# Responsive Round Complexity and Concurrent Zero-Knowledge

Tzafrir Cohen[1], Joe Kilian[2], and Erez Petrank[1]

[1] Dept. of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel, {tzafrir|erez}@cs.technion.ac.il
[2] Yianilos Labs, joe@pnylab.com

**Abstract.** The number of communication rounds is a classic complexity measure for protocols; reducing round complexity is a major goal in protocol design. However, when the communication time is inconstant, and in particular, when one of the parties intentionally delays its messages, the round complexity measure may become meaningless. For example, if one of the rounds takes longer than the rest of the protocol, then it does not matter if the round complexity is bounded by a constant or by a polynomial. In this paper, we propose a complexity measure called *responsive round complexity*. Loosely speaking, a protocol has responsive round complexity $m$ with respect to Party $A$, if it makes the following guarantee. If $A$'s longest delay in responding to a message in a run of the protocol is $t$, then, in that run, the overall communication time is at most $m \cdot t$. The logic behind this definition is that if a party responds quickly to a message, whether it has a good connection or it just chooses not to delay its messages, then this party deserves to get an overall quicker running time. Responsive round complexity is particularly interesting in a setting where a party may gain something by delaying its messages. In this case, the delaying party does not deserve the same response time as another party that behaves nicely.

We demonstrate the significance of responsive round complexity by presenting a new protocol for concurrent zero-knowledge. The new protocol is a black-box concurrent zero knowledge proof for all languages in NP with round complexity $\tilde{O}(\log^2 n)$ but responsive round complexity $\tilde{O}(\log n)$. While the round complexity of the new protocol is similar to what is known from previous works, its responsive round complexity is a significant improvement: all known concurrent zero-knowledge protocols require $\tilde{O}(\log^2 n)$ rounds. Furthermore, in light of the known lower bounds, the responsive round complexity of this protocol is basically optimal.

**Keywords:** Zero-knowledge, concurrent zero-knowledge, cryptographic protocols.

# 1 Introduction

In this work, we study a new measure related to the round complexity of protocols. We propose a notion of *responsive round complexity* that properly relates the running time of the protocol with the response time of each of the parties. Finally, we show how to improve state-of-the-art concurrent zero-knowledge protocols with respect to their responsive round complexity, and obtain an (almost) optimal protocol with respect to its responsive round complexity.

## 1.1 Round complexity

The number of rounds in a run of a protocol can be a major time-consuming component. Therefore, round complexity is one of the important complexity measures of protocols. However, a protocol's round complexity is not always directly proportional to its time complexity. The reason is that communication rounds do not always have the same length. Thus, for example, if the length of one of the rounds exceeds the accumulative length of all the other rounds, the round complexity does not tell us anything about the time complexity.

The difference in the length of communication rounds may be a result of two different reasons. One is that the network is unstable and communication times vary during the run of the protocol. The second possible reason is that one of the parties may delay its messages. It is sometimes useful for a party to delay its answer in the protocol until something happens. For example, it may delay its answer until it obtains information from another source, or it may try to foil timing assumptions made by other parties in the protocol.

We propose a new complexity measure called *responsive round complexity*. Our intention is to relate the overall running time of a party to its response time. By this measure, each party gets a guarantee on the overall communication time, which relates to the longest delay it imposes on the run of the protocol. A party that always responds quickly gets a good guarantee on the overall communication time, and a party that sometimes responds slowly gets a poor guarantee on the overall communication time.

In this extended abstract we concentrate on the two-party case. An extension of the definition to multi-party protocols is straightforwards.

**Definition 1.1. Response time:** *We say that the response time of a party $A$ in round $i$ of a specific run $\sigma$ of a protocol $\Pi$ is $t$, if $\Pi$ in run $\sigma$ tells $A$ to send a message in round $i$, and $t$ is the length of the time interval starting from the time $B$ sent its message in round $i-1$ and ending at the time $B$ received $A$'s response of round $i$. (If $A$ is not supposed to send a message in round $i$, then its response time is $0$ for round $i$.) The response time of $A$ in run $\sigma$ of protocol $\Pi$ is the maximum over all rounds $i$ of $A$'s response time in round $i$.*

**Definition 1.2. Responsive time complexity:** *We say that a protocol $\Pi$ has responsive round complexity $m$ with respect to Party $A$, if for any possible run $\sigma$ of Protocol $\Pi$, the overall communication time does not exceed $t \cdot m$ where $t$ is $A$'s response time in Run $\sigma$ of Protocol $\Pi$.*

Note that if all rounds are equally long in all runs of the protocol, then the responsive round complexity measure with respect to each of the two parties equals the (standard) round complexity measure[1].

Our primary interest in this notion is for cases when a party actually delays its messages to gain something; our goal is to develop protocols in which purposeful delays merely punishes the delayer. However, we note that the guarantee of our protocol also holds for networks with unstable or heterogeneous communication links. Parties will be (unfairly) punished for network delays beyond their control, but this punishment will be roughly proportional to the inherent delays. Thus, someone with a slow connection will obtain slow service, but will not be starved, as would be the case if a protocol simply timed-out on slow participants.

We demonstrate the usefulness of the new notion by using it for analyzing concurrent zero-knowledge protocols and constructing a new protocol with an (almost) optimal responsive round complexity.

### 1.2 Concurrent zero knowledge

Zero-knowledge interactive proofs as presented by Goldwasser, Micali, and Rackoff [16] are proofs that yield no knowledge but the validity of the proven assertion. These proof systems have proven important tools for a variety of cryptographic applications. However, the original definition of zero-knowledge considers security only in a restricted scenario in which the prover and the verifier execute the proof disconnected from the rest of the computing environment.

In recent years, several papers have studied the affect of a modern computing environment on the security of zero-knowledge. In particular, many computers today are connected through networks in which connections are maintained in parallel asynchronous sessions. It would be common to find several connections (such as FTP, Telnet, an internet browser, etc.) running together on a single workstation. Can zero-knowledge protocols be trusted in such an environment?

Zero-knowledge in a concurrent environment was first explored by Feige [12], and by Dwork, Naor, and Sahai [10]. Dwork, Naor and Sahai denoted zero-knowledge protocols that are robust to asynchronous composition *concurrent zero-knowledge* protocols. They observed that several known zero-knowledge proofs, with a straightforward adaptation of their original simulation to the asynchronous environment, may cause the simulator to work exponential time. Thus, it seems that the zero-knowledge property does not necessarily carry over to the asynchronous setting.

