indicating whether or not σ satisfies φ.

Definition A.1 (Publicly Verifiable Quasi-Arguments for NP). *A publicly verifiable non-interactive quasi-argument for* NP *consists of three algorithms* $\mathsf{QA} = (\mathsf{QA.G}, \mathsf{QA.P}, \mathsf{QA.P})$*:*

- $\mathsf{QA.G}(\lambda, \varphi, n, K)$ *is a randomized algorithm that given a security parameter λ, an M-variate 3CNF formula φ, input length n and a locality parameter K, outputs a prover key* pk *and a verifier key* vk.
- $\mathsf{QA.P}(\mathsf{pk}, \sigma)$ *is a deterministic algorithm that given a prover key* pk *and an assignment $\sigma : [M] \to \{0,1\}$, outputs a proof π.*
- $\mathsf{QA.V}(\mathsf{vk}, y, \pi)$ *is a deterministic algorithm that given a verifier key* vk*, an input $y \in \{0,1\}^n$ and an arbitrary proof π, outputs* ACC *or* REJ.

We make the following requirements:

1. ***Completeness:*** *For every $\lambda, K, n, M \in \mathbb{N}$ such that $K, n \leq M \leq 2^\lambda$, an M-variate 3CNF formula φ and an assignment $\sigma : [M] \to \{0,1\}$ satisfying φ:*

$$\Pr\left[\mathsf{QA.V}(\mathsf{vk}, y, \pi) = \mathsf{ACC} \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{QA.G}(\lambda, \varphi, n, K) \\ \pi \leftarrow \mathsf{QA.P}(\mathsf{pk}, \sigma) \end{array}\right] = 1 \ ,$$

   where $y \in \{0,1\}^n$ such that $\forall i \in [n] : y_i = \sigma(i)$.

2. ***Efficiency:*** *In the completeness experiment above:*
   - *The running time of* $\mathsf{QA.G}$ *is bounded by* $\mathrm{poly}(\lambda, |\varphi|)$ *and the verifier key size is at most* $n \cdot \mathrm{poly}(\lambda, K)$.
   - *The running time of* $\mathsf{QA.P}$ *is bounded by* $\mathrm{poly}(\lambda, |\varphi|)$ *and the proof size is at most* $\mathrm{poly}(\lambda, K)$.
   - *The running time of* $\mathsf{QA.V}$ *is bounded by* $n \cdot \mathrm{poly}(\lambda, K)$.

3. ***No-Signaling Extraction:*** *Let $M = M(\lambda), n = n(\lambda), K = K(\lambda)$ be polynomials. There exists a PPT oracle machine $\mathcal{E}$, called the no-signaling extractor, such that for any M-variate formula $\varphi = \varphi_\lambda$ and efficient adversary $\mathcal{A}$ we have the following:*

(a) ***Correct Distribution:*** *For every efficient distinguisher D there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$:*

$$\left| \Pr\left[D(y) = 1 \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{QA.G}(\lambda, \varphi, n, K) \\ (y, \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{vk}) \\ \textit{if}\, \mathsf{QA.V}(\mathsf{vk}, y, \pi) = \mathsf{REJ} : \\ \textit{set}\; y = \bot \end{array} \right] - \Pr_{(y,\sigma) \leftarrow \mathcal{E}^{\mathcal{A}}(\varphi, \emptyset)}[D(y) = 1] \right| \leq \mu(\lambda) \enspace .$$

(b) ***Local Consistency:*** *There exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and set $I \subseteq [M]$ of size at most K:*

$$\Pr_{(y,\sigma) \leftarrow \mathcal{E}^{\mathcal{A}}(\varphi, I)}\left[y = \bot \quad \textit{or} \quad \begin{array}{l} \forall i \in I \cap [n] : y_i = \sigma(i) \\ \llbracket \varphi \rrbracket_\sigma = 1 \end{array} \right] \geq 1 - \mu(\kappa) \enspace ,$$

where $\llbracket \varphi \rrbracket_\sigma$ is the bit indicating if σ satisfies φ.

(c) ***No-signaling:*** *For every efficient distinguisher D there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and sets $I' \subseteq I \subseteq [M]$ of size at most K:*

$$\left| \Pr_{(y,\sigma) \leftarrow \mathcal{E}^{\mathcal{A}}(\varphi, I)}[D(y, \sigma\big|_{I'}) = 1] - \Pr_{(y,\sigma') \leftarrow \mathcal{E}^{\mathcal{A}}(\varphi, I')}[D(y, \sigma') = 1] \right| \leq \mu(\lambda) \enspace ,$$

where $\sigma\big|_{I'}$ is the restriction of σ to the domain I'.

We recall the following theorem from [KPY19].

Theorem A.2 (Quasi-Arguments under KPY Assumption, [KPY19]). *Assuming the KPY assumption, there exists publicly verifiable quasi-arguments for* NP.

A.2 GIVC with Incremental Completeness

In this section we define generalized incremental verifiable computation (GIVC) schemes with incremental completeness. The generalization of incremental verifible computation scheme is more suitable for the recursive construction given below. It is a generalization in a couple of ways. First, the proofs in GIVC attest that for a pair of configurations C, C' and a natural number t, we have that C' is the result of applying M_x for t times starting with C. This is unlike the case of IVC, where the configuration C is fixed to be M_x^0, the first configuration of the computation. Second, the public parameters are split to a verifier key and a prover key. This turns to be important for the efficiency of the recursive construction.

Conventions. Let M be a Turing machine. Let $S = S(\lambda)$ be the configuration size of M for inputs of length λ. Let x be a string and denote by $M_x : \{0,1\}^S \to \{0,1\}^S$ the transition function of M for input x. Recursively, for every $i \in \mathbb{N}$ denote $M_x^{(i)} = M_x \circ M_x^{(i-1)}$ and $M_x^{(1)} = M_x$. Define

$$\mathcal{U}_M := \left\{ (x, i, C, C') \;\middle|\; \begin{array}{l} x \in \{0,1\}^* \\ i \in \mathbb{N} \\ C, C' \in \{0,1\}^{S(|x|)} \\ M_x^{(i)}(C) = C' \end{array} \right\} \enspace .$$

Definition A.2 (GIVC with Incremental Completeness). *Let $T = T(\lambda)$ be a time parameter. A Generalized Incremental Verifiable Computation (GIVC) scheme for M, with incremental completeness, consists of three algorithms* $\mathsf{GIVC} = (\mathsf{GIVC.G}, \mathsf{GIVC.P}, \mathsf{GIVC.V})$*:*

- $\mathsf{GIVC.G}(x)$ *is a randomized algorithm that given $x \in \{0,1\}^\lambda$, outputs a prover key* pk *and a verifier key* vk.
- $\mathsf{GIVC.P}(\mathsf{pk}, t, C, C', \pi)$ *is a deterministic algorithm that given a prover key* pk, *a natural number t, arbitrary configurations C, C' and arbitrary proof π, outputs a proof π'.*
- $\mathsf{GIVC.V}(\mathsf{vk}, t, C, C', \pi)$ *is a deterministic algorithm that given a verifier key* vk, *a natural number t, arbitrary configurations C, C' and arbitrary proof π, outputs* ACC *or* REJ.

