Concurrent Zero-Knowledge Proofs for NP*

Joe Kilian † Erez Petrank ‡ Ransom Richardson §

November 26, 2001

Abstract

A proof is concurrent zero-knowledge if it remains zero-knowledge when many copies of the proof are run in an asynchronous environment, such as the Internet. It is known that zero-knowledge is not necessarily preserved in such an environment [26, 31, 5]. Designing concurrent zero-knowledge proofs is a fundamental issue in the study of zero-knowledge since known zero-knowledge protocols cannot be run in a realistic modern computing environment. In this paper we first show that there exists a concurrent zero-knowledge proof system for all languages in NP. Our proof system is the first zero-knowledge proof system that retains the zero-knowledge property when copies of the proof are allowed to run concurrently in an asynchronous environment. Furthermore, our proof system has $\tilde{O}(\log^2 k)$ rounds (for a security parameter k), which is almost optimal, as it is shown in [5] that black-box concurrent zero-knowledge requires $\tilde{\Omega}(\log k)$ rounds. The existence of such a proof system solves an important open question about zero-knowledge protocols. Our protocol and analysis establish this fact with complexity that is polynomially related to the lower-bound for black-box zero-knowledge proof systems.

1 Introduction

Zero-knowledge proof systems, introduced by Goldwasser Micali and Rackoff [23], are efficient interactive proofs that yield no knowledge but the validity of the proven assertion. These proofs 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 one instance of 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?

1.1 Previous work

Composing zero-knowledge proofs in asynchronous environment was first studied by Feige [13], and explored in a more rigorous setting by Dwork, Naor, and Sahai [11]. Dwork, Naor and Sahai denoted

*This paper is a join of two works. The preliminary versions of these works appeared in the Proceeedings of *Advances in Cryptology - EUROCRYPT '99*, May 1999, Lecture Notes in Computer Science Vol. 1592 Springer 1999, pp. 415-431, and in the *Proceedings of the thirty third annual ACM Symposium on Theory of Computing.* ACM Press, 2001.

†Yianilos Labs, joe@pnylab.com. Work done in part while at the NEC Research Institute.

‡Dept. of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel. erez@cs.technion.ac.il. This research was supported by Technion V.P.R. Fund - N. Haar and R. Zinn Research Fund.

§Groove Networks, 800 Cummings Center, Beverly, MA 01915. rrichard@groove.net.

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 [26], building on techniques from [19], gave the first lower bound for concurrent zero-knowledge, showing that any language that has a 4-rounds concurrent black-box 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 [31] has improved this lower bound from from 4 rounds to 7, and Canetti, Kilian, Petrank and Rosen have recently improved the lower bound substantially showing that concurrent black-box zero-knowledge proof systems for non-trivial languages require $\tilde{\Omega}(\log k)$ rounds. A natural question is whether there exists a fully asynchronous (concurrent) zero-knowledge proof for NP.

1.2 This work

In this paper, we solve this important open question to the affirmative. We show that under a standard intractability assumption, particularly, the existence of a family of claw-free functions, there exists a concurrent zero-knowledge interactive proof system for all languages in NP. We prove this by constructing such a proof system.

Furthermore, in addition to providing an answer to this important open question, our protocol is a black box interactive proof for any language in NP with $\omega(\log^2 k)$ rounds, where k is the security parameter. This number of rounds is polynomially related to the lower bound on black-box zero-knowledge concurrent proof systems for languages that are not in BPP. Thus, our protocol achieves almost optimal round complexity.

The security parameter k determines the running time of all efficient parties and the number of concurrent sessions. All the above must be bounded by a polynomial in k. Saying that the number of rounds $m(k)$ is $\omega(\log^2 k)$ rounds we mean that $m(k)$ can be set to any function satisfying $m(k) = \omega(\log^2 k)$. More specifically, we only require of $m(k)$ that for any constant c there exists an integer k_c such that for any $k \geq k_c$ it holds that $m(k) \geq c \cdot \log^2 k$.

The concurrent zero-knowledge interactive proof for NP that we present relies on the existence of secure bit commitment schemes with statistical binding and bit commitment schemes with statistical secrecy (see Section 2.2 below). Using [27] the first can be efficiently based on any pseudo-random generator (and thus on any one-way function), and using [2, 28, 9], the latter can be efficiently based on the existence of a family of claw-free functions (and even only on collision-intractable hash functions). Our proof system may be run by an efficient prover that is given the witness for the (NP) assertion in the input.

Finally, the proof system we present is a family of proofs systems. This family is parameterized by the number of rounds. The proof as presented may be used with any number of rounds (above some required constant). The analysis we provide shows that it is concurrent zero-knowledge when the number of rounds is set to $\omega(\log^2 k)$. By the lower bound, we know that if we set the number of rounds too low, then the proof system does not remain concurrent zero-knowledge. Possibly, the same proof system remains zero-knowledge even if the number of rounds is set to $O(\log n)$, but we do not know how to show it. This is an interesting open question.

1.3 Subsequent work

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 with respect to the one presented in this paper is that the maximum number $\ell(k)$ of concurrent sessions tolerated must be predetermined in advance, and the communication required by this protocol is proportional to the chosen bound. Our protocol, is robust against any polynomial number of concurrent sessions, and its overall communication is independent of the actual number of sessions run.

In another subsequent work, Cohen Kilian and Petrank [6] present a new complexity measured for protocols called *responsive round complexity*. They provide a black box concurrent zero-knowledge protocols for all languages in NP building on the protocol presented in this paper and on the work of [11]. Their proof system has an optimal responsive round complexity for concurrent black box zero-knowledge proofs.

1.4 Alternative models

Due to the difficulty of designing an efficient concurrent zero-knowledge proof systems in the general model, several researchers have considered relaxations to the standard model, thus, "bypassing" the difficulty in constructing concurrent zero-knowledge protocols. Dwork, Naor and Sahai [11] with the so called "timing assumptions" have augmented the communication model by assuming a maximum delay on delivery of messages in the network [11, 12]. Damgård suggested a "setting assumptions" by using public key infrastructure [8]. This assumption has been somewhat relaxed by Canetti, Goldreich, Goldwasser, and Micali to a use of a public registry file [4].

A different approach is to weaken the security requirement. Instead of obtaining zero-knowledge, Feige and Shamir suggest the notion of *witness indistinguishability* [13, 16].

1.4.1 Resettable zero-knowledge

Canetti, Goldreich, Goldwasser and Micali [4] introduced the notion of *resettable* zero-knowledge. Such proofs are zero-knowledge proofs that on top of being concurrent, maintain zero-knowledge properties when the verifier is allowed to run the prover repeatedly on a fixed (yet, randomly chosen) random tape.

The practical motivation behind such robustness is a use of zero-knowledge in smartcards, where the prover (the card) can be reset by the verifier to run repeatedly without access to additional random coin tosses. It is assumed that the actual random tape of the card is hidden from the verifier, and it is shown in [4] how this hidden randomness can be used to allow such robustness of a zero-knowledge proof.

Canetti, Goldreich, Goldwasser and Micali modified an earlier version of our proof system to obtain the first resettable zero-knowledge proof system. This protocol requires $k^{\theta(1)}$ rounds. We note that their technique also applies to our current proof system, yielding a resettable zero-knowledge proof for NP with $\tilde{O}(\log^2 k)$ rounds. Previously, there was no sub-polynomial resettable zero-knowledge proof in a general asynchronous environment. We remark that as with concurrent zero-knowledge, if one makes set-up assumptions, then one may get more efficient proofs. For example, in the *public key model* there exist more efficient resettable zero-knowledge proofs, see [4].

The definition of resettable zero-knowledge and related issues appear in Section 7.1 below. For a more detailed discussion, motivation, and definitions the reader is referred to [4].

1.5 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. (See definitions in Section 2.1 below.) Our proof is black-box zero-knowledge (see Section 2.4 below). The proof will be perfectly sound, i.e., we will construct an interactive proof system, yet it will be possible to run the prover in polynomial time given a witness to the NP assertion that the prover is making.

1.6 Guide to the paper

In Section 2 we present some definitions and the tools we are using. In Section 3 we state our results. In Section 4 we present the concurrent zero-knowledge interactive proof for NP. In Section 5 we provide a simulator for the interaction between the prover and any (adversarial) verifier. The analysis of the simulator appears in Section 6. Finally, in Section 7, we discuss our protocol in the resettable zero-knowledge model.

2 Preliminaries

In this section we go over the definitions and the tools we are using. We postpone the definitions and discussion of resettable zero-knowledge to Section 7.

One important term in this paper is a *round*. We denote a message from the prover to the verifier or from the verifier to the prover by a round. In section 5 we start using the term *pairs of rounds* often. A pair of rounds relates to two messages: a message by the verifier followed by a message from the prover. For simplicity, at some point in the analysis (in Section 5) we will start abusing the term *round* to denote a pair of rounds. An appropriate remark is given at the appropriate location.

2.1 Zero-knowledge proofs

Let us recall the concept of interactive proofs, as presented by [23]. For formal definitions and motivating discussions the reader is referred to [23].

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 fraction ε such that*

- **Completeness:** *If $x \in L$ then*

$$\Pr\left[(P,V)(x) \text{ accepts}\right] \;\geq\; 1-\varepsilon(|x|)$$

- **Soundness:** *If $x \notin L$ then for any prover P^**

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

Brassard, Chaum, and Crépeau [2] suggested 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 [23, 29, 19, 22]. 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 probabilistic 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 computational zero-knowledge. In the sequel we will say *zero-knowledge* meaning *computational zero-knowledge*.

2.2 Bit commitments

We include a short and informal presentation of commitment schemes. For more details and motivation, see [17]. 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 (i.e., information theoretic) secrecy can be constructed from any collection of claw-free permutations; see [17]. 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 [9] 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. For simplicity, we adopt the simple commitment scheme as described in [17] for presentation in this paper. This commitment scheme will be used for the verifiers in our protocol. The scheme of Dåmgard, Pedersen and Pfitzmann [9] is quite practical and may be adopted for more efficient practical applications. 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 [27].

2.3 Witness Indistinguishability

Witness indistinguishable proofs were presented in [16]. 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 with 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 [16, 13].

We say that a relation R is polynomial time if there exists a machine that given (x, y) works in polynomial time in $|x|$ and determines whether $(x, y) \in R$. For any NP language there exists a polynomial time relation R_L such that L can be described as $L = \{x : \exists y, \quad R_L(x, y)\}$.

Definition 2.3 *A proof system (P, V) is witness indistinguishable over a polynomial time relation R is for any V', any large enough x, any w_1, w_2 such that $(x, w_1) \in R$ and $(x, w_2) \in R$, and for any auxiliary input y for V', the view of V' in the interaction with $P(x, w_1)$ is polynomially indistinguishable from the view of V' in the interaction with $P(x, w_2)$.*

It is shown in [16] that witness indistinguishability is preserved with asynchronous composition of proofs. More precisely, if a proof is auxiliary-input witness indistinguishable and if the prover can run in polynomial time given the witness, then asynchronous (concurrent) composition of such proofs remains auxiliary-input witness indistinguishable (with efficient provers). Note that any (auxiliary-input) zero-knowledge proof (even one that is not concurrent) is also (auxiliary-input) witness indistinguishable. In our proof system, we employ a constant round (auxiliary-input) zero-knowledge proof for the languages in NP in which the prover has an efficient procedure when given a witness to the NP-assertion such as in [18]. It follows that this proof is also (auxiliary-input) witness indistinguishable. Furthermore, since such witness indistinguishable proofs are also concurrent witness indistinguishable proof, we get that the sub-proof we employ is concurrent witness indistinguishable.

2.4 Blackbox simulation

The initial definition of zero-knowledge [22] requires that for any probabilistic polynomial time verifier $\hat{V}$, a simulator $S_{\hat{V}}$ exists that simulates $\hat{V}$'s view. Oren [29] 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.

At first glance, the limitations on S may seem to force S to be as powerful as a prover. However, S has important advantages over a prover P, allowing it to perform simulations in probabilistic polynomial time. First, it may set $\hat{V}$'s coin tosses as it wishes, and even run $\hat{V}$ on different sets of coin tosses. More importantly, S may conceptually "back up" $\hat{V}$ to an earlier point in the conversation, and then send different messages. This ability derives from S's control of $\hat{V}$'s coin tosses; since $\hat{V}$ otherwise operates deterministically, S can rerun it from the beginning, exploring different directions of the conversation by trying various messages.

2.5 Concurrent zero-knowledge

Informally, we consider a proof system that has many copies running. We call each copy a *session*. Each of them consists of a prover and a verifier. We require that the provers do not cooperate. This can be also thought of as one prover with many sessions, but the behavior of the prover in any of the proof does not depend only on the current proof and not on the other concurrent sessions. This complicates the construction of the proof system, since there is no central control that checks whether the verifiers are trying to cheat, or that tries to coordinate the timing of various sessions. However, modern systems necessitates this. Most operating systems allow multithreading and central control over the protocols run in each thread hinder efficiency, parallelism, and scalability. On the other hand, we allow the verifiers to coordinate their strategies and information. Following [11], 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 A will be a tuples of the form (V, α, t), indicating that P receives message α 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 $(P, \mathcal{A})(x)$. The following definition formalizes the above using the black-box formulation. We denote by $(P, A)(x)$ the distribution on the view of A in its interaction with P.