Kilian, Petrank, and Rackoff [19] gave the first lower bound for concurrent zero-knowledge, showing that any language that has a 4-round blackbox concurrent zero-knowledge interactive proof or argument is in BPP. Thus, a large class of known zero-knowledge interactive proofs and arguments for non-trivial languages do not remain zero-knowledge in an asynchronous environment. Rosen

[1] Here, we adopt the measure by which a round consists of two messages: one from Party $A$ to Party $B$ and the other is the response of $B$ to $A$.

[25] has improved this lower bound from from 4 rounds to 7. Canetti, Kilian, Petrank and Rosen [5] have substantially improved the lower bound to $\tilde{\Omega}(\log k)$.[2] The parameter $k$ is the security parameter. A polynomial in $k$ bounds the length of the inputs, the number of proofs that may start concurrently, and the time complexity that the parties spend in the protocol.

On the other hand, Richardson and Kilian [24] exhibited a concurrent zero-knowledge proof for any language in NP. Their protocol requires polynomially many rounds in $k$. Kilian and Petrank [18] substantially narrow the gap between the upper bound and the lower bounds. Using a different simulator, they provide a tighter security analysis for the Richardson-Kilian protocol, and show that it remains concurrent zero knowledge when run with only $\omega(\log^2 k)$ rounds.

How do these results translate to responsive round complexity? Zero-knowledge is about providing security to the prover. Thus, we expect the prover to follow the protocol and not delay its answers. The verifier is the bad guy, who may choose to deviate from what the protocol dictates in order to get knowledge from the prover. Thus, the verifier may delay its answers, and we would like to investigate how protocols behave in this case. It seems fair to provide quicker service (overall communication time) to verifiers that respond quickly and do not delay their answers. Verifiers that do delay their answers may get an overall slower run of the protocol. Responsive round complexity guarantees that the overall communication time is proportional to the longest delay of the verifier.

Looking at the best known upper bound protocol in [18], it is easy to see that the responsive round complexity with respect to the verifier is equivalent to its round complexity in stable networks. The verifier may simply keep its response time steady, and then the two measures equate. This is the best known protocol with respect to responsive time complexity, and it has responsive time complexity of any function $m$ satisfying $m = \omega(\log^2 k)$.

If we look at the best known lower bound in [5], it provides a specific schedule such that if the protocol does not have enough rounds, no black box simulator can simulate it in this schedule. In the demonstrated schedule each verifier has its own response time, but each of the verifiers does not change its response time during the proof. Thus, the lower bound holds also for responsive round complexity, and we cannot do better than $\tilde{\Omega}(\log k)$.

In this paper we present a new concurrent zero-knowledge proof for all languages in NP that has responsive round complexity $m$ for any function $m = \omega(\log k)$. Namely, the responsive round complexity of this protocol can be set to any function asymptotically larger than $\log k$. Thus, we get an algorithm whose responsive round complexity is almost optimal (up to a factor of at most $O(\log^2 \log k)$). Thus, any verifier that does not delay its messages (or even just does not change the delay from round to round) is guaranteed a round complexity of $\tilde{O}(\log k)$. Verifiers that do delay their messages get a protocol whose running time is at most $\tilde{O}(\log k)$ times the longest delay they choose to use.

[2] The "twiddle" notation neglects multiplicative factors that are polylogarithmic in the main term.

In a recent breakthrough work, Barak [1] gives a concurrent zero-knowledge for NP which is not black-box and requires only a constant number of rounds. A slight drawback of this protocol is that the maximum number $k$ of concurrent sessions tolerated must be predetermined in advance, and the communication required by this protocol is proportional to the chosen polynomial. Our protocol, like previous black-box protocols, is robust against any polynomial number of concrrent sessions, and its overall communication is independent of the number of sessions.

### 1.3 Techniques

One set of previous protocols [24,18] ignore the timing of the messages and consider only their order; black-box simulatable protocols exist in the general model, though with high round complexity. Another approach is taken by the protocols of [10,11]. In this approach, strong restrictions are enforced (or assumed) on the ratio between the slowest and longest response times, simplifying the task of producing a simulation, and allowing for constant-round protocols. One interpretation (and implementation) of this restriction is that verifiers with slow response times are treated as malicious; their responses are rejected.

Our approach is intermediate between these two approaches. As with the latter approach, we do take response times into consideration, but as with the first approach we place no restriction on these delays. Instead, we monitor the delays and "punish" verifiers with long delays, though in a proportionate fashion. Each verifier has an associated response time that is doubled when the verifier does not respond in time. Thus, there are $O(\log k)$ sets of verifiers, each set containing verifiers responding at around the same time. The prover may delay its answers to each verifier to match its delays with those of the verifier. For each set, we use techniques similar to those in [10,11] to simulate the conversations with verifiers in this set. We then show that simulating the $O(\log k)$ sets together is still doable in polynomial time.

### 1.4 Contributions

The first contribution of this work is in proposing the notion of responsive round complexity. We feel that this notion may be useful in settings when one of the parties may gain something from delaying its responses. A guarantee on the responsive round complexity provides a guarantee on the time complexity such that each party "gets what it deserves".

Our second contribution is in providing a concurrent blackbox zero-knowledge protocol with almost optimal responsive round complexity. Our design uses the protocol of [24,18] as a subroutine; its main technical contribution is a method for restarting this subroutine so as to obtain a better protocol in practice.

### 1.5 Related work

Our notion of responsive-round complexity is of course related to the vast literature on distributed algorithms, and continues the program of studying zero-knowledge in distributed settings.[3] We do point out the difference between our notion and the most commonly used distributed model. In the standard distributed model, an adversary can speed up responses in worst case fashion; we require that all parties give a "correct" output by the end of the protocol. In our model, we impose additional requirement on when individual parties finish (give a final output); parties whose responses have been sped up may have to finish long before the end of the protocol as a whole. (Here, the "protocol" is the collective set of interactive proofs)

Several recent works have overcome the difficulty of the asynchronous setting by putting limits on the asynchronisity of the system (timing assumptions) [10,11,6,9] or by making some set-up assumptions on the environment (such as a public key infrastructure) [7,4].

### 1.6 Terminology

Some words on the terminology we are using. By zero-knowledge we mean *computational* zero-knowledge, i.e., the distribution output by the simulation is polynomial-time indistinguishable from the distribution of the views of the verifier in the original interaction. Our proof is black-box zero-knowledge. The proof will be perfectly sound, i.e., we will construct an interactive proof, yet it will be possible to run the prover in polynomial time given a witness to the NP assertion that the prover is making.

