On the (Im)Plausibility of Constant-Round Public-Coin Straight-Line-Simulatable Zero-Knowledge Proofs

Yi Deng*‡, Juan Garay†, San Ling*, Huaxiong Wang* and Moti Yung♮

* School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
† AT&T Labs – Research, USA
‡ SKLOIS, Institute of Information Engineering, CAS, China
♮ Google Inc., USA

Abstract. In 2001, a breakthrough result by Barak [FOCS 2001] showed how to achieve public-coin zero-knowledge (ZK) arguments in constant rounds, a feature known to be impossible using black-box simulation. In this approach, the simulator makes use of the code of the malicious verifier in computing the prover messages (albeit without understanding it), and does not rewind the malicious verifier—and it is hence called a *straight-line* simulator.

Since then, however, we have witnessed little progress on the basic question whether Barak's technique can be extended to ZK *proof* systems. In this paper we make progress on this front, by providing strong evidence that such an extension is far from likely. Specifically, we show that for a natural class of constant-round public-coin ZK proofs (which we call "canonical," as all known non-black-box ZK protocols fall in this category), a straight-line simulator based on the known non-black-box technique for such a proof system can actually be used to solve a seemingly unrelated problem, namely, to figure out some non-trivial property of a verifier's program, and *without executing the target code*, a problem commonly viewed as notoriously hard.

A key tool in our reduction is an improved structure-preserving version of the well-known Babai-Moran Speedup (derandomization) Theorem, which essentially says that, for a constant-round public-coin interactive proof system in which the verifier sends m messages and each of the prover messages is of length p, if the cheating probability for an unbounded prover is ϵ, then there exist $(p/O(\log \frac{1}{\epsilon}))^m$ verifier random tapes such that the cheating probability for the unbounded prover over these random tapes is bounded away from 1—and this holds even when the prover knows this small set of random tapes in advance. (In our setting, the original Babai-Moran theorem yields a much larger size ($(O(p))^m$) of such set of verifier random tapes.) In addition, we show that this is tight with respect to round complexity, in the sense that there are public-coin proof systems with a super-constant number of rounds for which the prover's cheating probability is 1, over any polynomial number of verifier random tapes.

Finally, although the notion of straight-line simulation is intuitively clear and has been used several times in the literature, we are not aware of a formal definition of the process, perhaps due to the fact that thoroughly defining (as well as enforcing) "executing the verifier only once" does not appear to be straightforward. The notion of *generalized straight-line simulation* that we introduce not only overcomes those obstacles, but enables us to expose the limitations of the currently known non-black-box techniques.

1 Introduction

In their seminal paper [20], Goldwasser, Micali and Rackoff introduced the fascinating notion of a *zero-knowledge* (ZK) interactive proof, in which a party (called the prover) wishes to convince another party (called the verifier) of some statement, in such a way that the following two properties are satisfied: (1) zero knowledge— the prover does not leak any knowledge beyond the truth of the statement being proven, and (2) soundness—no cheating prover can convince the verifier of a false statement except with small probability. Shortly after the introduction of a ZK proof, Brassard, Chaum and Crépeau [3] defined a ZK proof system with relaxed soundness requirement, called a ZK *argument*, for which soundness is only required to hold against polynomial-time cheating provers. A vast amount of work ensued these pioneering results.

"Leaking no additional knowledge" is usually formalized in the following way: Given an arbitrary malicious verifier V^*, one can construct an efficient algorithm—the *simulator*, taking V^* as a subroutine, that is able to reconstruct the real interaction between an honest prover and V^* without knowledge of a witness to the common input whose validity the honest prover is trying to convince the verifier of. For most of the known ZK protocols, the associated simulator treats the subroutine V^* as a black box and uses a so-called rewinding technique in the simulation in order to achieve its goal; this kind of simulation (called *black-box simulation*) was in fact the only known technique to demonstrate "zero-knowledgeness" for a long while, reflecting the fact that if we want to take advantage of V^*'s code, we need to understand it first, which is commonly viewed as a notoriously hard problem.

A breakthrough result in 2001 changed the state of things. Indeed, in [2] Barak presented a non-black-box ZK argument in which the simulator makes use of the code of the malicious verifier in computing the prover messages, albeit without understanding it. Barak's construction is a constant-round public-coin argument, and its simulator does not rewind the malicious verifier (and it is hence called a *straight-line* simulator), and, furthermore, runs in strict polynomial time. These features have been proved impossible to achieve when using black-box simulation [18,6], thus emphasizing the significance of the result.

A basic question, however, with little progress since the introduction of non-black-box simulation, is the following:

> *Can Barak's technique be extended to ZK* proof *systems? I.e., can a non-black-box ZK proof system be constructed which overcomes some known impossibility result for black-box ZK proof systems?*

A naïve attempt. In order to illustrate the problem, let us take a look at Barak's protocol [2], which given a common input $x \in L$, proceeds as follows:

$V \to P$: Send a random hash function h.
$P \to V$: Send a $c = \mathrm{Com}(0^n)$.
$V \to P$: Send random string r.
$P \Rightarrow V$: Using a WI universal argument, prove that $x \in L$ OR that there exists a program Π such that $c = \mathrm{Com}(h(\Pi))$ and, given input c, Π outputs r in some super-polynomial time.

In order to perform the simulation, the simulator is given the code of the malicious verifier V^* as input, and at the first prover step, it commits to the hash value of the description of V^*, and then executes the WI universal argument using the witness to the second clause of the OR statement. Observe that, as defined, the simulator satisfies two remarkable properties: first, it runs in a straight-line manner, and, second, in the first prover step, when it commits to the hash value of the description of V^*, it can do so obliviously, without needing to figure out any non-trivial property of V^*'s program.

Could the above protocol be turned into a proof system? It seems unlikely, mainly due to the fact that since a Turing machine or algorithm may have an arbitrarily long representation, a computationally unbounded prover may, after receiving the second verifier message r, be able to find a program Π

(whose description may be different from the verifier's with which the prover is interacting) such that, $c = \mathrm{Com}(h(\Pi))$, and on input c, Π outputs r in the right amount of time.

A closer look at the approach above, however, shows that for the simulator to be successful, providing it with the entire description of V^* is not necessary, and just the description of V^*'s second message function is enough. Note also that for an honest verifier, this particular next-message function (e.g., the simple program "Output r") can be described in length of $|r| + l$, for some constant l. Thus, seemingly, one might be able to construct a straight-line-simulatable ZK proof system by putting more restrictions on the statement of the WI universal argument: after receiving r, we would now have the prover prove that either $x \in L$ OR that there exists a program Π of length $|r| + l$ such that $\Pi(c)$ outputs r in polynomial time.

The above change possibly prevents an unbounded prover from cheating (in this case at least), but in exchange for the simulator now having to solve a daunting program-understanding problem. Consider, for example, an arbitrary verifier V^* which has the same functionality as some honest verifier, but its description is much longer. Now, at the first prover step, the simulator needs to somehow figure out V^*'s functionality, rewrite it in a much shorter form, and commit to it. Achieving this seems very doubtful for the currently known non-black-box technique, which runs in a straight-line manner and *does not* execute V^* before this commitment step.

Our results and techniques. In this paper, we provide strong evidence that to construct constant-round public-coin straight-line simulatable ZK proof systems, we cannot go much farther than the naïve approach above. More specifically, we show that for a natural class of constant-round public-coin ZK proofs, (which we call "canonical," as all known non-black-box ZK protocols fall in this category), a straight-line simulator of such a proof system can actually be used to figure out some non-trivial property of the codes of verifier's next-message functions, which, as mentioned above, is very unlikely given the state of the art of non-black-box techniques.

A key tool in our reduction is an improved structure-preserving version of the well-known Babai-Moran Speedup (derandomization) Theorem [1,9,10], which essentially says that, for a constant-round public-coin interactive proof system in which the verifier sends m messages and each of the prover messages is of length p, if the cheating probability for an unbounded prover is ϵ, then there exist $(p/O(\log \frac{1}{\epsilon}))^m$ verifier random tapes such that the cheating probability for the unbounded prover over these tapes is bounded away from 1—and this holds even when the prover knows this small set of random tapes in advance. (In our setting, the original Babai-Moran theorem yields a much larger size ($(O(p))^m$) of such set of verifier random tapes.) In addition, we show that this is tight with respect to round complexity, in the sense that there are public-coin proof systems with a super-constant number of rounds for which the prover's cheating probability is 1, over any polynomial number of verifier random tapes.

As its name hints, straight-line simulation means that, during the simulation process, the simulator runs each step of a malicious verifier V^* only once. Although this notion has been used several times in the literature, we are not aware of a formal definition of the process, perhaps because this seems very obvious. Nonetheless, obstacles arise when trying to define (as well as enforce) "executing the verifier only once." The simulator, being in possession of V^*'s code, may for example further obfuscate V^* and then execute the obfuscated verifier many times, or may execute some sections of V^*'s code. Thus, formally capturing the essence of straight-line simulation requires placing various restrictions on the simulator's possible behaviors, making a simple formulation of the process uncertain at best.

We address this problem by so-to-speak taking a step back and formulating a more general process, in the sense that it will also capture non-straight-line simulation when the simulator runs the malicious verifier internally, but whose *straight-line appearance* enables us to expose the limitations of the currently known non-black-box techniques. Specifically, we introduce *generalized straight-line simulation*, where the simulator is of the oracle form $S^{O(\cdot)}(\cdot)$, and where both S and O are given the common input x and the code of the malicious verifier V^*. The simulation is then a *real world* interaction between S, which plays

the role of the prover, and O, which plays the role of the verifier, except that O also provides some actual computation of V^* upon being asked by S.

Equipped with these tools, and elaborating on what was anticipated at the beginning of the section, we then show a reduction from a natural class of constant-round public-coin zero-knowledge proof systems with the above generalized straight-line simulator $S^{O(\cdot)}(\cdot)$ to a certain *verifier-understanding* problem. Specifically, we show that there exists a set of polynomial number of honest (partial) verifiers $V^1, V^2, ..., V^q$ for which we can construct an "understanding" algorithm U, with oracle access only to S (not to O), such that for any polynomial-time constructible code V^* that is promised to be functionally equivalent to one of these honest verifiers, $U^{S(V^*)}(V^1, V^2, ..., V^q)$ can pin-point a verifier V^j from the set that is functionally *different* from V^*.

This reduction can be viewed as a negative answer to the question whether the currently known straight-line simulation techniques can be extended to ZK proof systems, for the following reasons:

- The known straight-line simulators can be re-written in our generalized straight-line simulation form wherein S does not run V^* at all. (Note that the actual computations of V^* needed for the simulation are provided by the oracle.)
- This further implies that the algorithm U^S does not run V^* when carrying out the above understanding task, making its success very unlikely even when U^S is allowed to run in exponential time[1].