We make the following requirements:

1. ***Incremental Completeness:***

   (a) *For every $(x, t, C, C') \in \mathcal{U}_M$ such that $t < T$ and candidate proof $\pi \in \{0,1\}^*$:*

   $$\Pr\left[\begin{array}{l}\mathsf{GIVC.V}(\mathsf{vk}, t, C, C', \pi) = \mathsf{ACC} \implies \\ \mathsf{GIVC.V}(\mathsf{vk}, t+1, C, M_x(C'), \pi') = \mathsf{ACC}\end{array} \;\middle|\; \begin{array}{l}(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{GIVC.G}(x) \\ \pi' = \mathsf{GIVC.P}(\mathsf{pk}, t, C, C', \pi)\end{array}\right] = 1 \enspace ,$$

   where M_x is the transition circuit of M with input x.

   (b) *For every security parameter $\lambda \in \mathbb{N}$, input $x \in \{0,1\}^\lambda$ and configuration $C \in \{0,1\}^S$:*

   $$\Pr\left[\mathsf{GIVC.V}(\mathsf{vk}, 0, C, C, \varepsilon) \mid (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{GIVC.G}(x)\right] = 1 \enspace ,$$

   where ε is the empty proof.

2. ***Soundness:*** *For every efficient adversary $\mathcal{A}$, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^\lambda$:*

   $$\Pr\left[\begin{array}{l}(x, t, C, C') \notin \mathcal{U}_M \\ \mathsf{GIVC.V}(\mathsf{vk}, t, C, C', \pi^*) = \mathsf{ACC}\end{array} \;\middle|\; \begin{array}{l}(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{GIVC.G}(x) \\ (t, C, C', \pi^*) \leftarrow \mathcal{A}(x, \mathsf{pk}, \mathsf{vk})\end{array}\right] \le \mu(\lambda) \enspace .$$

3. ***Efficiency:*** *The efficiency of the* GIVC*, denoted by $\mathbb{T}_{\mathsf{GIVC}}(\lambda)$, is the maximal worst-case run-time among* $\mathsf{GIVC.G}, \mathsf{GIVC.P}, \mathsf{GIVC.V}$ *for input having security parameter λ. We require $\mathbb{T}_{\mathsf{GIVC}}(\lambda) \le p(\lambda)$ for a fixed polynomial p.*

In the following we define the notion of final verifier efficiency for a generalized incremental computation scheme. Putting it simply, if $T(\lambda)$ is the time parameter for a GIVC, the final verifier efficiency is the run-time of the verifier when the time paramter is set to the final time $t = T(\lambda)$. The efficiency for the final time may very well be much smaller than the efficiency of the verifier for an arbitrary time smaller than $T(\lambda)$. Jumping forward, in the recursive construction we are able to minimize the final verifier efficiency that in turn let us construct an efficient GIVC for arbitrary time.

Definition A.3 (Final Verifier Efficiency). *Let* $\mathsf{GIVC} = (\mathsf{GIVC.G}, \mathsf{GIVC.P}, \mathsf{GIVC.V})$ *be a generalized incremental verifiable computation scheme with incremental completeness having time parameter $T(\lambda)$. The final verifier efficiency of* GIVC *is the worst-case run-time of* $\mathsf{GIVC.V}$ *for input having security parameter λ and time parameter set to $T(\lambda)$.*

Notation. Let C, C' be configurations, t time, π proof and vk verifier key of GIVC. Let us denote the following:

$$C \xrightarrow[t,\mathsf{vk}]{\pi} C' \iff \mathsf{GIVC.V}(\mathsf{vk}, t, C, C', \pi) = \mathsf{ACC}$$

A.3 Recursive GIVC Construction

Let M be a Turing machine with configuration size $S(\lambda) \le c\lambda$ for a constant c. Let $T = T(\lambda)$ and $B = B(\lambda)$ be polynomials. Let $\mathsf{GIVC} = (\mathsf{GIVC.G}, \mathsf{GIVC.P}, \mathsf{GIVC.V})$ be a generalized incremental verifiable computation scheme with incremental completeness for M having time parameter T. Assume without loss of generality that verification in GIVC for $t = 0$ is perfectly sound. Let $L(\lambda)$ be a bound on the final verifier efficiency of GIVC that is an efficiently computable function. Let $\mathsf{QA} = (\mathsf{QA.G}, \mathsf{QA.P}, \mathsf{QA.V})$ be publicly verifiable quasi-arguments for NP.

In the following we construct $\mathsf{GIVC}' = (\mathsf{GIVC}'.\mathsf{G}, \mathsf{GIVC}'.\mathsf{P}, \mathsf{GIVC}'.\mathsf{V})$, a generalized incremental verifiable computation scheme with incremental completeness for M having time parameter $T' = T \cdot B$ using GIVC and QA. We start by describing the 3CNF used by the quasi-argument.

Verification formula. Fix λ a security parameter and vk a verification key. Let φ be the 3CNF formula corresponding to the computation of $\mathsf{GIVC.V}(\mathsf{vk}, T, C, C', \pi)$ under Cook-Levin reduction. Namely, the input to φ is a string of length $\ell = 2S + L$ parsed as (C, C', π) for configurations C, C' and proof π padded to ℓ if needed. The witness to φ is a string of length polynomial in ℓ. For every input (C, C', π), there exists a witness w such that $\varphi(C, C', \pi, w) = 1$ if and only if $\mathsf{GIVC.V}(\mathsf{vk}, T, C, C', \pi) = \mathsf{ACC}$. Moreover, such witness w can be computed in time polynomial in the running time of GIVC.V. Let K be the size of φ. There exists such a formula φ with $K = \mathrm{poly}(L, S, \lambda)$ variables.

