The Round-Complexity of Black-Box Zero-Knowledge: A Combinatorial Characterization

Daniele Micciancio and Scott Yilek

Dept. of Computer Science & Engineering, University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093-0404, USA
{daniele,syilek}@cs.ucsd.edu
http://www-cse.ucsd.edu/users/{daniele,syilek}

Abstract. The round-complexity of black-box zero-knowledge has for years been a topic of much interest. Results in this area generally focus on either proving lower bounds in various settings (e.g., Canetti, Kilian, Petrank, and Rosen [3] prove concurrent zero-knowledge ($c\mathcal{ZK}$) requires $\Omega(\log n/\log\log n)$ rounds and Barak and Lindell [2] show no constant-round single-session protocol can be zero-knowledge with strict poly-time simulators), or giving upper bounds (e.g., Prabhakaran, Rosen, and Sahai [15] give a $c\mathcal{ZK}$ protocol with $\omega(\log n)$ rounds). In this paper we show that though proving upper bounds seems to be quite different from demonstrating lower bounds, underlying both tasks there is a single, simple combinatorial game between two players: a rewinder and a scheduler. We give two theorems relating the success of rewinders in the game to both upper and lower bounds for black-box zero-knowledge in various settings (sequential composition, concurrent composition, etc). Our game and theorems unify the previous results in the area, simplify the task of proving upper and lower bounds, and should be useful in showing future results in the area.

1 Introduction

Zero-Knowledge proofs, introduced by Goldwasser, Micali, and Rackoff [9], have been the focus of much research since their invention, both in cryptography and complexity theory. Interest in these proofs sparks from the fact that they provide both a useful tool for the construction of higher level security protocols, and a test-bed to explore new security issues, e.g., security under various forms of protocol composition. As a consequence, numerous variants of zero-knowledge have been considered and studied, resulting in many general possibility and impossibility results.

Informally, a zero-knowledge proof system is a two party protocol between a prover P and a verifier V that allows P to prove some assertion (e.g., membership of a string x in an $\mathcal{NP}$-language L) to V without leaking *any* information other than the truth of the assertion. This is typically proven by exhibiting a *black-box* simulator, i.e., an efficient procedure S that, given oracle access to the

R. Canetti (Ed.): TCC 2008, LNCS 4948, pp. 535–552, 2008.

program of any (possibly misbehaving) verifier V^*, produces (without interacting with the prover) an output which is essentially identical to that of V^* when interacting with P. Non-black-box simulation methods are also possible [1], but most theoretical work, as well as essentially all protocols of practical interest, fall in the black-box model, making black-box simulation an interesting area of research on its own.

Much of the work on black-box zero-knowledge focuses on *round-complexity*, as interaction is at the same time essential (to achieve the zero-knowledge property) and expensive (from a practical performance point of view). In the last decade, many general upper and lower bounds on the round complexity of black-box zero-knowledge proof systems have been established [8,12,18,16,11,15].

Negative results (i.e., round complexity lower bounds) typically show that no non-trivial[1] language admits a black-box zero-knowledge proof system with less than a given number of rounds. For example, Barak and Lindell have shown that if the black-box simulator is constrained to run in *strict* polynomial time, then only trivial languages admit constant round zero-knowledge proofs [2]. Similarly, [3] has shown that only trivial languages admit black-box concurrent zero-knowledge proofs with $o(\log n/\log\log n)$ rounds. Such negative results are proved by exhibiting a carefully crafted hard-to-simulate verifier V^* such that any efficient black-box algorithm S that simulates the interaction between V^* and P can be transformed into an efficient decision procedure for the language L being proven.

Positive results (i.e., round complexity upper bounds) typically assert that under general cryptographic assumptions (e.g., the existence of commitment schemes) every language in $\mathcal{NP}$ admits a black-box zero-knowledge proof system with low round complexity. Such positive results are usually proved by giving an explicit proof system for a single $\mathcal{NP}$-complete problem (e.g., 3-coloring or graph hamiltonicity), and proof systems for all other $\mathcal{NP}$ languages follow by reduction. For example, Goldreich and Kahan [7] give a constant-round zero-knowledge proof for 3-coloring, with an expected polynomial-time black-box simulator.

It would appear that the tasks of proving lower and upper bounds for black-box zero-knowledge are quite different: the former needs to be completely general and hold for any language and associated proof system; the latter considers a specific ($\mathcal{NP}$-complete) language and provides an explicit prover and simulator strategy for that language.

1.1 Our Results

We describe a simple combinatorial game (parameterized by an integer r) that closely characterizes (up to a small constant additive term) the round complexity of black-box zero-knowledge, in the sense that (under standard cryptographic

[1] In the context of zero-knowledge proofs, "non-trivial" typically refers to languages that are not known to be decidable in probabilistic polynomial time, since any such language admits a trivial zero-knowledge proof system where the verifier checks the validity of the assertion on its own, without the help of the prover.

assumptions) any non-trivial language (in $\mathcal{NP} \setminus \mathcal{BPP}$) admits a black-box zero-knowledge proof system with $r + O(1)$ rounds *if and only if* the combinatorial game admits a solution. (See Sect. 3 for a formal statement of the results.)

The game is simple, yet general enough to study, in a unified way, black-box zero-knowledge in many important settings, including

- zero-knowledge with sequential composition, where only a single prover interacts with a single verifier at any time (no interleavings of sessions),
- concurrent zero-knowledge, where many colluding cheating verifiers can interact with independent provers while interleaving their actions in the most adversarially possible way,
- a form of parallel zero knowledge, where n provers send all their first messages in order, before moving to the next round, and so on,
- zero-knowledge under various forms of bounded concurrency, where different sessions can be interleaved, provided not too many sessions are active at the same time.

All of these settings are treated in a uniform way simply by parameterizing the combinatorial game with a set Γ of "forbidden patterns", i.e., sequences of interleavings that are guaranteed not to occur. For example, in the case of concurrent zero-knowledge, $\Gamma = \emptyset$ and all interleavings are allowed, while in sequential zero-knowledge Γ is the set of all sequences in which labels are interleaved (i.e., between the first and last labels in some session, there is a node with a different label). For simplicity of exposition, we focus on the case of *concurrent* zero-knowledge, where $\Gamma = \emptyset$ and can be omitted. The reader can easily check that all our results and proofs immediately extend to arbitrary Γ.