Related work. Predicated on the conjectured existence of a certain family of hash functions, Barak, Lindell and Vadhan [8] proved that constant-round public-coin ZK proof systems do not exist for any non-trivial language, regardless of whether the associated simulator proceeds in a straight-line manner or needs rewinding. In contrast, our negative result only holds for a narrower class of constant-round public-coin ZK proofs, but relying on a very-unlikely-to-be-solved program-understanding problem.

A closely related problem to our understanding problem is program obfuscation, the theoretical study of which was initiated by Barak *et al.* [5]. At a high level, an obfuscator is an efficient compiler that takes a program as input and output an "unreadable" program with the same functionality as the input program. Recently, obfuscation has attracted a lot of research efforts (e.g., [11,22,24,27]) due to its wide range of applications, from software protection to providing a justification to the random oracle model. Hada [23], in particular, showed that the existence of a certain type of ZK protocol is tightly related to the existence of obfuscator for some specific functionality. Unfortunately, for a large class of functionalities, it has been shown that obfuscators do not exist.

If the currently known straight-line simulation techniques can actually be extended to a proof system, then we will have an algorithm that can solve our verifier-understanding problem *without* any execution of the target code. Again, this problem appears to be (much) tougher than breaking obfuscators.

Organization of the paper. Preliminaries, notation and definitions that are used throughout the paper are presented in Section 2. Our notion of generalized straight-line simulation as well as the "verifier-understanding" problem are formulated in Section 3. The main technical contributions—the improved derandomization lemma and the reduction of constant-round public-coin straight-line-simulatable ZK proofs to the verifier-understanding problem—are presented in Section 4. For ease of readability, only partial proofs and/or outlines are presented in the main body; the full proofs can be found in the appendix.

2 Preliminaries

In this section we recall some definitions and introduce notation that will be used throughout the paper. We say that function $\mathsf{neg}(n)$ is *negligible* if for every polynomial $q(n)$ there exists an N such that for all $n \geq N$, $\mathsf{neg}(n) \leq 1/q(n)$.

[1] We note that the actual running time of U^S (including the running time of S) is exponential in the length of the prover messages, rather than in the length of V^*'s code.

An interactive proof system $\langle P, V\rangle$ for a language L is a pair of interactive Turing machines in which the prover P wishes to convince the verifier V of some statement $x \in L$. In an interaction between P and V, the *view* of V, denoted by View_V^P, consists of the common input x,V's random tape, and all the prover messages it received. The *round complexity* of an interactive proof system $\langle P, V\rangle$ is the number of messages exchanged in an execution of $\langle P, V\rangle$. Without loss of generality, in this paper we assume that the verifier V sends the first message; thus, if the verifier sends m messages in total, the round complexity of this proof system is $2m$.

Definition 1 (Interactive Proofs). *A pair of interactive Turing machines $\langle P, V\rangle$ is called an* interactive proof system for language L *if V is a probabilistic polynomial-time (PPT) machine and the following conditions hold:*

- COMPLETENESS: *For every $x \in L$, $Pr[\langle P, V\rangle(x) = 1] = 1$.*
- SOUNDNESS: *For every $x \notin L$, and every (unbounded) prover P^*,*

$$Pr[\langle P^*, V\rangle(x) = 1] < \mathsf{neg}(|x|).$$

Public-coin proof systems and verifier decomposition. An interactive proof system is called *public-coin* if at every verifier step, the verifier sends only truly random messages.

We use boldface lowercase letters to refer to the verifier's random tapes (e.g., $\mathbf{r}$), and italic for each verifier message (e.g., r). Thus, for a $2m$-round public-coin interactive proof system $\langle P, V\rangle$, we have $\mathbf{r} = [r_1, r_2, ..., r_m]$, where r_i is the i-th verifier message. We use superscripts to distinguish different verifier's random tapes; e.g., $\mathbf{r}^i$, $\mathbf{r}^j$, etc.

Given a random tape $\mathbf{r} = [r_1, r_2, ..., r_m]$, we can "decompose" the verifier $V(\mathbf{r})$ into a collection of next-message functions, $V = [V_1, V_2, ..., V_m]$, with each V_i being defined as:

$$r_i \text{ or } \perp \leftarrow V_i(\mathsf{hist}, r_1, r_2, ..., r_i),$$

where hist refers to the current history; that is, given an accepting hist, $V_i(\mathsf{hist}, r_1, r_2, ..., r_i)$ outputs r_i, or otherwise it aborts. Notice that the next message function V_i needs the randomness $[r_1, r_2, ..., r_{i-1}]$ of previous verifier steps in order to check whether the current history is accepting or not.

We will sometimes abbreviate and use superscripts to distinguish verifiers running on different random tapes; that is, given two random tapes $\mathbf{r}^i = [r_1^i, r_2^i, ..., r_m^i]$ and $\mathbf{r}^j = [r_1^j, r_2^j, ..., r_m^j]$, we will use V^i and V^j as a shorthand for $V(\mathbf{r}^i)$ and$V(\mathbf{r}^j)$, respectively. Similarly, we will use V_k^i to denote the k-th next message function $V_k(r_k^i)$ of the verifier $V(\mathbf{r}^i)$.

Now, given a verifier $V^i = [V_1^i, ..., V_m^i]$, we will use $V_{[j,k]}^i$ to denote the partial verifier strategy starting with the j-th next message function and up to the k-th next message function. We will typically be concerned with the following partial strategies:

$$\text{prefix strategy: } V_{[1,k]}^i \triangleq [V_1^i, V_2^i, ..., V_k^i];$$
$$\text{suffix strategy: } V_{[k,m]}^i \triangleq [V_k^i, V_{k+1}^i, ..., V_m^i].$$

We conclude this section by recalling the following standard definition:

Definition 2 (Zero-Knowledge Proofs). *An interactive proof system $\langle P, V\rangle$ for a language L is said to be* zero knowledge *if there exists a probabilistic polynomial-time algorithm S such that for any probabilistic polynomial-time V^* and any $x \in L$, the distribution $\{\text{View}_{V^*}^P\}_{x\in L}$ is computationally indistinguishable from the distribution $\{S(x, V^*)\}_{x\in L}$.*

3 Generalized Straight-Line Simulation and the *Verifier-Understanding* Problem

In this section we put forth our notion of (generalized) straight-line simulation and formalize the "verifier-understanding" problem. First some additional notions.

When referring to a Turing machine V, we will slightly abuse notation and use V to represent both its code and its functionality. Specifically, if we write $V \in G$ for some set G, we will mean that there is a Turing machine in G whose code is identical to the code of V; on the other hand, if we say that V^* is "functionally equivalent" to V, both V^* and V will clearly refer to their functionality.

Consider now a zero-knowledge protocol $\langle P, V \rangle$ and a given malicious verifier V^*. It is convenient for us to think of a (non-black-box) simulator as a function at each prover step of the common input x, the description of V^*, the current session history and the actual execution (described in a computation tableau) of the malicious verifier. We call the prover step i a *PCP step* if the simulator S, in order to produce the i-th prover message, uses the actual execution (as described in the computation tableau) of (some of the steps of) V^*. We call it a "PCP step" because usually at this step the simulator needs to prepare a PCP proof for correctness of the executions of these verifier steps. In Barak's protocol [2], for example, the PCP step is the first prover step in the subprotocol of the WI universal argument.

Generalized straight-line simulation. The notion of straight-line simulation was put forth in and achieved by Barak's construction [2]. As its name hints, in such a simulation the simulator basically runs each step of the malicious verifier V^* *only once*. Now, as already mentioned in Section 1, although this notion has been used several times in the literature, we are not aware of a formal definition of the process, perhaps due to the fact that defining (as well as enforcing) "executing the verifier only once" seems challenging. The simulator may for example further obfuscate V^* and then execute the obfuscated verifier many times, or may execute some sections of the code of V^*.

We get around this problem by formulating a more general process, in the sense that it will also capture non-straight-line simulation when the simulator runs the malicious verifier internally, but its *straight-line appearance* will enable us to expose the limitations of the currently known non-black-box techniques. In a nutshell, in this *generalized* straight-line simulation (specified more formally below), we think of the simulation process as a *real world* interaction between an algorithm S and an oracle algorithm O, where S plays the role of the prover and O plays the role of the verifier, except that O also records the actual executions of V^* itself and provides some of them to S when being asked by S. Both S and O take the description of V^* as input; given the common input x, the simulator is the algorithm $S(x, V^*)$ with oracle access to $O(x, V^*)$, and its output is the interaction between them. (For simplicity, we will drop the common input x and write such a simulator as $S^{O(V^*)}(V^*)$ when clear from the context.)

Formally, for a PPT algorithm S and a deterministic polynomial-time algorithm O, a *generalized straight-line simulation* by $S^{O(V^*)}(V^*)$ is defined as follows:

- For each verifier step, $S(V^*)$ makes only one query to $O(V^*)$.
- $O(V^*)$ maintains a table $\mathcal{T}$. At each verifier step i, $1 \leq i \leq m$, $O(V^*)$ runs V^* to obtain its next verifier message r_i, and stores the actual computation of V^* at this step, denoted by $\mathcal{T}_i$, in the table $\mathcal{T}$. $O(V^*)$ answers the current query from S according to its type:
  1. Type 1: (hist, I_i), where I_i is a subset of $[1, 2, ..., i]$. For this type of query, $O(V^*)$ checks if the current history hist is accepting; if not, it returns $\perp$. Otherwise, it returns $r_i, \mathcal{T}_{I_i}$, where $\mathcal{T}_{I_i}$ is the set of tableaux $\{\mathcal{T}_j : j \in I_i\}$;
  2. Type 2: $(\mathsf{hist}, \emptyset)$. For this type of query, $O(V^*)$ returns only r_i.
- At each prover step i, upon receiving the reply from $O(V^*)$, S produces the current prover message p_i and obtains a new history hist which includes p_i. Furthermore, if the next prover step is a PCP step for which S requires the some tableaux $\mathcal{T}_{I_{i+1}}$, it sends a type-1 query (hist, I_{i+1}); otherwise, it sends a type-2 query $(\mathsf{hist}, \emptyset)$.

– The output of $S^{O(V^*)}(V^*)$ consists of the common input and the session transcript between $S(V^*)$ and $O(V^*)$ (except for those actual executions of V^* returned by O). Note that $S^{O(V^*)}(V^*)$ is a random variable defined over the randomness used by S.

