# A note on the round-complexity of Concurrent Zero-Knowledge

Alon Rosen

Department of Computer Science
Weizmann Institute of Science
Rehovot 76100, Israel
alon@wisdom.weizmann.ac.il

**Abstract.** We present a lower bound on the number of rounds required by Concurrent Zero-Knowledge proofs for languages in $\mathcal{NP}$. It is shown that in the context of Concurrent Zero-Knowledge, at least eight rounds of interaction are essential for black-box simulation of non-trivial proof systems (i.e., systems for languages that are not in $\mathcal{BPP}$). This improves previously known lower bounds, and rules out several candidates for constant-round Concurrent Zero-Knowledge. In particular, we investigate the Richardson-Kilian protocol [20] (which is the only protocol known to be Concurrent Zero-Knowledge in the vanilla model), and show that for an apparently natural choice of its main parameter (which yields a 9-round protocol), the protocol is not likely to be Concurrent Zero-Knowledge.

## 1 Introduction

Zero-knowledge proof systems, introduced by Goldwasser, Micali and Rackoff [14] are efficient interactive proofs which have the remarkable property of yielding nothing beyond the validity of the assertion being proved. The generality of zero-knowledge proofs has been demonstrated by Goldreich, Micali and Wigderson [12], who showed that every NP-statement can be proved in zero-knowledge provided that one-way functions exist [16, 19]. Since then, zero-knowledge protocols have turned out to be an extremely useful tool in the design of various cryptographic tasks.

The original setting in which zero-knowledge proofs were investigated consisted of a single prover and verifier which execute only one instance of the protocol at a time. A more realistic setting, especially in the time of the internet, is one which allows the concurrent execution of zero-knowledge protocols. In the concurrent setting (first considered by Dwork, Naor and Sahai [6]), many protocols (sessions) are executed at the same time, involving many verifiers which may be talking with the same (or many) provers simultaneously (the so-called parallel composition considered in [11] is a special case). This presents the new risk of an overall adversary which controls the verifiers, interleaving the executions and choosing verifiers queries based on other partial executions of the protocol. Since it seems unrealistic for the honest provers to coordinate their action so that zero-knowledge is preserved, we must assume that in each prover-verifier pair the prover acts independently. A zero-knowledge proof is said to be *concurrent zero-knowledge* if it remains zero-knowledge even when executed in the concurrent setting. Recall that in order to prove that a certain protocol is zero-knowledge it is required to demonstrate that every probabilistic polynomial-time adversary interacting with the prover can be simulated by a probabilistic polynomial-time machine (a.k.a. the *simulator*) which is in solitude. In the concurrent setting, the simulation task becomes even more complicated, as the adversary may have control over multiple sessions at the same time, and is thus able to determine their scheduling (i.e., the order in which the interleaved execution of these sessions should be conducted).

## 1.1 Previous Work

Coming up with an efficient concurrent zero-knowledge protocol for all languages in $\mathcal{NP}$ seems to be a challenging task. Indications on the difficulty of this problem were already given in [6], where it was argued that for a specific recursive scheduling of $n$ sessions a particular (natural) simulation of a particular 4-round protocol may require time which is exponential in $n$. Further evidence on the difficulty was given by Kilian, Petrank and Rackoff [18]. Using the same recursive scheduling as in [6], they were able to prove that for every language outside $\mathcal{BPP}$ there is no 4-round protocol whose concurrent execution is simulatable in polynomial-time (by a *black-box simulator*).

Recent works have (successfully) attempted to overcome the above difficulties by augmenting the communication model with the so-called *timing assumption* [6, 7] or, alternatively, by using various set-up assumptions (such as the *public-key model* [4, 5]).[1] For a while it was not clear whether it is even possible to come up with a concurrent zero-knowledge protocol (not to mention an efficient one) without making any kind of timing or set-up assumptions. It was therefore a remarkable achievement when Richardson and Kilian [20] proposed a concurrent zero-knowledge protocol for all languages in $\mathcal{NP}$ (in the vanilla model).[2]

Unfortunately, the simulator shown by Richardson-Kilian is polynomial-time only when a non-constant round version of their protocol is considered. This leaves a considerable gap between the currently known upper and lower bounds on the number of rounds required by concurrent zero-knowledge [20, 18]. We note that narrowing the above gap is not only of theoretical interest but has also practical consequences. Since the number of rounds is an important resource for protocols, establishing whether constant-round concurrent zero-knowledge exists is a well motivated problem.

[1] The lower bound of [18] (as well as our own work) applies only in a cleaner model, in which no timing or set-up assumptions are allowed (the so-called *vanilla* model).

[2] In fact, their solution is a family of protocols where the number of rounds is determined by a special parameter.

## 1.2 A closer look at the Richardson-Kilian Protocol

Being the only protocol known to be concurrent zero-knowledge in the vanilla model, versions of the Richardson-Kilian protocol are natural candidates for constant-round concurrent zero-knowledge. That is, it is still conceivable that there exists a (different) polynomial-time simulator for one of the protocol's constant-round versions.

**The Protocol:** The Richardson-Kilian (RK for short) protocol [20] consists of two stages. In the *first stage*, which is independent of the actual common input, the verifier commits to $k$ random bit sequences, $v_1, ..., v_k \in \{0,1\}^n$, where $n$ is the "security" parameter of the protocol and $k$ is a special parameter which determines the number of rounds. This is followed by $k$ iterations so that in each iteration the prover commits to a random bit sequence, $p_i$, and the verifier decommits to the corresponding $v_i$. The *result* of the $i^{\text{th}}$ iteration is defined as $v_i \oplus p_i$ and is known only to the prover. In the *second stage*, the prover provides a witness indistinguishable (WI) proof [8] that either the common input is in the language or that the result of one of the $k$ iterations is the all-zero string (i.e., $v_i = p_i$ for some $i$). Intuitively, since the latter case is unlikely to happen in an actual execution of the protocol, the protocol constitutes a proof system for the language. However, the latter case is the key to the simulation of the protocol in the concurrent zero-knowledge model: Whenever the simulator may cause $v_i = p_i$ to happen for some $i$ (this is done by the means of *rewinding* the verifier after the value $v_i$ has been revealed), it can simulate the rest of the protocol (and specifically Stage 2) by merely running the WI proof system with $v_i$ (and the prover's coins) as a witness.

**The Simulator:** The RK protocol was designed to overcome the main difficulty encountered whenever many sessions are to be simulated in the concurrent setting. As observed by Dwork, Naor and Sahai [6], rewinding a specific session in the concurrent setting may result in loss of work done for other sessions, and cause the simulator to do the same amount of work again. In particular, all simulation work done for sessions starting after the point to which we rewind may be lost. Considering a specific session of the RK protocol (out of $m = \text{poly}(n)$ concurrent sessions), there must be an iteration (i.e., an $i \in \{1, ..., k\}$) so that at most $(m-1)/k$ sessions start in the interval corresponding to the $i^{\text{th}}$ iteration (of this specific session). So if we try to rewind on the correct $i$, we will invest (and so waste) only work proportional to $(m-1)/k$ sessions. The idea is to abort the rewinding attempt on the $i^{\text{th}}$ iteration if more than $(m-1)/k$ sessions are initiated in the corresponding interval (this will rule out the incorrect $i$'s). The same reasoning applies *recursively* (i.e., to the rewinding in these $(m-1)/k$ sessions). Denoting by $W(m)$ the amount of work invested in $m$ sessions, we obtain the recursion $W(m) = \text{poly}(m) \cdot W(\frac{m-1}{k})$, which solves to $W(m) = m^{\Theta(\log_k m)}$. Thus, whenever $k = n$, we get $W(m) = m^{O(1)}$, whereas taking $k$ to be a constant will cause $W(m)$ to be quasi-polynomial.