### 1.7 Guide to the paper

In Section 2 we go over the preliminaries. We state our main result in Sect. 3. We provide an overview on the protocol and proof in Sect. 4. The protocol itself is presented in Sect. 5, the simulator to the protocol is presented in Sect. 6, and the analysis of the simulator is given in Sect. 7.

## 2 Preliminaries

### 2.1 Zero-knowledge proofs

Let us recall the concept of interactive proofs, as presented by [16]. For formal definitions and motivating discussions the reader is referred to [16].

**Definition 2.1.** *A protocol between a (computationally unbounded)* prover $P$ *and a (probabilistic polynomial-time)* verifier $V$ *constitutes an* interactive proof *for a language $L$ if there exists a negligible function $\varepsilon$ such that*

[3] Indeed, we would not be surprised if quite similar definitions have been proposed in this literature.

– **Completeness:** *If* $x \in L$ *then* $\Pr[(P,V)(x) \text{ accepts}] \geq 1 - \varepsilon(|x|)$.
– **Soundness:** *If* $x \notin L$ *then for any prover* $P^*$

$$\Pr[(P^*,V)(x) \text{ accepts}] \leq \varepsilon(|x|).$$

Brassard, Chaum, and Crépeau [2] introduced a modification of interactive proofs, called *arguments*, in which the prover is also polynomial time bounded. Thus, the soundness property is modified to be guaranteed only for probabilistic polynomial time provers $P^*$.

Let $(P,V)(x)$ denote the random variable that represents $V$'s view of the interaction with $P$ on common input $x$. The view contains the verifier's random tape as well as the sequence of messages exchanged between the parties.

We briefly recall the definition of black-box zero-knowledge [16,23,15,17]. The reader is referred to [17] for more details and motivation.

**Definition 2.2.** *A protocol* $(P,V)$ *is* computational zero-knowledge *(resp.,* statistical zero-knowledge*) over a language* $L$*, if there exists an oracle polynomial time machine* $S$ *(simulator) such that for any polynomial time verifier* $V^*$ *and for every* $x \in L$*, the distribution of the random variable* $S^{V^*}(x)$ *is polynomially indistinguishable from the distribution of the random variable* $(P,V^*)(x)$ *(resp., the statistical difference between* $M(x)$ *and* $(P,V)(x)$ *is a negligible function in* $|x|$*).*

In this paper, we concentrate on blackbox computational zero-knowledge, and use *zero-knowledge* as shorthand for *blackbox computational zero-knowledge*.

### 2.2 Bit commitments

We include a short and informal presentation of commitment schemes. For more details and motivation, see [22]. A commitment scheme involves two parties: The *sender* and the *receiver*. These two parties are involved in a protocol which contains two phases. In the first phase the sender commits to a bit, and in the second phase it reveals it. A useful intuition to keep in mind is the "envelope implementation" of bit commitment. In this implementation, the sender writes a bit on a piece of paper, puts it in an envelope and gives the envelope to the receiver. In a second (later) phase, the *reveal* phase, the receiver opens the envelope to discover the bit that was committed on. In the actual digital protocol, we cannot use envelopes, but the goal of the cryptographic machinery used, is to simulate this process.

More formally, a commitment scheme consists of two phases. First comes the *commit* phase and then we have the *reveal* phase. We make two security requirements which (loosely speaking) are:

**Secrecy:** At the end of the *commit phase*, the receiver has no knowledge about the value committed upon.
**Binding property:** It is infeasible for the sender to pass the commit phase successfully and still have two different values which it may reveal successfully in the reveal phase.

Various implementations of commitment schemes are known, each has its advantages in terms of security (i.e., binding for the receiver and secrecy for the receiver), the assumed power of the two parties etc.

Two-round commitment schemes with perfect secrecy can be constructed from any collection of claw-free permutations; see [22]. It is shown in [2] how to commit to bits with statistical security, based on the intractability of certain number-theoretic problems. Dåmgard, Pedersen and Pfitzmann [8] give a protocol for efficiently committing to and revealing strings of bits with statistical security, relying only on the existence of collision-intractable hash functions. This scheme is quite practical and we adopt it for the verifiers in our protocol. For the prover, we use a commitment scheme whose binding is information theoretic and security is computational. Such schemes can be constructed from any one-way function, see [20]. For simplicity, we simply speak of committing to and revealing bits when referring to the protocols of [8] for the verifier and [20] for the prover. We will need to use the properties of the commitment schemes in the concurrent setting.

**Theorem 2.3.** *The security of the bit commitments in [20] and [21] holds also in the concurrent setting.*

*Proof.* By definition, the binding property must be robust to asynchronous composition. Otherwise, the committer may play a mental game in which his real stand-alone commitment is part of an asynchronous game which he simulates, and then defeat the binding property in the normal stand-alone world.

As for the secrecy, a similar argument may be more complicated, since the receiver cannot simulate the behavior of the committer. Specifically, the committer has some information that the receiver does not have: the value of the committed string, which may be used in the other commitments. However, in our proof, the committer commits on uniformly chosen random strings. (And on nothing else.) Thus, if the committer follows the protocol, then the receiver is able to simulate the rest of the environment and the above argument holds for secrecy as well. □

### 2.3 Witness Indistinguishability

Witness indistinguishable proofs were presented in [13]. The motivation was to provide a cryptographic mechanism whose notion of security is similar though weaker than zero-knowledge, it is meaningful and useful for cryptographic protocols, and the security is preserved in an asynchronous composition. A witness indistinguishable proof is a proof for a language in NP such that the prover is using some witness to convince the verifier that the input is in the language, yet, the view of the verifier in case the prover uses witness $w_1$ or witness $w_2$ is polynomial time indistinguishable. Thus, the verifier gets no knowledge on which witness was used in the proof. The formal definition follows. For further discussion and motivation the reader is referred to [13].

### 2.4 Blackbox simulation

The initial definition of zero-knowledge [17] requires that for any probabilistic polynomial time verifier $\hat{V}$, a simulator $S_{\hat{V}}$ exists that simulates $\hat{V}$'s view. Oren [23] proposes a seemingly stronger, "better behaved" notion of zero-knowledge, known as *black-box* zero-knowledge. The basic idea behind black box zero-knowledge is that instead of having a new simulator $S_{\hat{V}}$ for each possible verifier, we have a single probabilistic polynomial time simulator $S$ that interacts with each possible $\hat{V}$. Furthermore, $S$ is not allowed to examine the internals of $\hat{V}$, but must simply look at $\hat{V}$'s input/output behavior. That is, it can have conversations with $\hat{V}$ and use these conversations to generate a simulation of $\hat{V}$'s view that is computationally indistinguishable from $\hat{V}$'s view of its interaction with $P$.