Let ϕ be the following formula over $M = O(K \cdot B)$ variables:

$$\phi(C_0, C_B, w = (\{C_i\}_{i \in [B-1]}, \{\pi_i\}_{i \in [B]}, \{w_i\}_{i \in [B]})) := \bigwedge_{i=1}^{B} \varphi(C_{i-1}, C_i, \pi_i, w_i) \ .$$

Where the witness w to ϕ is parsed to $w = (\{C_i\}_{i \in [B-1]}, \{\pi_i\}_{i \in [B]}, \{w_i\}_{i \in [B]})$. Let $n = 2S$ be the number of input variables for ϕ.

We are now ready to give the algorithm description of GIVC'. In the following ϕ, n, K are as defined in the verification formula paragraph.

$\mathsf{GIVC}'.\mathsf{G}(x)$

Input: $x \in \{0, 1\}^\lambda$.
Algorithm:

1. Sample $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{GIVC.G}(x)$.
2. Construct the formula ϕ using vk.
3. Sample $(\mathsf{qpk}, \mathsf{qvk}) \leftarrow \mathsf{QA.G}(\lambda, \phi, n, K)$.
4. Output $\mathsf{pk}' = (x, \mathsf{pk}, \mathsf{vk}, \mathsf{qpk})$ and $\mathsf{vk}' = (x, \mathsf{vk}, \mathsf{qvk})$.

$\mathsf{GIVC}'.\mathsf{P}(\mathsf{pk}', t, C, C', \pi)$

Input:

1. Prover key $\mathsf{pk}' = (x, \mathsf{pk}, \mathsf{vk}, \mathsf{qpk})$.
2. Time $t \in [0, T'-1]$ written as $t = qT + r$ for $r < T$.
3. Configurations $C, C' \in \{0,1\}^S$.
4. Candidate proof $\pi \in \{0,1\}^*$.

Algorithm:

1. Parse $\pi = (C_0, \pi_1, \dots, \pi_B, C_B)$.
2. Compute for every $i \in [B]$:

$$C'_i := \begin{cases} M_x(C') & i \geq q+1 \\ C_i & \text{otherwise} \end{cases} \quad \text{and} \quad \pi'_i := \begin{cases} \varepsilon & i > q+1 \\ \mathsf{GIVC}.\mathsf{P}(\mathsf{pk}, r, C_{i-1}, C_i, \pi_i) & i = q+1 \\ \pi_i & \text{otherwise} \end{cases} .$$

3. If $t < T'-1$, output $\pi' = (C'_0, \pi'_1, \dots, \pi'_B, C'_B)$.
4. Else $t = T'-1$, compute an assignment σ for ϕ using $(\pi'_i)_{i=1}^B, (C'_i)_{i=0}^B$ and vk. Output $\pi' = \mathsf{QA}.\mathsf{P}(\mathsf{qpk}, \sigma)$.

$\mathsf{GIVC}'.\mathsf{V}(\mathsf{vk}', t, C, C', \pi)$

Input:

1. Verifier key $\mathsf{vk}' = (x, \mathsf{vk}, \mathsf{qvk})$.
2. Time $t \in [0, T']$ written as $t = qT + r$ for $r < T$.
3. Configurations $C, C' \in \{0,1\}^S$.
4. Candidate proof $\pi \in \{0,1\}^*$.

Algorithm:

1. If $t = T'$, output the result of $\mathsf{QA}.\mathsf{V}(\mathsf{qvk}, (C, C'), \pi)$.
2. Otherwise $t < T'$, parse $\pi = (C_0, \pi_1, \dots, \pi_B, C_B)$.
3. Verify the following:

$$C = C_0 \xrightarrow[T,\mathsf{vk}]{\pi_1} \dots \xrightarrow[T,\mathsf{vk}]{\pi_q} C_q \xrightarrow[r,\mathsf{vk}]{\pi_{q+1}} C_{q+1} \xrightarrow[0,\mathsf{vk}]{\pi_{q+2}} \dots \xrightarrow[0,\mathsf{vk}]{\pi_B} C_B = C' .$$

If pass output ACC, otherwise output REJ.

Parsing The Empty Proof. In the above, when parsing the empty proof $\pi = \varepsilon$ as the vector $(C_0, \pi_1, \dots, \pi_B, C_B)$ we parse it as $(C, \varepsilon, C, \varepsilon, \dots, \varepsilon, C)$ where C is the first configuration given as input. Using this parsing, the empty proof pass as valid for $t = 0$ and $C = C'$, as required by incremental completeness.

Bound Final Verifier Size. Proofs for time $t = T'$ are verified using $\mathsf{QA.V}$ which has efficiency $n \cdot \mathrm{poly}(\lambda, K)$. We modify the above verifier in the case $t = T'$ to reject if the candidate proof π is of size larger than the efficiency of $\mathsf{QA.V}$. Doing so allow us to bound the final verifier efficiency of GIVC' in Claim A.1.

A.4 Analysis

In this section we analyze the construction given in appendix A.3

A.4.1 Recursive Step Efficiency

We assume w.l.o.g that $T' \leq 2^S$ where recall S is the configuration size of M. Let $L = L(\lambda)$ be a bound on the final verifier efficiency of GIVC. Let $\mathbb{T}_{\mathsf{GIVC}}, \mathbb{T}_{\mathsf{GIVC}'}$ be the efficiency of GIVC and GIVC' respectively. By the construction and by the efficiency guarantee of QA we have

$$\mathbb{T}_{\mathsf{GIVC}'} \leq \mathbb{T}_{\mathsf{GIVC}} + \mathrm{poly}(B, L, S, \lambda, |M|) \ ,$$

where the polynomial is independent of M, T and B. Next, in the following claim, we derive a recursive bound on final verifier efficiency.