Our game (described below) is similar, though not identical, to games implicitly defined in previous papers providing upper and lower bounds on the round complexity of black-box concurrent zero-knowledge [3,15]. We remark that the combinatorial games implicitly used in previous proofs differed from each other, leaving a (perhaps small, but interesting) gap between upper and lower bounds. A technical contribution of our paper is to identify (and precisely define) a variant of the game that simultaneously yields both upper and lower bounds. Moreover, our characterization result holds in a variety of settings (basically, any kind of protocol composition that can be described by a set of forbidden patterns). This can be useful not only to unify and simplify many previous results for the sequential and concurrent composition setting (e.g., [2,3,7,15,17],) but also as a starting point to study the round complexity under other forms of composition (e.g., parallel composition, bounded concurrency)

1.2 The Game

The game (parameterized by three integers d, h and r) is played by two players, called the scheduler and the rewinder. The scheduler strategy is described by a labeled tree, where each internal node v has exactly d children $v_1, \ldots, v_d$, and all leaves are at level h. Each node v is represented by a sequence $\{1, \ldots, d\}^*$ in the standard way, where nodes at level l of the tree are represented by sequences of

length l, and the parent of a node is obtained by removing the last element in the sequence. Each edge $v \to v_m$ (from a node v to its mth child) carries the label m, and every internal node carries a label $\pi(v)$ which equals either the node itself $\pi(v) = v$, or a previous label $\pi(v) = \pi(w)$, where w is an ancestor of v.

For every node $v \in \{1, \ldots, d\}^*$ in the tree, let $\mu(v)$ be the subsequence of v corresponding to the edges immediately following a node w with label $\pi(w) = \pi(v)$. (See Fig. 2. The reader is referred to Sect. 3 for a formal definition.) In an execution of the game, the rewinder explores the labeled tree defined by the scheduler, learning the label $\pi(v)$ when a node v is visited. Starting from the root node, the rewinder selects at every step a child of some previously visited node, subject to the following restriction:

- At any point during the game, if $|\mu(v)| = r$, then the rewinder is allowed to select a child of v only if he has already visited some other node w with the same label $\pi(v) = \pi(w)$ such that $\mu(w)$ is not a prefix of $\mu(v)$.

The rewinder is allowed to perform random choices during the game, with the goal of reaching a random leaf of the tree, while visiting the smallest possible number of nodes. Formally, we measure the success of the rewinder in the game by the number of visited nodes (which should be small) and the distance from random of the distribution over leaves defined by an execution of the game.

While the above game can be conveniently studied in a purely combinatorial setting, in order to establish a link between the game and computational zero-knowledge proof and argument systems one has to consider a natural computational version of the game where,

- the scheduler and rewinder strategies are required to be efficiently computable, and
- the leaf reached by the rewinder is only required to be pseudo-randomly distributed.

We remark that these computational restrictions are mostly a technicality, as they do not seem to affect the combinatorial complexity of solving the game. In particular, in all settings that we are aware of, the best known solution to the game is also efficiently computable and it generates distributions over leaves that are statistically close to (rather than simply indistinguishable from) random. We give examples in the full version of the paper [14].

2 Preliminaries

2.1 Notation

We denote by Σ^* the (infinite) set of all strings. We sometimes write $\{0,1\}^{\leq n}$ to denote the set of all bit-strings with length at most n. We will often use bold letters or overbars to represent (possibly empty) sequences of messages (e.g., $\mathbf{q}$, $\bar{\beta}$).

For interactive machines P and V, let $\langle P, V\rangle(x)$ be the local output of V after interacting with P on common input x. We denote by $\mathsf{view}_V^P(x)$ the random tape of V followed by the sequence of messages V receives while interacting with P

on common input x. We will generally use α to denote a message from V, while β will represent a message from P. We will often use next-message functions when dealing with interactive machines. Let $V(x, z, \bar{\beta}; r)$ denote the *next-message function* which takes as input a sequence of messages $\bar{\beta}$ and a random tape r and outputs the result of running interactive machine V with random tape r on common input x, auxiliary input z, and sequence of incoming message $\bar{\beta}$. When dealing with a joint computation between two interactive machines, we define a *round* to be two messages, one from each machine. So if we say a protocol is four rounds, for example, there are actually eight messages exchanged. It is sometimes convenient to run an interactive machine for a number of rounds which is higher than the number specified by its program. We use the convention that if an interactive machine implementing an r-round protocol is sent more than r messages, it replies to the messages beyond round r with an empty dummy message. For Turing machine M, we let $\mathsf{desc}(M)$ denote the description of M.

A function $\nu : \mathbb{N} \to [0, 1]$ is called *negligible* if $\nu(n) = n^{-\omega(1)}$, and *non-negligible* (or *noticeable*) if $\nu(n) = n^{-O(1)}$. We say two ensembles $\{X_w\}_{w \in S}$ and $\{Y_w\}_{w \in S}$ indexed by strings in some infinite set S, are *computationally indistinguishable* (denoted $\{X_w\}_{w \in S} \equiv_c \{Y_w\}_{w \in S}$) if for every probabilistic (strict) polynomial time (PPT) algorithm D (called the distinguisher) we have $|\mathbf{Pr}[D(X_w, w) = 1] - \mathbf{Pr}[D(Y_w, w) = 1]| < \nu(|w|)$ for some negligible function ν. Conversely, we say that the same ensembles are *computationally distinguishable* if there is some PPT distinguisher D such that $|\mathbf{Pr}[D(X_w, w) = 1] - \mathbf{Pr}[D(Y_w, w) = 1]| \geq \nu(|w|)$ for some non-negligible function ν.

2.2 Interactive Proofs and Black-Box Zero Knowledge

We use the standard definition of interactive proofs with negligible soundness error:

Definition 1. *A pair of interactive machines (P, V) is an interactive proof system for language L if machine V runs in polynomial time and there exists a negligible function $\nu : \mathbb{N} \to [0, 1]$ such that*

- *Completeness: For every $x \in L$, $\mathbf{Pr}[\langle P, V\rangle(x) = 1] \geq 1 - \nu(|x|)$*
- *Soundness: For every $x \notin L$ and every interactive machine B, $\mathbf{Pr}[\langle B, V\rangle(x) = 1] \leq \nu(|x|)$*

We say that such an interactive protocol has almost-perfect completeness or negligible completeness error, and negligible soundness error. If the soundness condition holds only against PPT B, then (P, V) is called an *argument* system (also known as a computationally sound proof). The results in this paper hold for both proof and argument systems, but for simplicity we focus on proof systems.

Definition 2. *Let (P, V) be an interactive proof system for some language L. We say that (P, V) is **black-box zero-knowledge** if there exists a PPT oracle machine S such that for every PPT interactive machine V^*, the ensembles*

$$\{\mathsf{view}_{V^*(z)}^{P(w_x)}(x)\}_{x \in L, z \in \{0,1\}^*} \text{ and } \{S^{V^*(x,z,\cdot)}(x)\}_{x \in L, z \in \{0,1\}^*}$$

are computationally indistinguishable.