For further definitions and motivations the reader is referred to [23]

### 2.5 Concurrent zero-knowledge

Following [10], we consider a setting in which a polynomial time adversary controls many verifiers simultaneously. The adversary $\mathcal{A}$ takes as input a partial conversation transcript of a prover interacting with several verifiers concurrently, where the transcript includes the local times on the prover's clock when each message was sent or received by the prover. The output of $\mathcal{A}$ will be a tuples of the form $(V, \alpha, t)$, indicating that $P$ receives message $\alpha$ from a verifier $V$ at time $t$ on $P's$ local clock. The adversary may either output a new tuple as above, or wait for $P$ to output its next message to one of the verifiers. The time that is written by the adversary in the tuple, must be greater than all times previously used in the system (by messages sent to $P$ or by $P$). The view of the adversary on input $x$ in such an interaction (including all messages and times, and the verifiers random tapes) is denoted $(P, \mathcal{A})(x)$.

**Definition 2.4.** *We say that a proof or argument system $(P, V)$ for a language $L$ is (computational)* concurrent zero-knowledge *if there exists a probabilistic polynomial time oracle machine $S$ (the simulator) such that for any probabilistic polynomial time adversary $\mathcal{A}$, the distributions $(P, A)(x)$ and $S^{\mathcal{A}}(x)$ are computational indistinguishable over the strings that belong to the language $L$.*

In what follows, we will usually refer to the adversary $\mathcal{A}$ as the *adversarial verifier* $V^*$ or just the *verifier* $V^*$. All these terms mean the same.

In our setting, the simulator will simulate a predetermined time interval which is polynomial in $k$. We assume that while rewinding the verifier, the simulator may also set its clock to the required rewound time.

### 2.6 The complexity parameters

In this paper, we simplify the discussion by using a single security parameter $k$. Our proof has (in worst case) $\omega(\log^2 k)$ rounds and it has responsive round complexity $\omega(\log k)$. The zero-knowledge simulation is guaranteed for a polynomial (in $k$) number of concurrent proofs. Also, the running time of the protocol

is polynomial in $k$. We will measure time by the smallest time units that are relevant in this setting. For example, one may think of the time unit as the minimal time a round in the protocol may take. But we may also use a much smaller time unit: the time of a computer cycle. In any of these time units, it holds that the running time of the protocol is polynomial (in $k$).

## 3 Main result

Our main result is the existence of black-box concurrent zero-knowledge interactive proof for all languages in NP with responsive round complexity $m$ for any $m$ satisfying $m = \omega(\log k)$. We state this explicitly in the following theorem.

**Theorem 3.1.** *Assume there exist secure two-round commitment schemes with statistical secrecy and secure two-round commitment schemes with statistical binding (such schemes follow from the existence of a family claw-free permutation pairs). Let $k$ be a complexity parameter bounding the size of the input. The verifier is polynomial time in $k$, and the concurrent proof may contain a polynomial (in $k$) number of proofs concurrently. Then there exists a black-box concurrent zero-knowledge interactive proof for all languages in NP, with:*

- *responsive round complexity $m(k)$, for any function $m(k)$ satisfying $m(k) = \omega(\log k)$, and*
- *a worst case round complexity of $m(k) \cdot \log k$.*

## 4 Overview of protocol and proof

We start with the protocol in [24,18]. We choose the following parameters for this protocol: the preamble consists of $m$ rounds for $m = \omega(\log k)$ (recall that a round consists of a message sent from the prover to the verifier followed by a response of the verifier). The body of the proof consists of a low error, constant round, auxiliary-input witness-indistinguishable interactive proof for NP in which the prover can be efficient given the witness to the proven assertion. The zero-knowledge protocol of [14] will do.

When a new copy of the protocol is initiated by the verifier, the verifier in the new protocol is associated with a response time which is initially the minimal possible response time, say the time of a computer cycle. When the verifier fails to respond within this time, the time associated with this verifier is doubled and the verifier is notified that it must start again with the doubled time. In this case we say that the verifier has been reset and has gone one level up. This may happen at most $O(\log k)$ times since at some such level the response time becomes greater than the running time of the adversary, or bigger than the time interval that has to be simulated. The verifiers may be viewed as working in levels of responsiveness. Level $i$ contains all verifiers with response time at most $\beta_i = 2^i$ and greater than $\beta_{i-1}$. The prover treats each verifier independently in light of its associated response time or level. For each verifier, the prover delays its answer according to its associated delay $\beta$ in a manner yet to be discussed.

The completeness and soundness of the interactive proof hold as in [24,18]. The worst case number of rounds for this protocol happens when the verifier goes through $m$ steps in each level and then delays its last message and is reset while going up to the next level. This yields $m \cdot O(\log k)$ rounds in the worst case. But since each level takes double the time of the previous level, the overall interaction time is dominated by the time of the highest level interaction, and is $O(m)$.

It remains to prove that the interactive proof is zero-knowledge. The delays imposed by the prover are similar to those suggested in [10]. Thus, simulating all protocols at the same level becomes possible in a way similar to that in [10]. In fact, these verifiers may be viewed as adhering to timing constraints. The delay imposed by the prover are not more than twice $\beta_i$ for a verifier in level $i$ and thus, do not increase the protocol time too much. It remains to show that rewinding protocols at higher levels do not force too many rewinding at lower levels. This is obtained with some care in the setting of the prover delays and by the fact that there are at most a logarithmic number of levels.

The reason we need a logarithmic number of rounds and cannot do with a constant number of rounds for each level as in [10] is the relation between the various levels. We allow ourselves one rewind only to any interval we wish to rewind. Any other constant will do, but rewinding a super-constant number of times (or polynomial as in [10]) will make the overall simulation time super-polynomial. Note also that this is an inherent problem since the lower bound in [5] uses verifiers that in each specific copy of the proof do not modify their response time. Thus the lower bound holds also for responsive round complexity and we cannot do with asymptotically less than $\log k / \log\log(k)$ responsive round complexity.