Definition 2.4 *We say that an interactive 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^**.

Simplifications:

To simplify the analysis, we will only care about the order of the messages sent in the interaction and not about the delays between messages. To make the output of the simulator contain the delays dictated by the verifier, our simulator may write the output with the required delays on a working tape (instead of actually writing to the output tape) and after the order of messages and their delays has been determined and written (by the simulator described in this paper), the simulator may run a final stage in which it outputs the messages written on the working tape, in the same order and while inserting the appropriate delays between the messages.

We further simplify the analysis by assuming that the prover responds immediately and with no delay to the verifier's messages. Of course, in real life, the adversarial verifier may insert a few quick messages before the prover outputs his message. However, this has no effect on our proof. Intuitively, this may only add information to the simulator before it has to produce a response. Two points should be noted: the simulator may be easily adapted to output the prover's message at times dictated by the verifier rather than immediately. The running time remains polynomial. The probability that the simulator is able to finish the simulation process after "solving" all protocols (see Section 5 below) does not change. Note that within one protocol such a changes is immaterial. The order of messages remain the same and cannot be changed within one protocol.

2.6 The complexity parameters

In this paper, we simplify the discussion by using a single security parameter k. Our proof has $m(k) = \omega(\log^2 k)$ rounds and the security is preserved with a polynomial $\ell(k)$ number of concurrent proofs. If we know that the number of concurrent proofs $\ell(k)$ is substantially smaller than the security parameter k, then the number of rounds $m(k)$ may be set to a poly-logarithmus in the number of possible proofs (and not length of the input or the security parameter).

3 Main result

Our main result is the existence of poly-logarithmic round concurrent (and black box) zero-knowledge interactive proof for NP. This result builds on the following assumptions.

Cryptographic Assumptions: We assume the existence of two rounds commitment schemes with statistical security and the existence of two rounds commitment schemes with statistical binding. Both assumptions are implied by the existence of a family of claw-free permutation pairs (see Section 2.2)

Let us state our main theorem given the above assumption.

Theorem 3.1 *Assume the above cryptographic assumptions. 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 ($\ell(k)$) number of proofs concurrently. Then for any function $m(\cdot)$ asymptotically greater than $\log^2 k$, i.e., $m(k) = \omega(\log^2 k)$, there exists an $m(k)$-round zero-knowledge interactive proof for all languages in NP which is: computational, black-box, and concurrent.*

Theorem 3.1 is proven in Sections 4, 5, and 6. First we present the protocol, next we present the simulator, and last, we analyze the simulator. Note that the proof transforms any round efficient auxiliary-input zero-knowledge interactive proof in which the prover can be implemented in polynomial time when given access to a witness to the NP theorem proven, into a concurrent zero-knowledge interactive proof. For simplicity, we use the proof system of Goldreich and Kahan [18] in the presentation.

For resettable zero-knowledge, one must use more restricted proof system. Our construction takes as a starting point the round-efficient proof of Goldreich and Kahan [18] and uses its specific structure. The theorem about round-efficient resettable zero-knowledge follows from [4] and is proven in Section 7:

Theorem 3.2 *Assume the above cryptographic assumptions. Let k be a complexity parameter bounding the size of the input. The verifier is polynomial time in k, and thus, may initiate at most a polynomial number (in k) of incarnations of the prover. Then for any function $t(\cdot)$ asymptotically greater than $\log^2 k$, i.e., $t(k) = \omega(\log^2 k)$, there exists a zero-knowledge argument for all languages in NP which is: computational, black-box, and resettable.*

4 The zero-knowledge proof system

4.1 An overview of the proof system

Our concurrent zero-knowledge interactive proof system follows the ideas presented by Feige, Lapidot and Shamir [14]. On an input theorem T, the proof consists of a proof-preamble of $2m = \omega(\log^2 k)$ rounds and a proof body being a "standard" constant round auxiliary-input zero-knowledge proof for a modified NP theorem T'. The parameter m determines the round complexity of the proof system. Our analysis shows that the proof system is concurrent zero-knowledge when m is set to any function $m = \omega(\log^2 k)$, where k is the security parameter. In the proof-body one may use any round efficient auxiliary-input zero-knowledge proof for NP. The proof system has to be of negligible error, of round complexity smaller than $O(\log^2 k)$, and in which the prover can run in polynomial time given a witness to the NP theorem T' that it must prove. All known constant round protocol such as [15, 3, 18] will do. All these (auxiliary-input) zero-knowledge proofs are also (auxiliary-input) witness indistinguishable, which is enough for us. Feige and Shamir showed that witness indistinguishability is preserved in this case also with concurrent composition [16].

Let us concentrate now on the preamble, which is the main tool in transforming a regular zero-knowledge proof into a concurrent one. Let T be the NP statement that the original prover would like to prove. We use a preamble with $2m$ rounds to start the proof. In this preamble, P and V each picks m strings in $\{0,1\}^k$ denoted $p_1, p_2, \ldots, p_m$ and $v_1, v_2, \ldots, v_m$ respectively. The prover P then proves that either T is true or that there exists an index i, $1 \le i \le m$, such that $v_i = p_i$. We denote this modified theorem T'. For each i, $1 \le i \le m$, P will have to determine p_i before v_i is revealed. Thus, this preamble will not give P a meaningful advantage in proving the theorem. However, the simulator will be able to learn v_i, and then rewind the proof and set $p_i = v_i$. Thus, the simulator will have a witness to the modified theorem T', and it may act as a real prover in the body of the proof. The full algorithm of the simulator is specified in Section 5 below.

The concurrent zero-knowledge argument for an input theorem T goes as follows:

$V \to P$: Commit to $v_1,\ v_2,\ \ldots, v_m$
$P \to V$: Commit to p_1

$V \to P$: Reveal v_1

$P \rightarrow V$: Commit to p_2
...
$V \rightarrow P$: Reveal v_i
$P \rightarrow V$: Commit to p_{i+1}
...
$V \rightarrow P$: Reveal v_m
$P \leftrightarrow V$: A zero-knowledge proof that T is true or $\exists i$ s.t. $v_i = p_i$.

In words: The verifier begins by committing to all its strings $v_1, \ldots, v_n$. After that, the prover commits to p_i and then the verifier reveals v_i for each i, $i = 1, 2, \ldots, m$. Finally, the prover gives a zero-knowledge proof that T is true or there exists an i s.t. $v_i = p_i$.

If the verifier fails to open one of its commitments properly, then the prover immediately aborts the proof. Ignoring the negligible chance that the commitments of the verifier turn out to fail the binding property, the strings $v_1, \ldots, v_m$ are fixed after the first round for the rest of the proof. Note that v_i is revealed only after the prover P commits on the value of p_i. Thus, if the security of the bit commitment holds, then P can fix $p_i = v_i$ with a negligible probability. Furthermore, ignoring the negligible chance that the commitment of the prover is not secure, the verifier does not learn the value of any of the p_i's so it can never tell whether $p_i = v_i$ holds for some $1 \leq i \leq m$.

4.2 A formal specification of the protocol

We start with the three building blocks used by the prover and verifier.

Statistically binding commitment scheme: For all commitments in which the prover is the committer we use statistically binding commitment schemes. (The reason for this is that we would like the scheme to hold even with computationally unbounded provers.) Any two-round such scheme will do. As an example we refer to Naor's scheme [27]. In this scheme, the committer commits bit by bit. To start the commitment, the receiver sends a uniformly chosen random string in $\{0,1\}^{3k}$ and then the sender responds with a commitment on a bit. The initial random string may be used for all further commitments. If the randomly chosen string does not belong to a small set of bad strings, then the commitment is perfectly binding: any commitment can be opened to only one bit. On the other hand, the scheme is semantically secure. Details appear in [27, 17]. We denote the first message in the prover commitment scheme by C_1^P and the second message by $C_2^P(s)$, where s is the committed string. We denote by $R^P(s)$ the message in which the committer reveals s.

Statistically hiding commitment scheme: For all commitments in which the verifier is the committer we use statistically hiding commitment schemes. (The reason for this is that we would like the scheme to hold even with computationally unbounded provers.) Any two round such scheme will do. As an example, we refer to the two-round scheme built from any claw-free collection [17]. To start the commitment, the receiver sends a uniformly chosen claw-free function. Then, the committer can respond with the actual commitment. We denote the first message in the verifier commitment scheme by C_1^V and the second message by $C_2^V(s)$, where s is the committed string. We denote by $R^V(s)$ the message in which the committer reveals s.

Witness indistinguishable proof system for NP: We need a general witness indistinguishable proof for all languages in NP. We choose to use the auxiliary-input zero-knowledge proof system for NP by Goldreich and Kahan [18]. Their proof relies on the existence of collections of claw-free functions, thus, it is implied by our cryptographic assumption. The good property of this protocol is that it is an interactive proof and so it implies that our proof is also an interactive proof. Another important property of this proof system is that the prover can run in polynomial time given a witness to the assertion made in the input. This is crucial for making the proof retain the witness indistinguishable property when composed concurrently [13]. We denote the five rounds of this proof system on a common input x by $P^1_{\text{body}}(\text{x})$, $V^1_{\text{body}}(\text{x})$, $P^2_{\text{body}}(\text{x})$, $V^2_{\text{body}}(\text{x})$, and $P^3_{\text{body}}(\text{x})$.

The NP language L' for which the witness indistinguishable proof system is used, consists of inputs of the form $(x, \text{preamble})$, where x is an input to L and preamble is a transcript of a preamble. An input $(x, \text{preamble})$ is in the language L' if one of the following two conditions hold.

1. $x \in L$, where $L \in$NP is the language for which we are building a concurrent zero-knowledge proof system.
2. The preamble is a valid preamble as described below, and there is a pair of rounds in the preamble, i.e., a revelation of v_i by the verifier followed by a commitment of the prover on a p_i, for $1 \leq i \leq m$ such that there exists a valid revelation of the commitment of the prover to the string v_i that was revealed by the verifier.

Note that if $L \in$NP then so is L' and thus L' can be proven with the Goldreich-Kahan zero-knowledge proof.

We now specify the protocol showing that an input x belongs to a language L using the above basic blocks.

The preamble:

Step P_0: P initiates a verifier commitment by sending C_1^V.
Step V_0: V chooses uniformly and independently at random m strings $v_1, v_2, \ldots, v_m \in \{0,1\}^k$ and sends $C_2^V(v_1)$, $C_2^V(v_2)$, $\ldots$, $C_2^V(v_m)$ to the prover. It also initiates the prover commitments by sending C_1^P.
Step P_1: P chooses uniformly at random a string $p_1 \in \{0,1\}^k$ and sends $C_2^P(p_1)$ to V.

Step V_1: V reveals v_1 by sending $R^V(v_1)$ to P.
Step P_2: P chooses uniformly at random a string $p_2 \in \{0,1\}^k$ and sends $C_2^P(p_2)$ to V.
$\ldots$
Step V_i: V reveals v_i by sending $R^V(v_i)$ to P.
Step P_{i+1}: P chooses uniformly at random a string $p_{i+1} \in \{0,1\}^k$ and sends $C_2^P(p_{i+1})$ to V.
$\ldots$
Step V_m: V reveals v_m by sending $R^V(v_m)$ to P.

At this stage, the preamble of the proof has ended. If, at any stage during the preamble, the verifier responds with an invalid message, the prover stops and aborts immediately. It does not respond to any further message of the verifier in this protocol. Next, the prover and the verifier construct the pair (x, ω), where x is the input and ω is the transcript of the preamble that was just finished including all the messages sent in Steps $P_0, V_0, \ldots, P_m, V_m$.

The body: P and V run a witness indistinguishable proof that $(x, \omega) \in L'$.

Step P_{m+1}: $P^1_{\text{body}}(x, \omega)$.
Step V_{m+1}: $V^1_{\text{body}}(x, \omega)$.
Step P_{m+2}: $P^2_{\text{body}}(x, \omega)$.
Step V_{m+2}: $V^2_{\text{body}}(x, \omega)$.
Step P_{m+3}: $P^3_{\text{body}}(x, \omega)$.

4.3 Completeness and soundness

Denote the probability that the prover fails to prove a true statement by *the completeness error* and the probability that the verifier accepts a false statement (when the prover uses any arbitrary strategy within its computational limits) *the soundness error*. We claim that these error probabilities are only slightly changed by the modification made to the proof.