Note that in this definition the simulator S is given oracle access to the next-message function of the adversarial verifier.

2.3 Composition of Interactive Proofs

We will consider a setting in which a verifier may interact with multiple, independent copies of the prover, each of which is attempting to prove the same theorem. Specifically, the prover's reply in some session should only depend on previous messages *from that session* and not messages from some other session. On the other hand, the verifier may make decisions based on messages it has seen from any session. To model this, we will consider a single adversarial verifier V^* which sends messages of the form (α, s) to the prover, where s is just some arbitrary string identifying the session. The prover's next message must be a reply for this session. The last message of the interaction is then considered the output of V^*, which we require to have the form (α^*, end) for some α^*. So, the transcript of an interaction between P and V^* will be of the form $((\alpha_1, s_1), \beta_1, (\alpha_2, s_2), \beta_2, \ldots, (\alpha_v, s_v), \beta_v, (\alpha^*, \mathsf{end}))$.

In this paper, we only consider adversarial verifiers which never abort, where by abort we mean send messages that are malformed in some session (or deviate in some other detectable way). This is without loss in the upper bound because a simulator can easily modify its verifier to be non-aborting. In the case of the lower bound, the adversarial verifiers we construct do not send abort messages at any time. Previous lower bounds (e.g., [3]) rely on aborting sessions to force the simulator to do extra work. Nevertheless, the act of the verifier *informing* the simulator that it is going to abort some session simply makes the task of simulating easier, since the simulator knows that on the current path it no longer needs to worry about the aborted session. In our paper, the verifier can still implicitly "abort" a session by never again sending a message for that session; it just doesn't tell the simulator it is doing this.

2.4 Black-Box Concurrent Zero Knowledge

In the most general form of concurrent composition, the verifier is allowed to interact with a polynomial number of independent provers while maintaining complete control over the scheduling of messages. Specifically, the verifier may interleave messages from different sessions any way it chooses. This situation was first explored for witness indistinguishability (a weaker notion than zero-knowledge) in [6,5] and later for zero-knowledge in [4].

As observed in [3], the standard definition of black-box zero-knowledge is insufficient in the concurrent setting, since the running time of the simulator must be a fixed polynomial and independent of its oracle. Thus, a verifier could initiate more sessions than the simulator has time to handle (yet still a polynomial number). To overcome this subtlety, we give the simulator an additional input representing an upper bound on the length of the interaction with the verifier (how many messages the verifier will send in all sessions before halting with

some local output). A similar approach was used in [3][2]. We then require the simulator to run in polynomial time in both the length of the common input and the maximum length of an interaction.

Definition 3. *Let (P, V) be an interactive proof system for some language L. We say that (P, V) is* ***black-box concurrent zero-knowledge*** *if there exists a PPT oracle machine S such that for every polynomial $h(\cdot)$ and every PPT interactive machine V^* sending at most $h(|x|)$ messages in any interaction, the ensembles*

$$\{\mathsf{view}^P_{V^*(z)}(x)\}_{x\in L, z\in\{0,1\}^*} \text{ and } \{S^{V^*(x,z,\cdot)}(x, h(|x|))\}_{x\in L, z\in\{0,1\}^*}$$

are computationally indistinguishable.

3 A Simple Combinatorial Game

In this section we formally define our combinatorial game. We first describe a purely combinatorial version, and then modify the game to satisfy some natural computational restrictions. The game is parameterized by positive integers h, r, and d and has two players: a rewinder $\mathbb{S}$ and a scheduler $\mathbb{V}$.

3.1 The Scheduler

The *scheduler* $\mathbb{V}$ uses a private random input to specify a labeled tree where each internal node v has exactly d children $v_1, \ldots, v_d$ and all leaves are at level h (see Fig. 1). Each edge $v \to v_m$ (from a node v to its mth child) carries the label m. Each internal node v (represented by a sequence in $D^{\leq h} = \{1, \ldots, d\}^{\leq h}$ of edges) carries a label $\pi(v)$ which equals either the node itself ($\pi(v) = v$), or a previous label $\pi(v) = \pi(w)$ for some ancestor w (i.e. prefix) of v. The labeling function $\mathbb{V} : D^{\leq h} \to D^{\leq h}$ takes as input some node v and returns its label $\pi(v)$.

We let $U_{d,h}$ denote the uniform distribution on leaves in a tree of degree d and height h. Leaves are sequences in D^h.

For any $v = (m_1, \ldots, m_j)$, let $\mu(v) = v[I]$, where I is the set of positions i in v satisfying $\pi(v[1, \ldots, i-1] = \pi(v)$. This means that $\mu(v)$ is the sequence of edges coming out of nodes with the same label as v, on the path from the root to v. See Fig. 2 for an illustration.

3.2 The Rewinder

The *rewinder* is a probabilistic oracle algorithm $\mathbb{S}^{\mathbb{V}}$ where $\mathbb{V} : D^{\leq h} \to D^{\leq h}$. The rewinder $\mathbb{S}$ uses the oracle to explore the scheduler tree, and it may query the oracle on a child of some previously visited node v subject to one restriction:

- If $|\mu(v)| = r$, then the rewinder must have already visited some other node w with the same label $\pi(w) = \pi(v)$ such that $\mu(w)$ is not a prefix of $\mu(v)$.

[2] In [3], the simulator takes an additional input representing an upper bound on the number of sessions the verifier will initiate during the interaction.

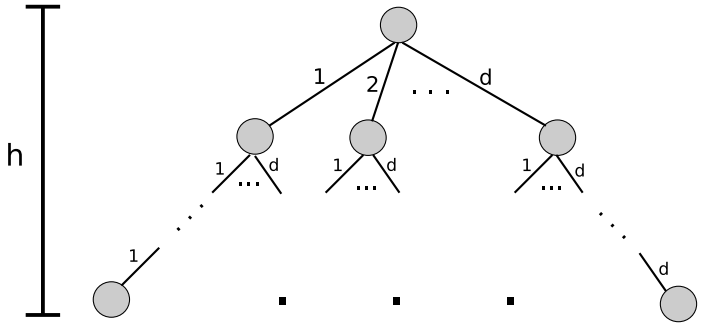

Fig. 1. The scheduler tree. Each internal node has degree d, with edges labeled 1 to d. The height of the tree is always h and the leaves are sequences in $\{1, \ldots, d\}^h$.