Remark 1. To reiterate, besides allowing us to overcome the difficulties mentioned earlier, the benefits of defining straight-line simulation as above are as follows:

1. Its generality, since we put no restriction on $S(V^*)$, thus also capturing non-straight-line simulation when S runs V^* internally. Note also that there may be proof systems with no PCP steps, in which S will not query O for any actual computation of V^*.
2. Its capability to expose the limitations of currently known straight-line simulation approaches due to its "straight-line" *appearance*. Indeed, when we re-write the known straight-line simulators (say, Barak's) in the oracle form above, it becomes clear that the component $S(V^*)$ does not have to run V^* at all since all actual executions of V^* needed for the simulation can be obtained from $O(V^*)$. As we will see, this feature will allow us to cast our main theorem (Theorem 1) as a negative result, as being able to extend the currently known straight-line simulation techniques to zero-knowledge proof systems seems very unlikely.

Equipped with the definition above, the corresponding notion of ZK proof naturally follows.

Definition 3 (Generalized Straight-Line-Simulatable (GSLS) ZK Proofs). *An interactive zero-knowledge proof system $\langle P, V\rangle$ for a language L is said to be* generalized straight-line-simulatable *if it admits a simulator $S^{O(\cdot)}(\cdot)$ satisfying the above conditions.*

In this paper we will focus on the following sub-class of GSLS ZK proof systems, which we call "canonical," since all known constructions (see below) fall in this category. Essentially, in this type of proofs the simulated session history before a PCP step k is "disconnected" from the code of the malicious verifier after step k. This property is indeed very natural: in computing the session history before prover step k, it seems to be enough for the simulator to use only the code of the first k steps of the malicious verifier and the actual computations of these verifier steps. Putting this formally:

Definition 4 (*Canonical* GSLS ZK Proofs). *Let $\langle P, V\rangle$ be a GSLS ZK proof system for a language L as above, and $S^{O(\cdot)}(\cdot)$ the associated simulator. We call $\langle P, V\rangle$* canonical *if for any common input x (not necessarily in L) the following two conditions hold:*

1. *Let prover step ℓ be the* last *PCP step. For any PPT verifier $V^* = [V^*_{[1,\ell]}, V^*_{[\ell+1,m]}]$, the output of $S^{O(V^*)}(V^*_{[1,\ell]})$ is indistinguishable from that of $S^{O(V^*)}(V^*)$.*
2. *For any PCP step k, $k < \ell$, and any PPT verifier $V^* = [V^*_{[1,k]}, V^*_{[k+1,\ell]}, V^*_{[\ell+1,m]}]$, the output of $S^{O(V^*_{[1,k]})}(V^*_{[1,k]})$ is the prefix session history of the output of $S^{O(V^*)}(V^*_{[1,\ell]})$, when S uses the same randomness in both executions.*

We stress that in the stand-alone setting[2], all known non-black-box ZK protocols, including Barak's protocol [2], the "two-slot" variant of Barak's protocol by Pass and Rosen [25], and the protocol by Deng, Goyal and Sahai [15], conform to the above definition. Although in the simulation for such protocols the simulator is given the entire code of V^*, a closer look reveals that it is not necessary to do so. Indeed, for all known non-black-box constructions, the simulator only needs the code of a *subset* of V^*'s steps (in addition to the session history so far) in order to produce the prover's messages.

The following definition relating the behavior of two verifiers will become handy in the sequel.

[2] In this paper we only consider stand-alone execution of ZK proof systems. It should be noted that in [25,15] some new techniques are introduced for the purpose of achieving stronger security properties, such as non-malleability and concurrency.

Definition 5 (Equivalence of Verifier Functionalities). *Given a proof system $\langle P, V\rangle$ for a language L, for two deterministic interactive algorithms V^1 and V^2 playing the role of the verifier, we say V^1 and V^2* have the same functionality*, or* are functionally equivalent*, if for any session prefix* hist*, the next verifier message produced by V^1 is identical to the one produced by V^2, i.e., $V^1(\mathsf{hist}) = V^2(\mathsf{hist})$ holds for any* hist*.*

We will use $V^1 \stackrel{\text{f}}{=} V^2$ as a shorthand for the above, and $V^1 \stackrel{\text{f}}{\neq} V^2$ as its negation.

We are now ready to formulate the "verifier-understanding problem," to which the existence of constant-round public-coin GSLS ZK proofs is reduced. In a nutshell, the problem resides, given a set of (partial) honest verifiers, in constructing an "understanding" algorithm U, with oracle access only to S (not to O), such that for any polynomial-time constructible program V^* that is promised to be functionally equivalent to one of the verifiers, is able to discern a verifier from the set that is functionally *different* from V^*. Formally:

Definition 6 (The *Verifier-Understanding* Problem). *Let n be the security parameter and t, p be some polynomials in n. Let $\langle P, V\rangle$ be a canonical GSLS zero-knowledge proof system as in Definition 4, the length of each prover message be p, and $S^{O(\cdot)}(\cdot)$ be the associated simulator.*

Given a common input x and t polynomial-time verifiers $(V^1, V^2, ..., V^t)$, with $V_i \stackrel{\text{f}}{\neq} V_j$ for all $1 \leq i \neq j \leq t$, the problem is to find a non-uniform algorithm U, running in time $2^{O(p)}$, such that for every polynomial-time algorithm C, the following holds:

- *First, C picks a machine V^i at random and outputs a polynomial-time Turing machine V^*, $V^* \leftarrow C(V^1, V^2, ..., V^t, i)$ such that $V^* \stackrel{\text{f}}{=} V^i$.*
- *U, taking $(V^1, V^2, ..., V^t)$ as input, makes at most a polynomial number of queries to the oracle $S(V^*)$ (not to $O(V^*)$) and outputs a machine V^j such that $V^j \stackrel{\text{f}}{\neq} V^*$ with probability negligibly close to 1. I.e.,*

$$\Pr\left[V^* \leftarrow C(V^1, V^2, ..., V^t, i);\ j \leftarrow U^{S(V^*)}(V^1, V^2, ..., V^t) : V^* \stackrel{\text{f}}{\neq} V^j\right] > 1 - \mathsf{neg}(n),$$

where the probability is taken over the random choice i and the randomness used by U and S.

It should be stressed that, in the above definition, algorithm U is not given V^*'s code as input. This captures U's difficulty in understanding the program, especially when its oracle S does not execute V^*.

4 (Im)Plausibility of Constant-Round Straight-Line-Simulatable Zero Knowledge Proofs

We are now ready to present our main result, which exhibits a reduction from canonical GSLS ZK proofs to the verifier-understanding problem, as defined above, a problem seemingly quite different in nature. We first fix some parameters and notation:

- $\langle P, V\rangle$: A $2m$-round canonical GSLS ZK proof sytem, where each prover message is of length p, for m a constant and p some polynomial;
- n is the security parameter, k, l are constants, $0 \leq k < l \leq m$, and q is a polynomial;
- G: a set of deterministic honest verifiers in $(V^1, V^2, ..., V^q)$ that share the same prefix verifier $V^j_{[1,k]}$, for some j;
- G': the set of partial verifiers on which the understanding algorithm U is asked to operate, defined as

$$G' = \{V_{[k+1,l]} : \exists V_{[l+1,m]} \text{ s.t. } [V^j_{[1,k]}, V_{[k+1,l]}, V_{[l+1,m]}] \in G\};$$

- $V'_{[1,k]}$: the auxiliary input to U and S (when $k = 0$, it is set to the empty string), which is the code of a prefix verifier such that $V'_{[1,k]} \stackrel{\text{f}}{=} V^j_{[1,k]}$.

We now show that, if $\langle P, V\rangle$ admits a generalized straight-line simulator $S^{O(\cdot)}(\cdot)$, then there is an algorithm U^S, taking $V'_{[1,k]}$ and G as auxiliary input, which can solve the verifier-understanding problem with respect to the set of partial verifiers G' with size $|G'| = t$ (cf. Definition 6; the verifier set in the definition is instantiated here with set G'). Formally, our main theorem can now be stated as follows.

Theorem 1. *Let $\langle P, V\rangle$ be a canonical GSLS ZK proof system with negligible soundness error for a non-trivial language $L \notin \mathcal{BPP}$ in the plain model, and $S^{O(\cdot)}(\cdot)$ be its generalized straigh-line simulator. Then, there exist $x \notin L$, constants k and l, sets G, G', a verifier program $V'_{[1,k]}$ as above, and an algorithm U, making at most a polynomial number of queries to $S(V'_{[1,k]}, V^*_{[k+1,l]})$ and running in time $2^{O(p)}$ such that, for any polynomial-time algorithm C that, on input (G', i), outputs $V^*_{[k+1,l]}$ satisfying $V^*_{[k+1,l]} \stackrel{\mathrm{f}}{=} V^i_{[k+1,l]} \in G'$:*

$$\Pr\left[V^*_{[k+1,l]} \leftarrow C(G', i); j \leftarrow U^{S(V'_{[1,k]}, V^*_{[k+1,l]})}(V'_{[1,k]}, G, G') : V^*_{[k+1,l]} \stackrel{\mathrm{f}}{\neq} V^j_{[k+1,l]} \in G'\right] > 1 - \mathsf{neg}(n),$$

where the probability is taken over the random choice i and the randomness used by U and S.

The proof of the theorem heavily relies on the new derandomization lemma presented in the next section, which essentially says that for a constant-round public-coin interactive proof systems $\langle P, V\rangle$ for some non-trivial language in which the verifier sends m messages and each of the prover's messages is of length p, if the cheating probability for an unbounded prover is ϵ, then there exists $q = (p/O(\log \frac{1}{\epsilon}))^m$ number of random tapes for the verifier such that the cheating probability for the unbounded prover over these verifier's random tapes is less than $1 - 1/q$.

Assuming the lemma, we now present an outline of the proof of Theorem 1.

Proof sketch. For simplicity, assume the case of a single PCP in the prover's strategy (the other cases—no PCP step, multiple PCP steps—are dealt with later on—Sections 4.2 and A.2, respectively). Let the PCP step be the l-th prover step. In this case,

- the step index k is set to 0, and the auxiliary input $V'_{[1,k]}$ is set to be the empty string[3];
- G is the set of deterministic *honest* verifiers $G = (V^1, V^2, ..., V^q)$ satisfying the condition of the derandomization lemma; and,
- $G' = \{V_{[1,l]} : \exists V_{[l+1,m]}$ s.t. $[V_{[1,l]}, V_{[l+1,m]}] \in G\}$ is the set of partial verifiers on which the understanding algorithm U is asked to work on.