## 1.3 Our First Result

Given the above state of affairs, one may be tempted to think that a better simulation method would improve the recursion into something of the form $W(m) = O(W(\frac{m-1}{k}))$. In such a case, taking $k$ to be a constant (greater than 1) would imply a constant-round protocol whose simulation in the concurrent setting requires polynomial-time (i.e., $W(m) = O(W(\frac{m-1}{k}))$ solves to $W(m) = m^{O(1)}$). This should hold in particular for $k = 2$ (which gives a 9-round version of the RK protocol). However, as we show in the sequel, this is not likely to be the case.

**Theorem 1 (informal) :** *If $L$ is a language such that concurrent executions of the 9-round version of the RK protocol (i.e., for $k = 2$) can be black-box simulated in polynomial-time, then $L \in \mathcal{BPP}$.*

Thus, in general, the RK protocol is unlikely to be simulatable by a recursive procedure (as above) that satisfies the work recursion $W(m) = O(W(\frac{m-1}{k}))$.

## 1.4 Our Second Result

The proof of Theorem 1 is obtained by extending the proof of the following general result.

**Theorem 2 :** *Suppose that $(P, V)$ is a 7-round proof system for a language $L$ (i.e., on input $x$, the number of messages exchanged is at most 7), and that concurrent executions of $P$ can be simulated in polynomial-time using black-box simulation. Then $L \in \mathcal{BPP}$. This holds even if the proof system is only computationally-sound (with negligible soundness error) and the simulation is only computationally-indistinguishable (from the actual executions).*

In addition to shedding more light on the reasons that make the problem of constant-round concurrent zero-knowledge so difficult to solve, Theorem 2 rules out several constant-round protocols which may have been previously considered as candidates. These include 5-round zero-knowledge proofs for $\mathcal{NP}$ [10], as well as 6-round perfect zero-knowledge *arguments* for $\mathcal{NP}$ [3].

## 1.5 Techniques

The proof of Theorem 2 builds on the works of Goldreich and Krawczyk [11] and Kilian, Petrank and Rackoff [18]. It utilizes a fixed scheduling of the concurrent executions. This scheduling is defined recursively and is more sophisticated than the one proposed by [6] and used by [18]. It also exploits a special property of the first message sent by the verifier.

Note that since the scheduling considered here is fixed (rather than dynamic), both Theorems 1 and 2 are actually stronger than stated. Furthermore, our argument refers to verifier strategies that never refuse to answer the prover's queries. Simulating a concurrent interaction in which the verifier may occasionally refuse to answer (depending on its coin tosses and on the history of the current and/or other conversations) seems even more challenging than the simulation task which is treated in this work. Thus, it is conceivable that one may use the extra power of the adversary verifier to prove stronger lower bounds.

### 1.6 Organization of the Paper

Theorem 2 is proved in Section 2. We then demonstrate (in Section 3) how to modify the proof so it will work for the 9-round version of the Richardson-Kilian protocol (i.e., with $k = 2$). We conclude with Section 4 by discussing additional issues and recent work.

## 2 Proof of Theorem 2

In this section we prove that in the context of concurrent zero-knowledge, at least eight rounds of interaction are essential for black-box simulation of non-trivial proof systems (i.e., systems for languages that are not in $\mathcal{BPP}$). We note that in all known protocols, the zero-knowledge feature is demonstrated via a black-box simulator, and that it is hard to conceive of an alternative (for demonstrating zero-knowledge).[3]

*Definitions:* We use the standard definitions of interactive proofs [14] and arguments (a.k.a computationally-sound proofs) [2], black-box simulation (allowing non-uniform, deterministic verifier strategies, cf. [11, 18]) and concurrent zero-knowledge (cf. [20, 18]). Furthermore, since we consider a fixed scheduling of sessions, there is no need to use formalism for specifying to which session the next message of the verifier belongs. Finally, by (computationally-sound) interactive proof systems we mean systems in which the soundness error is negligible.[4]

*Preliminary conventions:* We consider protocols in which 8 messages are exchanged subject to the following conventions. The first message is an initiation message by the prover[5], denoted $\mathtt{p}_1$, which is answered by the verifier's first message denoted $\mathtt{v}_1$. The following prover and verifier messages are denoted $\mathtt{p}_2, \mathtt{v}_2, ..., \mathtt{p}_4, \mathtt{v}_4$, where the last message (i.e., $\mathtt{v}_4$) is a single bit indicating whether the verifier has accepted the input (and will not be counted as an actual message). Clearly, any 7-round protocol can be modified to fit this form. Next, we consider black-box simulators which are restricted in several ways (but claim

[3] The interesting work of Hada and Tanaka [15] is supposedly an exception; but not really: They show that such a non-black-box simulation can be conducted if one makes an assumption of a similar nature (i.e., that for every machine which does X there exists a machine which does X along with Y). In contrast, starting from a more standard assumption (such as "it is infeasible to do X"), it is hard to conceive how one may use non-black-box simulators in places where black-box ones fail.

[4] We do not know whether this condition can be relaxed. Whereas we may consider polynomially-many parallel interactions of a proof system in order to decrease soundness error in interactive proofs (as such may occur anyhow in the concurrent model), this is not necessarily sufficient in order to decrease the soundness error in the case of arguments (cf. [1]).

[5] Being in control of the schedule, it would be more natural to let the verifier initiate each session. However, since the schedule is fixed, we choose to simplify the exposition by letting the prover send the first message of the session.

that each of these restrictions can be easily satisfied): Firstly, we allow only simulators running in *strict* polynomial-time, but allow them to produce output that deviates from the actual execution by at most a gap of 1/6 (rather than requiring the deviation to be negligible).[6] (The latter relaxation enables a simple transformation of any expected polynomial-time simulator into a simulator running in strict polynomial-time.) Secondly, we assume, without loss of generality that the simulator never repeats the same query. As usual (cf. [11]), the queries of the simulator are prefixes of possible execution transcripts (in the concurrent setting[7]). Such a prefix is a sequence of alternating prover and verifier messages (which may belong to different sessions as determined by the fixed schedule). Thirdly, we assume that before making a query $\overline{q} = (a_1, b_1, ..., a_t, b_t, a_{t+1})$, where the $a$'s are prover messages, the simulator makes queries to all relevant prefixes (i.e., $(a_1, b_1, ..., a_{i-1}, b_{i-1}, a_i)$, for every $i \leq t$), and indeed has obtained the $b_i$'s as answers. Lastly, we assume that before producing output $(a_1, b_1, ..., a_T, b_T)$, the simulator makes the query $(a_1, b_1, ..., a_T)$.