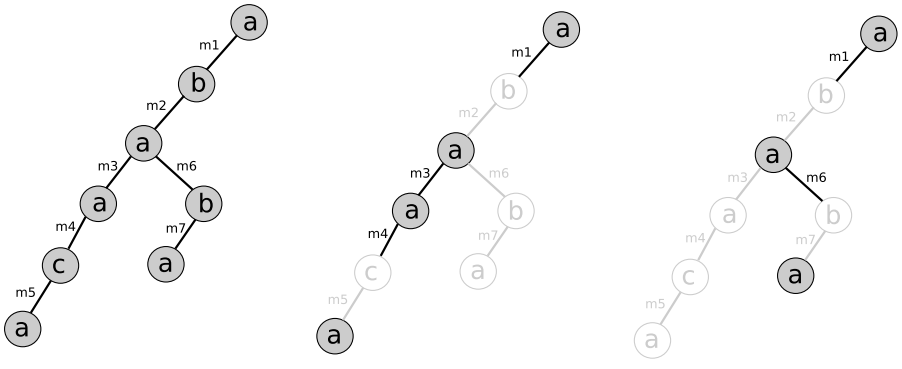

Fig. 2. The above depicts the explored portion of some tree. For brevity we let $a = \epsilon$, $b = m_1$, and $c = (m_1, m_2, m_3, m_4)$. The middle picture depicts $\mu(m_1, m_2, m_3, m_4, m_5) = (m_1, m_3, m_4)$. The right picture shows how the node (m_1, m_2, m_6, m_7) has the same label $a = \epsilon$ as the node $(m_1, m_2, m_3, m_4, m_5)$, but $\mu(m_1, m_2, m_6, m_7) = (m_1, m_6)$ is not a prefix of $\mu(m_1, m_2, m_3, m_4, m_5) = (m_1, m_3, m_4)$.

The restriction, which we often call the μ restriction, is illustrated in Fig. 2. It is easy to see that the condition can be efficiently checked by $\mathbb{S}$ (without making any additional queries) because it only depends on the previously issued queries and respective answers. We consider rewinders $\mathbb{S}$ that always terminate (or, at least terminate with probability 1), and always make at least one query $v_0 = \epsilon$.

We denote by $\mathbb{S}^{\mathbb{V}}$ the probability distribution over leaves D^h, specified by the last query asked by $\mathbb{S}$ to the oracle $\mathbb{V}$. The goal of the rewinder is to produce a distribution over leaves as close as possible to uniform $U_{d,h}$.

3.3 A Computational Version

To make the game useful in asymptotic settings (like black-box computational zero-knowledge proof systems), we need to impose some computational restrictions.

Instead of considering a single scheduler $\mathbb{V}$, we consider a family of schedulers $\{\mathbb{V}_x\}_{x\in\Sigma^*}$ indexed by a parameter string x. Each $\mathbb{V}_x$ is a defined as usual as a function $\mathbb{V}_x : D_x^{h_x} \to D_x^{h_x}$ where $D_x = \{1,\ldots,d_x\}$, i.e., $\mathbb{V}_x$ is a labeled tree of degree d_x and height h_x. We say that $\mathbb{V}$ is an *efficient* scheduler if $h_x \leq |x|^{O(1)}$, $d_x \leq 2^{|x|^{O(1)}}$ and the labeling function $(x,v) \mapsto \mathbb{V}_x(v)$ is computable in polynomial time $|x|^{O(1)}$. (Notice that by the bounds on h_x and d_x, the size of the second argument $|v| \leq h_x \cdot \log_2 d_x \leq |x|^{O(1)}$ is always polynomial in $|x|$.)

Similarly, we consider families of oracle algorithms $\mathbb{S}_x^{\mathbb{V}}(h)$ indexed by the string x. We say that $\mathbb{S}$ is an *efficient rewinder* if the strategy $\mathbb{S}_x^{(\cdot)}(h)$ can be implemented in PPT in $|x|$ and h. The goal of $\mathbb{S}$ is to produce a distribution over leaves $D_x^{h_x}$ which is computationally indistinguishable (in $|x|$) from the uniform distribution.

Finally, the round number r_x defining the rules of the game, can also depend on x. Typically, h_x, d_x and r_x are just functions of the parameter length $|x|$.

4 Two Theorems

In this section we give two theorems (for upper and lower bounds) which relate the success of rewinders in our combinatorial game to the success of black-box simulators in the more complicated concurrent zero-knowledge setting. For simplicity, in both of the following theorems, fix d_x to be some super-polynomial function of $|x|$.

Theorem 1 (Lower Bound). *For any round parameter r_x and some language L, if the following two conditions hold:*

1. *The game does not have a solution: For every efficient rewinder $\mathbb{S}$ there exists an efficient scheduler $(h_x, \mathbb{V}_x)$ such that the ensembles $\{\mathbb{S}_x^{\mathbb{V}_x}(h_x)\}_{x\in\Sigma^*}$ and $\{U_{d_x,h_x}\}_{x\in\Sigma^*}$ are distinguishable in polynomial time*
2. *There exists an (r_x+1)-round black-box concurrent zero-knowledge proof system for L*

Then $L \in \mathcal{BPP}$.

Theorem 2 (Upper Bound). *For any game parameter r_x, if the following two conditions hold:*

1. *The game has a solution: There exists an efficient rewinder $\mathbb{S}$ such that for all efficient schedulers $(h_x, \mathbb{V}_x)$, the ensembles $\{\mathbb{S}_x^{\mathbb{V}_x}(h_x)\}_{x\in\Sigma^*}$ and $\{U_{d_x,h_x}\}_{x\in\Sigma^*}$ are computationally indistinguishable*
2. *Perfectly-hiding commitment schemes exist*

Then there exists an $(r_x+O(1))$*-round black-box concurrent zero-knowledge proof system for all* $L \in \mathcal{NP}$*.*

We prove Thm. 1 in Sect. 5 and Thm. 2 in Sect. 6. Combining the two theorems gives us our main result: Any non-trivial language (in $\mathcal{NP} \setminus \mathcal{BPP}$) has an $(r + O(1))$-round black-box concurrent zero-knowledge proof system if and only if our combinatorial game with parameter r admits a solution.

5 Proof of Theorem 1 (Lower Bound)

In this section we show the relationship between the combinatorial game and black-box zero-knowledge lower bounds. Before getting into the details of the proof we will first provide some intuition for our results.

Like in the lower bound proof of [3], we run the simulator S for language L inside of a $\mathcal{BPP}$ decision procedure while simulating its oracle (an adversarial concurrent verifier) using sufficiently independent copies of the single-session honest verifier. However, the way we use the single-session honest verifier differs, as does our accepting criteria. We force the decision procedure to accept if and only if at some point S causes a single-session verifier to accept without successfully rewinding the session. Otherwise, the decision procedure rejects.