Claim 4.1 *If the witness indistinguishable proof system has soundness error ε_s and completeness error ε_c then the modified proof has completeness error at most ε_c, and soundness error at most $\varepsilon_s + \varepsilon$ for some negligible (in the security parameter k) ε.*

Proof: It is easy to see that the completeness property is not harmed by the modification. Regarding soundness, we start by assuming that the prover cannot break the binding property of its commitment. We will show later that this happens with overwhelming probability. Given this assumption, there is only one string to which a prover's commitment may be opened and the string p_i is well defined $C_2^P(p_i)$ has been sent. Since V is acting honestly, v_i is also well defined: it is the string that V chose and committed on. We partition the analysis into two disjoint events. Event 1 happens if there exists an index in the preamble for which $p_i = v_i$. Event 2 is the complement of Event 1. In case Event 2 happens (and we also assume $x \notin L$) then the theorem proven in the body of the proof is false. In this case the verifier accepts with probability at most ε_s. It remains to bound the probability that there exists an index i, $1 \leq i \leq m$, in the preamble for which $p_i = v_i$.

Assume for simplicity that the commitment scheme that the verifier uses is perfectly hiding, i.e., the distribution on the views of the prover in the interaction is independent of the actual string $s \in \{0,1\}^k$ on which V commits. In this case, the prover commits on p_i independently of v_i and the probability that $v_i = p_i$ is 2^{-n}, no matter what the prover strategy is. Now, if the commitment scheme is only statistically hiding, this means that the distribution of a verifier commitment message on two different strings in $\{0,1\}^n$ may have negligible statistical difference. This may change the success probability of the prover (i.e., the probability that the prover commits on a string $p_i = v_i$) by a negligible fraction. Thus, the probability that Event 1 happens is at most a negligible fraction.

To summarize, if the prover does not break the binding property of the commitment scheme, then the probability that the prover cheats is at most ε_s plus the probability that Event 1 happens which is negligible.

It remains to show that even in the concurrent setting of running our zero-knowledge proofs, the binding property of the prover's commitment scheme is still valid. Note that we must show this since we use commitment scheme that have been proven secure for the stand-alone environment, in which the committer and receiver run the protocol with no other protocols running concurrently. It must be shown that the use of these commitment scheme in the concurrent setting does not foil correctness. Suppose that our protocol is run concurrently with the verifier being honest and an arbitrary schedule of sessions determined by a polynomial time adversary. Suppose in a way of contradiction that there exists an adversary A that plays the prover role and for infinitely many inputs x, it manages with probability $1/p(k)$ for some polynomial p, in one of the rounds of one of these sessions to create an ambiguous view for the receiver which allows opening a commitment into two different strings. We would like to show that there exists an adversary that succeeds in breaking the commitment scheme with non-negligible probability in the stand-alone scenario, thus, contradicting the security of the given commitment scheme.

We construct an adversary A^* that breaks the commitment scheme in the stand-alone setting for infinitely many input lengths k. Our goal is to show that A^* can make ambiguous commitments to an honest receiver R in the stand-alone setting with non-negligible probability. We assume that both A and A^* are computationally unbounded, we discuss this issue more in the end of this proof. Let $\ell(k)$ be an upper bound on the number of sessions that may appear in the concurrent interaction. On input 1^k, A^* finds an input x on which A succeeds with non-negligible probability, a number i, $1 \leq i \leq m$, (representing a pair of rounds in a preamble), and a session number $1 \leq s \leq \ell(k)$. The strings (x, i, s) are chosen such that A on input x of length k succeeds with non-negligible probability to create an ambiguous commitment in round i of session s against the honest verifier. (Here, we assume that the sessions that the honest verifier initiates are numbered by their order of initiation.) A^* first runs the protocols of A, the honest verifier V and the adversarial scheduler. They are all run on the input x as well as independently and uniformly chosen random strings. A^* runs them all privately simulating the interaction between the original A and the verifier with the schedule provided by the scheduler in the concurrent case. A^* also runs the commitment scheme against a receiver R in the stand-alone setting. The receiver R is not aware of the whole background simulation. The receiver R initiates a commitment by sending the first message. A^* waits in the simulation till the initiation stage of Session s begins. It then sends R's message to the prover A. Note, first, that A is expecting this message from the verifier and that A^* is simulating the verifier. Note also that the message arriving from the honest receiver R in the stand-alone setting has the same distribution as the message that comes from the honest verifier V. After sending the message of R to initiate the prover commitments in Session s, A^* waits till the i'th pair of rounds in Session s occur. Other sessions may proceed in this simulation and A^* may continue the simulation of the verifier V and the scheduling of sessions. When the i'th pair of rounds arrive, A is supposed to present a commitment. When this commitment arrives, A^* may send it to R. If A succeeded in producing an ambiguous commitment in the concurrent setting, then A^* has managed to produce it in the stand-alone setting. Thus, for k's for which there exists an input $x \in \{0,1\}^k$ on which A succeeds, the success probability of A^* is equal to that of A. Thus, there are infinitely many k's for which A^* succeeds in breaking the bit commitment scheme with non-negligible probability and we are done.

The complexity of the machine A^* built is dominated by the complexity of the "commitment breaking" prover A that it runs and by the time it takes to fins a good x of length k. The indices i, s are chosen from a polynomial space and can be chosen at random by A^*. The simulated machines (the honest verifier and the adversarial scheduler) run in polynomial time. Thus, if A runs efficiently, then A^* can run efficiently but it must be non-uniform in order to obtain a good x for each length k. In our setting, since we are constructing an interactive proof, A is not computationally bounded. Of-course, we must employ a commitment scheme whose binding property holds against an all-powerful adversary in the stand-alone setting (since the prover is assumed to be all-powerful). The above shows that if there exists such am all-powerful machine A in the concurrent setting, then there exists another all-powerful machine A^* that breaks the commitment scheme in the stand-alone setting, thus contradicting the security of the bit commitment scheme and we are done. $\square$

5 The simulator

5.1 An overview of the simulator

We provide a black-box simulator for a run of $\ell(k)$ sessions of the above proof system (black-box simulation is discussed in Section 2.4). $\ell(k)$ may be any polynomial in the security parameter k, in the sequel we let $\ell = \ell(k)$. The simulator depends on $\ell(k)$ but the protocol does not. Therefore, the protocol is concurrent zero-knowledge for any polynomial number of concurrent sessions. The adversarial verifier V^* is given as a black box and the simulator interacts with it. This interaction will be used by the simulator to obtain a witness to the modified theorem T'. We assume that by the time the simulator gets to the body of the proof, it has such a witness. Thus, when simulating the main body, the simulator acts as the prover (which is an efficient algorithm given a witness to the NP theorem that has to be proven).

The simulator will succeed in "guessing" one of the v_i's by rewinding steps in the preamble. (Recall that the real prover cannot rewind the verifier, and cannot get this advantage.) The simulator runs the adversarial verifier V^* and then it may choose to rewind it to a previous round and run it again, possibly with different prover messages (determined by the simulator). Whenever such a rewind occurs, the adversarial verifier V^* is run again on several consecutive rounds. We call this series of rounds *the rewind interval.* We denote the interaction between the simulator and the verifier during these rounds until the rewind is applied *the first run* of the rewind interval and the interaction in which the verifier is forced to go through these rounds again (not knowing of-course that it has been rewound) *the second run* of the rewind interval. During the first run of the rewind interval, the simulator may record messages of V^* and use them to determine its messages in the second run of the rewind interval. The simulation continues after the rewind with the second run of the rewind interval. Namely, the second run is written into the transcript and used for continuing the interaction with the verifier.

The simulator will rewind the verifier at several points during their interaction. If the verifier reveals v_i before a rewind, i.e., in the first run of the rewind interval, and the simulator rewinds the verifier back far enough, it may change the value of p_i and commit on $p_i = v_i$. Since the verifier is committed to the value v_i (as of the beginning of the session), then unless the rewind goes beyond the first round of the particular proof, the simulator need not worry that v_i may change after it sets the commitment on p_i. Once the simulator has ensured that for some pair of rounds i $p_i = v_i$ in the preamble of a proof Π, we say that it has *solved* the proof Π. It can complete the rest of the simulation of Π without further rewinding, by choosing p_j arbitrarily for any $j \neq i$ and by playing the real prover in the main body of the proof Π (recall that after solving a proof, it has a witness to the theorem T' that has to be proven). We stress that the rest of the simulation of this session requires no further rewinding. A key feature of this protocol is that any rewind in any of the m pairs of rounds of the preamble suffices to solve the proof. Of course, if during a rewind the simulator is able to solve more than one session by setting p_j's of other sessions to values of v_j's that were discovered during the first run of rewound interval, then the simulator does that. It always solves the maximum number of sessions it can in a rewind.

Note that rewinding one step in one proof may render irrelevant the simulation of steps in other proofs that took place in between those steps. Thus, choosing a step to rewind according to the need to solve a proof Π is dangerous. It may lead the simulation to run an exponentially many steps as noted in [11] and proved for a set of protocols in [26]. We employ a different strategy of rewinding. We specify a fixed rewinding schedule regardless of the history of the interaction and the scheduling of the proofs so far. Running this rewinding schedule will guarantee a polynomial amount of work, so that the simulation is polynomial time. Nevertheless, whatever schedule of proofs the adversarial verifier-scheduler may use, the simulation is guaranteed to solve all proofs during their preamble with high probability.

Whenever the simulator rewinds the verifier, the second run of the rewind is the one that the simulator uses to continue the interaction (and keep for the transcript it builds). The first run is only used to get information and is then abandoned. The output is composed of the last full run, which is composed of second runs of relevant rewinds.

One of the most problematic issues in the design of the simulator is the following. During the run of the simulator, the adversarial verifier V^* may choose to send inappropriate messages. For example, it may choose not to reveal a value v_i that it has committed to in the beginning of the session. The run of the simulator is composed of rewinds: it executes an interaction with the verifier V^*, then it rewinds V^* and makes a second run, in which it may set the p_i's according to information on v_i's obtained in the first run. When the adversarial verifier V^* sends an inappropriate message for a proof Π the simulator aborts sending messages to V^* for this proof Π (as the normal prover would have done). If that happens in the first run of a rewind it bears a bad affect: the simulator cannot solve the proof Π after rewinding since it did not get to see the string v_i. However, if the verifier V^* sends a bad message in the second run of a rewound interval, then the proof Π is considered solved: the real prover aborts the interaction with the verifier in Π, and so does the simulator. This proof does not require solving since the body of the proof is not executed. Recall that the second run is kept in the transcript and used for continuing the simulation.

5.2 The adversarial scheduler uses round slots

We begin by simplifying our view of the adversarial schedule. Recall that we are running k preambles, each with $2m$ rounds. Like discussed in Section 2.5, we assume, w.l.o.g., that the real prover (and so also the simulator) always answers immediately and in the sequel we discuss pairs of rounds. In a pair of rounds, the verifier sends its message and the prover immediately answers. Also, we do not care about delays imposed by the verifier. We only care about the order of the various pairs of rounds in the full interaction. Thus, we get that the adversarial verifier V^* may schedule an overall number of $\ell \cdot m$ pairs of rounds in the preambles. When specifying the rewinding strategy, we are only interested in the schedule of the preambles. We do not care how the bodies of the proofs are scheduled and whether they are rewound. We will never need to rewind the bodies: the simulator will behave like the real prover in the bodies, however, bodies may be rewound due to requirements imposed by the preambles of other sessions that run concurrently.

Without loss of generality we assume that parallel verifier messages do not occur. If they do, the simulator may also respond in parallel, the response of the simulator to one session does not depend on its responses in other sessions.

For simplicity, from now on we will abuse the term *round* to denote a pair of rounds. Namely, in what follows, a round consists of a message of the verifier followed by an immediate response by the prover.

To summarize, we have reduced our view of the scheduled proofs to the adversarial verifier V^* scheduling ℓm preamble rounds, with the only constraint that within a proof the order of rounds is preserved. We think of this schedule as assigning rounds of the various (preambles of) proofs to ℓm "slots" of rounds. We consider the ℓm slots by their order in time, and specify the rewinding strategy with respect to these slots, regardless of how the adversary assigns actual proof rounds to these slots. For example, we may let the simulator rewind the verifier to the first round slot after running past the second round slot. More generally, After reading the verifier's message in any of the round-slots, the simulator may rewind the simulation (and the verifier) to any previous round-slot of the simulation. We will specify rewinding in the following manner. A rewind $(i \leftarrow j)$, for $1 \leq i < j \leq \ell m$, means that after reading the verifier's message of round slot j, the simulator rewinds the verifier back to just before the prover message in round slot i. When running the rewound interval the second time, the simulator may change its message in round slot i as well as any other message it made in the round slots between i and $j-1$.