Our basic reasoning behind the construction of the "understanding" algorithm U is as follows:

1. First, we prove that there is some false statement $x \notin L$ and a set of deterministic *honest* verifiers $G = (V^1, V^2, ..., V^q)$ satisfying the above derandomization lemma, such that for any $V^i \in G$ and any polynomial-time constructible prefix verifier $V^*_{[1,l]} \stackrel{\mathrm{f}}{=} V^i_{[1,l]}$, $S^{O(V^*_{[1,l]}, V^i_{[l+1,m]})}(V^*_{[1,l]}, V^i_{[l+1,m]})$ will output a transcript that is accepted by the verifier $[V^*_{[1,l]}, V^i_{[l+1,m]}]$ with probability negligibly close to 1.
2. Now, by the canonical property of the proof system, except with negligible probability, $S^{O(V^*_{[1,l]}, V^i_{[l+1,m]})}(V^*_{[1,l]})$ will also output an accepting transcript for any $V^*_{[1,l]}$ as above.
3. Assume $V^*_{[1,l]} \stackrel{\mathrm{f}}{=} V^i_{[1,l]}$, and let G_i be the set of verifiers in G that have the same prefix $V^i_{[1,l]}$. Then, we have that, for any $V^j \in G_i$, $S^{O(V^*_{[1,l]}, V^j_{[l+1,m]})}(V^*_{[1,l]})$ will also output an accepting transcript since it is (by item 1) a simulator for $[V^*_{[1,l]}, V^j_{[l+1,m]}]$, which is also polynomial-time constructible.

[3] For the multiple-PCP-step case, k and l are two consecutive PCP steps. For the single-PCP-step case, we set $k = 0$, which also means that the target code is a partial verifier from step $k + 1 = 1$ to step l.

4. The above implies that there exists a prover strategy P^*, that taking the description $V^*_{[1,l]}$ as auxiliary input, can make any verifier $V^j \in G_i$ accept with probability negligibly close to 1. P^* works as follows. It simply uses $S(V^*_{[1,l]})$ to produce the prover messages; upon receiving a message from V^j, it forwards it to $S(V^*_{[1,l]})$. When $S(V^*_{[1,l]})$ is instructed to query $O(V^*_{[1,l]}, V^j_{[l+1,m]})$ at prover step $l-1$ for the actual computations of $V^*_{[1,l]}$, then $P^*(V^*_{[1,l]})$ generates them for $S(V^*_{[1,l]})$[4].

   Observe that all the sessions between P^* and different verifiers in G_i share the same history prefix up to the l-th prover step (the PCP step), denoted by $(r_1, p_1, ..., r_l, p_l)$, which is actually the same prefix output by $S^{O(V^*_{[1,l]}, V^j_{[l+1,m]})}(V^*_{[1,l]})$, since the prefix strategy up to the l-th step of any verifier in G_i is functionally equivalent to $V^*_{[1,l]}$.

5. The above gives us an understanding algorithm $U^{S(V^*_{[1,l]})}$ (note that U is not being given the code of $V^*_{[1,l]}$): For each verifier V^j in G, U plays the role of the oracle O using strategy V^j and interacts with $S(V^*_{[1,l]})$ until obtaining a session history $(r_1, p_1, ..., p_{l-1})$[5]. It then generates the next verifier message r_l (by running V^j on the current history) and uses a careful exhaustive search to check whether there exists a (residual) prover strategy P^* such that, based on the history $(r_1, p_1, ..., r_l)$, P^* can make any verifier with the prefix strategy $V^j_{[1,l]}$ accept. If this is not the case, U outputs $V^j_{[1,l]}$.
   U works due to the following two reasons:

   (1) By items 3 and 4, if $V^*_{[1,l]} \stackrel{\text{f}}{=} V^i_{[1,l]}$ for some $V^i_{[1,l]} \in G'$, then there is prover strategy that can make any verifier in G sharing the same prefix verifier $V^i_{[1,l]}$ accept. Thus, $V^j_{[1,l]}$, U's output, must be functionally different from $V^*_{[1,l]}$ (and $V^i_{[1,l]}$). This guarantees the correctness of the algorithm.

   (2) By the derandomization lemma, there must exist at least one prefix $V^j_{[1,l]} \in G'$, $V^j_{[1,l]} \stackrel{\text{f}}{\neq} V^*_{[1,l]}$, and a set of verifiers that share the same prefix $V^j_{[1,l]}$, for which the above prover strategy that can make any verifier in this set accept does not work. Otherwise, we could construct a prover strategy with auxiliary input $V^*_{[1,l]}$ that could make all verifiers in G accept, contradicting the derandomization lemma.

This concludes the proof outline, which reduces the existence of a constant-round canonical GSLS ZK proof system to the existence of a successful program-understanding algorithm in the setting of Definition 6. The detailed proof for the case of (at most) one PCP step is presented in Section 4.2. □

4.1 An Improved Derandomization Lemma for Interactive Proofs

In this section we prove a structure-preserving version of the well-known Babai-Moran "Speedup Theorem" [1,9] with improved parameters for our application. Essentially, the result says that for any constant-round public-coin interactive proof system with small soundness error, there exists a polynomial set of random verifier tapes such that the cheating probability for the unbounded prover over these verifier tapes is bounded away from 1—and this holds even when the prover knows this small set of random tapes in advance.

[4] Note that S only queries for some actual computations of $V^*_{[1,l]}$ (since after prover step l there are no more PCP steps), and that, when $V^*_{[1,l]} \stackrel{\text{f}}{=} V^j_{[1,l]}$, the verifier messages up to the l-th verifier step in the interaction between P^* and V^j can be viewed as produced by $V^*_{[1,l]}$. These facts enable P^* to execute $V^*_{[1,l]}$ internally and provide actual computation of $V^*_{[1,l]}$ for the correctness of previous verifier messages (up to the l-th verifier step).

[5] Note that in the simulation S queries O for some actual computations of $V^*_{[1,l]}$ only at prover step $l-1$, and is supposed to receive them at prover step l. Thus, until prover step $l-1$, U is not yet required to provide S with some computation of $V^*_{[1,l]}$, which U would actually be unable to produce since it does not have the code of $V^*_{[1,l]}$.

We first recall the Babai-Moran theorem. Let $\mathrm{AM}[k]$ denote the set of languages whose membership can be proved via a k-round public-coin proof system.

Theorem 2 ([9]). *For any polynomial $t(n)$, $\mathrm{AM}[t+1] = \mathrm{AM}[t]$. In particular, for any constant k, $\mathrm{AM}[k] = \mathrm{AM}[2]$.*

For our application, we wish to de-randomize the verifier while keeping the original proof system structure intact (that is, without "collapsing" the round complexity). The $\mathrm{AM}[k] = \mathrm{AM}[2]$ proof—and its randomness-efficient variant in [10][6]—actually yield such a result: for any $2m$-round public-coin proof system with small soundness error ϵ, there exist $(O(p))^m$ verifier random tapes over which the cheating probability of an unbounded prover is still bounded away from 1, where p is the length of the prover's messages.

Next, we will present an improvement to this result, in which the number of such verifier random tapes reduces to $(p/O(\log \frac{1}{\epsilon}))^m$. In addition, we show that this de-randomization lemma is essentially tight with respect to the round complexity, as there are super-constant-round public-coin proof systems for which the prover's cheating probability is 1, over any polynomial number of verifier random tapes.

Before stating the lemma, we introduce some additional notation:

- $V_{|(\mathbf{r}^1,\mathbf{r}^2...,\mathbf{r}^t)}$ denotes the honest verifier that is restricted to choose *uniformly at random* one of $\mathbf{r}^1, \mathbf{r}^2..., \mathbf{r}^t$ as its random tape, where t is a polynomial; we use $V_{|(\mathbf{r}^1,\mathbf{r}^2...,\mathbf{r}^t)}(\mathbf{r}^i)$ to denote the verifier that takes $\mathbf{r}^i$, $1 \leq i \leq t$, as its random tape.
- $P^*(\mathbf{r}^1, \mathbf{r}^2..., \mathbf{r}^t)$ denotes the unbounded *cheating* prover with auxiliary input $(\mathbf{r}^1, \mathbf{r}^2..., \mathbf{r}^t)$, indicating that it will interact with $V_{|(\mathbf{r}^1,\mathbf{r}^2...,\mathbf{r}^t)}$.

We now state the result formally. For simplicity, we assume that all the prover messages are of equal length.

Lemma 1. *Let m be a constant and $\langle P, V\rangle$ be a $2m$-round public-coin interactive proof system for language L with negligible soundness error ϵ. Let p denote the length of the prover's messages. Then for every $x \notin L$, there exist $q = (p/O(\log \frac{1}{\epsilon}))^m$ different random tapes, $\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q$, such that for every unbounded prover P,*

$$\Pr[\langle P(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), V_{|(\mathbf{r}^1,\mathbf{r}^2,...\mathbf{r}^q)}\rangle(x) = 1] \leq 1 - \frac{1}{q}\ .$$

In the main body we present the intuition and basic inequalities that yield the proof for the case of a 3-round proof system[7] (same ideas also appeared in [1,9]), and defer the full proof of the lemma to Appendix A.1.

Let us consider a 3-round public-coin proof system $\langle P, V\rangle$ with negligible soundness error for some language L[8], in which the prover sends the first message p_1 and the last message p_2, and the verifier sends the second message $\mathbf{r}$ (its public coins). Without loss of generality, we assume $|p_1| = |p_2| = p$, and $|\mathbf{r}| = n$. We now prove that there exists a number p[9] of verifier random tapes $(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^p)$ over which the cheating probability is at most $1 - 1/p$.

[6] In [10], Bellare and Rompel present a randomness-efficient approach to transform $\mathrm{AM}[k]$ into $\mathrm{AM}[2]$: to halve the number of rounds of an Arthur-Merlin proof system, they introduce a so-called "oblivious sampler" and use a small amount of randomness to specify roughly $O(p)$ verifier messages in the original proof system. Their proof, however, yields almost the same result as the Speedup Theorem in our setting where we want to maintain the structure of the original proof system, and only care about the number of original verifier random tapes that are needed to make sure the resulting protocol after derandomization is still a proof system.

[7] The basic reasoning here applies to a proof system of even number (4) of rounds as well, by having the verifier send a dummy message first.

[8] For example, the n-folded parallel version of Blum's 3-round proof system for Graph Hamiltonicity [7] or the 3-round proof system for Graph Isomorphism [19].

[9] For the sake of simplicity, we do not optimize this parameter here.