It is fairly straightforward to show that if x is not in L, then our decision procedure will reject with overwhelming probability, due to the soundness of the proof system. More difficult is showing that if x is in the language, then the decision procedure accepts with noticeable probability. To accomplish this, we want to make it difficult for the simulator to successfully rewind every session. This is where we use the combinatorial game and our assumption that for any rewinder there is a difficult scheduler.

Intuitively, a simulator making oracle queries to an adversarial verifier (and possibly rewinding) is similar to a rewinder exploring a scheduler tree. Following this intuition, we show that given S we can build a rewinder $\mathbb{S}$. The rewinder runs S internally and uses its scheduler oracle to help simulate a concurrent adversarial verifier oracle for S. Given such a rewinder $\mathbb{S}$, our assumption guarantees a corresponding difficult scheduler $\mathbb{V}^*$. The hope is that this difficult scheduler will make the corresponding adversarial verifier hard-to-simulate. We then simply make our $\mathcal{BPP}$ decision procedure run S and simulate its oracle in the same way as the rewinder $\mathbb{S}$, and using the difficult scheduler $\mathbb{V}^*$.

There is still one issue to overcome. Our assumption tells us that it is difficult for $\mathbb{S}$ to reach a random leaf in $\mathbb{V}^*$, so we need to make sure the simulator's task of properly simulating and the rewinder's task of reaching a random leaf are related. To do this, we have the simulated adversarial concurrent verifier randomize queries before querying the difficult scheduler $\mathbb{V}^*$. As long as the independence used to randomize is greater than the maximum length of an interaction between the honest prover and the verifier, the prover's interaction with this verifier will correspond to reaching a random leaf in the scheduler tree. Since S must simulate this interaction properly (because of the zero-knowledge property), we have our desired relationship between the game and the proof

system. It should then be the case that if x is in L, S will have difficult time rewinding every session, and thus it will be forced to make a single-session honest verifier accept without rewinding. This, as we said above, causes the decision procedure to accept x.

5.1 Details of the Proof

We will now expand on the ideas explained in the previous section. Fix some function r_x, fix function d_x to be super-polynomial in $|x|$, and let L be some language. Assume the following:

1. For every probabilistic oracle machine $\mathbb{S}_x(h)$ running in polynomial time in $|x|$ and h, there exists a polynomial time (in $|x|$) computable $\mathbb{V}_x(\cdot)$, such that the ensembles $\{\mathbb{S}_x^{\mathbb{V}_x}(h_x)\}_{x\in\Sigma^*}$ and $\{U_{d_x,h_x)}\}_{x\in\Sigma^*}$ are distinguishable in polynomial time.
2. There exists an (r_x+1)-round black-box concurrent zero-knowledge proof system for L.

We aim to show that $L \in \mathcal{BPP}$. Since we are assuming L has a black-box zero-knowledge proof, let S be its simulator. As we explained above, we will define a decision procedure D which runs S and simulates for it a verifier oracle V, with the help of some scheduler $\mathbb{V}$. We would like our decision procedure to simulate a verifier oracle which makes it difficult for S to properly rewind every session. To do this we first show how to construct a difficult verifier after placing a few restrictions on S.

Restrictions on S. We make some assumptions about S which are without loss of generality. First, we assume that S only makes queries for which it has already queried all shorter prefixes. Second, we assume that before S outputs a view with prover messages $\bar{\beta}$, it queries its oracle one last time with $\bar{\beta}$ (and all shorter prefixes). Clearly, any PPT S can be transformed into a simulator S' that satisfies the above properties and still runs in polynomial time.

Let $t(|x|,h)$ and $m(|x|)$ be polynomial bounds on the number of queries and the message size for all queries made by S.

A Difficult Verifier. As with past lower bound proofs, we wish to describe a verifier which will be difficult for the simulator S to simulate inside of the decision procedure. We first describe an oracle verifier $\hat{V}$ which is given oracle access to a scheduler $\mathbb{V}$ which uses p bits of its private random input.

We will rely extensively on two hash functions, which we say are given to $\hat{V}$ as auxiliary input. Let $\mathcal{F} = \{F_{n,h}\}_{n,h\in\mathbb{N}}$ be a family of $t(n,h)$-wise independent hash functions. Let $\mathcal{G} = \{G_{n,h}\}_{n,h\in\mathbb{N}}$ be a family of $t(n+h)$-wise independent hash functions. Each function $f \in F_{n,h}$ will be from $\{0,1\}^{\leq h\cdot m}$ to $\{0,1\}^{\rho}$, where ρ is an upper bound on the number of bits the single-session honest verifier reads from its random tape. Functions from $\mathcal{F}$ will be used to hash all of the messages leading up to the start of a session in order to generate randomness for the single-session honest verifier used in that session. Each function $g \in G_{n,h}$

will be from $\{0,1\}^{\leq h \cdot m} \times \{0,1\}^{\leq h \cdot m} \times m$ to $\{0,1\}^{\log d}$. Functions from $\mathcal{G}$ are used to randomize messages for use as edges in a scheduler tree. They contain independence greater than the running time of S and therefore also greater than the maximum length of an interaction. In particular, for a randomly chosen function from the family, queries of length h will lead to a uniformly random leaf as long as the inputs are unique.

Given any function g from the family just described, for simplicity we define another function $\hat{g}_{\mathbb{V}}$ which is with respect to some scheduler $\mathbb{V}$. Now, define $\mathbb{V} \circ g(\bar{\beta}) = \mathbb{V}(\hat{g}_{\mathbb{V}}(\bar{\beta}))$ and recursively define $\hat{g}$ as

$$\hat{g}_{\mathbb{V}}(\epsilon) = \epsilon$$
$$\hat{g}_{\mathbb{V}}(\bar{\beta}, \beta) = (\hat{g}_{\mathbb{V}}(\bar{\beta}), g(\pi_{\mathbb{V} \circ g}(\bar{\beta}), \mu_{\mathbb{V} \circ g}(\bar{\beta}), \beta)),$$

where π and μ are defined as they were in Sect. 3. We are ready to describe the verifier $\hat{V}$ with oracle access to some scheduler $\mathbb{V}$. We describe the next-message function, which takes as input a sequence of messages $\bar{\beta}, \beta$. The verifier is given as auxiliary input two hash functions f and g from the families we described above.