5.3 Specification of the rewind timing - an overview

Let us use recursion to specify the rewinding timing. At the top level of the recursion, the simulator runs all the round slots $1..\ell m$. The simulator rewinds the first half of the round slots and then the second half of these round slots (regardless of which rounds of which proofs appear in the round-slots). It then "feeds" each of these $\ell m/2$ round slots to the recursion. Namely, at the second level of the recursion, each of the halves is split into halves and each quarter is rewound. In case the number of round slots is odd, we let the first half contain $\lceil \ell m/2 \rceil$ round slots and the second half contain $\lfloor \ell m/2 \rfloor$ round slots. Finally, at the bottom of the recursion there is an interval containing one round slot. There is no need to rewind one round slot (yet an interval of two round slots is rewound).

A formal procedure to determine the rewinding appears in Subsection 5.4.5 below. Let us explain this rewinding schedule with some examples. Suppose the number of round slots, ℓm, is 4. Then the round slots are run by the simulator in the following order: 1, 2, 1, 2, 3, 4, 3, 4. Using the rewinding syntax with the above sequence, we may write: $(1 \leftarrow 2)$, $(3 \leftarrow 4)$. When $\ell m = 8$, the round slots have the following order: 1, 2, 1, 2, 3, 4, 3, 4, 1, 2, 1, 2, 3, 4, 3, 4, 5, 6, 5, 6, 7, 8, 7, 8, 5, 6, 5, 6, 7, 8, 7, 8. Using the rewinding syntax with the above sequence, we may write: $(1 \leftarrow 2)$, $(3 \leftarrow 4)$, $(1 \leftarrow 4)$, $(1 \leftarrow 2)$, $(3 \leftarrow 4)$, $(5 \leftarrow 6)$, $(7 \leftarrow 8)$, $(5 \leftarrow 8)$, $(5 \leftarrow 6)$, $(7 \leftarrow 8)$.

5.4 A detailed specification of the simulator

The simulator invokes the adversarial verifier V^* as a black box. We assume without loss of generality that the verifiers roles in all the sessions in the interaction are run by the verifier V^*. The sequence of events is as follows. V^* generates the next message for session i, $1 \leq i \leq \ell$. The simulator receives this message and produces the response of the prover for this session. For simplicity, we assume that the simulator responds before V^* generates the next message (for a different session). (Alternatively, the simulator may produce the response, but not put it on the transcript until instructed to do so by V^*.)

The simulator receives four types of messages from V^*.

- **Type 0:** An initiation of a new session (to which the prover is supposed to respond by sending C_1^V according to the verifier's commitment scheme).
- **Type 1:** A commitment on $v_1, v_2, \ldots, v_m$ plus a string C_1^P to initiate the prover's commitments.
- **Type 2:** A preamble message that contains a revelation of a string v_i.
- **Type 3:** A message of the body of the proof for some session.

The simulator runs the verifier V^* and responds to its messages as specified in Subsection 5.4.3 below. The simulator records the transcript so far. After each verifier message is received, the simulator decides whether to rewind and how far back to go as specified in Subsection 5.4.5 below. Let α and β be integers $1 \leq \alpha \leq \beta \leq \ell m$. If a rewind is required by the rewinding schedule at round number β back to round number α, then the simulator rewinds after receiving β verifier preamble messages (i.e., only messages of types 1 and 2 are considered) back to the exact point it was after receiving the α'th message (and before responding to it). The numbers α and β are integers $1 \leq \alpha \leq \beta \leq \ell m$.

In the body of the proofs, the simulator invokes the algorithm of the prover in the witness indistinguishable proof. This prover is efficient. It gets the input x, the transcript built so far, and the witness for the NP statement $(x, \omega) \in L'$, and it produces a response.

We start by discussing the term *session* and describing what the simulator keeps in its data structure. Then, we describe how the simulator responds to each type of message. Namely, how it modifies its data structure, how it generates the response message, and how it builds the transcript that is eventually output. Next, we describe how rewinds affect the data structure and the transcript. Finally, we describe how the the simulator determines whether a rewind is required and how far back to go with the rewind.

5.4.1 Sessions in the simulation

In what follows we make frequent use of the term *session*. The term session and what distinguishes one session from another may seem vague during the complex run (with rewind) of the adversarial verifier. The adversarial verifier is being rewound frequently, running fourth and back. Thus, we feel that a few words should be said on how a session is defined. We define a new session whenever the verifier sends a message of type 1 (initiating a session). At that time, a unique identity is given to the session. This identity is used in any further messages sent by the verifier or simulator. Whenever a message arrives with an invalid session identity, the message is ignored. If the verifier is rewound back beyond the initiation message, the session seize to exist. When the verifier runs again and delivers a new initiation message, the simulator treats it as a new session and assigns it a new session identity.

We remark that a similar approach is to identify a session according to the prefix of the transcript, i.e., the transcript until the beginning of the session. (This includes the verifier's random tape.) Note that this determines the operation of the verifier in this session. This alternative approach is less wasteful. For example, if a rewind makes the verifier run an initiation with the same prefix, our approach will call it a different session, whereas this alternative approach identifies the two sessions. Both views work well, we adapt the first view in which the simulator assigns a unique id whenever a new session is initiated.

5.4.2 Simulator's data structure

The simulator keeps a list of sessions. Each session s has the following information associated with it.

1. An identifying unique session number.

2. A solution, which is initially null and later may contain a number of preamble round i, $1 \leq i \leq m$, and a value σ_i^s. Intuitively, this is the value of v_i, the string that the verifier reveals in round i in session s.

3. A witness, which is initially null, and later contains the random tape used to commit on p_i in round i of session s, where i is the round number mentioned in the solution entry of this session.

4. A flag BAD signifying whether the session has been aborted because of a verifier bad message. Initially this flag is false.

5. Information for the commitment schemes, and in particular, the initiation strings of the prover C_1^P and of the verifier C_1^V commitment schemes. Initially, these fields are null.

In addition to the list of sessions, the simulator keeps the transcript built so far. Initially, it contains the random tape that the simulator chose for the adversarial verifier. Eventually, this will be the output.

Finally, the simulator keeps information allowing it to decide when to rewind and how far back to go. This information is kept in $\log(m\ell)$ bit vectors of length $m\ell$. We will discuss this algorithm in section 5.4.5 below. We now assume that after any message is received, the simulator may check if a rewind is required. In case the answer is positive, the simulator also gets the rewinding scope.

5.4.3 Response to a message

When the next message is sent by the verifier V^*, the simulator responds according to the type of the message. However, the simulator starts by verifying that the message is valid. If the message is invalid or the *bad* flag is set, then the simulator does not respond to the message and it sets the *bad* flag on for this session. If a rewind is required for this message number, then the rewind is performed. Otherwise, i.e., the message is valid, the simulator responds as follows.
Messages of type 0: The simulator opens a new entry in the session list and assigns the session with a new unique session identity. It then initiates the verifier commitments by selecting C_1^V according to the commitment scheme of the verifier, and sending it to V.
Messages of type 1: When a commitment on $v_1, v_2, \ldots, v_m$ and C_1^P arrives, C_1^P is saved, and the message is added to the (end of the) transcript. The simulator checks if a rewind is required for this message number. If yes, a rewind is performed. Otherwise, the simulator responds by committing on a uniformly chosen number in $\{0,1\}^k$. The commitment messages are added to the (end of the) transcript.
Messages of type 2: When a message arrives revealing a committed value v_i (for round i of session s), the simulator checks the session list for the solution entry of this session. If this entry is null, the simulator writes the value (i, v_i) as the solution of this session. The simulator then checks if a rewind is required for this message number. If yes, a rewind is performed. Otherwise, the simulator prepares a response message. It checks if the solution entry of this session contains the value of $i+1$ as the index of the solution. If this is the case, the simulator sets $r = \sigma_{i+1}^s$ (which is read from the solution entry in the session list). Else, it sets r to be a uniformly chosen value in $\{0,1\}^k$. Then, the simulator commits on r. The commitment messages are added to the (end of the) transcript. If r has been set to σ_{i+1}^s, then the random coins used in the commitment on r are stored in the witness entry of the session list.

Messages of type 3: When a message in a body of a session arrives, the simulator checks if the sessions has a solution. If not, the simulator stops outputting "Failure". Otherwise, (the session has a solution), the simulator invokes the prover algorithm of the witness indistinguishable proof with the input x, the suffix of the transcript containing messages of type 3 (the body), and the witness for this proof. This witness is taken from the witness entry in the session list. the simulator uses the output of the prover's algorithm as its response to V^*. The verifier message and the response are added to the transcript.

5.4.4 Handling a rewind

When the simulator discovers that a rewind is requested in the current round β back to round α, the simulator checks the transcript and sets the rewind interval (α, β) to include all messages starting from the α's preamble (type 1 or type 2) message in the transcript till the most recent message received from the verifier (i.e., the β'th preamble message in the transcript). The simulator takes the following actions:

1. All sessions that were initiated during this interval are marked deleted in the session list. (It is enough to mark them bad.)
2. All messages in this interval (including type 3 messages) are deleted from the transcript.
3. The simulator rewinds V^* to just before this interval. Recall that the rewind gets to a point just after V^* has sent a message M.
4. The simulator responds to the message M as if it was just sent by V^*, as in Subsection 5.4.3 above.

5.4.5 Determination of rewinds

The simulator uses the following procedure to determine whether a rewind is required and how far back the rewind should go. The input to this procedure is the transcript so far, the current verifier message, and the overall number of messages γ required in this simulation. The value of γ, $m\ell \leq \gamma \leq 2m\ell$ is a power of 2 that is an upper bound on the number of round slots in the simulation. The output of the procedure is either null if a rewind is not required for the current round, or a number α saying that a rewind is required and it should go back to the point just after the α'th preamble verifier message. Note that we count only preamble messages of the verifier. We do not care to which session the messages belong. We do not count messages of the body of proofs.

The procedure keeps an inner state with information about the history of rewinds. This data structure consists of $\lceil\log(\gamma)\rceil$ bit vectors of length γ. These vectors are numbered $1, 2, \ldots, \log(\gamma)$. Initially, vector i is initiated with 1's in all entries divisible by 2^i, and 0's otherwise. For example, vector number 1 contains alternating 1's and 0's starting with a zero. We denote the j's entry in the i'th vector by $\text{bit\_vector}_i(j)$. As the simulation proceeds, the entries in the vectors are modified. We will sometime say that a vector is reset to its initial value meaning that it is assigned 1's and 0's as its initial values have been described above. The setting to the initial values will usually happen within an interval (α, β) meaning that the initial values are reset only into entries between α and β in the specified vectors.

Before presenting the algorithm, let us provide some intuition. Our goal is to implement the schedule as described (recursively) in Section 5.3. At level i of the recursion, we would like to rewind any interval of length 2^i of the type $(c \cdot 2^i, (c+1) \cdot 2^i)$. The i'th vector is dedicated to this task. It contains a 1 at locations that should be rewound. For example, at the lowest level, vector 1 makes the simulator rewind each even round. Once a rewind is done, the bit is set to 0, so that the rewind is not repeated a third time. When rewinding an interval, we must start again all lower level rewinds. Thus, when a rewind at level i is executed, it is of length 2^i and all rewinds of lower levels $1, 2, \ldots, i-1$ must be started from scratch, i.e., their indicators must be reset to their initial values. The algorithmic description follows.

The rewinding determination starts by concatenating the current verifier message to the transcript so far and counting the number of verifier preamble messages (of type 1 or type 2). This number is the only part of the input which is considered by this procedure, and we denote it by β. The rewind determination works as follows.

1. $i = 1$
2. If (bit_vector$_i(\beta) == 0$) then
   - If ($i == \log(\gamma)$) then return(NULL)
   - $i = i + 1$; goto Step (2).
3. Otherwise [bit_vector$_i(\beta) == 1$, a level i rewind is required]:
   - For $j = 1$ to $i - 1$: reset bit_vector j to its initial values in the interval $(\beta - 2^i, \beta)$.
   - bit_vector$_i(\beta) = 0$
   - return$(\beta - 2^i)$

5.4.6 Termination

If the number of preamble verifier messages is γ then termination is easy. The simulator terminates when all rewindings are done. The output of the simulation is the transcript string.

When the number of messages is smaller than γ and the verifier has halted, we must make some rounding (or padding) effort to finish the simulation. In this case, the simulator produces to itself dummy verifier preamble messages that do not belong to any session and are answered by dummy prover messages. The dummy messages are omitted from the transcript eventually and do not appear in the output. However, they allow finishing all the rewinding schedule for the real (and dummy) messages.

6 Analysis of the simulator

6.1 Time complexity

The work of the simulator consists of running the verifier again and again as the rewinding schedule instructs plus determination of rewinds interval plus playing the prover in the body of the proofs. We show that the number of rewinds is bounded by γ^2. Recall that $\gamma \leq 2m\ell$ and therefore γ is polynomial in the security parameter k. The running time to determine whether to rewind and also for producing a response to a verifier message (both for preamble messages and for body messages) is also polynomial time. Thus, even if between any two rewinds, the simulator has to work on producing γ prover responses and it must check for a rewind at each message, the overall running time between two rewinds is polynomial.