## 2.1 The schedule, aversary verifiers and decision procedure

**The fixed schedule:** For each $x \in \{0,1\}^n$, we consider the following concurrent scheduling of $n$ sessions all run on common input $x$. The scheduling is defined recursively, where the scheduling of $m$ sessions (denoted $\mathcal{R}_m$) proceeds in 3 phases:

**First phase:** Each of the first $m/\log m$ sessions exchanges three messages (i.e., $\mathtt{p}_1, \mathtt{v}_1, \mathtt{p}_2$), this is followed by a recursive application of the scheduling on the next $m/\log m$ sessions.

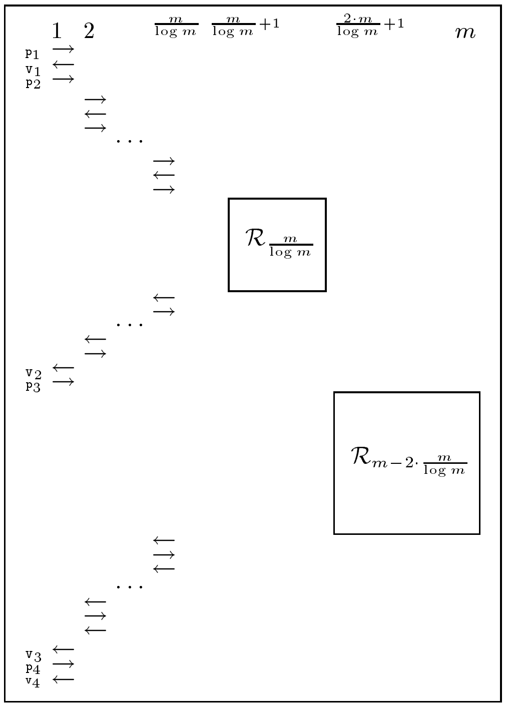
**Second phase:** Each of the first $m/\log m$ sessions exchanges two additional messages (i.e., $\mathtt{v}_2, \mathtt{p}_3$), this is followed by a recursive application of the scheduling on the last $m - 2 \cdot \frac{m}{\log m}$ sessions.

**Third phase:** Each of the first $m/\log m$ sessions exchanges the remaining messages (i.e.,$\mathtt{v}_3$,$\mathtt{p}_4$,$\mathtt{v}_4$).

The schedule is depicted in Figure 1. We stress that the verifier typically postpones its answer (i.e., $\mathtt{v}_j^{(i)}$) to the last prover's message (i.e., $\mathtt{p}_j^{(i)}$) till after a recursive sub-schedule is executed, and that it is crucial that in the first phase each session will finish exchanging its messages before the next sessions begins (whereas the order in which the messages are exchanged in the second and third phases is immaterial).

[6] We refer to the deviation gap, as viewed by any polynomial-time distinguisher. Such a distinguisher is required to decide whether its input consists of a conversation corresponding to real ececutions of the protocol, or rather to a transcript that was produced by the simulator. The computational deviation consists of the fraction of inputs which are accpted by the distinguisher in one case but rejected in the other.

[7] Indeed, for sake of clarity, we adopt a redundant representation. Alternatively, one may consider the subsequence of all prover's messages appearing in such transcripts.



1
2
$\frac{m}{\log m}$
$\frac{m}{\log m}+1$
$\frac{2\cdot m}{\log m}+1$
$m$
p1
v1
p2
v2
p3
v3
p4
v4
$\mathcal{R}_{\frac{m}{\log m}}$
$\mathcal{R}_{m-2\cdot\frac{m}{\log m}}$

**Fig. 1.** The fixed schedule – recursive structure for $m$ sessions.

**Definition 3** (identifiers of next message): *The fixed schedule defines a mapping from partial execution transcripts ending with a prover message to the* identifiers of the next verifier message*; that is, the session and round number to which the next verifier message belongs.* (Recall that such partial execution transcripts correspond to queries of a black-box simulator and so the mapping defines the identifier of the answer:) *For such a query $\overline{q} = (a_1, b_1, ..., a_t, b_t, a_{t+1})$, we let $\pi_{\text{sn}}(\overline{q}) \in \{1, ..., n\}$ denote the session to which the next verifier message belongs, and by $\pi_{\text{msg}}(\overline{q}) \in \{1, ..., 4\}$ its index within the verifier's messages in this session.*

**Definition 4** (initiation-prefix): *The* initiation-prefix $\overline{ip}$ of a query $\overline{q}$ *is the prefix of $\overline{q}$ ending with the prover's initiation message of session $\pi_{\text{sn}}(\overline{q})$. More formally, $\overline{ip} = a_1, b_1, ..., a_\ell, b_\ell, a_{\ell+1}$, is the initiation-prefix of $\overline{q} = (a_1, b_1, ..., a_t, b_t, a_{t+1})$ if $a_{\ell+1}$ is of the form $\mathsf{p}_1^{(i)}$ for $i = \pi_{\text{sn}}(\overline{q})$.* (Note that $\pi_{\text{msg}}(\overline{q})$ may be any index in $\{1, ..., 4\}$, and that $a_{t+1}$ need not belong to session $i$.)

**Definition 5** (prover-sequence): *The* prover-sequence *of a query $\overline{q}$ is the sequence of all prover's messages in session $\pi_{\text{sn}}(\overline{q})$ that appear in the query $\overline{q}$. The length of such a sequence is $\pi_{\text{msg}}(\overline{q}) \in \{1, \ldots, 4\}$. In case the length of the prover-sequence equals 4, both query $\overline{q}$ and its prover-sequence are said to be* terminating (*otherwise, they are called* non-terminating). *The prover-sequence is said to cor-*

*respond to the initiation-prefix $\overline{ip}$ of the query $\overline{q}$.* (Note that all queries having the same initiation-prefix agree on the first element of their prover-sequence, since this message is part of the initiation-prefix.)

We consider what happens when a black-box simulator (for the above schedule) is given oracle access to a verifier strategy $V_h$ defined as follows (depending on a hash function $h$ and the input $x$).

**The verifier strategy $V_h$:** On query $\overline{q} = (a_1, b_1, ..., a_t, b_t, a_{t+1})$, where the $a$'s are prover messages (and $x$ is implicit in $V_h$), the verifier answers as follows:

1. First, $V_h$ checks if the execution transcript given by the query is legal (i.e., consistent with $V_h$'s prior answers), and answers with an error message if the query is not legal. (In fact this is not necessary since by our convention the simulator only makes legal queries. From this point on we ignore this case.)
2. More importantly, $V_h$ checks whether the query contains the transcript of a session in which the last verifier message indicates rejecting the input. In case such a session exists, $V_h$ refuses to answer (i.e., answers with some special "refuse" symbol).
3. Next, $V_h$ determines the initiation-prefix, denoted $a_1, b_1, ..., a_\ell, b_\ell, a_{\ell+1}$, of query $\overline{q}$. It also determines $i = \pi_{\rm sn}(\overline{q})$, $j = \pi_{\rm msg}(\overline{q})$, and the prover-sequence of query $\overline{q}$, denoted $\mathtt{p}_1^{(i)}, ..., \mathtt{p}_j^{(i)}$.
4. Finally, $V_h$ determines $r_i = h(a_1, b_1, ..., a_\ell, b_\ell, a_{\ell+1})$ (as coins to be used by $V$), and answers with the message $V(x, r_i; \mathtt{p}_1^{(i)}, ..., \mathtt{p}_j^{(i)})$ that would have been sent by the honest verifier on common input $x$, random-pad $r_i$, and prover's messages $\mathtt{p}_1^{(i)}, ..., \mathtt{p}_j^{(i)}$.