Algorithm $\hat{V}^{\mathbb{V}_x}(x, (f, g), (\bar{\beta}, \beta))$
1. If $|\bar{\beta}, \beta| = h$ or $V(x, (\mu_{\mathbb{V} \circ g}(\bar{\beta}), \beta); f(\pi_{\mathbb{V} \circ g}(\bar{\beta}))) = \mathsf{reject}$ then return $((\bar{\beta}, \beta), \mathsf{end})$
2. Else return $(V(x, \mu_{\mathbb{V} \circ g}(\bar{\beta}, \beta); f(\pi_{\mathbb{V} \circ g}(\bar{\beta}, \beta))), \mathbb{V} \circ g(\bar{\beta}, \beta))$

The adversarial concurrent verifier intuitively executes multiple sessions and interleaves them based on queries to its scheduler oracle (the queries are first randomized using g before querying the oracle). The content of the replies from $\hat{V}$ comes from a single-session honest verifier V which $\hat{V}$ runs internally. Each session's messages are determined by a sufficiently independent copy of V (the independence is due to the hash function f).

The adversarial verifier $\hat{V}$ sends final output (a sequence of messages) in two cases. One is if at any time a single-session honest verifier rejects. The other is if the end of the interaction is reached (the sequence of messages is length h). We should mention that it is possible a valid rewinding of some session by S might not yield a valid rewinding in $\mathbb{V}$. This is because each message in the queries from S is at most size m (a polynomial in $|x|$), but when g randomizes it maps these messages into a message space of size d (which is super-polynomial in $|x|$). However, since S makes at most polynomially many queries, the probability of a collision is negligible, so we ignore this event for the rest of the proof.

Now, as we said earlier, we want to run the simulator inside of a decision procedure, giving it oracle access to a difficult-to-simulate verifier. To make the verifier just described "difficult", we want it to have oracle access to a difficult scheduler. To get such a scheduler, we need to define a rewinder and then use our assumption about the combinatorial game not admitting a solution. The rewinder is as follows.

Algorithm $\mathbb{S}_x^{\mathbb{V}_x}(h)$
1. Randomly choose f and g from their respective families.
2. Run $[S^{\hat{V}}(x,h)]^{\mathbb{V}_x}$
3. Watch for queries to $\mathbb{V}_x$ which violate the condition of the combinatorial game.
4. If such a query is made then halt and output $\perp$.

The rewinder $\mathbb{S}$ internally runs S, giving it oracle access to $\hat{V}$. Since $\hat{V}$ expects a scheduler oracle, $\mathbb{S}$ will simulate it using its own scheduler oracle $\mathbb{V}$. However, $\mathbb{S}$ will monitor these queries to $\mathbb{V}$ and if at any time there is a query which violates the condition of the game (the scheduler was not properly "rewound"; see Sect. 3) $\mathbb{S}$ will halt and fail. Otherwise $\mathbb{S}$ will continue until S halts with some final output.

Given the efficient rewinder above, our assumption about the game not admitting a solution guarantees there is some difficult efficient scheduler $\mathbb{V}^*$. Our adversarial concurrent verifier $\hat{V}$ should now, when given oracle access to $\mathbb{V}^*$, be hard-to-simulate for S. Let p be an upper bound on the number of private random bits that $\mathbb{V}^*$ uses.

The Decision Procedure. We are ready to give the $\mathcal{BPP}$ decision procedure D for language L.

Algorithm $D(x)$
1. Choose f, g randomly from the required families.
2. Choose private random input $R \xleftarrow{\$} \{0,1\}^p$.
3. Run $[S^{\hat{V}(x,(f,g),\cdot)}(x,h_x))]^{\mathbb{V}_x^*}$
4. If there is ever an attempted query to $\mathbb{V}_x^*$ which violates the conditions of the game, then halt and ACCEPT

The decision procedure D, on input x, runs the simulator S on input x and with interaction length equal to the height h_x of the difficult scheduler tree. The simulator expects an oracle so to simulate it D uses $\hat{V}$ with auxiliary input two randomly chosen hash function f and g. Scheduler oracle queries from $\hat{V}$ are answered using the difficult scheduler $\mathbb{V}^*$. Like in the rewinder defined above, D monitors the oracle queries to $\mathbb{V}^*$. If there is ever a query which would violate the game condition, D halts and accepts. Otherwise, S will eventually halt with some output $(\bar{\beta}, \beta)$, in which case D rejects. The correctness of our decision procedure follows from two lemmas.

Lemma 1. *For all but finitely many* $x \in L$, $\mathbf{Pr}[D(x)$ *accepts*$] \geq 1/q(|x|)$ *for some polynomial* $q(\cdot)$.

Lemma 2. *For all but finitely many* $x \notin L$, $\mathbf{Pr}[D(x)$ *accepts*$] < \nu(|x|)$ *for some negligible function* $\nu(\cdot)$.

Proof of Lemma 1. The idea of the proof is simple. The decision procedure only accepts if there is some query which violates the game condition. Notice that this is the only difference between the execution of the rewinder $\mathbb{S}_x^{\mathbb{V}_x^*}(h_x)$ and

the execution of $S^{\hat{V}^{\mathbb{V}^*}}(x, h_x)$. We show that this latter execution (without any check for game conditions) results in a random leaf when x is in the language. The inability of the rewinder to reach a random leaf then allows us to argue that the difference between the two executions above is noticeable.

We now give details. Consider some adversarial verifier V^* which is identical to $\hat{V}$ except that it does not make oracle calls to a scheduler. Instead, it is given the code of a scheduler as auxiliary input (as well as hash function f and g and private random input for the scheduler).

We first claim that for randomly chosen f and g, the ensembles

$$\{\hat{g}_{\mathbb{V}^*}(\langle P, V^*_{f,g,\mathsf{desc}(\mathbb{V}^*),R}\rangle(x))\}_{x\in L} \text{ and } \{U_{d_x,h_x}\}_{x\in L}$$

are computationally indistinguishable. To see this, notice first that the output of V^* when interacting with the honest prover on input $x \in L$ will be a prover query of length h. This is because the proof system has negligible completeness error, so any invocation of the honest verifier will accept with overwhelming probability. Thus, with high probability V^* will only reply with final output when the length of the query is h. This means that with high probability P will cause V^* to query its scheduler at a leaf. Now, because the independence of g is greater than h, g was randomly chosen from the family, and because each input to g (inside the definition of $\hat{g}$) is distinct (appending π and μ ensures this), then g, when applied to the query of length h will result in a uniformly random leaf.