So it remains to prove that the number of rewinds is bounded by γ^2. Let $\delta = \log(\gamma)$. We prove that the number of rewinds is at most γ^2 by an induction on δ (and $\gamma = 2^\delta$). When $\delta = 1$ (and $\gamma = 2$) there are two rounds to run and they are repeated once, thus the overall number of rewinds is at most 1. Suppose the claim is correct for $\delta = \delta'$ and let us determine how many rewinds occur for $\delta = \delta' + 1$. Looking at the structure of the vectors, all vectors of levels $1, 2, \ldots, \delta'$ have twice the length with their first half equal their second half. On the top level we have an additional vector with all zeros except for 1 in the end. Ignoring the top level vector, since all vectors at level $1, 2, \ldots, \delta'$ are doubled, the simulator must do twice the work as in the previous level. Since at the top level it is instructed to repeat it all again once again, we get a factor of 4 between the number of rewinds at level δ' and the number of rewinds at level $\delta' + 1$.

To summarize, for $\delta = 1$ the number of rewinds is 1 and this number if multiplied by 4 whenever the level is incremented. Thus the overall number of rewinds for a given γ and $\delta = \log(\gamma)$ is $4^\delta = \gamma^2$ and we are done.

6.2 Indistinguishability

Intuitively, the simulator plays almost exactly the role of the prover in the second run of all rewinds (which also include the output of the simulation). There are two differences. One is that some of the committed values p_i satisfy $p_i = v_i$. However, if $\hat{V}$'s behavior is changed because of this fact in a way that is noticeable in polynomial time, then the commitment scheme of the prover is not secure. The other change in the protocol is that the simulator uses different witnesses than the prover normally uses in the body of the protocol. But that difference is polynomially indistinguishable since the proof body is witness indistinguishable (also in the concurrent setting). So this gives good intuition that $\hat{V}$'s behavior and view in the simulation is polynomial time indistinguishable from its behavior and view when interacting with the real prover (both with respect to the content of the messages and to their schedule). This intuition works if the simulator solves all sessions of the proof. We now show this formally.

Claim 6.1 *If the simulator does not halt outputting "FAILURE", i.e., it solves each session before the session body begins, then the view of the verifier produced by the simulator $S(x)$ is indistinguishable from the view of the verifier as in the interactive proof $(P, V^*)(x)$.*

Proof: Assume that the simulator solves all sessions before their body begins. (We show in Sections 6.3 and 6.4 below that this happens with overwhelming probability.) We restrict the discussion to the honest prover P and we denote by $(P_w, V^*)(x)$ the view of the verifier V^* when interacting with P on input x and when P uses w as the witness for $x \in L$.

Suppose in a way of contradiction that there exists an efficient distinguisher D such that for infinitely many (x, w), $x \in L$, and $w \in \Sigma^*$ being a witness for $x \in L$, the distinguisher D distinguishes between the distribution $(P_w, V^*)(x)$ and the output of the simulator $S(x)$ with non negligible advantage. Although not specifically mentioned, we assume the simulator has blackbox access to V^*. We use standard hybrid argument to show that either the commitment scheme used by the prover does not have semantic security for infinitely many lengths, or the proof body is not witness indistinguishable for infinitely many inputs $x \in L$.

For any $x \in L$, consider a third set of distribution spaces $\Omega(x, w)$ (in addition to $(P_w, V^*)(x)$ and $S(x)$). In the distribution $\Omega(x, w)$ the preamble contains an index i for which $p_i = v_i$, but in the body of the proofs, the prover uses the witness w for $x \in L$. This distribution can be generated by a modified simulator that gets as an auxiliary input the witness w. The modified simulator behaves exactly like the normal simulator, except that when it invokes the given witness indistinguishable prover for the bodies of the concurrent proof, it uses the witness from the auxiliary input rather than the witness obtained while rewinding.

The distinguisher D must distinguish with non negligible advantage either between $\Omega(x, w)$ and $(P_w, V^*)(x)$ or between $\Omega(x, w)$ and $S(x)$ for infinitely many pairs (x, w) with $x \in L$ and $w \in \Sigma^*$. We show that if the first is true then the commitment scheme used by the prover is not semantically secure, whereas if the latter is true then the proof used in the body is not witness indistinguishable.

Case I: D **can distinguish** $\Omega(x, w)$ **from the original interaction.** We start with the first case. Suppose there exists a distinguisher D that can tell with advantage $1/p(k)$ for some polynomial p between (P_w, V^*) and $\Omega(x, w)$ for infinitely many pairs (x, w), $x \in L$ and $w \in \Sigma^*$. We build an non-uniform adversary A that breaks the (semantic) security of the prover commitment scheme. A gets as input a security parameter 1^k and an auxiliary input (x, w, j). Intuitively, (x, w) is a pair on which D succeeds with $|x| = k$, and j is an integer (an index to be discussed later). Adversary

A begins by selecting a string $\sigma \in \{0,1\}^k$ and sending σ to the committer C. Next, A engages with C in the commitment phase of the commitment scheme. After this phase, A distinguishes with non negligible advantage between the following two cases. Case 1: C has committed on σ, case 2: C has committed on a randomly chosen string in $\{0,1\}^k$. This contradicts semantic security.

To explain how A is built and what the index j is, we use further hybrid arguments. Recall that ℓ is a polynomial that upper bounds the number of possible sessions. We number each session according to the time its first message is scheduled. We define the distribution $\Omega_j(x,w)$, $0 \le j \le \ell$, to be a distribution similar to $\Omega(x,w)$. It is generated by a modified simulator that runs as the normal simulator for the preamble messages of the first j sessions. For the preamble messages of sessions $j+1$ and on, it uses the prover strategy, i.e., it commits on random strings and does not try to solve these proofs. The simulator stops with "Failure" if one of the first j sessions is not solved before its body arrives. In all sessions, the simulator uses the witness w for invoking the witness indistinguishable prover in the proof body. Thus, for the first j sessions each session has an index i, $1 \le i \le m$ with $v_i = p_i$ in the preamble. In the other sessions (Session $j+1$ and on), the p_i's are chosen uniformly at random in $\{0,1\}^k$. Note that $\Omega_0(x,w)$ is exactly $(P_w, V^*)(x)$, whereas $\Omega_\ell(x,w) = \Omega(x,w)$.

For each pair (x,w) on which D can tell between (P_w, V^*) and $\Omega(x,w)$ with advantage $1/p(k)$, there must exist an index j, $0 \le j \le \ell - 1$, for which our distinguisher D has advantage $1/(p(k) \cdot \ell(k))$ in distinguishing $\Omega_j(x,w)$ from $\Omega_{j+1}(x,w)$. Such a j is provided in the auxiliary input of the adversary A. We proceed with constructing A.

Recall that A gets as input a security parameter 1^k and an auxiliary input (x,w,j). A starts by choosing uniformly at random a preamble round index i, $1 \le i \le m$. A uses the verifier V^* as a black box and A uses a generator of $\Omega_{j+1}(x,w)$ as described above. A lets the generator of $\Omega_{j+1}(x,w)$ interact with V^* except for a special treatment of the $j+1$'st session. Recall that the generator for $\Omega_{j+1}(x,w)$ tries to solve the $j+1$'st session. A runs the generator of $\Omega_{j+1}(x,w)$ and also solves it. The solution gets written to the session list as usual. A then lets the generator use this solution until the final run comes in which the solution should be used for what will be written as the final transcript. (This point in time is easy to determine: each bit vector i should contain zeros from the location of the current round slot for an interval of length 2^i.) At that time, A does not let the generator commit on the solution as dictated by the algorithm of the generator. Instead, A slices in its interaction with the message of the real committer C. A fixes σ to be the solution provided by the session table for Session $j+1$. It engages in a commitment scheme with committer C in the following manner. The receiver's messages are determined by the messages of V^* as it behaves in the interaction with the generator of $\Omega_{j+1}(x,w)$ in this specific point in time, i.e., round i of session $j+1$. The response of the generator is modified to be the message of the committer C. After that, the generator continues as usual. Recall that in the bodies of the proofs the witness w is always used, thus, the modification does not affect the continuation of this generation.

After the final transcript is output by the generator, A lets D say whether this transcript belongs to $\Omega_j(x,w)$ or to $\Omega_{j+1}(x,w)$. Note that if the committer committed on σ then the transcript belongs to $\Omega_{j+1}(x,w)$ since the commitment is of the same distribution of commitments as the generator would have produced: a commitment on σ. However, if the committer commits on a random string in $\{0,1\}^k$, then this transcript is generated according to $\Omega_j(x,w)$: the commitments of the generator in this case are on uniformly chosen random strings in the j'th session. The adversary outputs the output of D on the transcript.

The advantage of A in distinguishing commitments on σ from commitments on a uniformly chosen random string in $\{0,1\}^k$ equals the advantage of D in distinguishing $\Omega_{j+1}(x,w)$ from $\Omega_j(x,w)$ and this advantage is a non negligible fraction. Also, since there is an infinite set of pairs (x,w) on which D does well, there is an infinite set of lengths for which A succeeds in the above task. Finally, similar hybrid arguments may be used to constructs an adversary A' that can predict one of the bits of a committed string with non-negligible probability, thus, contradicting the security of the commitment scheme.

Case II: D can distinguish $\Omega(x,w)$ from the output of the simulation. Here, we also use a hybrid argument, which is basically the same as the one used by Feige [13] to show that witness indistinguishability is preserved with concurrent composition. Consider the distributions $\Omega'_0(x,w), \Omega'_1(x,w), \ldots, \Omega'_\ell(x,w)$, where ℓ is an upper bound on the number of possible sessions. The distribution $\Omega'_j(x,w)$ is defined by its generator which is again a modified simulator. The modified simulator solves all sessions as usual. The difference is in the bodies of proofs. The modified simulator uses the witness indistinguishable prover for the proof bodies as follows. In the first j sessions it uses the solution that the simulator has obtained, i.e., a witness that $v_i = p_i$ for some round i, whereas in the other sessions ($j+1$ and on) the string w is used as the witness that $x \in L$. All distributions $\Omega'_j(x,w)$, $1 \leq j \leq \ell$ can be produced in polynomial time given an auxiliary input with $x \in L$ and a witness w for $x \in L$. This is because the prover in the proof body can run in polynomial time given a witness to $x \in L$, and since the simulator runs in polynomial time. We note that $\Omega'_0(x,w) = \Omega(x,w)$ and $\Omega'_\ell(x,w)$ is $S(x)$. Thus, there must exist an index j for which the distinguisher D can tell between $\Omega'_j(x,w)$ and $\Omega'_{j+1}(x,w)$ with an non negligible advantage. Similarly to [13] we let a new prover simulate the whole interaction except for the j'th session which is run against a real verifier. Now, D can be used to distinguish the case that the output transcript is in $\Omega'_j(x,w)$ or in $\Omega'_{j+1}(x,w)$, which translates to distinguishing the witness w from the witness that there exists an i such that $p_i = v_i$. This contradicts the witness indistinguishability of the proof system that is used in the body of the concurrent proof and we are done. $\square$

It remains to show that the simulator succeeds in solving all sessions with overwhelming probability. We show this in the following two subsections.

6.3 Analysis with respect to a static schedule

To simplify the presentation of this analysis, we start by showing that the simulator works well for a static schedule. In a static schedule, the adversarial verifier V^* chooses the (worst possible) schedule for the simulator, but this schedule is fixed and does not change during the simulation. In Subsection 6.4 below, we extend the argument to the case that the schedule is dynamic and may change as a function of the adversary's random coins and the history of the simulation so far.

Our goal is to show that with overwhelming probability the simulator will manage to obtain a witness for $(w, \omega) \in L'$ during the simulation of the preamble. We start with some properties of the rewinding schedule. We denote the intervals that are rewound *rewind intervals.* Because of the manner we defined the rewinding schedule, the rewind intervals are either disjoint or contained within each other. So for any two rewind intervals $(i_1 \leftarrow i_2)$ and $(j_1 \leftarrow j_2)$ if $i_1 < j_2 \leq i_2$, then it holds that j_1 must be greater or equal to i_1. In the above case, in which the rewind interval $(j_1 \leftarrow j_2)$ is contained within the rewind interval $(i_1 \leftarrow i_2)$ we will say that the rewind $(i_1 \leftarrow i_2)$ *dominates* the rewind $(j_1 \leftarrow j_2)$.

Definition 6.2 *We say that a rewind* $(j_1 \leftarrow j_2)$ dominates *the rewind* $(i_1 \leftarrow i_2)$ *if* $j_1 \leq i_1 < i_2 \leq j_2$.

We call a run of the simulator against a (black-box) verifier V^* *good* if the simulator solves each of the proofs during the preamble and before it gets to simulating the main body of the proof. We would like to show that the rewinding schedule presented in Subsections 5.3 and 5.4.5 lets the simulator get "good" runs with overwhelming probability, no matter what schedule is chosen for the messages in the sessions. During the simulation, we do not need to rewind bodies of proofs. Rewindings of a proof body that happens while rewinding a preamble of another session does not hurt the simulation.