## 5 The zero-knowledge protocol

We start by presenting the protocol. It consists of a preamble of $m$ rounds where $m$ is any function satisfying $m = \omega(\log k)$ and a body consisting of a (not concurrent) constant round zero-knowledge proof. If this were the full picture, we would get that the overall number of rounds is dominated by $m$ and is thus almost logarithmic. However, we sometimes let the prover say "RESET". This happens only during the preamble, and is caused by a long delay in the verifier response. When such a delay occurs, the protocol starts from the beginning with a delay parameter doubled. At this point we say the the proof has gone up one *level*. Generally a proof is at level $i$ if it has gone through $i$ resets.

To see that the overall round complexity is $m \cdot O(\log k)$ it is enough to note that the maximum number of resets is logarithmic. This is true since the delay can only be doubled a logarithmic number of times. The logarithm is in the length of the simulated period. We denote this length by $\Delta$ and measure it in units of $\beta_0$, i.e., the time of a computer cycle. In Figure 1 we describe the protocol. This is the protocol presented in [24,18] enhanced with time monitoring and possible

| | |
|---|---|
| Step V-0: | V $\rightarrow$P: V Selects $m$ strings, $v_1, \ldots, v_m \in \{0,1\}^n$ uniformly and independently at random, and send Commit $(v_1), \ldots,$ Commit $(v_m)$ to the prover. |
| Step P-1: | P $\rightarrow$V: Send Commit $(p_1)$ exactly $T$ after the Step V-0 message was sent. |
| Step V-1: | V $\rightarrow$P: Reveal $v_1$. |
| $\vdots$ | |
| Step P-$j$: | P $\rightarrow$V: If V's message from Step V-$(j-1)$ was received more than $T$ time units after P's message from Step P-$(j-1)$ was sent then goto RESET. Else, send Commit $(p_j)$ exactly $2T$ time units after P's round $(j-1)$ message was sent. |
| Step V-$j$: | V $\rightarrow$P: Reveal $v_j$. |
| $\vdots$ | |
| Step P-$m$: | V $\rightarrow$P: If V's message from Step V-$(m-1)$ was received more than $T$ time units after P's message from Step P-$(m-1)$ was sent then goto RESET. Else, send Commit $(p_j)$ exactly $2T$ time units after P's round $(j-1)$. |
| Step V-$m$: | V $\rightarrow$P: Reveal $v_m$. |
| Proof body: | P waits $T$ time units and then proves to V in zero-knowledge that $x \in L$ or that $\exists i$, $1 \leq i \leq m$, such that $p_i = v_i$.<br>(No delays or time monitoring is used during the course of this proof.) |
| End of proof | |
| | |
| RESET: | |
| P $\rightarrow$V: | A reset message with parameter $2T$. Both P and V continue by setting $T = 2T$ and starting the protocol from Step V-0 again. |

**Fig. 1.** The protocol

resets. All commitments from the verifier to the prover are statistically secret and all commitments from the prover to the verifier are statistically binding.

**Theorem 5.1.** *If the zero-knowledge proof used in the body of the protocol has completeness error $\epsilon_c$ and soundness error $\epsilon_s$ then our interactive proof as in Fig. 1 has completeness error $\epsilon_c$ and soundness error at most $\varepsilon_s + \varepsilon$ for some negligible fraction $\varepsilon$.*

*Proof.* Clearly, the completeness error cannot increase. As for the soundness, the prover may gain extra strength by managing to set $p_i = v_i$ for some $1 \leq i \leq m$. However, since the verifier is using statistically hiding commitment scheme this may happen with negligible probability only, and we are done. □

**Lemma 5.2.** *The protocol has responsive round complexity $5m$.*

*Proof.* We show that it holds for the preamble. The additional constant number of rounds in the body of the proof cannot increase the responsive round complexity since the prover answers with no delays at that stage of the protocol.

Consider a proof that ended the preamble at level $\ell$, i.e., had $\ell$ resets. (We will discuss later the case that the preamble has not ended at all within the time

$\Delta$.) If $\ell = 0$ then the proof had $m$ rounds, the length of each equals the minimum possible response time, that was actually matched by the verifier. Otherwise, we have $\ell \neq 0$. The proof was last reset at level $\ell - 1$ which means that the verifier did not respond within time $\beta_{\ell-1} = 2^{\ell-1} = \frac{1}{2}\beta_\ell$. Thus the response-time of the verifier is at least $\beta_{\ell-1}$. We now compute the overall communication time and show that it is smaller than $5m \cdot \beta_{\ell-1}$.

At each of the levels $i = 1, 2, \ldots, \ell - 1$ the protocol ran for at most $m$ rounds. At level $\ell$ we assume it finished the preamble and thus had $m$ rounds. Summing over all the communication times during the preamble we get that the communication time is bounded by

$$\sum_{i=1}^{\ell} m \cdot \beta_i \;\; = m \cdot \sum_{i=1}^{\ell} 2^i \;\; \leq \;\; m \cdot 2^{i+1} \;\; = \;\; 4 \cdot m \cdot \beta_{\ell-1}.$$

We bound the additional communication time of the proof body by $m\beta_{\ell-1}$. This is correct for the constant round body if the verifier does not pose a delay longer than $\beta_{\ell-1}$; if it does, the responsive round complexity may only decrease.

Last, we deal with the case that the verifier does not finish. We assume that the simulation time $\Delta$ is much larger than the running time of the adversarial verifier. Thus, a particular verifier that has not yet responded will never respond and its responsive round complexity is much better than $5m$. □

We next show that the protocol is concurrent zero-knowledge, by presenting a simulator for the concurrent interaction.

## 6 The simulator

We present a black box simulation of the above protocol. We assume the worst, i.e., that there is one adversary that controls all verifiers (whose number is polynomial in $k$). This adversary deviates from the protocol as it wishes and is limited only by being a polynomial time machine. The simulator interacts with this adversary (or with these verifiers) and its goal is to produce a transcript distribution which is indistinguishable from the real interaction between the adversary and the original prover $P$. Note that each message in the transcript is associated with a time telling when it is produced after the beginning of the interaction.

The simulator simulates the body of the proof simply by playing the real prover. The reason it may do that is that it rewinds each of the verifiers so that it manages to get a round $i$ in which $p_i = v_i$. After that we say that this particular copy of the proof has been "solved", or that this particular verifier has been *neutralized*. Our goal is to ensure that there will be enough rewinding so that all proofs will be solved, while taking care that the rewinding does not exceed polynomial time.