The zero-knowledge property then ensures us that ensembles

$$\{\hat{g}(S^{V^*(x,(f,g),\mathsf{desc}(\mathbb{V}^*),R,\cdot)}(x, h_x))\}_{x\in L} \text{ and } \{\hat{g}(\langle P, V^*_{f,g,\mathsf{desc}(\mathbb{V}^*),R}\rangle(x))\}_{x\in L}$$

must be computationally indistinguishable as well (again for randomly chosen f and g as above). Now, we know by our assumption that there must be some Δ which can distinguish between U_{d_x,h_x} and $\mathbb{S}^{\mathbb{V}^*}_x(h_x)$ with noticeable probability for all sufficiently large strings x. So there must be some $\tilde{\Delta}$ which can distinguish between the ensembles $\mathbb{S}^{\mathbb{V}^*_x}_x(h_x)$ and $\hat{g}(S^{V^*(x,(f,g),\mathsf{desc}(\mathbb{V}^*),R,\cdot)}(x, h_x))$ for all sufficiently large $x \in L$.

Recall that the random variable $\mathbb{S}^{\mathbb{V}^*}_x(h_x)$ is considered to be the function $\hat{g}$ applied to the last query from S, which we have said must be the same as the sequence of messages S outputs. Yet for any x the statistical distance between $\hat{g}(S^{V^*(x,(f,g),\mathsf{desc}(\mathbb{V}^*),R,\cdot)}(x))$ and $\mathbb{S}^{\mathbb{V}^*}_x(h_x)$ is at most the probability that some invalid query from S causes an honest-verifier to accept plus the probability of a collision, and the probability of collision is negligible (since d is a fixed super-polynomial) so it must be the case that the former probability is noticeable. The lemma immediately follows.

Proof Sketch of Lemma 2. The proof uses the standard technique from past lower bound papers (cf. [3]), which relies on the soundness of the (single-session) proof system. Since D accepts if and only if S is able to make the single-session honest verifier accept without rewinding, we can use S to build a single-session "cheating prover" which, while interacting with some single-session honest verifier, runs S

internally. The cheating prover uses the actual verifier it is interacting with to simulate one session with S, while for all other sessions the cheating prover follows the strategy of D and runs copies of the single-session honest verifier internally. The negligible soundness error of the proof system ensures that S must only be able to make an unrewound session accept with negligible probability, and the lemma follows.

6 Proof of Theorem 2 (Upper Bound)

In this section we show how our combinatorial game relates to black-box zero-knowledge upper bounds. We follow the usual procedure for proving there is a black-box zero-knowledge proof system for all languages in $\mathcal{NP}$ by giving such a proof system for an $\mathcal{NP}$-complete language (this was first done in [10]).

We want this proof system to closely resemble our combinatorial game. Specifically, we desire a proof system in which

- for r rounds, an honest prover sends uniformly random messages, and
- if a simulator executes one successful rewind during these r rounds, it will be able to successfully simulate that session.

Intuitively, we want the first property since in our combinatorial game the rewinder must reach a uniformly random leaf. The second property matches our μ restriction on the rewinder.

The Proof System. As a starting point, we will focus on $\mathcal{NP}$-complete languages with 3-message, public-coin, committed-verifier zero-knowledge (CVZK) proof systems (formalized in [13]). This means the proof system (P, V) has three messages, the verifier simply outputs bits from its random tape, and there exists a simulator S such that for all challenges c the ensembles $\{S(x, c)\}_{x\in L}$ and $\{\mathsf{view}^P_{V_c}(x)\}_{x\in L}$ are computationally indistinguishable. Hamiltonicity is one such $\mathcal{NP}$-complete language.

We then follow the technique of Rosen [17] (which also appeared in [15]) and augment the above proof system with a preamble which proceeds as follows. The verifier initially commits (using a perfectly-hiding commitment scheme) to a challenge σ and shares $\sigma^0_{i,j}, \sigma^1_{i,j}$ for $1 \le i \le l$ and $1 \le j \le r$ of σ such that $\sigma^0_{i,j} \oplus \sigma^1_{i,j} = \sigma$ for all i and j. Numerous rounds of challenge-response follow. In each round, the prover sends a random bit-string and the verifier opens the shares corresponding to this bit-string. The prover then executes the 3-message, public-coin, committed verifier zero-knowledge proof system (which we now call the second stage). In this second stage, the verifier sends σ as its challenge, as well as openings for the rest of the shares to show that it did not cheat in the preamble. See Fig. 3 for details.

The Simulator. For an $\mathcal{NP}$-complete language L with the property mentioned above, we wish to show that there is an $(r(\cdot)+O(1))$-round black-box concurrent zero-knowledge proof system, assuming that there is an efficient rewinder $\mathbb{S}$ that

Preamble:
$V \to P$: commit to σ and $\sigma^0_{i,j}, \sigma^1_{i,j}$ for $1 \le i \le l$ and $1 \le j \le r$.
For $j = 1, \ldots, r$:
$P \to V$: Send $b_{1,j}, \ldots, b_{l,j} \xleftarrow{\$} \{0,1\}^l$.
$V \to P$: Decommit to $\sigma^{b_{1,j}}_{1,j}, \ldots, \sigma^{b_{l,j}}_{l,j}$.
Second Stage:
$P \to V$: Send first message of three-round protocol.
$V \to P$: Decommit to σ and $\sigma^{1-b_{i,j}}_{i,j}$ for $1 \le i \le l$ and $1 \le j \le r$.
$P \to V$: Answer according to the value of σ.

Fig. 3. The Rosen protocol

can win our combinatorial game with parameter $r(\cdot)$, and that perfectly-hiding commitment schemes exist.

To accomplish this task, we will need to build a successful simulator S with an output distribution computationally indistinguishable from the view of any adversarial concurrent verifier V^* interacting with the real prover P. The simulator is given oracle access to V^* to aid it in this task. We assume without loss that V^* always sends exactly h messages in any concurrent interaction.

From a high level, our simulator S will make queries to V^* and use its one advantage over a prover (the ability to rewind V^*) to help it achieve its goal. At some point S will make one final query to V^* and output a view corresponding to this query.

The rewinding strategy employed by S will be dictated by the efficient, successful rewinder $\mathbb{S}$, which S will run internally. Since $\mathbb{S}$ expects a scheduler oracle which it uses to explore a labeled tree, we make S simulate the oracle using an internal tree data structure which it labels with the help of its own oracle, V^*.

Each node $v = (m_1, \ldots, m_j)$ (denoted by the sequences of edges leading to it) in the tree will correspond to some query made by $\mathbb{S}$, and will contain three pieces of information. The first is the label of the node $\pi(v)$, which is either v or $\pi(w)$, the label of some ancestor. The other information will be a sequence of prover message $\bar{\beta}$ and the corresponding reply (α, s) from V^*. Intuitively, s is the session label which S uses to determine the node label $\pi(v)$. If s is the same as in some ancestor node w, then $\pi(v) = \pi(w)$. If s is a new label, then $\pi(v) = v$.