Assuming towards the contradiction that a black-box simulator, denoted $S$, contradicting Theorem 2 exists, we now descibe a probabilistic polynomial-time decision procedure for $L$, based on $S$. Recall that we may assume that $S$ runs in strict polynomial time: we denote such time bound by $t_S(\cdot)$. On input $x \in L \cap \{0,1\}^n$ and oracle access to any (probabilistic polynomial-time) $V^*$, the simulator $S$ must output transcipts with distribution having computational deviation of at most $1/6$ from the distribution of transcripts in the actual concurrent executions of $V^*$ with $P$.

*A slight modification of the simulator:* Before presenting the procedure, we slightly modify the simulator so that it never makes a query that is refused by a verifier $V_h$. Note that this condition can be easily checked by the simulator, and that the modification does not effect the simulator's output. From this point on, when we talk of the simulator (which we continue to denote by $S$) we mean the modified one.

**Decision procedure for $L$:** On input $x \in \{0,1\}^n$, proceed as follows:

1. Uniformly select a function $h$ out of a small family of $t_S(n)$-wise independent hash functions mapping poly$(n)$-bit long sequences to $\rho_V(n)$-bit sequences, where $\rho_V(n)$ is the number of random bits used by $V$ on an input $x \in \{0,1\}^n$.
2. Invoke $S$ on input $x$ providing it black-box access to $V_h$ (as defined above). That is, the procedure emulates the execution of the oracle machine $S$ on input $x$ along with emulating the answers of $V_h$.
3. Accept if and only if all sessions in the transcript output by $S$ are accepting.

By our hypothesis, the above procedure runs in probabilistic polynomial-time. We next analyze its performance.

**Lemma 6** (performance on YES-instances): *For all but finitely many $x \in L$, the above procedure acccepts $x$ with probability at least* $2/3$.

**Proof Sketch:** The key observation is that for uniformly selected $h$, the behavior of $V_h$ in actual (concurrent) interactions with $P$ is identical to the behavior of $V$ in such interactions. The reason is that, in such actual interactions, a randomly selected $h$ determines uniformly and independently distributed random-pads for all $n$ sessions. Since with high probability (say at least $5/6$), $V$ accepts in all $n$ concurrent sessions, the same must be true for $V_h$, when $h$ is uniformly selected. Since the simulation deviation of $S$ is at most $1/6$, it follows that for every $h$ the probability that $S^{V_h}(x)$ is a transcript in which all sessions accept is lower bounded by $p_h - 1/6$, where $p_h$ denotes the probability that $V_h$ accepts $x$ (in all sessions) when interacting with $P$. Taking expectation over all possible $h$'s, the lemma follows. ■

**Lemma 7** (performance on NO-instances): *For all but finitely many $x \notin L$, the above procedure rejects $x$ with probability at least* $2/3$.

We can actually prove that for every polynomial $p$ and all but finitely many $x \notin L$, the above procedure accepts $x$ with probability at most $1/p(|x|)$. Assuming towards the contradiction that this is not the case, we will construct a (probabilistic polynomial-time) strategy for a cheating prover that fools the honest verifier $V$ with success probability at least $1/\text{poly}(n)$ (in contradiction to the computational-soundness of the proof system). Loosely speaking, the argument capitalizes on the fact that rewinding of a session requires the simulator to work on a new simulation sub-problem (one level down in the recursive construction). New work is required since each different message for the rewinded session forms an unrelated instance of the simulation sub-problem (by virtue of definition of $V_h$). The schedule causes work involved in such rewinding to accumulate to too much, and so it must be the case that the simulator does not rewind some (full instance of some) session. In this case the cheating prover may use such a session in order to fool the verifier.

## 2.2 Proof of Lemma 7 (performance on NO-instances)

Let us fix an $x \in \{0,1\}^n \setminus L$ as above.[8] Define by $\mathtt{AC} = \mathtt{AC}_x$ the set of pairs $(\sigma, h)$ so that on input $x$, coins $\sigma$ and oracle access to $V_h$, the simulator outputs a transcript, denoted $S_\sigma^{V_h}(x)$, in which all $n$ sessions accept. Recall that our contradiction assumption is that $\Pr_{\sigma,h}[(\sigma,h) \in \mathtt{AC}] > 1/p(n)$, for some fixed polynomial $p(\cdot)$.

**The cheating prover:** The cheating prover starts by uniformly selecting a pair $(\sigma, h)$ and hoping that $(\sigma, h)$ is in $\mathtt{AC}$. It next selects uniformly two elements $\xi$ and $\zeta$ in $\{1, ..., q_S(n)\}$, where $q_S(n) < t_S(n)$ is a bound on the number of queries made by $S$ on input $x \in \{0,1\}^n$. The prover next emulates an execution of $S_\sigma^{V_{h'}}(x)$ (where $h'$, which is essentially equivalent to $h$, will be defined below), while interacting with the honest verifier $V$. The prover handles the simulator's queries as well as the communication with the verifier as follows: Suppose that the simulator makes query $\overline{q} = (a_1, b_1, ..., a_t, b_t, a_{t+1})$, where the $a$'s are prover messages.

1. Operating as $V_h$, the cheating prover first determines the initiation-prefix, $\overline{ip} = a_1, b_1, ..., a_\ell, b_\ell, a_{\ell+1}$, corresponding to the current query $\overline{q}$. (Note that by our convention and the modification of the simulator there is no need to perform Steps 1 and 2 of $V_h$.)
2. If $\overline{ip}$ is the $\xi^{\text{th}}$ distinct initiation-prefix resulting from the simulator's queries so far then the cheating prover operates as follows:
   (a) The cheating prover determines $i = \pi_{\text{sn}}(\overline{q})$, $j = \pi_{\text{msg}}(\overline{q})$, and the prover-sequence of $\overline{q}$, denoted $\mathtt{p}_1^{(i)}, ..., \mathtt{p}_j^{(i)}$ (as done by $V_h$ in Step 3).
   (b) If the query $\overline{q}$ is non-terminating (i.e., $j \le 3$), and the cheating prover has only sent $j-1$ messages to the actual verifier then it forwards $\mathtt{p}_j^{(i)}$ to the verifier, and feeds the simulator with the verifier's response (i.e., which is of the form $\mathtt{v}_j^{(i)}$).[9]
   (c) If the query $\overline{q}$ is non-terminating (i.e., $j \le 3$), and the cheating prover has already sent $j$ messages to the actual verifier, the prover retrieves the $j^{\text{th}}$ message it has received and feeds it to the simulator.[10]

[8] In a formal proof we need to consider infinitely many such $x$'s.