A proof Π may be solved via a rewind $(i_1 \leftarrow i_2)$ if there are at least two rounds of Π appearing within the round slots $i_1, i_1+1, \ldots, i_2$, and the proof Π does not begin or end during the round slots $i_1+1, i_1+2, \ldots, i_2$. The reason for precluding the first round is that if we rewind past the first round, the verifier may pick a new vector $v_1, \ldots, v_m$ and make the information obtained in the first run of the rewind useless. The proof is *actually* solved in such a rewind if the verifier behaves "well" (i.e., follows the protocol) in both runs of the rewind interval. In this case, we have two consecutive rounds of the proof Π: rounds a and $a+1$ ($2 \leq a \leq m-2$) of Π within the rewind interval. Thus, in the second run of these rounds, the simulator can set p_a in the preamble to the value v_a obtained in the first round and solve the proof. The reason we preclude the first round of the proof Π from the rewind interval is that if we rewind past the first round of the proof, then V^* gets to its first round again and it may choose new values for $v_1, \ldots, v_m$. In particular, v_a may change, and the simulator would not know the new value of v_a to set as p_a. The reason that we require that the preamble does not end before the rewind, i.e., that round m of the proof Π is not within the rewind interval, is that a proof must be solved before the preamble ends. Else, the main body may start, and the simulator will noticeably fail to simulate the proof body, possibly causing the verifier to stop cooperating with the rest of the simulation.

We would like to point out that a rewind may solve the proof in any level of the recursion. If there exists a rewind $(i_1 \leftarrow i_2)$ that may solve the proof, and there exists a larger rewind $(j_1 \leftarrow j_2)$ that dominates it, then the fact that we rewind $(j_1 \leftarrow j_2)$ does not "ruin" the solution of the proof obtained in rewind $(i_1 \leftarrow i_2)$. This is true since in both runs of the rounds $j_1, j_1+1 \ldots, j_2$ in the dominating rewind interval we rewind $(i_1 \leftarrow i_2)$. So even if the rewind $(i_1 \leftarrow i_2)$ happens again and again because of dominating rewinds, in each of the runs it may solve the proof again.

In what follows, we will restrict our attention to the minimal rewinding intervals that may solve a proof. If a proof may be solved by a rewind $(j_1 \leftarrow j_2)$, then sometimes it may also be solved by several rewinds that dominate $(j_1 \leftarrow j_2)$ just because they dominate it. However, we will be interested only in the smallest rewind interval that may solve a proof. Minimality is expressed in Conditions (1) and (4) of the following definition. This minimality property will be used to get independence between the relevant intervals.

Definition 6.3 *We say that a rewind* $(j_1 \leftarrow j_2)$ may solve *a proof* Π *if the following four conditions hold:*

1. *Exactly two rounds of the preamble of* Π *take place during round slots* $j_1, j_1+1, \ldots, j_2$,
2. *the first round of* Π *takes place at a round slot* $i < j_1$,
3. *the last round of* Π *takes place at a round slot* $j > j_2$, *and*
4. *The first round of* Π *appears in the first half of the rewind interval* $(j_1 \leftarrow j_2)$ *and the second round of* Π *appears in the second half of the rewind interval* $(j_1 \leftarrow j_2)$.

Note that if Π has two rounds in the same half of the rewind interval $(j_1 \leftarrow j_2)$, then there exists a dominated interval that may solve Π. This the ratio behind Part (4) of the definition. We now show that many rewinds may solve Π.

Lemma 6.4 *For any schedule of k sessions of the proof preambles (in the ℓm round slots), if a preamble of a specific proof* Π *completes in round slot* β, *then there are at least* $\left\lceil \frac{m}{\log(\ell m)+1} \right\rceil - 2$ *rewind intervals that complete by round* β *and that may solve* Π.

Proof: We first show that there are at least $\left\lceil \frac{m}{\log(\ell m)+1} \right\rceil$ rewind intervals that satisfy Conditions (1) and (4) in Definition 6.3 above. We then note that at most two of these intervals may foil Conditions (2) or (3), thus the number of rewinds that may solve the proof Π is at least $\left\lceil \frac{m}{\log(\ell m)+1} \right\rceil - 2$ as required. Clearly, any relevant interval must end by round slot number β, since the preamble terminates at round slot β.

Fix a session Π and any schedule of the rounds for all the sessions. We denote a rewind interval *good* if it satisfies Conditions (1) and (4) in Definition 6.3 above (with respect to Π). Consider the rewinds by the height of the recursion. At the top level, i.e., recursion height $\lceil \log(\ell m) \rceil$, we have ℓm

round slots. In these round slots we have m rounds of the proof Π. In each recursion invocation, all round slots of the current level rewind interval are split into two almost[1] equal parts and participate in two rewind intervals of a lower recursive level. This splitting goes down the recursion until we are left with one or two round slots at recursion level 1. If we consider the rounds of the specific proof Π as scheduled in the round slots, then there are m rounds scattered at the top level, which are split into two in each recursion invocation. The split of these rounds of Π is not necessarily equal (or even close to equal), since there may be other rounds of other proofs that appear in the (equal) split of the round slots.

In the following, we claim that if there are r rounds of Π in a rewind interval of level h, then these rounds participate in at least $\left\lceil \frac{r}{h+1} \right\rceil$ good rewind intervals with respect to Π. Assigning the recursion level $h \leq \log(\ell m)$ of the top level, and the number $r = m$ of rounds in the preamble of Π in the top level, we get the validity of the assertion in Lemma 6.4.

Claim 6.5 *For any schedule of k sessions of the proof preamble (in ℓm round slots), and for any specific proof Π. Let r be an integer, $2 \leq r \leq m$, and let h be an integer such that $r \leq 2^h$. Suppose there are r rounds of a proof Π in a rewind interval of recursion level h. Then these rounds participate in at least $\left\lceil \frac{r}{h+1} \right\rceil$ good rewind intervals with respect to the proof Π.*

Proof: We prove the claim by an induction on r. Let $r = 2$. if the two rounds are split in the current recursion invocation, then the current rewind interval is good. Otherwise, the two rounds may stay together for several invocations of the recursion and then get split, thus, making a good rewind interval at some lower level. Finally, they may stay together until the bottom level, which makes the bottom level a good rewind interval with respect to the proof Π. Thus, these 2 rounds participate in at least 1 good rewind interval, as required.

Now, suppose that the claim is correct for all $2 \leq r' < r$ and let us prove that it holds for r rounds. Consider the partitioning of the r rounds of the current rewind interval into two rewind intervals when invoking the next recursion. (Recall that each rewind interval is split into two rewind intervals.) Denote by r_1 the number of rounds that go into the first rewind interval, and by r_2 the number of rounds that are assigned into the second rewind interval. We know that $r_1 + r_2 = r$ and assume w.l.o.g. that $r_1 \leq r_2$. We split the analysis into 3 possible cases.

Case 1: $r_1 \geq 2$. In this case, we may use the induction hypothesis. The recursion level of the two rewind intervals that contain the r_1 and r_2 rounds is $h-1$. By the induction hypothesis, the number of good rewind intervals is at least:

$$\left\lceil \frac{r_1}{h} \right\rceil + \left\lceil \frac{r_2}{h} \right\rceil \geq \left\lceil \frac{r_1 + r_2}{h} \right\rceil \geq \left\lceil \frac{r}{h+1} \right\rceil$$

and we are done with Case 1.

Case 2: $r_1 = 1$. In this case, we know that $r_2 = r - 1 \geq 2$ (since $r \geq 3$), thus, we may use the induction hypothesis for the second rewind interval. Nothing is guaranteed for the first rewind interval to which only one round was assigned. By the induction hypothesis, we get that the number of good rewind intervals is at least:

$$\left\lceil \frac{r_2}{h} \right\rceil = \left\lceil \frac{r-1}{h} \right\rceil \geq \left\lceil \frac{r}{h+1} \right\rceil$$

and we are done with Case 2.

Case 3: $r_1 = 0$. In this case, we cannot use the induction hypothesis, since $r_2 = r$. Thus, we check what may happen to these r rounds as we go down the recursion. These rounds may stay together in a single rewind interval only at recursion levels greater than $\lceil \log(r) \rceil$, since there are at

[1] If the number of round slots is odd, then the left interval has one more round slot than the right interval.

most $2^{h'}$ round slots at a rewind interval of recursion level h'. So there exists a level $2 < h' < h$ at which the rounds r are split into $r_1 \geq 1$ rounds and $r_2 \geq 1$ rounds for the rewind intervals of level $h'-1$. By the same argument as in Cases 1 and 2, we get that the number of good intervals that these r rounds participate in is at least:

$$\left\lceil \frac{r}{h'+1} \right\rceil \geq \left\lceil \frac{r}{h+1} \right\rceil$$

and we are done with Case 3 and with the proof of Claim 6.5. □

As mentioned above, this also concludes the proof of Lemma 6.4 since for any proof Π there are m rounds at recursion level $\lceil \log(\ell m) \rceil$, and since only two of them may contain the first or last round of the preamble. □

6.3.1 Why the rewinding works

We would like to claim now that the simulator is able to solve each proof during its preamble and before it is required to simulate the main body of the proof with high probability. By Lemma 6.4, for each of the k proofs, there are at least $\left\lceil \frac{m}{\log(\ell m)} \right\rceil - 2$ rewind intervals that may solve it. Of course, it is enough that for each proof there is one rewind that actually solves it during the preamble (rather than may solve it). If we have one such rewind for each proof, the simulator can properly simulate each proof and all of them together no matter what the schedule is.

However, it is not always the case that a proof is solved in a rewind that may potentially solve it. The reason is that the adversarial verifier V^* may sometimes not open the commitment of a round of a proof Π. If the verifier V^* does not open the commitment, then the real prover aborts the proof Π. In a rewind interval that may solve Π there are exactly two rounds of Π (which are not the first or last round). Denote the number of these rounds in the proof Π by a and $a+1$. The proof is solved in this rewind unless the following event happens:

1. the verifier *does not* reveal the committed value v_a in the first run of the rewind interval, but
2. the verifier *does* reveal the committed value v_a in the second run of the rewind interval.

All three other alternatives (i.e., the verifier reveals the committed values in both runs of the rewind interval, or does not reveal the committed value in both runs of the rewind interval, or reveals the committed value only in the first run of the rewind interval) allow the simulator solve the proof Π in this rewind. If the verifier reveals the committed value in the first run of the rewind interval, then the proof is solved, since the simulator may set the value of its string p_a to v_a that it has learned. If the verifier does not reveal the committed value both in the first and second run of the rewind interval, then the proof Π is also solved, since the prover does not answer any of the following rounds of the proof session Π, and the simulator may easily "simulate" that.

We stress that the following naive solution would not work here: output an aborted proof if either in the first or in the second run of the rewind interval V^* does not reveal the committed value. This solution is not good, since it increases the probability of aborting the session Π above the probability of aborting Π in the real interaction. Thus, the simulation may become polynomially distinguishable from the original proof.

Let us compute the probability that a rewind that may solve the protocol fails to solve it. When we solve a proof, the second run of the rewind is different from the first run. In particular, the value of some p_i equals the value of some v_i and the verifier may note that an interval is run the second time by noting that some other proof Π' has been solved in this rewind interval. However, the prover is using a commitment scheme to secretly commit on the strings p_i's in all the proofs. Using the secrecy of the commitment scheme, the verifier cannot tell that it has been rewound, so it cannot make an effort to abort the first run and behave well on the second run. Therefore, the probability that the verifier aborts in the first run is similar to the probability that it aborts in the second run of the rewind interval. These two probabilities are equal up to an (additive) negligible fraction (representing the probability that the commitment scheme fails). Whatever the probability p that V^* chooses not to reveal the committed value is, the probability that it does not reveal in the first run of a rewind, yet it does reveal in the second run, is $p(1-p+\varepsilon) \leq 1/4+\varepsilon$ for some negligible fraction ε. In the sequel, we assume that any rewind that may solve the proof indeed solves it with probability at least $2/3$.

We go on and compute the probability that the simulation succeeds, i.e., that each proof is solved before its preamble terminates. Note that a preamble of a proof Π may terminate several times, since Π may be completely (or partially) rewound several times and in particular, its last round of the preamble may be run by the simulator several times. At the worst case, the preamble of each of the k proofs terminates a number of times that equals the overall number of times that a rewind interval is executed. This number is at most $2^{\lceil \log(\ell m) \rceil} \leq 2\ell m$, i.e., a polynomial in k. We will show that the simulator fails to solve any particular proof with a negligible probability. Thus, it fails to solve any of (the polynomial number of) the proofs with negligible probability as well.