The difficulty in the construction and in describing the simulator lies in the rewinding schedule. Other than that the operation of the simulator is quite simple. The simulator runs the adversary on a randomly chosen random string

while performing all rewinds in the rewind schedule. The simulator breaks the entire sequence of time steps into sections. Each section is simulated twice by the simulator. The first run is used to obtain information, and the second run is used to produce the actual output transcript. During simulation of each such section, the simulator recursively divides it into smaller subsections.

During the first time a section is simulated, the simulator records the strings revealed by the verifiers during this run. Then, while running the second run of the rewind, the simulator solves all proofs that may be solved by setting $p_i$ to equal $v_i$ for known values of $v_i$'s. The second run of the section is used to produce the transcript obtained thus far. When a body of a proof arrives, if the proof has been solved, then the simulator acts as the prover while proving the existence of $i$ such that $p_i = v_i$ (the simulator has a witness to this fact). If the proof has not been solved, the simulation aborts and declares failure.

We will show that the probability that any of the proofs remains unsolved is negligible. Thus, the simulator rarely fails. When it does not fail, its output will be indistinguishable from the real interaction. One difference between the simulated transcripts and the real ones is in the preambles: in the simulation there is an $i$ with $p_i = v_i$. But by the secrecy of the commitment schemes this difference cannot be detected by a polynomial-time bounded machine. Note that these strings are never revealed, avoiding difficulties arising when partial subsets are revealed. The second difference is in the witness used in the bodies of the proofs. However, a zero-knowledge proof is witness indistinguishable. This property is preserved in a concurrent setting and is thus indistinguishable by a polynomial time distinguisher.

It remains to show that there exists a rewinding schedule by which the simulator is efficient and still all proofs are solved with overwhelming probability.

### 6.1 The rewinding schedule

The schedule of the rewinds is given as a pseudo-code in Fig. 2 and is illustrated in Fig. 3.

The X axis represents the time (as viewed by the verifiers or listed in the output transcript produced by the simulator), and the numbers in the graph represent the X coordinate (=the time) of an event. The Y axis represents the order of events of the simulator itself. The advances of the simulation are shown as thick arrows, whereas the rewinds are shown as thin backward arrow.

In the example of Fig. 3 the top-level run has exactly two recursive sections. At the top level this is not always the case, but in any other level the recursion is invoked exactly twice. The top level of this run is $\log \Delta$ (all logarithms are base 2), where $\Delta$ is the length of the interval we simulate. In the example $\Delta = 4$ and the top level is 2. The first section starts at the beginning and ends when the simulator advances from 1 to 2 after the after the seventh rewind (the second $(1 \leftarrow 3)$). The second section begins in the advancement from 2 to 4 and ends at the end of the simulation.

During the run of the top-level sections there are also rewinds of lower levels. In this example there is only one lower level: level 1. For each rewind of level

```
0:  // Recall that Δ is the overall simulation time interval.
1:  top_level = log(Δ)
2:
3:  // This is a recursive algorithm. Top-level call follows:
4:  simul(0, Δ, top_level)
5:  output transcript
6:
7:  // Definition of recursive function:
8:  simul(location, length, level)
9:    β = 2^level
10:   if (level < 1)
11:     // here comes the simulation of interaction with V* for time interval length
12:     return
13:   else
14:     // recursively run lower-level simulations
15:     for i = 0 to length/β - 1
16:       simul(location + i·β, β,level-1)
17:       simul(location + (i+1)·β, β,level-1)
18:       rewind_to_location(location + i·β)
19:       simul(location + i·β, β, level-1)
20:     end for
21:   end if
```

**Fig. 2.** Description of the Rewinding Schedule

$\ell$ there are 6 rewinds of level $\ell - 1$ (4 before the actual level-$\ell$ rewind, and two after it). Thus before the first level-2 rewind ($(0 \leftarrow 4)$) there are 4 level-1 rewinds ($(0 \leftarrow 2)$, $(1 \leftarrow 3)$, $(2 \leftarrow 4)$ and $(3 \leftarrow 5)$) and more two after it ($(0 \leftarrow 2)$ and $(1 \leftarrow 3)$).

Note that the rewinding schedule does not depend on the schedule of proofs as determined by the adversarial verifier. It may be the case that no proof ran and the simulator would still behave the same. The rewinding schedule depends on the time only.[4]

### 6.2 The effects of the rewinds

Each proof may start its $i$-th level run in an arbitrary point in time. However, by the delay of $\beta_i$ imposed by the prover, they all have the same look during the preamble: The prover sends a message, then the verifier responds within time $\beta_i$ and then the prover sends its next message exactly $2\beta_i$ time units after its

[4] We remark that one may obtain better efficiency by checking if the rewind is helpful to the simulation and avoid rewinding it it's not. Even if one does not try to check the messages, a scrutiny of the schedule may lead to other improvements. However all we care about is that the simulation is polynomial time and this is guaranteed by our simple non-optimized simulation procedure.

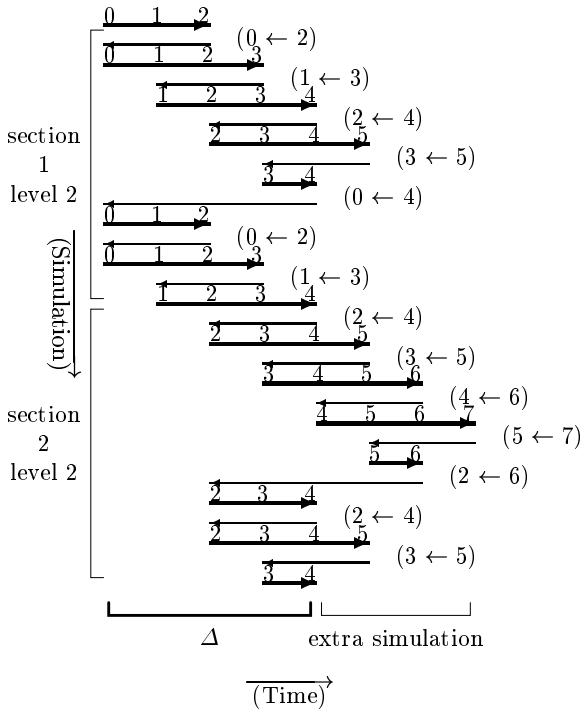


**Fig. 3.** The rewind schedule of two rounds **of** level 2

previous message. Thus, the time between two prover messages is always $2\beta_i$ and the response time of the verifier is less than $\beta_i$.