[9] We comment that by our conventions regarding the simulator, it cannot be the case that the cheating prover has sent less than $j-1$ messages to the actual verifier: The prefixes of the current query dictate $j-1$ such messages.

[10] We comment that the cheating prover may fail to conduct Step 2c. This will happen whenever the simulator makes two queries with the same initiation-prefix and the same number of prover messages in the corresponding session, but with a different sequence of such messages. Whereas this will never happen when $j = 1$ (as once the initiation-prefix is fixed then so is the value of $\mathtt{p}_1^{(i)}$), it may very well be the case that for $j \in \{2,3\}$ a previous query regarding initiation-prefix $\overline{ip}$ had a different $\mathtt{p}_j^{(i)}$ message. In such a case the cheating prover will indeed fail. The punchline of the analysis is that with noticeable probability this will not happen.

(d) Whenever the query $\overline{q}$ is terminating (i.e., $j = 4$), the cheating prover operates as follows:
   i. As long as the $\zeta^{\text{th}}$ terminating query corresponding to the above initiation-prefix has not been made, the cheating prover feeds the simulator with $\mathtt{v}_4^{(i)} = 0$ (i.e., session rejected).
   ii. Otherwise, the cheating prover operates as in Step 2b (i.e., it forwards $\mathtt{p}_4^{(i)}$ to the verifier, and feeds the simulator with the verifier's response – some $\mathtt{v}_4^{(i)}$ message).[11]

3. If $\overline{ip}$ is NOT the $\xi^{\text{th}}$ distinct initiation-prefix resulting from the queries so far then the prover emulates $V_h$ in the obvious manner (i.e., as in Step 4 of $V_h$): It first determines $r_i = h(a_1, b_1, ..., a_\ell, b_\ell, a_{\ell+1})$, and then answers with $V(x, r_i; \mathtt{p}_1^{(i)}, ..., \mathtt{p}_j^{(i)})$, where all notations are as above.

*Defining $h'$ (mentioned above):* Let $(\sigma, h)$ and $\xi$ be the initial choices made by the cheating prover, and suppose that the honest verifier uses coins $r$. Then, the function $h'$ is defined to be uniformly distributed among the functions $h''$ which satisfy the following conditions: The value of $h''$ on the $\xi^{\text{th}}$ initiation-prefix equals $r$, whereas for every $\xi' \neq \xi$, the value of $h''$ on the ${\xi'}^{\text{th}}$ initiation-prefix equals the value of $h$ on this prefix. (Here we use the hypothesis that the functions are selected in a family of $t_S(n)$-wise independent hash functions. We note that replacing $h$ by $h'$ does not effect Step 3 of the cheating prover, and that the prover does not know $h'$.)

The probability that the cheating prover makes the honest verifier accept is lower bounded by the probability that both $(\sigma, h') \in \mathtt{AC}$ and the messages forwarded by the cheating prover in Step 2 are consistent with an accepting conversation with $V_{h'}$. For the latter event to occur, it is necessary that the $\xi^{\text{th}}$ distinct initiation-prefix will be useful (in the sense hinted above and defined now). It is also necessary that $\zeta$ was "successfully" chosen (i.e., the $\zeta^{\text{th}}$ terminating query which corresponds to the $\xi^{\text{th}}$ initiation-prefix is accepted by $V_{h'}$).

**Definition 8** (accepting query): *A terminating query $\overline{q} = (a_1, b_1, ..., a_t, b_t, a_{t+1})$ (i.e., for which $\pi_{\text{msg}}(\overline{q}) = 4$) is said to be* accepting *if $V_{h'}(a_1, b_1, ..., a_t, b_t, a_{t+1})$ equals 1 (i.e., session $\pi_{\text{sn}}(\overline{q})$ is accepted by $V_{h'}$).*

**Definition 9** (useful initiation-prefix): *A specific initiation-prefix $\overline{ip}$ in an execution of $S_\sigma^{V_{h'}}(x)$ is called* useful *if the following conditions hold:*

1. *During its execution, $S_\sigma^{V_{h'}}(x)$ made at least one accepting query which corresponds to the initiation-prefix $\overline{ip}$.*

[11] We note that once the cheating prover arrives to this point, then it either succeds in the cheating task or completely fails (depending on the verifier's response). As a consequence, it is not essential to define the cheating prover's actions from this point on (as in both cases the algorithm will be terminated).

2. *As long as no accepting query corresponding to the initiation-prefix $\overline{ip}$ was made during the execution of $S_\sigma^{V_{h'}}(x)$, the number of (non-terminating) different prover-sequences that correspond to $\overline{ip}$ is at most 3, and these prover-sequences are prefixes of one another.*[12]

*Otherwise, the prefix is called* unuseful.

**The success probability:** Define a Boolean indicator $\chi(\sigma, h', \xi)$ to be true if and only if the $\xi^{\text{th}}$ distinct initiation-prefix in an execution of $S_\sigma^{V_{h'}}(x)$ is useful. Define an additional Boolean indicator $\psi(\sigma, h', \xi, \zeta)$ to be true if and only if the $\zeta^{\text{th}}$ terminating query among all terminating queries that correspond to the $\xi^{\text{th}}$ distinct initiation-prefix (in an execution of $S_\sigma^{V_{h'}}(x)$) is the first one to be accepting. It follows that if the cheating prover happens to select $(\sigma, h, \xi, \zeta)$ so that both $\chi(\sigma, h', \xi)$ and $\psi(\sigma, h', \xi, \zeta)$ hold then it convinces $V(x, r)$; the first reason being that the $\zeta^{th}$ such query is answered by an accept message[13], and the second reason being that the emulation does not get into trouble (in Steps 2c and 2d). To see this, notice that all first $(\zeta - 1)$ queries having the $\xi^{\text{th}}$ distinct initiation-prefix satisfy exactly one of the following conditions:

1. They have non-terminating prover-sequences that are prefixes of one another (which implies that the cheating prover never has to forward such queries to the verifier twice).
2. They have terminating prover-sequences which should be rejected (recall that as long as the $\zeta^{\text{th}}$ terminating query has not been asked by $S_\sigma^{V_{h'}}(x)$, the cheating prover automatically rejects any terminating query).