Claim A.1 (Recursive Efficiency). *Let $L(\lambda)$ be a bound on the final verifier efficiency of* GIVC. *Then*

$$L' \leq \mathrm{poly}(L, S, \lambda) \ ,$$

is a bound on the final verifier efficiency of GIVC'. *The polynomial is independent of M, T and B.*

Proof sketch. We have that locality parameter $K = \mathrm{poly}(L, S, \lambda)$. It follows by the efficiency guarantee of the quasi-arguments that $L' = \mathrm{poly}(n, K, \lambda)$ bounds the final verifier efficiency of GIVC'. Hence, $L' = \mathrm{poly}(n, K, \lambda) = \mathrm{poly}(L, S, \lambda)$. □

A.4.2 Incremental Completeness

Fix a tuple $(x, t, C, C') \in \mathcal{U}_M$ such that $t < T' - 1$, a proof $\pi \in \{0,1\}^*$ and keys $(\mathsf{pk}', \mathsf{vk}') \in \mathsf{Supp}(\mathsf{GIVC}'.\mathsf{G}(x))$. Parse $\pi = (C_0, \pi_1, \ldots, \pi_B, C_B)$, $\mathsf{pk}' = (x, \mathsf{pk}, \mathsf{vk}, \mathsf{qpk})$ and $\mathsf{vk}' = (x, \mathsf{vk}, \mathsf{qvk})$. Assume $\mathsf{GIVC}'.\mathsf{V}(\mathsf{vk}', t, C, C', \pi) = \mathsf{ACC}$. We prove $\mathsf{GIVC}'.\mathsf{V}(\mathsf{vk}', t+1, C, C', \pi') = \mathsf{ACC}$ where $\pi' = \mathsf{GIVC}'.\mathsf{P}(\mathsf{pk}', t, C, C', \pi)$.

Start with the case $\pi \neq \varepsilon$. Since π pass verification, for $t = qT + r$ where $r < T$ we have:

$$C = C_0 \xrightarrow[T,\mathsf{vk}]{\pi_1} \ldots \xrightarrow[T,\mathsf{vk}]{\pi_q} C_q \xrightarrow[r,\mathsf{vk}]{\pi_{q+1}} C_{q+1} \xrightarrow[0,\mathsf{vk}]{\pi_{q+2}} \ldots \xrightarrow[0,\mathsf{vk}]{\pi_B} C_B = C' \ . \tag{1}$$

By our assumption that verification for time 0 in GIVC is of perfect soundness we have:

$$C_{q+1} = \cdots = C_B = C' \ .$$

By incremental completeness of GIVC:

$$C_q \xrightarrow[r+1,\mathsf{vk}]{\pi'_{q+1}} M_x(C_{q+1}) \quad \text{and} \quad M_x(C_{q+1}) \xrightarrow[0,\mathsf{vk}]{\varepsilon} M_x(C_{q+1}) \ ,$$

where $\pi'_{q+1} = \mathsf{GIVC.P}(\mathsf{pk}, r, C'_q, C'_{q+1}, \pi_{q+1})$. Therefore:

$$C = C_0 \xrightarrow[T,\mathsf{vk}]{\pi_1} \dots \xrightarrow[T,\mathsf{vk}]{\pi_q} C_q \xrightarrow[r+1,\mathsf{vk}]{\pi'_{q+1}} M_x(C_{q+1}) \xrightarrow[0,\mathsf{vk}]{\varepsilon} \dots \xrightarrow[0,\mathsf{vk}]{\varepsilon} M_x(C_{q+1}) = M_x(C') \ .$$

By construction, this implies that π' pass verification. For the case $\pi = \varepsilon$, note that by the way the empty proof is parsed, (1) holds and hence the same argument proves π' is accepted.

For $t = T' - 1$, by the above argument, the assignment σ computed by $\mathsf{GIVC'.P}$ satisfies ϕ. By perfect completeness of QA the proof π' is accepting. This concludes incremental completeness of $\mathsf{GIVC'}$. $\square$.

A.4.3 Security Analysis

In the security reductions given in this section, it will be more convenient to consider randomized (non-uniform) adversaries. Although we model adversaries as deterministic circuits, this is without loss of generality. Indeed, by hard-wiring the random coins that maximize the success probability of the randomized adversary, we can construct a deterministic (non-uniform) adversary with no worse success probability.

Let $\mathcal{A}'$ be an efficient adversary that attempts to break the soundness of $\mathsf{GIVC'}$. The following claim is due to the soundness of GIVC.

Claim A.2. *There exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^\lambda$ we have:*

$$\Pr\left[\begin{array}{l} t < T' \\ (x,t,C,C') \notin \mathcal{U}_M \\ \mathsf{GIVC.V}(\mathsf{vk}', t, C, C', \pi^*) = \mathsf{ACC} \end{array} \middle| \begin{array}{l} (\mathsf{pk}', \mathsf{vk}') \leftarrow \mathsf{GIVC'.G}(x) \\ (t, C, C', \pi^*) \leftarrow \mathcal{A}'(x, \mathsf{pk}', \mathsf{vk}') \end{array}\right] \le \mu(\lambda) \ .$$

Proof of claim. Let us denote the above probability by $\varepsilon(x)$. Consider an adversary $\mathcal{A}$ that attempts to break GIVC as follows. On input $(x, \mathsf{pk}, \mathsf{vk})$ proceed to generate $\mathsf{pk}', \mathsf{vk}'$ as $\mathsf{GIVC'.G}$ would after sampling from $\mathsf{GIVC.G}$. Simulate $(t, C, C', \pi^*) \leftarrow \mathcal{A}(x, \mathsf{pk}, \mathsf{vk})$. Assume π^* pass verification, $t < T'$ and $(x,t,C,C') \notin \mathcal{U}_M$. Parse $\pi^* = (C_0, \pi_1^*, \dots, \pi_B^*, C_B)$. Write $t = qT + r$ for $r < T$. We then have that:

$$C = C_0 \xrightarrow[t_1 = T,\mathsf{vk}]{\pi_1^*} \dots \xrightarrow[t_q = T,\mathsf{vk}]{\pi_q^*} C_q \xrightarrow[t_{q+1} = r,\mathsf{vk}]{\pi_{q+1}^*} C_{q+1} \xrightarrow[t_{q+2} = 0,\mathsf{vk}]{\pi_{q+2}^*} \dots \xrightarrow[t_B = 0,\mathsf{vk}]{\pi_B^*} C_B = C' \ .$$

Since $(x,t,C,C') \notin \mathcal{U}_M$, there must exists $i \in [B]$ such that $(x, t_i, C_i, C_{i+1}) \notin \mathcal{U}_M$. This index can be found in polynomial time since $T' = \mathrm{poly}(\lambda)$. The adversary $\mathcal{A}$ finds the first such index i and outputs $(t_i, C_i, C_{i+1}, \pi_i^*)$.