A *rewind* operation at level $i$ that makes the simulator run the interval $(T, T+2\beta_i)$ twice is meant to solve all proofs of level $i$ in which the verifier sent a preamble message "`Reveal` $v_\ell$" (for some $1 < \ell \leq m$) in response to a prover message that was sent in-between the times $T$ and $T+\beta_i$. Note that in this case the verifier must respond before $T+2\beta_i$ and thus simulator has learned the value of $v_\ell$. Since $p_\ell$ is still within the rewind. The simulator may modify the commitment on $p_\ell$ in the second run of the rewind and commit on $p_\ell = v_\ell$. For example, the rewind $(0 \leftarrow 4)$ may solve proofs that run in the top level and whose prover has sent a message between the times 0 and 2.

Had this always worked, we wouldn't need so many preamble rounds. A couple of them would have been enough. However, here is what may go wrong: the verifier may delay its answer in the first run of the rewind interval, thus getting a reset message from the prover (simulator), yet, in the second run, provide an answer in time. In this case, the simulator would not know the value of $v_\ell$ in the second run since it was not exposed in the first run. The second run is the one that prevails and written to the final transcript. Thus, solving the proof in this round of the preamble fails in this case. In Sect. 7 below, we argue that this happens with constant probability in each round and with negligible

probability in all the $m$ rounds of the preamble. Note that setting $p_\ell = v_\ell$ in any one of the rounds suffices to solve the proof.

It remains to analyze the probability that the simulator succeeds in solving each of the proofs (before getting to the proof body) and to verify that the rewinding schedule results in a polynomial time simulator. This analysis is provided in the following section.

# 7 Analysis of the Simulator

## 7.1 Efficiency

We start by showing that the rewinding schedule of Section 6.1 results in a polynomial time simulator. Note that each step of the simulation, i.e., committing on strings, revealing them, and playing the prover in the proof-body are all polynomial time. Thus, if the rewinding is polynomial time, we get that the whole algorithm is efficient. We will actually show that the number of rewinds is polynomial. Since each rewind time is polynomial this is enough.

**Lemma 7.1.** *The overall number of rewinds during the simulation run on time interval $\Delta$ is at most $\Delta^3$.*

*Proof.* We use the recursive description of the rewinding schedule as in Figure 2. Consider a run of a time interval $t$ at level $\ell$. Using the notation of Figure 2 this is a run of $simul(location, t, \ell)$. Note that the number of rewinds is independent of the location. Thus, we denote the number of rewinds in this run by $X(t, \ell)$. In a run of $t$ time units at level $\ell$ there are $\frac{t}{\beta_\ell}$ iterations of the main loop of the *simul* procedure. In each iteration there is a level-$\ell$ rewind and 3 calls to $simul(\cdot, \beta_\ell, \ell - 1)$ are performed (recall that $\beta_\ell = 2^\ell$). Thus,

$$X(t, \ell) \;\leq\; \frac{t}{\beta_\ell} \cdot \left(1 + 3 \cdot X\left(2\beta_{\ell-1}, \ell - 1\right)\right).$$

This recursion inequality gives the bound: $X(t, \ell) \;\leq\; \frac{t}{\beta_\ell} \cdot 7^{\ell-1}$. At the top level, $t = \Delta, \ell = \log \Delta$, and we obtain

$$\frac{t}{\beta_\ell} \cdot 7^{\ell-1} = \frac{\Delta}{2^{\log \Delta}} \cdot 7^{\log \Delta} = 7^{\log \Delta} \leq \Delta^3,$$

as required. □

## 7.2 Indistinguishability

We show that no polynomial-time algorithm can distinguish the output of the simulator from $V^*$'s view of its interaction with the original prover, $P$.

Assume, first, that the simulator always manages to solve all proofs before getting to the bodies of the proofs. We show later that this assumption holds with overwhelming probability.

We separate the discussion of the preambles and the proof bodies. The difference in the *preambles* is that in the simulation one or more of the rounds has $p_i = v_i$. In the real interaction, this seldom happens. The difference in the bodies is that the simulator always proves that "$\exists i$ such that $p_i = v_i$" whereas the original prover (almost) always proves that $x \in L$.

The prover never opens its commitments on the $p_i$'s. By the secrecy of the commitment scheme, a polynomial time distinguisher cannot tell between preambles generated by the simulators and real preambles. Since this is the case, the adversarial verifier itself cannot distinguish between the first and the second runs of a rewind. We will use this fact to show that the simulator solves all proofs with high probability.

Finally, the *proof bodies* are witness indistinguishable. By [13] this property holds also in the concurrent setting. Thus, an efficient distinguisher cannot tell between using a witness to "$\exists i$ such that $p_i = v_i$" and using a witness to "$x \in L$" and we are done.

It remains to show that the simulator may fail to solve one of the proofs only with negligible probability. We first argue (in Claim 7.1) that for each proof, each round of its preamble that appears in its final level is rewound. We then argue (in Claim 7.2) that a rewind of such a round does not solve the proof with probability at most $1/3$. Since all rewinds are rewound independently, and since solving the proof in one of them is enough, and since there are $m = \omega(\log k)$ such rewinds before the body of the proof, we get that the proof remains unsolved with probability $(1/3)^m$, which is negligible. Any of the proofs may be run a polynomial number of times by the simulator (since intervals are rewound) and there are a polynomial number of proofs. By the summation bound, the probability that any of these proofs is not solved by the end of the preamble remains negligible.

**Claim 7.1.** *If a proof preamble terminates at level $\ell$, then each of its $m$ rounds at level $\ell$ is rewound.*

**Proof Sketch:** By the delays posed by the prover, each round takes exactly $2\beta_\ell$ time units. For a rewind $(T \leftarrow T + 2\beta_\ell)$ to properly rewind a proof round, both the prover's message and the verifier's message have to be within the interval $(T, T + 2\beta_\ell)$.

By the requirement of the $\ell$-th level, the answer of the verifier must arrive within $\beta_\ell$. Thus if the prover has sent $Commit(p_i)$ at the interval $(T, T + \beta_\ell)$, it is guaranteed that the verifier's reply (Reveal $v_i$) will arrive within the rewind interval, and that the next prover message will be sent after the rewind interval ends. Thus the round will be properly rewound by a rewind $(T \leftarrow T + 2\beta_\ell)$.