Thus, the probability that when selecting $(\sigma, h, \xi, \zeta)$ the cheating prover convinces $V(x, r)$ is at least

$$\begin{aligned}
&\Pr[\psi(\sigma, h', \xi, \zeta) \;\&\; \chi(\sigma, h', \xi)] \\
&= \Pr[\psi(\sigma, h', \xi, \zeta) \mid \chi(\sigma, h', \xi)] \cdot \Pr[\chi(\sigma, h', \xi)] \\
&\geq \Pr[\psi(\sigma, h', \xi, \zeta) \mid \chi(\sigma, h', \xi)] \cdot \Pr[(\sigma, h') \in \texttt{AC} \;\&\; \chi(\sigma, h', \xi)] \qquad (1)
\end{aligned}$$

Note that if the $\xi^{\text{th}}$ distinct initiation-prefix is useful, and $\zeta$ is uniformly (and independently) selected in $\{1, ..., q_S(n)\}$, the probability that the $\zeta^{\text{th}}$ query corresponding to the $\xi^{\text{th}}$ distinct initiation–prefix is the first to be accepting is at least $1/q_S(n)$. Thus, Eq. (1) is lower bounded by

$$\frac{\Pr[(\sigma, h') \in \texttt{AC} \;\&\; \chi(\sigma, h', \xi)]}{q_S(n)} \qquad (2)$$

[12] In other words, we allow for many different terminating queries to occur (as long as they are not accepting). On the other hand, for $j \in \{1, 2, 3\}$ only a single query that has a prover sequence of length $j$ is allowed. This requirement will enable us to avoid situations in which the cheating prover will fail (as described in Footnote 10).

[13] We use the fact that $V(x, r)$ behaves exactly as $V_{h'}(x)$ behaves on queries for the $\xi^{\text{th}}$ distinct initiation-prefix.

Using the fact that, for every value of $\xi$ and $\sigma$, when $h$ and $r$ are uniformly selected the function $h'$ is uniformly distributed, we infer that $\xi$ is distributed independently of $(\sigma, h')$. Thus, Eq. (2) is lower bounded by

$$\Pr[(\sigma, h') \in \mathtt{AC}] \cdot \frac{\Pr[\exists i \text{ s.t. } \chi(\sigma, h', i) \,|\, (\sigma, h') \in \mathtt{AC}]}{q_S(n)^2} \tag{3}$$

Thus, Eq. (3) is noticeable (i.e., at least $1/\mathrm{poly}(n)$) provided that so is the value of $\Pr[\exists i \text{ s.t. } \chi(\sigma, h', i) \,|\, (\sigma, h') \in \mathtt{AC}]$. The rest of the proof is devoted to establishing the last hypothesis. In fact we prove a much stronger statement:

**Lemma 10** *For every $(\sigma, h') \in \mathtt{AC}$, the execution of $S_\sigma^{V_{h'}}(x)$ contains a useful initiation-prefix (that is, there exists an $i$ s.t. $\chi(\sigma, h', i)$ holds).*

## 2.3 Proof of Lemma 10 (existence of useful initiation prefixes)

The proof of Lemma 10 is by contradiction. We assume the existence of a pair $(\sigma, h') \in \mathtt{AC}$ so that all initiation-prefixes in the execution of $S_\sigma^{V_{h'}}(x)$ are unuseful and show that this implies that $S_\sigma^{V_{h'}}(x)$ made at least $n^{\Omega(\frac{\log n}{\log\log n})} \gg \mathrm{poly}(n)$ queries which contradicts the assumption that it runs in polynomial-time.

**The query–and–answer tree:** Throughout the rest of the proof, we fix an arbitrary $(\sigma, h') \in \mathtt{AC}$ so that all initiation-prefixes in the execution of $S_\sigma^{V_{h'}}(x)$ are unuseful, and study this execution. A key vehicle in this study is the notion of a query–and–answer tree introduced in [18]. This is a rooted tree in which vertices are labeled with verifier messages and edges are labeled by prover's messages. The root is labeled by the empty string, and it has outgoing edges corresponding to the possible prover's messages initializing the first session. In general, paths down the tree (i.e., from the root to some vertices) correspond to queries. The query associated with such a path is obtained by concatenating the labeling of the vertices and edges in the order traversed. We stress that each vertex in the tree corresponds to a query actually made by the simulator.

*Satisfied sub-path:* A sub-path from one node in the tree to some of its descendants is said to satisfy session $i$ if the sub-path contains edges (resp., vertices) for each of the messages sent by the prover (resp., verifier) in session $i$, and if the last such message (i.e., $\mathtt{v}_4^{(i)}$) indicates that the verifier accepts session $i$. A sub-path is called satisfied if it satisfies all sessions for which the first prover's message appears on the sub-path.

*Forking sub-tree:* For any $i$ and $j \in \{2, 3, 4\}$, we say that a sub-tree $(i, j)$-forks if it contains two sub-paths, $\overline{p}$ and $\overline{r}$, having the same initiation-prefix, so that

1. Sub-paths $\overline{p}$ and $\overline{r}$ differ in the edge representing the $j^{\text{th}}$ prover message for session $i$ (i.e., a $\mathtt{p}_j^{(i)}$ message).
2. Each of the sub-paths $\overline{p}$ and $\overline{r}$ reaches a vertex representing the $j^{\text{th}}$ verifier message (i.e., some $\mathtt{v}_j^{(i)}$).

In such a case, we may also say that the sub-tree $(i, j)$-forks on $\overline{p}$ (or on $\overline{r}$).

*Good sub-tree:* Consider an arbitrary sub-tree rooted at a vertex corresponding to the first message in some session so that this session is the first at some level of the recursive construction of the schedule. The full tree is indeed such a tree, but we will need to consider sub-trees which correspond to $m$ sessions in the recursive schedule construction. We call such a sub-tree $m$-good if it contains a sub-path satisfying all $m$ sessions for which the prover's first message appears in the sub-tree (all these first messages are in particular contained in the sub-path). Since $(\sigma, h') \in \mathtt{AC}$ it follows that the full tree contains a path from the root to a leaf representing an accepting transcript. The path from the root to this leaf thus satisfies all sessions (i.e., 1 through $n$) which implies that the full tree is $n$-good. The crux of the entire proof is given in the following lemma.

**Lemma 11** *Let $T$ be an $m$-good sub-tree, then at least one of the following holds:*

1. *$T$ contains at least two different $\left(m - 2 \cdot \frac{m}{\log m}\right)$-good sub-trees.*
2. *$T$ contains at least $\frac{m}{\log m}$ different $\left(\frac{m}{\log m}\right)$-good sub-trees.*

Denote by $W(m)$ the size of an $m$-good sub-tree (where $W(m)$ stands for the work actually performed by the simulator on $m$ concurrent sessions in our fixed scheduling). It follows (from Lemma 11) that any $m$-good sub-tree must satisfy

$$W(m) \geq \min\left\{\frac{m}{\log m} \cdot W\left(\frac{m}{\log m}\right), 2 \cdot W\left(m - 2 \cdot \frac{m}{\log m}\right)\right\} \qquad (4)$$

Since Eq. (4) solves to $n^{\Omega\left(\frac{\log n}{\log\log n}\right)}$ (proof omitted), and since every vertex in the query–and–answer tree corresponds to a query actually made by the simulator, then the assumption that the simulator runs in poly($n$)-time (and hence the tree is of poly($n$) size) is contradicted. Thus, Lemma 10 follows from Lemma 11.

## 2.4 Proof of Lemma 11 (the structure of good sub-trees)

Considering the $m$ sessions corresponding to an $m$-good sub-tree, we focus on the $m/\log m$ sessions dealt explicitly at this level of the recursive construction (i.e., the first $m/\log m$ sessions, which we denote by $\mathcal{F} \overset{\text{def}}{=} \{1, ..., m/\log m\}$).