Note, the distribution of the generated $\mathsf{pk}', \mathsf{vk}'$ by $\mathcal{A}$ is the same those sampled from $\mathsf{GIVC'.G}(x)$. Therefore, the success probability of $\mathcal{A}$ is at least $\varepsilon(x)$. By the soundness of GIVC, since $\mathcal{A}$ is efficient, there exists a negligible function μ, such that the success probability of $\mathcal{A}$ is at most $\mu(\lambda)$. Hence $\varepsilon(x) \le \mu(\lambda)$. This concludes the proof of the claim. $\square$

The next claim deals with the case $t = T'$. The analysis is essentially the same as in the bootstrapping theorem by Kalai et al. replacing delegation schemes with GIVC schemes (can be viewed as a more expressive delegation scheme). See Section 2.2.2 in [KPY19] for high-level overview of the analysis. For the sake of completeness, we give the slightly modified version of the analysis by Kalai et al. in what follows.

Claim A.3. *There exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^\lambda$ we have:*

$$\Pr\left[\begin{array}{l} t = T' \\ (x,t,C,C') \notin \mathcal{U}_M \\ \mathsf{GIVC.V}(\mathsf{vk}', t, C, C', \pi^*) = \mathsf{ACC} \end{array} \middle| \begin{array}{l} (\mathsf{pk}', \mathsf{vk}') \leftarrow \mathsf{GIVC}'.\mathsf{G}(x) \\ (t, C, C', \pi^*) \leftarrow \mathcal{A}'(x, \mathsf{pk}', \mathsf{vk}') \end{array}\right] \leq \mu(\lambda) \ .$$

Proof of claim. Assume toward contradiction that there exists a polynomial p such that for infinitely many $\lambda \in \mathbb{N}$ there exists $x \in \{0,1\}^\lambda$ such that

$$\Pr\left[\begin{array}{l} t = T' \\ (x,t,C,C') \notin \mathcal{U}_M \\ \mathsf{GIVC.V}(\mathsf{vk}', t, C, C', \pi^*) = \mathsf{ACC} \end{array} \middle| \begin{array}{l} (\mathsf{pk}', \mathsf{vk}') \leftarrow \mathsf{GIVC}'.\mathsf{G}(x) \\ (t, C, C', \pi^*) \leftarrow \mathcal{A}'(x, \mathsf{pk}', \mathsf{vk}') \end{array}\right] \geq \frac{1}{p(\lambda)} \ . \tag{2}$$

Fix such λ and x. A random tape r of $\mathsf{GIVC.G}(x)$ is said to be bad if, when fixing r, the probability in (2) is at least $1/2p(\lambda)$. By averaging, a $1/2p(\lambda)$ fraction of r's are bad. Fix such bad r and let $(\mathsf{pk}, \mathsf{vk})$ be the keys defined by r. Consider the following adversary for QA:
$\underline{\mathcal{A}_{\mathsf{QA}}(\mathsf{qpk}, \mathsf{qvk}):}$

1. Use $(\mathsf{qpk}, \mathsf{qvk})$ and $(\mathsf{pk}, \mathsf{vk})$ to obtain $(\mathsf{pk}', \mathsf{vk}')$.
2. Simulate $(t, C, C', \pi) \leftarrow \mathcal{A}'(x, \mathsf{pk}', \mathsf{vk}')$.
3. Let $y = (C, C')$ and output (y, π).

Obtaining $(\mathsf{pk}', \mathsf{vk}')$ is done the same way as by $\mathsf{GIVC}'.\mathsf{G}$. Let $\mathcal{E}$ be the no-signaling extractor for the quasi-arguments. By the correct distribution property, we have for every efficient distinguisher D and λ:

$$\left|\begin{array}{c} \Pr\left[D(y) = 1 \middle| \begin{array}{l} (\mathsf{qpk}, \mathsf{qvk}) \leftarrow \mathsf{QA.G}(\lambda, \phi, n, K) \\ (y, \pi) \leftarrow \mathcal{A}_{\mathsf{QA}}(\mathsf{qpk}, \mathsf{qvk}) \\ \text{if } \mathsf{QA.V}(\mathsf{qvk}, y, \pi) = \mathsf{REJ} : \\ \text{set } y = \bot \end{array}\right] \\ - \Pr\left[D(y) = 1 \mid (y, \sigma) \leftarrow \mathcal{E}^{\mathcal{A}_{\mathsf{QA}}}(\phi, \emptyset)\right] \end{array}\right| \leq \operatorname{negl}(\lambda) \ . \tag{3}$$

If $y \neq \bot$ it is parsed as (C, C'). Given C, we denote for every $j \in [0, B]$ the configuration $\overline{C}_j = M_x^{(T \cdot j)}(C)$ that follows $T \cdot j$ steps after C.

Let CHEAT be the event that $y \neq \bot$ and parsed to (C, C') such that $C' \neq \overline{C}_B$. By (2) and (3) we have:

$$\Pr_{(y,\sigma) \leftarrow \mathcal{E}^{\mathcal{A}_{\mathsf{QA}}}(\phi, \emptyset)}[\mathsf{CHEAT}] \geq \frac{1}{\operatorname{poly}(\lambda)} \ . \tag{4}$$

For $j \in [B]$ let $I_j \subseteq [M]$ be the set of variables in ϕ describing C_{j-1}, C_j, π_j, w_j. Note that by definition $|I_j| = K$. For $j \in [B]$ let Exp_j be the experiment where sampling $(y, \sigma) \leftarrow \mathcal{E}^{\mathcal{A}_{\mathsf{QA}}}(\phi, I_j)$.

By the no-signaling property and (4), it follows that for every $j \in [B]$:

$$\Pr_{\mathsf{Exp}_j}[\mathsf{CHEAT}] \geq \frac{1}{\operatorname{poly}(\lambda)} \ . \tag{5}$$

Since CHEAT implies that $y \neq \perp$, by applying the local consistency property, for every $j \in [B]$ we have:

$$\Pr_{\mathsf{Exp}_j}\left[\mathsf{CHEAT} \implies \begin{array}{l} \forall i \in I_j \cap [n] : y_i = \sigma(i) \\ [\![\phi]\!]_\sigma = 1 \end{array}\right] \geq 1 - \mathrm{negl}(\lambda) \ .$$

In the following, let C_{j-1}, C_j, π_j, w_j be the values assigned by σ to the variables in I_j. If σ locally satisfies ϕ then $\varphi(C_{j-1}, C_j, \pi_j, w_j) = 1$ and hence $\mathsf{GIVC.V}(\mathsf{vk}, T, C_{j-1}, C_j, \pi) = \mathsf{ACC}$:

$$\Pr_{\mathsf{Exp_i}}\left[\begin{array}{l} \mathsf{CHEAT} \\ \mathsf{GIVC.V}(\mathsf{vk}, T, C_{j-1}, C_j, \pi_j) = \mathsf{REJ} \end{array}\right] \leq \mathrm{negl}(\lambda) \ . \tag{6}$$