For any proof Π, if the preamble of Π is completed, then the number of rounds that may solve Π is at least $a \stackrel{\text{def}}{=} \left\lceil \frac{m}{\log(\ell m)} \right\rceil - 2$. Since we set $m = \omega(\log^2 k)$ and since a realistic value of m satisfies $m < k$, then this number is

$$a \;\geq\; \frac{\omega(\log^2 k)}{\log(k) \;+\; \log(m)} \;=\; \omega(\log k).$$

We note now that all rewind intervals that may solve the proofs are disjoint in time. This follows since overlapping intervals must contain one another by the definition of the rewinding intervals. On the other hand, by the minimality of the intervals (Requirements (1) and (4) of Definition 6.3) a rewind that may solve the proof Π does not contain another rewind that may solve the proof Π. Thus all these a rewind intervals are disjoint and the probability that the proof is actually solved in any of them is independent and at least $1/3$. So for any occurrence of a proof Π, the probability that the simulator fails to solve it is at most $(1/3)^a$, which is a negligible fraction (in k). By the summation bound, the probability that the simulator fails in any of the (polynomially many) occurrences of proofs is also negligible.

6.4 Extending the analysis for the dynamic schedule

We now move to the more difficult, yet realistic case, in which the verifier does not fix the schedule of the messages in the ℓm round slots in advance, but may determine which message to schedule in the next round slot depending on the history so far and its random coins. Looking back at the analysis of the previous section, the problem now is that the rewind intervals in which a proof may be solved constitute a random variable. Each time a new rewind interval is started, there is a probability that the interval will include two rounds of the proof (which are not the first or last round). This probability depends on the random tape of the adversarial verifier, the history so far, and the behavior of the prover (or the simulator) during the rewind interval. It is possible that in the first run of the rewind interval V^* will choose to include two rounds of the proof but in the second round it will choose not to. The security of the prover's bit commitment gives us, again, a guarantee that the first run and the second run of the rewind have similar behavior.

As before, we ask ourselves what is the probability that a preamble of a proof Π ends without the proof being solved by the simulator. At each point of the simulation one or more rewinds may start. The simulator solves the proof Π during a rewind interval ρ if the first run of ρ includes exactly two rounds of the proof Π that are not the first or the last rounds, and the verifier reveals its committed value properly. Let us present the explicit definitions.

Definition 6.6 *(Dynamic analogue of Definition 6.3:) we say that a run of a rewind ρ (either first or second run) is* interesting *with respect to a proof Π if it includes exactly two rounds of the* *preamble of Π that are neither the first nor the last round of the preamble, and each being in a different half of the rewind interval.*

Definition 6.7 *We say that a run of a rewind ρ (either first or second run) is* good *with respect to a proof Π if it is interesting and the verifier properly reveals its commitments in the second of these two rounds. If a run is not good with respect to Π, we call it* bad *with respect to Π.*

If the first run is good with respect to a proof Π, then the proof Π is solved (no matter what the second run is). For each run of a rewind ρ, depending on the history so far, there is a probability p_ρ, determined by the adversarial verifier, that a run of this rewind is be good with respect to Π. By the security of the prover's commitment scheme, the probability that the first run is good is equal up to an (additive) negligible fraction to the probability that the second run is good. Lemma 6.8 is similar to Lemma 6.4.

Lemma 6.8 *In any schedule of ℓ sessions of the proof preambles (in ℓm round slots), if a preamble of a specific proof Π completes in round β, then there are at least $\left\lceil \frac{m}{\log(\ell m)+1} \right\rceil - 2$ rewind intervals that completed before round β with a second good run with respect to Π.*

Proof: In the same way as in the proof of Lemma 6.4 such number of intervals must exist in any transcript that has a completed preamble in it. It remains to recall that the simulation always continues with the second run of all previous rewinds. Thus, at any point in the simulation time, all rewind intervals that have been finished have been fixed by the second run of the rewind. This means that a transcript that has a completed preamble in it must contains at least $\left\lceil \frac{m}{\log(\ell m)+1} \right\rceil - 2$ such rewind intervals and they all contain the second runs of all rewinds. □

By Lemma 6.8, before a preamble may complete, the history must contain at least $a \stackrel{\text{def}}{=} \left\lceil \frac{m}{\log(\ell m)+1} \right\rceil - 2$ good second runs. However, for the proof to be completed unsolved, all the first runs of all previously completed rewinds must be bad with respect to Π. We will show that this happens with negligible probability.

Lemma 6.9 *The probability that there exists a preamble of a proof Π that ends well during the simulation but is not solved is negligible.*

Proof: We show that for any specific copy Π of the proof whose preamble has completed, the probability that the preamble ends well, yet Π remains unsolved is negligible. Since there is a polynomial number of proofs and each of the preambles may end a polynomial number of times, then we get that the probability that a preamble of any of the proofs remains unsolved when it ends is negligible.

Consider the run of the simulator. At each point of the simulation, one or more rewind intervals may start. At each of these points there is some probability p that the run of one of the rewinds interval will be good with respect to Π. As discussed before, if the commitment scheme that the prover uses is secure, then the probability that the first run is good is equal to the probability that the second round is good up to an additive negligible fraction. We would like to compute the probability that the preamble of the proof instance Π ends well without being solved. By Lemma 6.8, for any possible schedule of the proof instance Π, it must include at least a intervals that were good in the second run with respect to Π. By our definition of a good interval, these intervals are non-overlapping (Recall the minimality condition of Definition 6.6).

We may think of the adversarial verifier as running the following stochastic experiment, which we denote *the sequential experiment*. It runs serially through tests (which are the rewinds). For the ith test, based on the history so far and its random tape, the adversary chooses a probability p_i. (This is the probability that the first run of the interval ends well. The probability that the second run ends well is at most $p_i + \varepsilon_i$ for some negligible fraction ε_i.) Then, with probability

$(1-p_i)(p_i+\varepsilon_i)$ the adversary wins the test, for some negligible fraction ε_i. (The first run is bad and the second is good.) With probability p_i it looses the whole experiment (the first run is good and the simulator has solved the proof Π). In this case we say that the adversary dies. Finally, with probability $(1-p_i)(1-p_i-\varepsilon_i)$ nothing happens, i.e., the adversary neither wins nor dies (both runs are not interesting). The goal of the adversary is to win at least a tests in the experiment without dying. The probability that the adversary succeeds in the sequential experiment is an upper bound on the probability that the preamble of a proof Π ends without being solved. The reason is that when a preamble completes without being solved, all first runs must be bad and at least a runs must be good. The number of tests run during the sequential experiment is b. In our case $b \leq 2\ell m$.

We now analyze the sequential experiment with parameters a and b.

Claim 6.10 *Let b and a be two positive integers such that $a < b$ and b is bounded by a polynomial (in k). Then the probability that the adversary wins the sequential experiment with parameters a and b is at most $(2/3)^a$.*

Proof: In the sequential experiment, the adversary chooses a probability p_i in each round $1 \leq i \leq b$. In each of the tests, with probability p_i the adversary fails the whole experiment. With probability $(1-p_i)(p_i+\varepsilon_i)$ it wins the ith test, where ε_i is a negligible fraction (in k). With probability $(1-p_i)(1-p_i-\varepsilon_i)$ nothing happens and we move to the next test.

We will show that for any $j \geq 0$, the probability that the adversary goes from winning j tests to winning $j+1$ tests without getting killed in between, is at most $2/3$, regardless of the choice of the probabilities p_i's. From that we get that the probability that the adversary wins a tests without getting killed is at most $(2/3)^a$.

Suppose the adversary has won j tests without getting killed and it is now trying to win one more. The adversary chooses probabilities p_i's and runs the tests. In each test it either dies, or it wins, or nothing happens. Let β be the number of rounds remaining before the b tests of the experiment end. The probability that the adversary wins one test before it dies and before the game ends is:

$$\mu_1 \stackrel{\text{def}}{=} \sum_{t=1}^{\beta}\left(\prod_{j=1}^{t-1}(1-p_j)(1-p_j-\varepsilon_j)\right)\cdot(1-p_t)(p_t+\varepsilon_t) \tag{1}$$

To show that this probability is less than $2/3$ no matter what the choice of the p_j's is, we compute the probability of a disjoint event. The event that the adversary dies before it wins the $j+1$ test. (Note that there is a third disjoint event in which the adversary does not die and does not win during the remaining β tests.) The probability of the adversary dying before winning is:

$$\mu_2 \stackrel{\text{def}}{=} \sum_{t=1}^{\beta}\left(\prod_{j=1}^{t-1}(1-p_j)(1-p_j-\varepsilon_j)\right)\cdot p_t \tag{2}$$

Comparing μ_1 and μ_2, we see that for each term in the summation, all the factors are the same but the last. Since the ε_i's are negligible (in k) and β is bounded by a polynomial (in k), then we get that

$$\mu_1 - \mu_2 \leq \varepsilon \tag{3}$$

for some negligible fraction ε. Since μ_1 and μ_2 represent the probabilities of disjoint events, then we also get

$$\mu_1 + \mu_2 \leq 1. \tag{4}$$

Combining Equations 3 and 4 we get

$$\mu_1 \leq \frac{1}{2}+\frac{\varepsilon}{2} < \frac{2}{3}$$

and we are done with the proof of Claim 6.10. $\square$

To summarize, the probability that the preamble of any proof instance Π ends well without being solved, is at most $\left(\frac{2}{3}\right)^a$. Recall that $a = \left\lceil \frac{m}{\log(\ell m)+1} \right\rceil - 2 = \omega(\log k)$ (and $b \le (2\ell m)^2$), so we get that the above is a negligible fraction in k. Since we have at most ℓm instances of any of the k proofs, the probability that the preamble of any of these proofs ends well without being solved by our simulator is also negligible and we are done with the proof of Lemma 6.9. □

Using Lemma 6.9, we get that the simulator fails with negligible probability. Also, as in the static case, when the simulator succeeds, it outputs an interaction that is polynomially indistinguishable from the real interaction between the adversarial verifier and the real prover.

7 Resettable zero-knowledge

In this section, we explain how to modify our interactive proof system to make it resettable zero-knowledge. This is similar to the proof given in [4] for an earlier version of this work. We start with the definitions and proceed with the construction.

7.1 Definitions

We provide the definitions of resettable zero-knowledge. For more detailed discussion and motivation the reader is referred to [4].

Definition 7.1 *An interactive proof system (P, V) for a language L is called* resettable zero-knowledge *if for any probabilistic polynomial-time adversary V^* there exists a probabilistic polynomial time simulator M^* so that the following two distribution ensembles are computational indistinguishable: let t be a polynomial $t = poly(n)$, let each distribution be indexed by a sequence of common inputs $x_1, \ldots, x_t \in L \cap \{0,1\}^n$ and a corresponding sequence of prover's auxiliary-inputs $y_1, \ldots, y_t$.*

Distribution 1 *is defined by the following random process which depends on P and V^*:*

1. *Randomly select and fix t random tapes $\emptyset_1, \ldots, \emptyset_t$, for P, resulting in deterministic strategies $P^{(i,j)} = P_{x_i, y_i, \emptyset_i}$ defined by $P_{x_i, y_i, \emptyset_i}(\alpha)$ being the output of P on input x_i auxiliary y_i, random tape $\emptyset_i$ and a history so far α, for all $i, j \in \{1, \ldots, t\}$. Each $P^{(i,j)}$ is called an incarnation of P.*

2. *Machine V^* is allowed to run polynomially many sessions with the $P^{(i,j)}$'s. V^* is allowed to send arbitrary messages to each of the of the $P^{(i,j)}$ and obtain the responses of $P^{(i,j)}$ to such messages.*

Distribution 2: *the output of $M^*(x_1, \ldots, x_t)$.*

We say that an interactive proof is black-box resettable zero-knowledge if there is a single universal simulator that can simulate an adversarial verifier V^* by using V^* as a black box.

As will be shown in Section 7.2, the interactive proof that we present in this paper can be easily modified into an interactive proof that has a specific structure denoted an *admissible* proof system in [4]. Canetti et. al. provide an easy way to prove that an admissible interactive proof is resettable zero-knowledge. Intuitively, an admissible proof is a proof in which the verifier restricts itself in the beginning of the protocol to its answers in the rest of the protocol. In our modified proof (described in Section 7.2 below), the verifier makes a commitment on all its future messages. In the rest of the proof system it sends its messages by revealing the committed values from the beginning of the proof. The prover aborts in case the verifier fails to reveal one of its strings appropriately.

The formal definition of admissible proofs partitions the consideration of the verifier commitments and the rest of the proof, by considering two independent prover modules. One of them deals with the commitments and verification of string revelation, and the other with the rest of the proof. Each revelation of a string contains the string itself (called the main part of the message) and a part witnessing that the revelation is correct (called the authenticator part of the message).