For the sake of contradiction, assume that for some false statement $x \notin L$ there is an unbounded prover $P^\diamond$ such that for any p-tuple $(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^p)$, $P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^p)$ can cheat $V_{|(\mathbf{r}^1, \mathbf{r}^2, ... \mathbf{r}^p)}$ with probability 1. Now note that the number of such successful cheating provers is $\binom{2^n}{p}$, and that there are at most 2^p different first prover messages p_1. Thus, there is a number of at least $\binom{2^n}{p}/2^p$ $P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^p)$'s that produce the same first prover message, denote it p_1^*, for which if the verifier is using a random tape in any of the p-tuples

$$\{(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^p) : p_1^* \leftarrow P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^p)\},$$

we have an unbounded prover that can produce second prover message p_2^* to make the verifier accept.

On the other hand, the number of p-tuple choices $(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^p)$ out of $1/2e$ fraction of all possible verifier random tapes is at most $\binom{\frac{2^n}{2e}}{p}$. Since

$$\binom{\frac{2^n}{2e}}{p} < (\frac{2^n}{2p})^p < \frac{\binom{2^n}{p}}{2^p},$$

we have that the set $\{(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^p) : p_1^* \leftarrow P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^p)\}$ covers at least a $1/2e$ fraction of all possible verifier random tapes.

In sum, we are able conclude that there is an unbounded prover, which sends p_1^* as its first message, that can make the verifier accept the false statement with probability at least $1/2e$. This contradicts the negligible soundness error of $\langle P, V \rangle$.

The proof of the lemma for the general (arbitrary constant rounds) case can be found in Appendix A.1, and the tightness result, i.e., the counterexample for superconstant-round proof systems, appears in Appendix B.

4.2 Proof of Theorem 1

Given Lemma 1, we now present the detailed proof of Theorem 1 for the case where there is at most one PCP step in the proof system. The general case (multiple PCP steps) is presented in Appendix A.2.

Let $\langle P, V \rangle$ be a canonical $2m$-round public-coin GSLS ZK proof system with a single PCP step for some non-trivial (outside $\mathcal{BPP}$) language L, and $S^{O(\cdot)}(\cdot)$ be its associated generalized straight-line simulator. We first prove the following claim, where Lemma 1 is used.

Lemma 2. *Let $\langle P, V \rangle$ be as above. Then there exist a false statement $x \notin L$ and q honest verifiers $V^1, V^2, ..., V^q$ (recall that we use V^i as a shorthand for $V(\mathbf{r}^i)$, $1 \leq i \leq q$), such that given the description of any polynomial-time constructible $V^* \stackrel{\text{f}}{=} V^i$ for a random i as input, the simulator $S^{O(\cdot)}(\cdot)$ will output an accepting transcript with probability negligibly close to 1, while the unbounded prover can cheat only with probability at most $1 - 1/q$.*

Proof. We first prove that there is some false statement $x \notin L$ so that for every PPT algorithm C which takes a random $V(\mathbf{r})$ as input and outputs V^* such that $V^* \stackrel{\text{f}}{=} V(\mathbf{r})$, the simulator $S^{O(V^*)}(V^*)$ will output an accepting transcript with probability (over the randomness used by S and the choices of verifier random tapes) negligibly close to 1. Otherwise, if that were not the case, we could use the following simple algorithm to decide membership in L efficiently[10]: Pick a verifier random tape $\mathbf{r}$ at random and run C to construct $V^* \stackrel{\text{f}}{=} V(\mathbf{r})$, and then run $S^{O(\cdot)}(\cdot)$ on input x and V^*; if V^* accepts, output "$x \in L$," otherwise output "$x \notin L$."

[10] Although the error probability here may be high, it can be reduced by standard parallel repetition.

Now fix the above $x \notin L$, and set Q to be the set of verifier random tapes such that for any $\mathbf{r} \in Q$, and any polynomial-time constructible $V^* \stackrel{\text{f}}{=} V(\mathbf{r})$, $S^{O(V^*)}(V^*)$ will output an accepting transcript with probability negligibly close to 1 (over only the randomness of S).

We now show that the size of Q is larger than a $(1 - \mathsf{neg}(n))$ fraction of all possible random tapes. Assume the verifier's random tape $\mathbf{r}$ and S's random tape R are uniformly distributed over $\{0,1\}^s$ and $\{0,1\}^t$, respectively, and denote by $S^{O(V^*)}(V^*, R) = 1$ the event that the simulator outputs an accepting transcript. We have

$$
\begin{aligned}
& \Pr_{\substack{\mathbf{r}\leftarrow\{0,1\}^s \\ R\leftarrow\{0,1\}^t}}[V^* \leftarrow C(\mathbf{r}) : S^{O(V^*)}(V^*, R) = 1] \qquad (1) \\
= & \Pr_{\substack{\mathbf{r}\leftarrow\{0,1\}^s \\ R\leftarrow\{0,1\}^t}}[V^* \leftarrow C(\mathbf{r}) : S^{O(V^*)}(V^*, R) = 1 | \mathbf{r} \in Q] \Pr[\mathbf{r} \in Q] \\
& + \Pr_{\substack{\mathbf{r}\leftarrow\{0,1\}^s \\ R\leftarrow\{0,1\}^t}}[V^* \leftarrow C(\mathbf{r}) : S^{O(V^*)}(V^*, R) = 1 | \mathbf{r} \notin Q] \Pr[\mathbf{r} \notin Q] \\
\leq & \Pr[\mathbf{r} \in Q] + (1 - \frac{1}{\mathsf{poly}(n)}) \Pr[\mathbf{r} \notin Q] \\
= & \frac{|Q|}{2^s} + (1 - \frac{1}{\mathsf{poly}(n)})(1 - \frac{|Q|}{2^s}) \\
= & 1 - \frac{1}{\mathsf{poly}(n)}(1 - \frac{|Q|}{2^s}).
\end{aligned}
$$

Given that the probability in expression (1) is greater than $1 - \mathsf{neg}(n)$, so is the quantity $\frac{|Q|}{2^s}$.

Thus, given $x \notin L$, for any unbounded prover, the cheating probability, taken over the choices of verifier random tapes in Q, is still negligible. Applying Lemma 1, we can find q random tapes $\mathbf{r}_i \in Q$, $1 \leq i \leq q$, such that the probability, taken over these q random tapes, that the unbounded prover makes the verifier accept is at most $1 - 1/q$. This completes the proof of the lemma. □

Now to the proof of the theorem. We start with the case where there is no PCP step in the prover's strategy, which admits a simpler proof.

No PCP step. In this case, the proof is relatively straightforward. we can set parameters k and l in our main theorem to 0 and m respectively, auxiliary input $V'_{[1,k]}$ to be the empty string, and U will work on the set of *complete* verifier strategies $G = (V^1, ..., V^q)$, guaranteed by Lemma 2. (In this case, the sets G and G' in the proof outline of Theorem 1 above are identical.)

Now, given a polynomial-time constructible verifier code V^* that is promised to be functionally equivalent to one of the verifiers in G, we construct an understanding algorithm $U(G)$ with oracle access to $S(V^*)$ as follows. For each verifier V^j in G, U, playing the role of verifier V^j, obtains a complete interaction (without exhaustive search) with S to determine if V^j accepts. If not, it outputs V^j. Note that in this case, in simulation the oracle O does not provide any actual computation of V^* to S; thus, U can play the role of O by simply acting as an external verifier.

Lemma 2 ensures that this U^S algorithm works as desired.

Single PCP step. Now to the proof of the single-PCP case in detail. We construct the algorithm U^S with the following parameters:

- $k = 0$, and auxiliary input $V'_{[1,k]}$ set to be the empty string;
- l, the PCP step index;
- $x \notin L$ and $G = (V^1, V^2, ..., V^q)$, the set of deterministic honest verifiers satisfying Lemma 2;

- $G' = \{V_{[1,l]} : \exists V_{[l+1,m]}$ s.t. $[V_{[1,l]}, V_{[l+1,m]}] \in G\}$, the set of partial verifiers algorithm U^S is asked to work on;
- $V^*_{[1,l]}$, the output of an arbitrary PPT algorithm C on input (G', i) for random i such that $V^*_{[1,l]} \stackrel{\text{f}}{=} V^i_{[1,l]}$.

The understanding algorithm U^S.

Input to U: G, G' and an initially empty set T[11].
Oracle access to $S(V^*_{[1,l]})$.

1. Group $V^1, V^2..., V^q$ as follows: If two verifiers, say, V^i and V^j, share the same randomness used in their first l next-message functions they are placed in the same group.
   Assume this results in h, $h \leq q$, different verifier groups G_1,G_2,...,G_h.
2. For each group G_f, $1 \leq f \leq h$, do:
   2.1. Pick a verifier, say, V^j in group G_f[12]. U plays the role of oracle O and uses V^j as verifier strategy to interact with $S(V^*_{[1,l]})$ until obtaining a session prefix history $(r_1, p_1, ..., p_{l-1}, r_l)$.
   2.2. Run each verifier in G_f exhausting all possible prover messages after the l-th verifier step, and see if the prefix history enjoys the following **Nice Properties** with respect to G_f:
   - For every verifier V^j in G_f, there exists a session continuation $(p_l, r_{l+1}, ..., p_m)$ such that V^j accepts the transcript $(r_1, p_1..., r_l, p_l, r_{l+1}, ..., p_m)$.
   - For any $i \geq l$, if two verifiers in G_f sharing the same prefix $(r_1, p_1..., r_i)$, the next prover message p_i is the same in the two successful continuations (i.e., accepting by these two verifiers).

   If these two properties do NOT hold, add $V^j_{[1,l]}$ to the set T.
3. Output an arbitrary $V^j_{[1,l]}$ in T.

First note that if the above two "nice properties" hold with respect to group G_f, then there exists a single (inefficient) prover strategy such that, given the prefix $(r_1, p_1..., r_l)$, the prover can cheat all verifiers in G_f.

By Lemma 1, it is easy to see that T is not empty; i.e., there must exist a group, G, such that given the session prefix $(r_1, p_1..., r_l)$ generated in the interaction between $S(V^*_{[1,l]})$ and a verifier in G, there does NOT exist session continuations for which the above two "nice properties" hold (in other words, there does not exist a single prover strategy that can cheat all verifiers in G), as otherwise we would have a single prover with x and $V^*_{[1,l]}$ as auxiliary input who takes $S(V^*)$ as a partial prover strategy (up to the l-th prover step) and could make all of $V^1, V^2..., V^q$ accept, contradicting Lemma 1.

We now prove that the probability (taken over the randomness of U and the randomness of S) that the output $V^j_{[1,l]}$ of U^S is not correct, i.e.,

$$Pr[V^j_{[1,l]} \in T \wedge V^*_{[1,l]} \stackrel{\text{f}}{=} V^j_{[1,l]}]$$

is negligible.