When $\mathbb{S}$ asks its oracle for the label of v, some previously unexplored node, the simulator S adds the node v to the tree (recall nodes are specified by a sequence of edges), and looks up the information $(\pi(v'), \bar{\beta}, (\alpha, s))$ stored at the parent node $v' = (m_1, \ldots, m_{j-1})$. There are now two cases.

In the first case, there are less than $r+1$ nodes on the path to v labeled $\pi(v')$,i.e., $|\mu(v')| < r$, in which case the simulator appends m_j to $\bar{\beta}$ and queries it to V^*. This corresponds to taking some previous query $\bar{\beta}$ and making a longer query with the addition of preamble message m_j.

In the second case, there are either $r+1$ or $r+2$ nodes with label $\pi(v')$ on the path, so the preamble is finished. In this case, S searches the tree for a

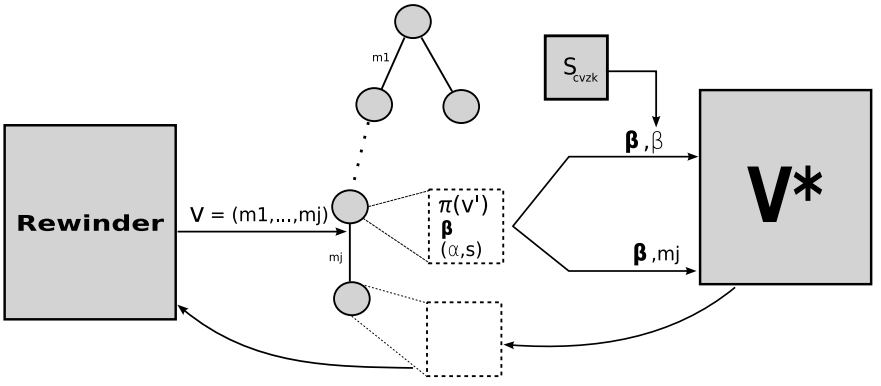

Fig. 4. Illustration of how S uses $\mathbb{S}$ to make queries to V^*

node y with label $\pi(y) = \pi(v')$ and $\mu(y)$ not a prefix of $\mu(v')$. This constitutes a proper rewinding, so S should have V^*'s challenge revealed to it by comparing the messages stored at each node. The simulator, armed with the challenge, uses the committed-verifier simulator to generate a message to send to V^*. The simulator will find the required information in the tree, since we are assuming verifiers that do not abort and rewinders that do not make invalid queries. In either case, the simulator will make some query (call it $\bar{\beta}_j$) to V^*.

We still need to give $\mathbb{S}$ an answer to its query v. To do so we need to find out which information to store at node v in the tree data structure. We will use V^* to accomplish this task. Let (α_j, s) be the reply from V^* on query $\bar{\beta}_j$. If there exists some ancestor w of v with a message (α_i, s) stored at it, then at node v store $\pi(w), \bar{\beta}_j, (\alpha_j, s)$ and otherwise store $v, \bar{\beta}_j, (\alpha_j, s)$. Finally, once the tree has been updated, S returns the label $\pi(v)$ to $\mathbb{S}$. The sequences of messages S ultimately outputs will be the $\bar{\beta}$ stored at the last node $\mathbb{S}$ queries. The entire process is illustrated in Fig. 4.

Showing zero-knowledge is straightforward. To do so we gradually modify the simulator until it acts identically to the honest prover, arguing indistinguishability for each modification. We first use a hybrid argument to argue that we can replace the invocations of the CVZK simulator with the honest prover given a real witness. Then we use the fact that $\mathbb{S}$ is a good rewinder, meaning it can reach a random leaf given any scheduler, to replace the $\mathbb{S}$ inside of S with a single, random path down the tree. It then is the case that S follows a single path down the tree, preamble messages are randomly chosen, and second-stage messages are simulated using the honest prover with an actual witness. This is the same as the behavior of the honest prover, so the zero-knowledge property follows.

Acknowledgements

The authors would like to thank the anonymous referees for their comments. The first author is supported in part by NSF Cybertrust grant CNS–0430595.

The second author is supported in part by the first author's grant, as well as by an Irwin and Joan Jacobs Fellowship.

References

1. Barak, B.: How to go beyond the black-box simulation barrier. In: FOCS 2001, pp. 106–115 (2001)
2. Barak, B., Lindell, Y.: Strict polynomial-time in simulation and extraction. SIAM J. Comput. 33(4), 783–818 (2004)
3. Canetti, R., Kilian, J., Petrank, E., Rosen, A.: Black-box concurrent zero-knowledge requires (almost) logarithmically many rounds. SIAM J. Comput. 32(1), 1–47 (2003)
4. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: STOC 1998, pp. 409–418 (1998)
5. Feige, U.: Alternative Models for Zero-Knowledge Interactive Proofs. PhD thesis, Weizmann Institute of Science (1990)
6. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: STOC 1990, pp. 416–426. ACM Press, New York (1990)
7. Goldreich, O., Kahan, A.: How to construct constant-round zero-knowledge proof systems for NP. Jour. of Cryptology 9(2), 167–189 (1996)
8. Goldreich, O., Krawczyk, H.: On the composition of zero-knowledge proof systems. SIAM J. Comput. 25(1), 169–192 (1996)
9. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. 18(1), 186–208 (1989)
10. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. J. ACM 38(3), 690–728 (1991)
11. Kilian, J., Petrank, E.: Concurrent and resettable zero-knowledge in poly-logorithmic rounds. In: STOC 2001, pp. 560–569. ACM, New York (2001)
12. Kilian, J., Petrank, E., Rackoff, C.: Lower bounds for zero knowledge on the internet. In: FOCS 1998, pp. 484–492 (1998)
13. Micciancio, D., Ong, S.J., Sahai, A., Vadhan, S.: Concurrent zero knowledge without complexity assumptions. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 1–20. Springer, Heidelberg (2006)
14. Micciancio, D., Yilek, S.: The round-complexity of black-box zero-knowledge: A combinatorial characterization. Full Version of this paper, `http://www-cse.ucsd.edu/users/syilek/`
15. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: FOCS 2002, pp. 366–375. IEEE, Los Alamitos (2002)
16. Richardson, R., Kilian, J.: On the concurrent composition of zero-knowledge proofs. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 415–431. Springer, Heidelberg (1999)
17. Rosen, A.: A note on constant-round zero-knowledge proofs for NP. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 191–202. Springer, Heidelberg (2004)
18. Rosen, A.: A note on the round-complexity of concurrent zero-knowledge. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 451–468. Springer, Heidelberg (2000)