Let us set some conventions before continuing with the formal definition. We assume that the proof always starts with a verifier message specifying an incarnation of a prover $P^{(i,j)}$. The second message is then sent by the prover and is called the *initialization message*. (In the special interesting case the proof starts with a commitment, this message contains the random string of the prover required to make the verifier commitment). The third message is a verifier message denoted the *determining message*. We require that this message includes as a prefix the first two messages of the proof.

Definition 7.2 *A proof system (P, V) is called* admissible *if the following requirements hold:*

1. *The prover P consists of two modules P_1, P_2. Similarly, the random input $\o$ is partitioned into two disjoint parts $\o^{(1)}, \o^{(2)}$, where $\o^{(i)}$ is given to P_i. The initialization message is sent by P_1.*

2. *Each verifier message other than the first is first received by P_1 and is interpreted as consisting of two parts, called* main *and* authenticator. *P_1 decides whether to accept the message or to abort. If P_1 accepts, it forwards the main part of the message to P_2 who generates the next prover message.*

3. *Let V^* be an arbitrary (deterministic) polynomial-size circuit representing a possible strategy for the verifier in the interactive proof (P, V). Then, except with negligible probability, V^* is unable to generate two different messages for some round ℓ that are accepted by P_1 with respect to the same determining message.*

We think of an incarnation of a prover P in an admissible proof as having three indices $P^{(i,j,k)}$. The first index stands for the input and matching auxiliary input x_i and y_i. The second index stands for the random tape of the first prover $\o_j^{(1)}$ and the third index stands for the random tape of the second prover $\o_k^{(2)}$.

In Section 7.2 below, we will present a modification of our protocol that is an admissible proof. It is shown in [4] that for admissible proofs it is simpler to show that the proof is resettable zero-knowledge. In particular, any admissible proof that is *hybrid* zero-knowledge is also resettable zero-knowledge. In the hybrid model of zero-knowledge the verifier is allowed to use many incarnations of provers $P^{(i,j,k)}$ but with a restriction: No two different incarnations may have the same k. Namely, the verifier may run the prover many times, but the random coins of the second prover must be random and independent in each run.

Let us define hybrid zero-knowledge. In Section 7.2 below we will show that our modified proof is both admissible and hybrid zero-knowledge and therefore it is also resettable zero-knowledge. Let us define hybrid zero-knowledge and quote the related assertion from [4].

Definition 7.3 *Let (P, V) be an admissible interactive proof with P_1, P_2 being the two modules of the prover as in Definition 7.2. We say that (P, V) is hybrid zero-knowledge if Definition 7.1 holds when V^* in Distribution 1 of Definition 7.1 is restricted to interacting with incarnations of provers $P^{(i,j,k)}$ such that no two incarnations $P^{(i,j,k)}$ and $P^{(i',j',k')}$ satisfy $k = k'$.*

In Section 7.2 We will use the following corollary of [4]:

Corollary 7.4 *[4], Corollary 9: If an admissible proof (P, V) is hybrid zero-knowledge then (P, V) is resettable zero-knowledge.*

7.2 Constructing resettable zero-knowledge

Let us now show how to modify our interactive proof system to make it resettable zero-knowledge. Our proof is an easy extension of the proof in [4]. The details are provided for self containment. We make the following two modifications. First, we fix the body of the proof to use the zero-knowledge proof for 3-Colorability by Goldreich and Kahan [18]. Recall that in the body of the proof the prover provides a zero-knowledge proof that the input theorem T is true or $\exists i$ s.t. $v_i = p_i$. (See Section 4 above.) Since the original theorem T is an NP statement then so is the assertion "T is true or $\exists i$ s.t. $v_i = p_i$", and thus, given the preamble and the theorem T, it can be reduced to Graph 3-Colorability, and proven in zero-knowledge via the Goldreich-Kahan proof system. A property of this proof, is that the verifier commits on its queries, which are edges in the graph. (For the honest verifier these are edges chosen uniformly at random.)

The second modification we make is that we modify the verifier to commit on its queries in the Goldreich-Kahan proof system in the beginning of the protocol preamble together with the commitments on the values $v_1, \ldots, v_m$. Since the graph is not known at this time of the protocol (recall that the reduction uses the messages sent by the parties during the preamble), the verifier commits to n^4 random pairs of vertices in the graph. Once the graph is determined, the proof will consider only pairs of vertices that are edges in the graph. We remark that the soundness is not substantially changed since the commitments of the verifier are done with statistical secrecy, and since the probability to end up with a quadratic number of edge-challenges is overwhelming.

We would like to show that the resulting protocol is resettable zero-knowledge. But by Corollary 7.4, if an admissible proof (P, V) is hybrid zero-knowledge then (P, V) is resettable zero-knowledge. We will first argue that this protocol is admissible, and then that it is hybrid zero-knowledge, and we are done.

To show that the modified protocol is admissible, we have to define the two provers. We define the first prover module to handle the verifier's commitments. It supplies the random string required for the verifier commitments and then handles the later revelations of committed values (with no need for more random coins). The second prover module proceeds with the preamble only if the first prover accepts the revelations. Then, during the body of the proof, the second prover gets the revealed challenges and answers them. It can be easily verified that this satisfies the definition of admissible proofs.

To show that the protocol is hybrid zero-knowledge we must modify the simulator to handle the modifications in the protocol, and be able to deal with multiple incarnations of the prover using (perhaps) the same random tape for the first prover's module. We first note that the modifications in the protocol do not interfere with the simulation as described. The simulator acts as a receiver in the proof system and that can be done in polynomial time. "Solving a proof" carries the same meaning as before, namely, setting a value p_i to equal v_i for some preamble round i.

The more interesting change is the additional power of the verifier, who may ask that the first module of the prover be used with the same random string in several interactions. In terms of efficiency, this bears no effect on the simulator. The simulator can keep a list of the random coins used so far in the first prover module (i.e., to receive commitments from the verifier) and use these strings repeatedly upon request from the verifier. But will the output still be of (almost) the same distribution as the real interaction?

The distribution of the commitments as played by the first prover module is perfectly simulated since the simulator plays exactly the same algorithm as the prover. It remains to check that the influence of this power given to the verifier is not destructive to the behavior of the simulator with respect to the second prover module. To see that it is not, one must verify that the secrecy and binding properties of the commitment scheme still hold even when the receiver must receive various commitments with the same random string. Looking into the commitment scheme with perfect secrecy in [17] and [9] it turns out that this is a property of these commitment schemes. Namely, the receiver starts by choosing a random string (or a random succinct description of a function) and sends his choice to the verifier. If that choice is good (and that happens with overwhelming probability) then the committer can use that to make a polynomial number of commitments with secrecy and binding properties preserved. (Actually, the same goes also for the computational secrecy commitment scheme in [27], but that is not relevant to the argument.)

One additional issue, that has not been discussed, is whether the simulator may simulate several proofs that do not have the same input. To see that this bears no additional difficulty on the simulator, note that the input is only relevant for the body of the proof and there, the simulator behaves like the real prover, regardless of which specific input the proof uses.

Thus, we get that the modified protocol is resettable zero-knowledge, and we are done with the proof of Theorem 3.2.

8 Acknowledgment

We thank Cynthia Dwork, Uri Feige, Oded Goldreich, Yuval Rabani, and Amit Sahai for many illuminating conversations on this subject. We thank the anonymous referees for their constructive and most helpful comments.

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] G. Brassard, D. Chaum and C. Crépeau. *Minimum Disclosure Proofs of Knowledge.* In *JCSS*, pages 156–189. 1988.

[3] C. Brassard, C. Crepeau and M. Yung, "Constant-Round Perfect Zero-Knowledge Computationally Convincing Protocols", *Theoretical Computer Science*, Vol. 84, 1991, pp. 23-52.

[4] R. Canetti, O. Goldreich, S. Goldwasser and S. Micali. Resettable zero-knowledge. Revision 1 of Report TR99-042, the *Electronic Colloquium on Computational Complexity (ECCC)* `ftp://ftp.eccc.uni-trier.de/pub/eccc/`, June 2000. A preliminary version appears in *Proc. 32nd Annual ACM Symposium on Theory of Computing* May 2000.

[5] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowldege Requires $\tilde{\Omega}(\log n)$ Rounds. *Thirty-Third Annual ACM Symposium on the Theory of Computing*, July 6-8, 2001.

[6] T. Cohen, J. Kilian and E. Petrank. Responsive Round Complexity and Concurrent Zero-Knowledge. Proceedings of *Advances in Cryptology - ASIACRYPT 2001 7th International Conference on the Theory and Application of Cryptology and Information Security*, Australia, December 2001.

[7] G. Di Crescenzo and R. Ostrovsky. "On Concurrent Zero-Knowledge with Pre-Processing". *Proceedings of Advances in Cryptology (CRYPT0-99)*, pp. 485-502, Springer-Verlag Lecture Notes in Computer Science, Vol 1666. 1999.

[8] I. Dåmgard. "Efficient Concurrent Zero-Knowledge in the Auxiliary String Model." *Advances in Cryptology – Eurocrypt 2000 Proceedings*, Lecture Notes in Computer Science, Berlin: Springer-Verlag, 2000.

[9] I. Dåmgard, T. Pedersen and B. Pfitzmann. On the Existence of Statistically Hiding Bit Commitment Schemes and Fail-Stop Signatures. Advances in Cryptology – CRYPTO '93 Proceedings, pp. 250-265. Lecture Notes in Computer Science #773, Berlin: Springer-Verlag, 1994.

[10] D. Dolev, C. Dwork, and M. Naor. "Non-malleable cryptography". In *Proceedings of the 23rd Symposium on Theory of Computing*, ACM STOC, 1991.

[11] C. Dwork, M. Naor and A. Sahai. Concurrent Zero-Knowledge. Proceedings,30*th Symposium on Theory of Computing*, pp. 409–428, 1998.

[12] C. Dwork and A. Sahai. Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints. Proceedings, *Advances in Cryptology – Crypto '98.*

[13] U. Feige. Ph.D. thesis, Weizmann Institute of Science, 1990.

[14] U. Feige, D. Lapidot and A. Shamir. Multiple non-interactive zero-knowledge proofs based on a singe random string. In *Proceedings of the 31st Annual IEEE Symposium on the Foundations of Computer Science*, pages 308–317, 1990.

[15] U. Feige and A. Shamir, "Zero Knowledge Proofs of Knowledge in Two Rounds", *Advances in Cryptology – Crypto 89 proceedings*, pp. 526-544, 1990.

[16] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In Baruch Awerbuch, editor, *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, pages 416–426, Baltimore, MY, May 1990. ACM Press.

[17] O. Goldreich. Foundation of Cryptography - Basic Tools. Cambridge Unievrsity Press. 2001.

[18] O. Goldreich and A. Kahan, "How to Construct Constant-Round Zero-Knowledge Proof Systems for NP", *Journal of Cryptology*, Vol. 9, No. 2, 1996, pp. 167–189.

[19] O. Goldreich, H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. SIAM J. on Computing, Vol. 25, No.1, pp. 169-192, 1996

[20] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that Yield Nothing But their Validity or All Languages in NP Have Zero-Knowledge proof Systems", *Jour. of ACM.*, Vol. 38, 1991, pp. 691–729.

[21] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, Winter 1994.

[22] S. Goldwasser, S. Micali, C. Rackoff. The Knowledge Complexity of Interactive Proofs. Proc. 17th STOC, 1985, pp. 291-304.

[23] S. Goldwasser, S. Micali, and C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems", *SIAM J. Comput.*, 18 (1):186–208, 1989.

[24] J. Kilian. Zero-Knowledge with Log-Space Verifiers *Proceedings, 29th annual IEEE Symposium on the Foundations of Computer Science.*

[25] J. Kilian and E. Petrank. Concurrent zero-knowledge in poly-logarithmic rounds. In *Proceedings of the thirty third annual ACM Symposium on Theory of Computing.* ACM Press, 2001.

[26] J. Kilian, E. Petrank, and C. Rackoff. "Lower Bounds for Zero-Knowledge on the Internet", *Proceedings of the 39nd IEEE Conference on the Foundations of Computer Science*, November 1998.

[27] M. Naor. "Bit Commitment Using Pseudo-Randomness,", *Journal of Cryptology*, vol. 4, 1991, pp.151-158.

[28] Moni Naor and Moti Yung. "Universal one-way hash functions and their cryptographic applications", Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing, 1989, pp. 33-43.

[29] Y. Oren. On the cunning powers of cheating verifiers: Some observations about zero knowledge proofs. In Ashok K. Chandra, editor, *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 462–471, Los Angeles, CA, October 1987. IEEE Computer Society Press.

[30] Ransom Richardson and Joe Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In Proceeedings of *Advances in Cryptology - EUROCRYPT '99*, May 1999, Lecture Notes in Computer Science Vol. 1592 Springer 1999, pp. 415-431

[31] A. Rosen. A Note on the Round-Complexity of Concurrent Zero-Knowledge. *Advances in Cryptology – Crypto 2000 proceedings*, Lecture Notes in Computer Science 1880, p. 451