It remains to show that any message that the prover sends on level $\ell$ has an associated level-$\ell$ rewind. Details are omitted. □

**Claim 7.2.** *If in a proof $\Pi$ at level $\ell$ the prover's message for round $i$ is sent at time $T$, then a rewind $(T' \leftarrow T' + 2\beta_\ell)$, where $0 \leq T' - T < \beta_\ell$, solves $\Pi$ in this rewind with probability at least $2/3$.*

*Proof.* If the verifier answers in time during both runs of the rewind then the proof is solved: the first run reveals the value $v_j$ (for round $j$ of the proof) and in the second run of the rewind the prover may commit on a modified $p_j = v_j$. Note that the verifier cannot modify $v_j$, except with negligible probability, since it is committed to this value as of the first round of the proof $\Pi$. If the verifier does not answer in the second run of the rewind, then it is actually reset into level $\ell + 1$ and this proof does not need to be solved at level $\ell$. The only bad case is when the verifier delays its answer in the first run, but does not delay it in the second run of the rewind. In this case, the simulator does not learn the value of $v_j$ in the first run and thus, cannot set $p_j = v_j$ in the second run.

What is the probability of this bad incident? Since the prover commits to $p_j$, the verifier cannot tell if $p_j = v_j$, it cannot tell between the first and second run, except with negligible probability in which the secrecy of the commitment scheme fails. Suppose the verifier delays it's message beyond $\beta_\ell$ with probability $p$ at the first run. It then delays its message with probability at least $p - \varepsilon$ for some negligible fraction $\varepsilon$ in the second run. The probability that the simulator does not solve the proof is thus at most $p \cdot (1 - p + \varepsilon) \leq 1/4 + \varepsilon \leq 1/3$, and we are done. □

## References


1. Boaz Barak: How to Go Beyond The Black-Box Simulation Barrier. To appear in IEEE, *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, October, 2001.
2. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. JCSS **37** (1988) 156–189
3. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge. Record 99-22, Theory of Cryptography Library (1999) received October 25th, 1999. Supercedes Theory of Cryptography Library Record 99-15.
4. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge (extended abstract). In ACM, ed.: Proceedings of the thirty second annual ACM Symposium on Theory of Computing: Portland, Oregon, May 21–23, [2000], New York, NY, USA, ACM Press (2000) 235–244 see also [3].
5. Canetti, R., Kilian, J., Petrank, E., Rosen, A.: Concurrent zero-knowledge requires $\tilde{\Omega}(\log n)$ rounds. In: Proceedings of the thirty third annual ACM Symposium on Theory of Computing, ACM Press (2001)
6. Crescenzo, G.D., Ostrovsky, R.: On concurrent zero-knowledge with pre-processing. In Wiener, M., ed.: Advances in Cryptology – CRYPTO ' 99. Lecture Notes in Computer Science, International Association for Cryptologic Research, Springer-Verlag, Berlin Germany (1999) 485–502
7. Damgård, I.B.: Efficient concurrent zero-knowledge in the auxiliary string model. In Preneel, B., ed.: Advances in Cryptology – EUROCRYPT ' 2000. Lecture Notes in Computer Science, Brugge, Belgium, Springer-Verlag, Berlin Germany (2000) 418–430
8. Damgård, Pedersen, T.P., Pfitzmann, B.: On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In Stinson, D.R., ed.: Proc. CRYPTO 93, Springer (1994) 250–265 Lecture Notes in Computer Science No. 773.

9. Dwork, C., Naor, M.: Zaps and their applications. In IEEE, ed.: Proceedings of the 41st Annual Symposium on Foundations of Computer Science: proceedings: 12–14 November, 2000, Redondo Beach, California, IEEE Computer Society Press (2000) 283–293
10. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In ACM, ed.: Proceedings of the thirtieth annual ACM Symposium on Theory of Computing: Dallas, Texas, May 23–26, 1998, New York, NY, USA, ACM Press (1998) 409–418
11. Dwork, C., Sahai, A.: Concurrent zero-knowledge: Reducing the need for timing constraints. Lecture Notes in Computer Science **1462** (1998) 442–457
12. Feige, U.: Alternative models for zero knowledge interactive proofs. PhD thesis, Weizmann Institute of science (1990)
13. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In ACM, ed.: Proceedings of the twenty-second annual ACM Symposium on Theory of Computing, Baltimore, Maryland, May 14–16, 1990, New York, NY, USA, ACM Press (1990) 416–426
14. Goldreich, O., Kahan, A.: How to construct constant-round zero-knowledge proof systems for NP. Journal of Cryptology: the journal of the International Association for Cryptologic Research **9** (1996) 167–189
15. Goldreich, O., Krawczyk, H.: On the composition of Zero-Knowledge Proof systems. SICOMP **25** (1996) 169–192
16. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive systems. SIAM Journal of Computing **18** (1989) 186–208
17. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: ACM Symposium on Theory of Computing (STOC '85), Baltimore, USA, ACM Press (1985) 291–304
18. Kilian, J., Petrank, E.: Concurrent zero-knowledge in poly-logarithmic rounds. In: Proceedings of the thirty third annual ACM Symposium on Theory of Computing, ACM Press (2001)
19. Kilian, J., Petrank, E., Rackoff, C.: Lower bounds for zero knowledge on the Internet. In IEEE, ed.: 39th Annual Symposium on Foundations of Computer Science: proceedings: November 8–11, 1998, Palo Alto, California, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, IEEE Computer Society Press (1998) 484–492
20. Naor, M.: Bit commitment using pseudorandomness. Journal of Cryptology **4** (1991) 151–158
21. Naor, M., Yung, M.: Universal one-way hash functions and their cryptographic applications. In: 21th Annual Symposium on Theory of Computing (STOC), ACM Press (1988) 33–43
22. Goldreich, O.: Foundation of cryptography — fragments of a book. Available from the *Electronic Colloquium on Computational Complexity (ECCC)* `http://www.eccc.uni-trier.de/eccc/`, February 1995. (1995)
23. Oren, Y.: On the cunning powers of cheating verifiers: Some observations about zero knowledge proofs. In Chandra, A.K., ed.: Proceedings of the 28th Annual Symposium on Foundations of Computer Science, Los Angeles, CA, IEEE Computer Society Press (1987) 462–471
24. Richardson, R., Kilian, J.: On the concurrent composition of zero-knowledge proofs. Lecture Notes in Computer Science **1592** (1999) 415–431
25. Rosen, A.: A note on the round-complexity of concurrent zero-knowledge. In: CRYPTO: Proceedings of Crypto. (2000)