Also, in Exp_1, if σ is consistent with the input $y = (C, C')$ then $C_0 = C = \overline{C}_0$:

$$\Pr_{\mathsf{Exp}_1}\left[\begin{array}{l} \mathsf{CHEAT} \\ C_0 \neq \overline{C}_0 \end{array}\right] \leq \mathrm{negl}(\lambda) \ . \tag{7}$$

Similarly, in Exp_B, if σ is consistent with the input $y = (C, C')$ then $C_B = C'$. Recall that in the event of CHEAT we have $C' \neq \overline{C}_B$. Hence:

$$\Pr_{\mathsf{Exp}_B}\left[\begin{array}{l} \mathsf{CHEAT} \\ C_B = \overline{C}_B \end{array}\right] \leq \mathrm{negl}(\lambda) \ . \tag{8}$$

By no-signaling, for every $i \in [B-1]$ we have:

$$\left|\Pr_{\mathsf{Exp}_i}\left[\begin{array}{l} \mathsf{CHEAT} \\ C_i = \overline{C}_i \end{array}\right] - \Pr_{\mathsf{Exp}_{i+1}}\left[\begin{array}{l} \mathsf{CHEAT} \\ C_i = \overline{C}_i \end{array}\right]\right| \leq \mathrm{negl}(\lambda) \ . \tag{9}$$

By combining (5) to (9) and the fact B is a polynomial, it follows there exists $j \in [B]$ such that:

$$\Pr_{\mathsf{Exp}_j}\left[\begin{array}{l} \mathsf{CHEAT} \\ C_{j-1} = \overline{C}_{j-1} \\ C_j \neq \overline{C}_j \\ \mathsf{GIVC.V}(\mathsf{vk}, T, C_{j-1}, C_j, \pi_j) = \mathsf{ACC} \end{array}\right] \geq \frac{1}{\mathrm{poly}(\lambda)} \ . \tag{10}$$

Given this, we can construct an adversary $\mathcal{A}$ for GIVC as follows.
$\underline{\mathcal{A}(x, \mathsf{pk}, \mathsf{vk})}$

1. Emulate $(y, \sigma) \leftarrow \mathcal{E}^{\mathcal{A}_{\mathsf{QA}}}(\phi, I_j)$.

2. Let C_{j-1}, C_j, π_j be the values assigned by σ to the corresponding variables in ϕ.

3. Output (T, C_{j-1}, C_j, π_j).

Recall x is the fixed string such that (2) holds. Since (10) holds for every set of keys sampled by $\mathsf{GIVC.G}$ with a bad random tape r and since a $1/2p(\lambda)$ fraction of r's are bad:

$$\Pr\left[\begin{array}{l} (x, T, C, C') \notin \mathcal{U}_M \\ \mathsf{GIVC.V}(\mathsf{vk}, t, C, C', \pi^*) = \mathsf{ACC} \end{array} \middle| \begin{array}{l} (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{GIVC.G}(x) \\ (T, C, C', \pi^*) \leftarrow \mathcal{A}(x, \mathsf{pk}, \mathsf{vk}) \end{array}\right] \geq \frac{1}{\mathrm{poly}(\lambda)} \ .$$

A contradiction to the soundness of GIVC. □

Together, Claim A.2 and Claim A.3 proves the soundness of GIVC'.

A.5 Efficient IVC

In this section, we prove Theorem A.1. Fix constants $c, \varepsilon > 0$. Let α be a large enough constant to be chosen later. Let M be any Turing machine with run-time $T \leq c\lambda^\alpha$, configuration size $S \leq c\lambda$ and description size $|M| \leq c \log \alpha$. In the following, we construct a GIVC scheme with incremental completeness for M with efficiency $\mathbb{T}_{\mathsf{GIVC}}(\lambda) \leq \lambda^{\varepsilon\alpha}$. This implies the existence of an IVC scheme with incremental completeness for M with efficiency $O(\lambda^{\varepsilon\alpha})$. Indeed, we may hardwire the first configuration C to be M_x^0 and take the public parameters pp to be the tuple $(\mathsf{vk}, \mathsf{pk})$. It is not hard to check that by doing so, with appropriate syntax adjustments, we result in an IVC scheme with incremental completeness for M with said efficiency. By taking α to be larger and using the assumption $\lambda > 1$, we get the desired result (without the big O notation).

We proceed to construct the GIVC scheme.

Claim A.4 (Base Case). *There exists a GIVC scheme with incremental completeness for M with time parameter 1 of efficiency* $\mathrm{poly}(S, \lambda, |M|)$.

Proof. The content of the proofs of the scheme is ignored, verification is by computing the next configuration and comparing. Implementing M_x, the successor circuit of M with input x, can be done in time $\mathrm{poly}(S, \lambda, |M|)$. □

Let d be a constant to be chosen later. Let GIVC be the scheme resulted in applying the recursive construction, for d times, with parameter $B = T^{1/d}$, starting with the base case scheme. By the results in appendix A.4.1 we have

$$\mathbb{T}_{\mathsf{GIVC}}(\lambda) \leq d \cdot T^{O(1/d)} \cdot (S|M|\lambda)^{2^{O(d)}} = d \cdot T^{O(1/d)} \cdot (c \cdot \log_2 \alpha \cdot \lambda)^{2^{O(d)}} \quad ,$$

The last equality is due to $S \leq c\lambda$ and $|M| \leq c \log_2 \alpha$. Let η be a constant guaranteed by the O notation. Note that η is independent of M, T and B. We have

$$\mathbb{T}_{\mathsf{GIVC}}(\lambda) \leq d \cdot T^{\eta/d} \cdot (c \cdot \log_2 \alpha \cdot \lambda)^{2^{\eta \cdot d}} \leq d \cdot (c\lambda)^{\eta \cdot \alpha/d} \cdot (c \cdot \log_2 \alpha \cdot \lambda)^{2^{\eta \cdot d}} \quad ,$$

where the last inequality is for $\alpha > 1$. Since $\log_\lambda$ is an increasing function, it suffices to show that by choice of d we have $\log_\lambda \mathbb{T}_{\mathsf{GIVC}} \leq \varepsilon\alpha$. For $d = \log_2 \alpha / 2\eta$ we have:

$$\log_\lambda \mathbb{T}_{\mathsf{GIVC}}/\alpha \leq \frac{\log_\lambda(\log_2(\alpha)/2\eta)}{\alpha} + \frac{2\eta^2}{\log \alpha} + \frac{\log_\lambda(c \cdot \log_2 \alpha \cdot \lambda)}{\sqrt{\alpha}} \xrightarrow[\alpha \to \infty]{} 0 \ .$$

Proving that $\log_\lambda \mathbb{T}_{\mathsf{GIVC}} \leq \varepsilon\alpha$ for large enough α. Note that we used the fact c and η are independent of α and ε. This concludes the proof of Theorem A.1. □

A.6 The KPY Assumption

The KPY assumption is over a group G of prime order p equipped with bilinear map.