Assume otherwise, i.e., U's output $V^j_{[1,l]} \stackrel{\text{f}}{=} V^*_{[1,l]}$. Denote by G_j the set of (complete) verifiers sharing the same prefix $V^j_{[1,l]}$, and by G'_j the suffix verifiers in G_j, i.e., $G'_j = \{V_{[l+1,m]} : [V^j_{[1,l]}, V_{[l+1,m]}] \in G_j\}$. Then, if we fix S's randomness, and write down the session outputs by $S^{O(V^*_{[1,l]}, V^i_{[l+1,m]})}(V^*_{[1,l]})$ for all $V^i_{[l+1,m]} \in G'_j$, we obtain the following transcripts:

[11] Keep in mind that here we omit inputs x and randomness to S and U for simplicity.

[12] Since verifiers in the same group share the same randomness used in the first verifier steps, we always obtain the same prefix no matter which verifier in this group we choose.

$$\{(r_1, p_1, ..., p_m) \leftarrow S^{O(V^*_{[1,l]}, V^i_{[l+1,m]})}(V^*_{[1,l]}) : V^i_{[l+1,m]} \in G'_j\},$$

which satisfy the following three properties:

1. All these sessions share the same prefix history, say, $(r_1, p_1, ..., r_l)$ (actually they share the same prefix $(r_1, p_1, ..., r_l, p_l)$, up to the l-th prover step, since S uses the same randomness in all these sessions).
2. Every session is accepting except with negligible probability. This is because $S^{O(V^*_{[1,l]}, V^i_{[l+1,m]})}(V^*_{[1,l]})$ is a simulator for $[V^*_{[1,l]}, V^i_{[l+1,m]}]$ (by the canonical property), which is also polynomial-time constructible, and thus, by Lemma 2, it outputs an accepting transcript except with negligible probability.
3. All sessions satisfy the second "nice property." This is due to the fact that S interacts with O in a straight-line manner, and that it produces the prover messages after the l-th verifier step without being given any information about the description of $V^i_{[l+1,m]}$.

Taken together, these properties mean that, given the prefix $(r_1, p_1..., r_l)$, U can always find (albeit inefficiently) session continuations such that the two "nice properties" in step 2.2 of algorithm U hold. Thus, except with negligible probability, if $V^*_{[1,l]} \stackrel{\mathrm{f}}{=} V^j_{[1,l]}$, then we have $V^j_{[1,l]} \notin T$.

In sum, with probability negligibly close to 1, algorithm U, running in time at most $q \cdot \mathsf{poly} \cdot 2^{mp} \in 2^{O(p)}$, will output at least one $V^j_{[1,l]}$ whose functionality is *different* from that of $V^*_{[1,l]}$.

This concludes the proof of Theorem 1 for the case of a single PCP step. The general case (multiple PCP steps) is presented in Appendix A.2.

5 Conclusions

A natural question which arises from our reduction is: How hard is the verifier-understanding problem? It depends. Note that what we wish to understand is the partial code of an honest verifier algorithm, which is simply a set of *constant functions*, since the proof system is public-coin. Thus, if the algorithm S actually runs this code internally, it can figure out the functionality of the target code of the partial verifier fairly easily, and thus our understanding problem should be doable in moderately exponential time by U^S.

However, if S does *not* run the target code at all in the simulation process (neither does U^S—recall that U is not even being given the target code), then it seems very unlikely for such U^S to be able to solve our understanding problem, even in exponential time, when required to solve *arbitrary* polynomial-time-constructible code.

Note that all known straight-line simulators can be rewritten into our oracle algorithm form, S^O, for which S does not need to execute the malicious verifier V^* on its own. For example, the simulator for Barak's protocol [2] only needs the actual computation of V^* in its second prover step (the PCP step) in order to prepare a PCP proof; for any other prover steps, it can go through without any actual computation of V^*. Thus, when we view Barak's simulation as an interaction between S and O, and further have O provide S with the actual computation of the malicious verifier, we can have an algorithm S that goes through the entire simulation without any execution of V^*. Hence, we conclude that the currently known straight-line simulation technique cannot be extended to ZK proof system unless our verifier-understanding problem can be solved.

References

[1] L. Babai: Trading Group Theory for Randomness. STOC 1985, pp. 421-429, 1985.
[2] B. Barak: How to go beyond the black-box simulation barrier. FOCS 2001, pp.106-115.
[3] G. Brassard, D. Chaum, and C. Crépeau Minimum disclosure proofs of knowledge. J. Comput. Syst. Sci., 37(2):156-189, 1988.
[4] B. Barak, O. Goldreich, S. Goldwasser, Y. Lindell: Resettably sound ZK and its Applications. FOCS 2001, pp. 116-125, 2001.
[5] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, K. Yang: On the (Im)possibility of Obfuscating Programs. CRYPTO 2001, pp.1-18, 2001.
[6] B. Barak, Y. Lindell: Strict polynomial-time in simulation and extraction. STOC 2002, pp.484-493. 2002.
[7] M. Blum: How to prove a theorem so no one else can claim it. Proceedings of theInternational Congress of Mathematicians, pp.444-451, 1986.
[8] B. Barak, Y. Lindell, S. P. Vadhan: Lower Bounds for Non-Black-Box Zero Knowledge. FOCS 2003, pp.384-393,2003.
[9] L. Babai, S. Moran: Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes. J. Comput. Syst. Sci. 36(2): 254-276, 1988.
[10] M. Bellare, J. Rompel: Randomness-Efficient Oblivious Sampling FOCS 1994: 276-287.
[11] R. Canetti and R. R. Dakdouk: Obfuscating Point Functions with Multibit Output. EUROCRYPT 2008, pp.489-508, 2008.
[12] R. Canetti, O. Goldreich, S. Goldwasser, S. Micali. Resettable Zero Knowledge. STOC 2000, pp.235-244, 2000.
[13] R. Canetti, J. Kilian, E. Petrank and A. Rosen. Concurrent Zero-Knowledge requires $\Omega(logn)$ rounds. STOC 2001, pp.570-579, 2001.
[14] I. Damgard. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. EUROCYPT 2000, pp.174-187, 2000.
[15] Y. Deng, V. Goyal, A. Sahai: Resolving the Simultaneous Resettability Conjecture and a New Non-Black-Box Simulation Strategy. FOCS 2009: 251-260
[16] G. Di Crescenzo, Ivan Visconti. Concurrent ZK in the Public-Key Model. ICALP 2005, pp.816-827, 2005.
[17] C. Dwork, M. Naor and A. Sahai. Concurrent Zero-Knowledge. STOC 1998, pp.409-418, 1998.
[18] O. Goldreich and H. Krawczyk: On the Composition of Zero-Knowledge Proof Systems. SIAM J. Comput. 25(1), pp.169-192, 1996.
[19] O. Goldreich, S. Micali and A. Wigderson. Proofs that yield nothing but their validity or All languages in NP have zero-knowledge proof systems. J. ACM, 38(3), pp.691-729, 1991.
[20] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. SIAM. J. Computing, 18(1):186-208, February 1989.
[21] O. Goldreich, S. Vadhan and A. Wigderson. On Interactive Proofs with a Laconic Prover. ICALP 2001, pp. 334-345.
[22] S. Goldwasser, Guy N. Rothblum: On Best-Possible Obfuscation. TCC 2007, pp.194-213, 2007.
[23] S. Hada: Zero-Knowledge and Code Obfuscation. ASIACRYPT 2000, pp.443-457, 2000.
[24] S. Hohenberger, Guy N. Rothblum, A. Shelat, V. Vaikuntanathan: Securely Obfuscating Re-encryption. TCC 2007, pp.233-252, 2007.
[25] R. Pass, A. Rosen: New and improved constructions of non-malleable cryptographic protocols. STOC 2005: 533-542.
[26] M. Prabhakaran, A. Rosen, A. Sahai: Concurrent Zero Knowledge with Logarithmic Round-Complexity. FOCS 2002, pp.366-375, 2002.
[27] H. Wee: On obfuscating point functions. STOC 2005, pp.523-532, 2005.

A Proofs

A.1 Proof of Lemma 1

We first introduce some definitions and additional notation that will be used in the proof.

We assume that the length of each prover message is greater than any constant, in particular, $p > 10$. Note that this assumption is without loss of generality because if the length of the prover message in a constant-round interactive proof for a language L is constant, then L is trivial (see [21]), which in turn implies our lemma immediately.

Throughout this subsection, we consider only *structured* q-tuples of verifier's random tapes, which are selected in the following way:

1. For each verifier step i, $1 \leq i \leq m$, if $|\{0,1\}^{l_i}| > \frac{m^2 p}{2 \log \frac{1}{\epsilon}}$, set $t_i = \frac{m^2 p}{2 \log \frac{1}{\epsilon}} \in p/O(\log \frac{1}{\epsilon})$; otherwise, set $t_i = 2^{l_i}$, where l_i is the length of the i-th verifier message;
2. Choose t_i *distinct* strings $r_{1i}, r_{2i}, ..., r_{t_i i}$ from $\{0,1\}^{l_i}$;

3. Choose an i-th verifier message $r_{j_i i} \in (r_{1_i i}, r_{2i}, ..., r_{t_i i})$, $1 \leq j_i \leq t_i$ for each step i, and set random tape $\mathbf{r}^j = [r_{j_1 1}, r_{j_2 2}, ..., r_{j_m m}]$.

4. A q-tuple of random tapes is now the set of all possible random tapes set in step 3, $(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^q)$. Note that the size q of this set is $\prod_{i=1}^{m} t_i$, which is determined by Step 2.

We identify $(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$ with $(\mathbf{r}^{\pi(1)}, \mathbf{r}^{\pi(2)}, ...\mathbf{r}^{\pi(q)})$ for any permutation π on $\{1, 2, ..., q\}$. Two q-tuples, $(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$ and $(\mathbf{r'}^1, \mathbf{r'}^2, ...\mathbf{r'}^q)$, are said to be *distinct* if there exists at least one $\mathbf{r}^i$ such that $\mathbf{r}^i \in (\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$ but $\mathbf{r}^i \notin (\mathbf{r'}^1, \mathbf{r'}^2, ...\mathbf{r'}^q)$, or vice-versa. Thus the number of all possible distinct such structured q-tuples is

$$\prod_{i=1}^{m} \binom{2^{l_i}}{t_i}.$$

Some more basic notation before the proof:

- $\text{prefix}_i(\mathbf{r}^j)$: the first i messages from the verifier using random tape $\mathbf{r}^j$, that is, for $\mathbf{r}^j = [r_1^j, r_2^j, ..., r_m^j]$, $\text{prefix}_i(\mathbf{r}^j) = [r_1^j, r_2^j, ..., r_i^j]$.
- $\overrightarrow{T}$ and its size $|\overrightarrow{T}|$: $\overrightarrow{T}$ is a *set* of structured q-tuples of verifier's random tapes, The size of $\overrightarrow{T}$, denoted by $|\overrightarrow{T}|$, is simply defined to be the number of *distinct* q-tuples in $\overrightarrow{T}$.
- $p_k \leftarrow \langle P(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), V\rangle_{|\mathsf{hist}}$ denotes the k-th prover message produced by the prover $P(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$ (the prover strategy taking q-tuple $(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$ as auxiliary input), conditioned on hist being the current history so far.