**Claim 12** *Let $T$ be an $m$-good sub-tree. Then for any session $i \in \mathcal{F}$, there exists $j \in \{2, 3\}$ such that the sub-tree $(i, j)$-forks.*

**Proof:** Consider some $i \in \mathcal{F}$, and let $\overline{p}_i$ be the first sub-path reached during the execution of $S_\sigma^{V_{h'}}(x)$ which satisfies session $i$ (since the sub-tree is $m$-good such a sub-path must exist, and since $i \in \mathcal{F}$ every such sub-path must be contained in the sub-tree). Recall that by the contradiction assumption for the proof of Lemma 10, all initiation-prefixes in the execution of $S_\sigma^{V_{h'}}(x)$ are unuseful. In particular, the initiation-prefix corresponding to sub-path $\overline{p}_i$ is unuseful. Still, path $\overline{p}_i$ contains vertices for each prover message in session $i$ and contains an accepting message by the verifier. So the only thing which may prevent the above initiation-prefix from being useful is having two (non-terminating) queries with the very same initiation-prefix (non-terminating) prover-sequences of the same length. Say that these sequences first differ at their $j^{\text{th}}$ element, and note that $j \in \{2,3\}$ (as the prover-sequences are non-terminating and the first prover message, $\mathtt{p}_1^{(i)}$, is constant once the initiation-prefix is fixed). Also note that the two (non-terminating) queries were answered by the verifier (rather than refused), since the (modified) simulator avoids queries which will be refused. By associating a sub-path to each one of the above queries we obtain two different sub-paths (having the same initiation-prefix), that differ in some $\mathtt{p}_j^{(i)}$ edge and eventually reach a $\mathtt{v}_j^{(i)}$ vertex (for $j \in \{2,3\}$). The required $(i,j)$-forking follows. ■

**Claim 13** *If there exists a session $i \in \mathcal{F}$ such that the sub-tree $(i,3)$-forks, then the sub-tree contains two different $(m-2\cdot\frac{m}{\log m})$-good sub-trees.*

**Proof:** Let $i \in \mathcal{F}$ such that the sub-tree $(i,3)$-forks. That is, there exist two sub-paths, $\overline{p}_i$ and $\overline{r}_i$, that differ in the edge representing a $\mathtt{p}_3^{(i)}$ message, and that eventually reach some $\mathtt{v}_3^{(i)}$ vertex. In particular, paths $\overline{p}_i$ and $\overline{r}_i$ split from each other before the edge which corresponds to the $\mathtt{p}_3^{(i)}$ message occurs along these paths (as otherwise the $\mathtt{p}_3^{(i)}$ edge would have been identical in both paths). By nature of the fixed scheduling, the vertex in which the above splitting occurs precedes the first message of all (nested) sessions in the second recursive construction (that is, sessions $2\cdot\frac{m}{\log m}+1,\ldots,m$). It follows that both $\overline{p}_i$ and $\overline{r}_i$ contain the first and last messages of each of these (nested) sessions (as they both reach a $\mathtt{v}_3^{(i)}$ vertex). Therefore, by definition of $V_h$, all these sessions must be satisfied by both these paths (or else $V_h$ would have not answered with a $\mathtt{v}_3^{(i)}$ message but rather with a "refuse" symbol). Consider now the corresponding sub-paths of $\overline{p}_i$ and $\overline{r}_i$ which begin at edge $\mathtt{p}_1^{(k)}$ where $k = 2\cdot\frac{m}{\log m}+1$ (i.e., $\mathtt{p}_1^{(k)}$ is the edge which represents the first message of the first session in the second recursive construction). Each of these new sub-paths is contained in a disjoint sub-tree corresponding to the recursive construction, and satisfies all of its $(m-2\cdot\frac{m}{\log m})$ sessions. It follows that the (original) sub-tree contains two different $(m-2\cdot\frac{m}{\log m})$-good sub-trees and the claim follows. ■

**Claim 14** *If for every session $i \in \mathcal{F}$ the sub-tree $(i,2)$-forks, then the sub-tree contains at least $|\mathcal{F}| = \frac{m}{\log m}$ different $(\frac{m}{\log m})$-good sub-trees.*

In the proof of Claim 14 we use a special property of $(i,2)$-forking: The only location in which the splitting of path $\overline{r}_i$ from path $\overline{p}_i$ may occur, is a vertex which represents a $\mathtt{v}_1^{(i)}$ message. Any splitting which has occured at a vertex which precedes the $\mathtt{v}_1^{(i)}$ vertex would have caused the initiation-prefixes of (session $i$ along) paths $\overline{p}_i$ and $\overline{r}_i$ to be different (by virtue of the definition of $V_h$, and since all vertices preceding $\mathtt{v}_1^{(i)}$ are part of the initiation-prefix of session $i$).

**Proof:** Since for all sessions $i \in \mathcal{F}$ the sub-tree $(i, 2)$-forks, then for every such $i$ there exist two sub-paths, $\overline{p}_i$ and $\overline{r}_i$, that split from each other in a $\mathtt{v}_1^{(i)}$ vertex and that eventually reach some $\mathtt{v}_2^{(i)}$ vertex. Similarly to the proof of Claim 13, we can claim that each one of the above paths contains a "special" sub-path (denoted $\overline{\overline{p}}_i$ and $\overline{\overline{r}}_i$ respectively), that starts at a $\mathtt{v}_1^{(i)}$ vertex, ends at a $\mathtt{v}_2^{(i)}$ vertex, and satisfies all $\frac{m}{\log m}$ sessions in the first recursive construction (that is, sessions $\frac{m}{\log m}+1,\ldots,2\cdot\frac{m}{\log m}$). Note that paths $\overline{\overline{p}}_i$ and $\overline{\overline{r}}_i$ are completely disjoint. Let $i_1, i_2$ be two different sesions in $\mathcal{F}$ (without loss of generality $i_1 < i_2$), and let $\overline{\overline{p}}_{i_1}, \overline{\overline{r}}_{i_1}, \overline{\overline{p}}_{i_2}, \overline{\overline{r}}_{i_2}$ be their corresponding "special" sub-paths. The key point is that for every $i_1, i_2$ as above, it cannot be the case that both "special" sub-paths corresponding to session $i_2$ are contained in the sub-paths corresponding to session $i_1$ (to justify this, we use the fact that $\overline{\overline{p}}_{i_2}$ and $\overline{\overline{r}}_{i_2}$ split from each other in a $\mathtt{v}_1^{(i_2)}$ vertex and that for every $i \in \{i_1, i_2\}$, paths $\overline{\overline{p}}_i$ and $\overline{\overline{r}}_i$ are disjoint).

This enables us to associate a distinct $(\frac{m}{\log m})$-good sub-tree to every $i \in \mathcal{F}$ (i.e., which either corresponds to path $\overline{\overline{p}}_i$, or to path $\overline{\overline{r}}_i$). Which in particular means that the tree contains at least $|\mathcal{F}|$ different $(\frac{m}{\log m})$-good sub-trees. ■

We are finally ready to analyze the structure of the sub-tree $T$. Since for every $i \in \mathcal{F}$ there must exist $j \in \{2, 3\}$ such that the sub-tree $(i, j)$-forks (Claim 12), then it must be the case that either $T$ contains two distinct $(m-2\cdot\frac{m}{\log m})$-good sub-trees (Claim 13), or $T$ contains at least $\frac{m}{\log m}$ distinct $(\frac{m}{\log m})$-good sub-trees (Claim 14). This completes the proof of Lemma 11 which in turn implies Lemmata 10 and 7. The proof of Theorem 2 is complete.