Assumption A.1 (KPY Assumption). *For every $\alpha(\lambda) = O(\log \lambda)$, given the following matrix of group elements:*

$$\left(g^{s^j t^i}\right)_{\substack{i \in [0,2] \\ j \in [0,\alpha]}} = \begin{pmatrix} g^{s^0} & g^{s^1} & \cdots & g^{s^\alpha} \\ g^{s^0 t} & g^{s^1 t} & \cdots & g^{s^\alpha t} \\ g^{s^0 t^2} & g^{s^1 t^2} & \cdots & g^{s^\alpha t^2} \end{pmatrix} \quad ,$$

for random $g \in G$ and $s \in \mathbb{Z}_p$, efficient adversaries fail to distinguish between the case $t = s^{2\alpha+2}$ and the case t is random independent element in $\mathbb{Z}_p$, except with negligible advantage.

B Hardness from a Computational Reduction

In this section we prove Proposition 4.1 and Proposition 4.2. Both propositions are meant to show that a computational Karp reduction from an underlying hard search problem (in worst-case/average-case) to LS, implies the hardness of LS (in worst-case/average-case, respectively).

B.1 Worst-Case Hardness

In this subsection we prove Proposition 4.1 establishing non-uniform worst-case hardness given a computational Karp reduction. We use n to denote the size of LS instances and λ the size of R instances.

Proposition 4.1. *Let R be an FNP search problem having a computational Karp reduction to LS. Assume there exists a adversary $\mathcal{A} = \{\mathcal{A}_n\}_{n\in\mathbb{N}}$ of polynomial-size $s(n)$ solving LS in the worst-case. Then there exists an adversary $\mathcal{A}' = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ solving R in the worst-case. The size of $\mathcal{A}'$ is*

$$\mathsf{size}(\mathcal{A}') = \mathrm{poly}(s(\mathbb{T}_{\mathsf{Red}}), \mathbb{T}_R, \lambda) \enspace ,$$

where $\mathbb{T}_{\mathsf{Red}}(\lambda)$ is the efficiency of the reduction and $\mathbb{T}_R(\lambda)$ is the efficiency of the NP verification $R(x, y)$.

Proof. The proof relies on the non-uniform derandomization technique employed by Adleman's theorem proof ($\mathsf{BPP} \subseteq \mathsf{P/Poly}$). Let $\mathcal{A} = \{\mathcal{A}_n\}_{n\in\mathbb{N}}$ be an efficient adversary of size $s(n)$ solving LS. Consider the following randomized circuit family attempting to solve R.

$\underline{\mathcal{C}(x;r)}$:

1. Sample $(\mathcal{S}, \mathcal{F}) \leftarrow \mathcal{X}(x)$.
2. Simulate $w' \leftarrow \mathcal{A}(\mathcal{S}, \mathcal{F})$.
3. Output $w = W(w')$.

Since $\mathcal{A}$ is an efficient adversary, we can apply the computational soundness of the reduction. Therefore, there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^\lambda$ for which R_x is non-empty:

$$\Pr_r [w \notin R_x \mid w = \mathcal{C}(x;r)] \leq \mu(\lambda)$$

Consider the randomized circuit family $\mathcal{C}'$, that amplify the soundness error of $\mathcal{C}$ using repetition with independent random coins. In the following we partition $r = r_1 || r_2 || \dots || r_\lambda$.

$\underline{\mathcal{C}'(x;r)}$:

1. For $i = 1, \dots, \lambda$:
   (a) Sample $w \leftarrow C(x; r_i)$
   (b) If $R(x, w) = 1$, output w.
2. Output $\perp$

Since we use independent random coins in the above, we have for every $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^\lambda$ for which R_x is non-empty:

$$\Pr_r\left[w \notin R_x \mid w = \mathcal{C}'(x,r)\right] \le \mu(\lambda)^\lambda \underbrace{<}_{\text{for large } \lambda} 2^{-\lambda}$$

By union bound, for large enough $\lambda \in \mathbb{N}$, there exists random coin tosses r_λ such that $\mathcal{C}'(x;r_\lambda)$ is correct for every input $x \in \{0,1\}^\lambda$ such that R_x is non empty. Indeed for large enough $\lambda \in \mathbb{N}$:

$$\Pr_r\left[\exists x \in \{0,1\}^\lambda, R_x \neq \emptyset \text{ s.t. } \mathcal{C}'(x;r) \notin R_x\right] \le \sum_{x\in\{0,1\}^\lambda, R_x \neq \emptyset} \Pr_r\left[\mathcal{C}'(x;r) \notin R_x\right] < 2^\lambda \cdot 2^{-\lambda} = 1$$

Therefore, we can consider $\mathcal{A}'$ that simulates $\mathcal{C}'$ with fixed coins being r_λ. We have that for large enough λ, the adversary $\mathcal{A}'$ returns a correct witness w for every x that has witness. Since we can verify if w is a witness for x, we can also handle inputs with no witness with the same asymptotic efficiency. For small value λ we simply hard-wire $\mathcal{A}'$ to answer correctly. We have that $\mathcal{A}'$ solves R for every input.

Efficiency Analysis. We have that $s(n) = \Omega(n)$ by trivial LS instances. The size of $\mathcal{C}$ is:

$$\mathsf{size}(C) = O(\mathbb{T}_{\mathsf{Red}}(\lambda) + s(\mathbb{T}_{\mathsf{Red}}(\lambda))) = O(s(\mathbb{T}_{\mathsf{Red}}(\lambda))) \ .$$

The circuit $\mathcal{C}'$ simulate $\mathcal{C}$ for λ times and every time applies the verification algorithm. Therefore of size:

$$\mathsf{size}(C') = O\left(\lambda s(\mathbb{T}_{\mathsf{Red}}(\lambda)) + \lambda p(\mathbb{T}_R(\lambda))\right) \ ,$$

for a fixed polynomial p (the polynomial p is due to Turing machine simulation). Hence, the size of $\mathcal{A}'$ is $\mathrm{poly}(s(\mathbb{T}_{\mathsf{Red}}), \mathbb{T}_R, \lambda)$ for a fixed polynomial.