The proof of the lemma is by contradiction. Assume that there exists an unbounded prover, call it $P^\diamond$, and $x \notin L$, such that for any q-tuple $(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$, $r_i \neq r_j$ for $i \neq j$:

$$\Pr[\langle P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), V_{|(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)}\rangle(x) = 1] > 1 - \frac{1}{q}\,. \tag{2}$$

First note that $V_{|(\mathbf{r}^1, \mathbf{r}^2, ..., \mathbf{r}^t)}(\mathbf{r}^i)$ acts exactly the same as $V(\mathbf{r}^i)$. Therefore

$$\Pr[\langle P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), V_{|(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)}\rangle(x) = 1] \tag{3}$$

$$= \sum_i \Pr[\langle P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), V_{|(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)}(\mathbf{r}^i)\rangle(x) = 1]\frac{1}{q} \tag{4}$$

$$= \sum_i \Pr[\langle P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), V(\mathbf{r}^i)\rangle(x) = 1]\frac{1}{q}\,. \tag{5}$$

Further, observe that the probability $\Pr[\langle P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), V(\mathbf{r}^i)\rangle(x) = 1]$ is either 0 or 1 because in this interaction the tapes are fixed and both prover and verifier are deterministic. Thus, if inequality (2) holds, we have

$$\Pr[\langle P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), V_{|(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)}\rangle(x) = 1] = 1\,, \tag{6}$$

and, by (5),

$$\Pr[\langle P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), V(\mathbf{r}^i)\rangle(x) = 1] = 1\,. \tag{7}$$

Now, given prover $P^\diamond$ such that (7) holds for any q-tuple $(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$, we describe a prover P^* that will cheat V with probability greater than ϵ.

The Cheating Prover P^*.

Input: x, as in inequality (2).

1. Set $\overrightarrow{T^0}$ to be the set of all possible distinct structured q-tuples over $\{0,1\}^{l_1+l_2+...+l_m}$, and G_1 the set of all possible first verifier's messages (i.e., the set $\{0,1\}^{l_1}$).

2. For $k = 1$ to m, do

   2.1. Upon receiving the k-th verifier message r_k, set hist to be the current history $(r_1, p_1^*, ..., r_k)$. Check if $r_k \in G_k$. If $r_k \notin G_k$, abort and output "$\perp$". Otherwise, for every q-tuple $(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q) \in \overrightarrow{T^0}$ such that: a) it contains some $\mathbf{r}^i$ such that $\text{prefix}_{k-1}(\mathbf{r}^i) = [r_1, r_2, ..., r_k]$, and, b) the current hist is consistent with the interaction between $P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$ and V, set $t' = \prod_{k+1}^m t_i$, compute the k-th prover message by running $P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$, and obtain the set of k-th prover messages

$$\{p_k \leftarrow \langle P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), V\rangle_{|\mathsf{hist}} :$$
$$(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q) \in \overrightarrow{T^0} \text{ and } \exists(\mathbf{r}^{i_1}, \mathbf{r}^{i_2}, ...\mathbf{r}^{i_{t'}}) \in (\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q) \text{ s.t.}$$
$$\text{prefix}_k(\mathbf{r}^{i_j}) = [r_1, r_2, ..., r_k] \text{ for all } 1 \le j \le t' = \prod_{k+1}^{m} t_i\}^{13}.$$

   Set p_k^* to be the p_k that maximizes the size of the set

$$\{(\mathbf{r}^{i_1}, \mathbf{r}^{i_2}, ...\mathbf{r}^{i_{t'}}) : p_k \leftarrow \langle P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), V\rangle_{|\mathsf{hist}}, \text{ and}$$
$$(\mathbf{r}^{i_1}, \mathbf{r}^{i_2}, ...\mathbf{r}^{i_{t'}}) \in (\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), \text{ and}$$
$$\text{prefix}_k(\mathbf{r}^{i_j}) = [r_1, r_2, ..., r_k] \text{ for all } 1 \le j \le t' = \prod_{k+1}^{m} t_i\}$$

   2.2. If $k < m$, denote by $\overrightarrow{T^k}$ the above set that achieves its maximum size, and set (guessing the next verifier messages)

$$G_{k+1} \leftarrow \{r_{k+1} \in \{0,1\}^{l_{k+1}} : |\{(\mathbf{r}^{i_1}, \mathbf{r}^{i_2}, ...\mathbf{r}^{i_{t'}}) \in \overrightarrow{T^k} :$$
$$\text{prefix}_{k+1}(\mathbf{r}^{i_j}) = [r_1, r_2, ..., r_k, r_{k+1}] \text{ for all } 1 \le j \le t' = \prod_{k+1}^{m} t_i\}| \ge \frac{\prod_{i=k+1}^m \binom{2^{l_i}}{t_i}}{2^{1.1kp}}\}.$$

In a nutshell, the above algorithm just tries many different cheating provers $P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$ to make the current history accepted by as many verifiers as possible.

Analysis of algorithm P^*. Let us now analyze the success probability of the prover's strategy outlined above. We first show that the size of G_k is large enough for every k.

Claim. For every $1 \le k \le m$, conditioned on P^* not outputting $\perp$, $|G_k| \ge \frac{2^{l_k}}{2^{1.1kp/t_k e}}$.

Proof. When $k = 1$, $|G_1| = |\{0,1\}^{l_1}| > \frac{2^{l_k}}{2^{1.1kp/t_k e}}$.

When $k \ge 2$, the condition of P^* not outputting "$\perp$" implies that, for $j \le k$, r_j is in G_j, and that

$$|\{(\mathbf{r}^{i_1}, \mathbf{r}^{i_2}, ...\mathbf{r}^{i_{t''}}) \in \overrightarrow{T^{k-1}} : \text{prefix}_k(\mathbf{r}^{i_j}) = [r_1, r_2, ..., r_k] \text{ for all } 1 \le j \le t'' = \prod_{k}^{m} t_i\}| \ge$$
$$\frac{\prod_{i=k}^m \binom{2^{l_i}}{t_i}}{2^{1.1(k-1)p}}.$$

which in turn leads to (recall that the length of prover messages is p), for $k \ge 2$,

[13] Observe that, by the structure of q-tuple, if there exists a $\mathbf{r}^i \in (\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$ such that $\text{prefix}_k(\mathbf{r}^i) = [r_1, r_2, ..., r_k]$, then there exist $t' = \prod_{k+1}^m t_i$ many such random tapes.

$$|\overrightarrow{T^k}| \geq \frac{\prod_{i=k}^{m}\binom{2^{l_i}}{t_i}}{2^{1.1(k-1)p+p}} = \frac{\prod_{i=k}^{m}\binom{2^{l_i}}{t_i}}{2^{1.1kp-0.1p}}\,. \tag{8}$$

Now assume that, for $k \geq 2$, conditioned on P^* not outputting "$\perp$" (i.e., for $j \leq k$, r_j is in G_j), $|G_k| < \frac{2^{l_k}}{2^{1.1kp/t_k e}}$.

Set $t' = \prod_{k+1}^{m} t_i$. Recall that all t'-tuples $(\mathbf{r}^{i_1}, \mathbf{r}^{i_2}, ..., \mathbf{r}^{i_{t'}}) \in \overrightarrow{T^k}$ share the same prefix $[r_1, r_2, ..., r_k]$, and that, by the structure of q-tuple of random tapes, within a t'-tuple $(\mathbf{r}^{i_1}, \mathbf{r}^{i_2}, ..., \mathbf{r}^{i_{t'}}) \in \overrightarrow{T^k}$, there are only t_k distinct k-th verifier messages, say $(r_k^1, r_k^2, ..., r_k^{t_k})$. We partition these t'-tuples in $\overrightarrow{T^k}$ in two classes by the property of $(r_k^1, r_k^2, ..., r_k^{t_k})$:

1. Every $r_k^i \in (r_k^1, r_k^2, ..., r_k^{t_k})$ is in G_k (which implies $t_k \leq |G_k|$). The number of t'-tuples in $\overrightarrow{T^k}$ satisfying this condition is at most
$$\binom{|G_k|}{t_k}\prod_{i=k+1}^{m}\binom{2^{l_i}}{t_i}.$$

2. There is at least one $r_k^i \in (r_k^1, r_k^2, ..., r_k^{t_k})$ that is *not* in G_k. Then by the definition of G_k, and by the fact that, within a t'-tuple $(\mathbf{r}^{i_1}, \mathbf{r}^{i_2}, ..., \mathbf{r}^{i_{t'}}) \in \overrightarrow{T^k}$, for every i, the number of random tapes in this t'-tuple with each prefix $[r_1, r_2, ..., r_{k-1}, r_k^i]$ is the same (equal to $\prod_{k+1}^{m} t_i$), then the number of t'-tuples in $\overrightarrow{T^k}$ satisfying this condition is at most
$$\binom{2^{l_k}}{t_k}\frac{\prod_{i=k+1}^{m}\binom{2^{l_i}}{t_i}}{2^{1.1kp}}.$$

Thus, we have

$$\begin{aligned}
|\overrightarrow{T^k}| &\leq \binom{|G_k|}{t_k}\prod_{i=k+1}^{m}\binom{2^{l_i}}{t_i} + \binom{2^{l_k}}{t_k}\frac{\prod_{i=k+1}^{m}\binom{2^{l_i}}{t_i}}{2^{1.1kp}} \\
&< \binom{\frac{2^{l_k}}{2^{1.1kp/t_k e}}}{t_k}\prod_{i=k+1}^{m}\binom{2^{l_i}}{t_i} + \binom{2^{l_k}}{t_k}\frac{\prod_{i=k+1}^{m}\binom{2^{l_i}}{t_i}}{2^{1.1kp}} \\
&< (\frac{2^{l_k}}{2^{1.1kp/t_k}t_k})^{t_k}\prod_{i=k+1}^{m}\binom{2^{l_i}}{t_i} + \binom{2^{l_k}}{t_k}\frac{\prod_{i=k+1}^{m}\binom{2^{l_i}}{t_i}}{2^{1.1kp}} \\
&< \frac{(\frac{2^{l_k}}{t_k})^{t_k}}{2^{1.1kp}}\prod_{i=k+1}^{m}\binom{2^{l_i}}{t_i} + \binom{2^{l_k}}{t_k}\frac{\prod_{i=k+1}^{m}\binom{2^{l_i}}{t_i}}{2^{1.1kp}} \\
&< \frac{\prod_{i=k}^{m}\binom{2^{l_i}}{t_i}}{2^{1.1kp-1}},
\end{aligned}$$

which contradicts (8) when $p > 10$, which we can always assume without loss of generality (otherwise our lemma holds trivially; see [21]). □

Now observe that, for every prover step $k \leq m$, if $G_k \geq \frac{2^{l_k}}{2^{1.1kp/t_k e}}$, then the probability that P^* guesses the next verifier message correctly, i.e., the probability that $r_k \in G_k$, is $|G_k|/2^{l_k} = \frac{1}{2^{1.1kp/t_k e}}$. Therefore P^*

guesses all the next verifier messages correctly with probability at least

$$\prod_{k=1}^{m} \frac{|G_k|}{2^{l_k}} = \prod_{k=1}^{m} \frac{1}{2^{1.1kp/t_k e}},$$

which is greater than ϵ for $t_k \leq \frac{m^2 p}{2\log\frac{1}{\epsilon}}$. (Recall that either $t_k = \frac{m^2 p}{2\log\frac{1}{\epsilon}}$, or $t_k = 2^{l_k}$ when $2^{l_k} \leq \frac{m^2 p}{2\log\frac{1}{\epsilon}}$.)