## 3 Extending the proof for the Richardson-Kilian protocol

Recall that the Richardson-Kilian protocol [20] consists of two stages. We will treat the first stage of the RK protocol (which consists of 6 rounds) as if it were the first 6 rounds of any 7-round protocol, and the second stage (which consists of a 3-round WI proof) as if it were the remaining 7th message. An important property which is satisfied by the RK protocol is that the coin tosses used by the verifier in the second stage are independent of the coins used by the verifier in the first stage. We can therefore define and take advantage of two (different) types of initiation-prefixes. A first-stage initiation prefix and a second-stage initiation prefix (which is well defined only given the first one). These initiation-prefixes will determine the coin tosses to be used by $V_h$ in each corresponding stage of the protocol (analogously to the proof of Theorem 2).

The cheating prover will pick a random index for each of the above types of initiation-prefixes (corresponding to $\xi$ and $\zeta$ in the proof of Theorem 2). The first index (i.e., $\xi$) is treated exactly as in the proof of Theorem 2, whereas the second index (i.e., $\zeta$) will determine which of the WI session corresponding to the second-phase initiation-prefix (and which also correspond to the very same $\xi^{\text{th}}$ first-phase initiation-prefix) will be actually executed between the cheating prover and the verifier. As long as the $\zeta^{\text{th}}$ second-stage initiation prefix will not be encountered, the cheating prover will be able to impersonate $V_h$ while always deciding correctly whether to reject or to accept the corresponding "dummy" WI session (as the second-stage initiation-prefix completely determines the coins to be used by $V_h$ in the second stage of the protocol). As in the proof of Theorem 2, the probability that the $\zeta^{\text{th}}$ second-stage initiation prefix (that correponds to the $\xi^{\text{th}}$ first-phase initiation-prefix) will make the verifier accept is non-negligible. The existence of a useful pair of initiation-prefixes (i.e., $\xi$ and $\zeta$) is proved essentially in the same way as in the proof of Theorem 2.

## 4 Concluding Remarks

*Summary:* In this work we have pointed out the impossibility of black-box simulation of non-trivial 7-round protocols in the concurrent setting. The result which is proved is actually stronger than stated. Not only because we consider a fixed scheduling in which the adversarial verifier never refuses to answer (and thus should have been easier to simulate, as argued in Section 1.5), but also because we are considering simulators which may have as much as a constant deviation from actual executions of the protocol (rather than negligible deviation, as typically required in the definition of Zero-Knowledge).

*On the applicability of the RK protocol:* We note that the above discussion does not imply that the $k = 2$ version of the RK protocol is completely useless. As noted by Richardson and Kilian, for security parameter $n$, the simulation of $m = \text{poly}(n)$ concurrent sessions may be performed in quasi-polynomial time (recall that the simulation work required for $m$ sessions is $m^{\Theta(\log_k m)}$). Thus, the advantage that a polynomial-time adversary verifier may gain from executing $m$ concurrent sessions is not significant. As a matter of fact, if one is willing to settle for less than polynomially-many (in $n$) concurrent sessions, then the RK protocol may be secure in an even stronger sense. Specifically, as long as the number of sessions is $m = 2^{O(\sqrt{\log_2 n})}$, then simulation of the RK protocol can be performed in polynomial-time even if $k = 2$. This is considerably larger than the logarithmic number of concurrent sessions enabled by straightforward simulation of previously known constant-round protocols.

*Improved simulation of the RK protocol:* Recently, Kilian and Petrank [17] have proved that the Richardson-Kilian protocol will remain concurrent zero-knowledge even if $k = O(g(n) \cdot \log^2 n)$, where $g(\cdot)$ is any non-constant function (e.g., $g(n) = \log n$). Thus, the huge gap between the known upper and lower bounds on the number of rounds required by concurrent zero-knowledge has been considerably narrowed.

## 5 Acknowledgements

I would like to thank Oded Goldreich for introducing me to the subject, and for the generous assistance in writing this manuscript. I would also like to thank Yoav Rodeh for significant discussions, and Yehuda Lindell (as well as the anonymous referees) for commenting on the earlier version of this paper.

## References

1. M. Bellare, R. Impagliazzo and M. Naor. Does Parallel Repetition Lower the Error in Computationally Sound Protocols? In *38th FOCS*, pages 374–383, 1997.
2. G. Brassard, D. Chaum and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *JCSS*, Vol. 37, No. 2, pages 156–189, 1988.
3. G. Brassard, C. Crépeau and M. Yung. Constant-Round Perfect Zero-Knowledge Computationally Convincing Protocols. *Theoret. Comput. Sci.* , Vol. 84, pp. 23-52, 1991.
4. R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable Zero-Knowledge. In *32nd STOC*, 2000.
5. I. Damgard. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. In *EuroCrypt2000*.
6. C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In *30th STOC*, pages 409–418, 1998.
7. C. Dwork, and A. Sahai. Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints. In *Crypto98*, Springer LNCS 1462 , pages 442–457, 1998.
8. U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd STOC*, pages 416–426, 1990.
9. O. Goldreich. Foundations of Cryptography – Fragments of a Book. Available from `http://theory.lcs.mit.edu/~oded/frag.html`.
10. O. Goldreich and A. Kahan. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Jour. of Cryptology*, Vol. 9, No. 2, pages 167–189, 1996.
11. O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM J. Computing*, Vol. 25, No. 1, pages 169–192, 1996.
12. O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *JACM*, Vol. 38, No. 1, pp. 691–729, 1991.
13. O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Jour. of Cryptology*, Vol. 7, No. 1, pages 1–32, 1994.
14. S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, Vol. 18, No. 1, pp. 186–208, 1989.
15. S. Hada and T. Tanaka. On the Existence of 3-Round Zero-Knowledge Protocols. In *Crypto98*, Springer LNCS 1462, pages 408–423, 1998.
16. J. Hastad, R. Impagliazzo, L.A. Levin and M. Luby. Construction of Pseudorandom Generator from any One-Way Function. *SIAM Jour. on Computing*, Vol. 28 (4), pages 1364–1396, 1999.
17. J. Kilian and E. Petrank. Concurrent Zero-Knowledge in Poly-logarithmic Rounds. In Cryptology ePrint Archive: Report 2000/013. Available from `http://eprint.iacr.org/2000/013`
18. J. Kilian, E. Petrank, and C. Rackoff. Lower Bounds for Zero-Knowledge on the Internet. In *39th FOCS*, pages 484–492, 1998.
19. M. Naor. Bit Commitment using Pseudorandomness. *Jour. of Cryptology*, Vol. 4, pages 151–158, 1991.
20. R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *EuroCrypt99*, Springer LNCS 1592, pages 415–431, 1999.