□

B.2 Average-Case Hardness

In this subsection we prove Proposition 4.2.

Proposition 4.2. *If there exists hard-on-average* FNP *problem R, with a computationally sound Karp reduction from R to* LS*, then* LS *is hard-on-average.*

Proof of Proposition 4.2. Let D be an efficient sampler such that (R, D) is hard-on-average. In the following, we construct an efficient sampler HARD of LS such that $(\mathsf{LS}, \mathsf{HARD})$ is hard-on-average. Let $(\mathcal{X}, \mathcal{W})$ be the computational sound Karp from R to LS guaranteed by the statement. The sampler $\mathsf{HARD}(1^\lambda)$ start by sampling $x \leftarrow D(1^\lambda)$ and output $(\mathcal{S}, \mathcal{F}) \leftarrow \mathcal{X}(x)$. It is efficient by the efficiency of $\mathcal{X}$ and D.

Let $\mathcal{A}$ be an efficient adversary attempting to find a local maximum for $(\mathcal{S}, \mathcal{F}) \leftarrow \mathsf{HARD}(1^\lambda)$ with success probability $\varepsilon(\lambda)$. That is

$$\varepsilon(\lambda) = \Pr\left[\mathcal{F}(\mathcal{S}(w')) \le \mathcal{F}(w') \;\middle|\; \begin{array}{l}(\mathcal{S}, \mathcal{F}) \leftarrow \mathsf{HARD}(1^\lambda) \\ w' \leftarrow \mathcal{A}(\mathcal{S}, \mathcal{F})\end{array}\right] \ .$$

Let us denote

$$p_1(\lambda) = \Pr\left[\begin{array}{l} \mathcal{F}(\mathcal{S}(w')) \le \mathcal{F}(w') \\ w \in R_x \end{array} \middle| \begin{array}{l} x \leftarrow D(1^\lambda) \\ (\mathcal{S}, \mathcal{F}) \leftarrow \mathcal{X}(x) \\ w' \leftarrow \mathcal{A}(\mathcal{S}, \mathcal{F}) \\ w = \mathcal{W}(w) \end{array} \right]$$

$$p_2(\lambda) = \Pr\left[\begin{array}{l} \mathcal{F}(\mathcal{S}(w')) \le \mathcal{F}(w') \\ w \notin R_x \end{array} \middle| \begin{array}{l} x \leftarrow D(1^\lambda) \\ (\mathcal{S}, \mathcal{F}) \leftarrow \mathcal{X}(x) \\ w' \leftarrow \mathcal{A}(\mathcal{S}, \mathcal{F}) \\ w = \mathcal{W}(w) \end{array} \right]$$

By law of total probability and the definition of HARD we have that:

$$\varepsilon(\lambda) = p_1(\lambda) + p_2(\lambda) \ .$$

We have that p_1 is negligible since (R, D) is hard-on-average and p_2 is negligible by the computational soundness of the reduction.

Claim B.1. $p_1(\lambda) \le \text{negl}(\lambda)$

Proof of claim. Let $\mathcal{A}'$ be an efficient adversary attempting to solve R for instances $x \leftarrow D(1^\lambda)$ defined as follows.

$\underline{\mathcal{A}'(x)}$:

1. Sample $(\mathcal{S}, \mathcal{F}) \leftarrow \mathcal{X}(x)$.
2. Simulate $w' \leftarrow \mathcal{A}(\mathcal{S}, \mathcal{F})$.
3. Output $w = \mathcal{W}(w')$.

Take note that the above is randomized while we model adversaries as deterministic circuits. Instead of sampling we hardwire into $\mathcal{A}'$ the random coins that maximize the success probability. We now have by that (R, D) is hard-on-average:

$$p_1(\lambda) = \Pr\left[\begin{array}{l} \mathcal{F}(\mathcal{S}(w')) \le \mathcal{F}(w') \\ w \in R_x \end{array} \middle| \begin{array}{l} x \leftarrow D(1^\lambda) \\ (\mathcal{S}, \mathcal{F}) \leftarrow \mathcal{X}(x) \\ w' \leftarrow \mathcal{A}(\mathcal{S}, \mathcal{F}) \\ w = \mathcal{W}(w) \end{array} \right]$$
$$\le \Pr\left[w \in R_x \middle| \begin{array}{l} x \leftarrow D(1^\lambda) \\ w \leftarrow \mathcal{A}'(x) \end{array} \right] \le \text{negl}(\lambda) \ .$$

This concludes the proof of the claim. □

Claim B.2. $p_2(\lambda) \le \text{negl}(\lambda)$

Proof of claim. We have, by the computational soundness of the reduction, proven in Theorem 4.1, that there exists a negligible function μ such that for every $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^\lambda$ with non-empty R_x:

$$q_x = \Pr\left[\begin{array}{l} \mathcal{F}(\mathcal{S}(w')) \le \mathcal{F}(w') \\ w \notin R_x \end{array} \middle| \begin{array}{l} (\mathcal{S}, \mathcal{F}) \leftarrow \mathcal{X}(x) \\ w' \leftarrow \mathcal{A}(\mathcal{S}, \mathcal{F}) \\ w = \mathcal{W}(w) \end{array} \right] \le \mu(\lambda) \ .$$

By averaging we get the result.

$$p_2(\lambda) = \Pr\left[\begin{array}{l|l} & x \leftarrow D(1^\lambda) \\ \mathcal{F}(\mathcal{S}(w')) \leq \mathcal{F}(w') & (\mathcal{S}, \mathcal{F}) \leftarrow \mathcal{X}(x) \\ w \notin R_x & w' \leftarrow \mathcal{A}(\mathcal{S}, \mathcal{F}) \\ & w = \mathcal{W}(w) \end{array} \right] = \underset{x \leftarrow D(1^\lambda)}{\mathbb{E}}[q_x] \leq \mu(\lambda) \ .$$

This concludes the proof of the claim. □

We conclude that $\varepsilon(\lambda) = p_1(\lambda) + p_2(\lambda) \leq \mathrm{negl}(\lambda)$. This concludes the proof of the proposition. □