Notice also that, in case that all guesses of the next verifier messages are correct, there exists at least one q-tuple $(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$ such that the complete transcript $(r_1, p_1^*...r_m, p_m^*)$ is generated in the interaction between $P^\diamond(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$ and $V(\mathbf{r}^i)$, $\mathbf{r}^i = [r_1, r_2...r_m] \in (\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$, which is guaranteed by our assumption to be accepting.

In sum, our cheating prover P^* will cheat with probability greater than ϵ, which breaks the soundness of the proof system $\langle P, V\rangle$, thus yielding the lemma.

A.2 Proof of Theorem 1: The General Case

We now generalize the approach described in Section 4.2 to prove Theorem 1 in the general case of multiple PCP steps in the $\langle P, V\rangle$ interaction. The proof relies on the following somewhat technical lemma. Again, Assume $\langle P, V\rangle$ is a canonical GSLS ZK proof system of $2m$ rounds.

Lemma 3. *Fix the false statement x and the verifiers $V^1, V^2, ..., V^q$ guaranteed by Lemma 2. Then there exist two consecutive PCP step indexes k and l, $0 \leq k < l \leq m$, and a prefix verifier $V'_{[1,k]}$, $V'_{[1,k]} \stackrel{\text{f}}{=} V^j_{[1,k]}$ for some j, $1 \leq j \leq q$ (when $k = 0$, $V'_{[1,k]}$ is set to empty string), such that the following two conditions hold:*

1. *Let G denote the set of honest verifiers $(V^1, V^2, ..., V^q)$ that share the same prefix strategy $V^j_{[1,k]}$. Except with negligible probability (over the randomness of S), the history $(r_1, p_1, ..., r_k)$[14] generated by $S^{O(V'_{[1,k]})}(V'_{[1,k]})$[15] does NOT satisfy algorithm U's "nice properties" with respect to G. That is, given this history, no unbounded prover can cheat a random verifier in G with probability 1.*
2. *For any polynomial-time constructible suffix verifier $V^*_{[k+1,l]}$ such that the verifier $[V'_{[1,k]}, V^*_{[k+1,l]}]$ is functionally equivalent to the prefix strategy of some verifiers in G, except for negligible probability, the prefix $(r_1, p_1, ..., r_k, p_k, r_{k+1}, ..., r_l)$ generated by $S^{O(V'_{[1,k]}, V^*_{[k+1,l]})}(V'_{[1,k]}, V^*_{[k+1,l]})$ satisfies U's "nice properties," with respect to the set of verifiers in G whose prefixes are functionally equivalent to $[V'_{[1,k]}, V^*_{[k+1,l]}]$.*

Proof. We prove the lemma by examining the PCP steps in the simulation process one by one.

Fix the randomness for the simulator S. We first look at the first PCP step, say, prover step d. If for any polynomial-time constructible $V^*_{[1,d]}$, the session prefix $(r_1, p_1, ..., r_d)$ produced by the simulator $S^{O(V^*_{[1,d]})}(V^*_{[1,d]})$ enjoys U's "nice properties" (i.e., there exist a single prover strategy that can make a random verifier in $(V^1, V^2, ..., V^q)$ that share the same functionality up to d-th step as $V^*_{[1,d]}$ accept with probability 1), then we set k to 0, l to d, and $V'_{[1,k]}$ to the empty string, and lemma 3 follows from this fact, which satisfies the second condition of lemma 3, and lemma 2, which guarantees that the first condition holds with respect to $G = (V^1, V^2, ..., V^q)$, i.e., no unbounded prover can make a random verifier in G accept with probability 1.

Otherwise, i.e., there exists $V'_{[1,d]}$ such that the prefix $(r_1, p_1, ..., r_d)$ produced by $S^{O(V'_{[1,d]})}(V'_{[1,d]})$ does not satisfy U's "nice properties," we then examine the second PCP step, say, prover step e. If for

[14] When k is set to 0, this history is also set to empty string.

[15] Note that, by the canonical property of $\langle P, V\rangle$, S works with only the partial verifier $V'_{[1,k]}$ as input.

any polynomial-time constructible $V^*_{[d+1,e]}$, the prefix history $(r_1, p_1, ..., r_e)$ produced by $S^{O(V'_{[1,d]}, V^*_{[d+1,e]})}$ $(V'_{[1,d]}, V^*_{[d+1,e]})$ satisfies U's "nice properties," Lemma 3 follows from the "failure"(to satisfy the nice properties) at PCP step d and the "success" at this PCP step e. In this case, we set k to d, l to e.

Otherwise, we extend $V'_{[1,d]}$ to $V'_{[1,e]}$ (for which the simulator generates a session prefix up to the e-th verifier step that does not enjoy the nice properties), and continue to examine the third PCP step.

We continue with this process until we find a PCP step for which for any polynomial time constructible code of partial verifier up to this PCP step, our simulator can generate a prefix satisfying the nice properties, and once we find it, we are done. Finding such a PCP step is guaranteed by the following fact: Assume the last PCP step is the prover step h. Then, for any polynomial-time constructible $V^*_{[1,h]}$, the session prefix $(r_1, p_2, ..., r_h)$ produced by $S^{O(V^*_{[1,h]})}(V^*_{[1,h]})$ satisfies U's "nice properties" with respect to the group of verifiers in $(V^1, V^2, ...V^q)$ that share the same functionality up to the h-th step as $V^*_{[1,h]}$, due to the natural property and the fact that S interacts with O in a straight-line manner (the same reasons given in the analysis of U, see section 4.2 for details).

In case we reach the last PCP step h, we set k to the index of next-to-last PCP step, l to h, and $V'_{[1,k]}$ to the code we found when we examined the next-to-last PCP step. □

We now turn to the proof of Theorem 1 in the multiple-PCP-step case.

Again, we will construct an understanding algorithm U'^S with respect to the following parameters:

- $x \notin L$, k, l and $V'_{[1,k]}$ as defined in Lemma 3;
- G: the set of deterministic honest verifiers in $(V^1, V^2, ..., V^q)$ that share the same prefix verifier $V^j_{[1,k]}$, which is functionally equivalent to $V'_{[1,k]}$;
- G': the set of the partial verifiers on which algorithm U' is asked to operate, defined as

$$\{V_{[k+1,l]} : \exists V_{[l+1,m]} \text{ s.t. } [V^j_{[1,k]}, V_{[k+1,l]}, V_{[l+1,m]}] \in G\}.$$

- $V^*_{[k+1,l]}$: the output of an arbitrary PPT C on input (G', i) for a random i such that $V^*_{[k+1,l]} \stackrel{\text{f}}{=} V^i_{[k+1,l]} \in G'$.

Taking $V'_{[1,k]}$ and G as auxiliary input, algorithm U' on input G', with oracle access to $S(V'_{[1,k]}, V^*_{[k+1,l]})$, carries out the verifier-understanding task with respect to the set of partial verifiers G'. It proceeds the same as algorithm U with the following modifications:

1. In step 1 of U, U' subgroups the verifiers in G according to the randomness used in verifier step $k+1$ up to verifier step l. Note that for every verifier in G, the randomness used in the first k verifier steps is the same.
2. In step 2.1 of U, for each subgroup G_f, U' picks a partial strategy $V^t_{[k+1,l]}$ of a verifier in G_f, plays the role of the oracle O and uses the prefix verifier $[V'_{[1,k]}, V^t_{[k+1,l]}]$ to interact with $S(V'_{[1,k]}, V^*_{[k+1,l]})$ until obtaining a session prefix $(r_1, p_1, ..., p_{l-1}, r_l)$. In this process, when S asks for some actual computation of $V'_{[1,k]}$, U' generates them for S.
   Note that during this step, U' does not need to provide S with any actual computation of $V^*_{[k+1,l]}$ since k and l are two consecutive PCP steps.

Lemma 3 now guarantees that, with probability negligibly close to 1, U' will output a verifier $V^t_{[k+1,l]}$ such that $V^t_{[k+1,l]} \stackrel{\text{f}}{\neq} V^*_{[k+1,l]}$. In addition, note that the running time of U' is at most $q' \cdot \text{poly} \cdot 2^{mp} \in 2^{O(p)}$. This completes the proof of Theorem 1.

B Interactive Proof Systems with Super-Constant Rounds

In this section we give a simple super-constant-round public-coin interactive proof system for which Lemma 1 does not hold.

Preamble: For $1 \leq k \leq s$, do:
$P \rightarrow V$: Send n random strings p_1^k,...,p_n^k of length n each.
$V \rightarrow P$: Send a random string r_k of length n.
Main proof: If there is some $p_i^k = r_k$, V accepts; otherwise execute a 3-round Blum protocol [7] with negligible soundness error.

Observe that for any q, if q different verifier random tapes $(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q)$ are fixed in advance and known to an all-powerful prover, then for the cheating probability to be strictly less than 1, there must be at least $n+1$ different verifier messages at any verifier step $k \leq s$ (i.e., the entropy $H(r_k|(\mathbf{r}^1, \mathbf{r}^2, ...\mathbf{r}^q), \mathsf{hist})$ is greater than $\log n$), which leads to $q \geq (n+1)^s$. That is, if s is super-constant, for any polynomial number of verifier's random tapes that are fixed in advance we have a prover with cheating probability 1.