Concurrent Zero-Knowledge in Poly-logarithmic Rounds

(Extended Abstract)

Joe Kilian * Erez Petrank †

May 28, 2000

Abstract

A proof is concurrent zero-knowledge if it remains zero-knowledge when run in an asynchronous environment, such as the Internet. It is known that zero-knowledge is not necessarily preserved in such an environment; Kilian, Petrank and Rackoff have shown that any **4** rounds zero-knowledge interactive proof (for a non-trivial language) is not concurrent zero-knowledge. On the other hand, Richardson and Kilian have shown that there exists a concurrent zero-knowledge argument for all languages in NP, but it requires a **polynomial** number of rounds. In this paper, we present a concurrent zero-knowledge proof for all languages in NP with a drastically improved complexity: our proof requires only a poly-logarithmic, specifically, $\omega(\log^2 k)$ number of rounds. Thus, we narrow the huge gap between the known upper and lower bounds on the number of rounds required for a zero-knowledge proof that is robust for asynchronous composition.

1 Introduction

Zero-knowledge proofs, presented in [19], are proofs that yield no knowledge but the validity of the proven assertion. Zero-knowledge proofs and arguments [1] have been proven an important tool in various cryptographic applications. However, the original definition of zero-knowledge considers security only in a restricted scenario in which the prover and the verifier execute the proof disconnected from the rest of the computing environment.

In recent years, several papers have studied the affect of a modern computing environment on the security of zero-knowledge. In particular, many computers today are connected through networks (and it may be a small local area networks or the big internet) 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 a common environment?

*NEC Research Institute. E-mail: joe@research.nj.nec.com

†Dept. of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel. Email: erez@cs.technion.ac.il.

1.1 Previous work

Dwork, Naor, and Sahai [8] were the first to explore zero-knowledge in the asynchronous setting. They denoted zero-knowledge protocols that are robust to asynchronous composition *concurrent zero-knowledge* protocols. It was noticed in [8] 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. In order to provide a protocol that may be used in a modern environment, they presented a compromise: a protocol that is not zero-knowledge in a fully asynchronous setting, but is zero-knowledge in an environment with bounds on the asynchronisity. In particular, they present a 4 round zero-knowledge argument for NP assuming that there are two constants α and β such that the fastest message arrives within time at least α and the slowest message arrives within time at most β.

Dwork and Sahai [9] reduced the limitation on the asynchronisity. They presented a proof which has a preprocessing phase and a proof-body phase. Their proof is concurrent zero-knowledge argument for NP such that the (α, β) limitation is required only during the preprocessing stage. Then, the body of the proofs can be run in a fully asynchronous environment.

Kilian, Petrank, and Rackoff [22] presented the first lower bound on concurrent zero-knowledge. They showed 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. The question that rose was: does there exist a fully asynchronous (concurrent) zero-knowledge proof for NP?

The answer was given by Richardson and Kilian [25] who presented concurrent zero-knowledge arguments for all languages in NP, that is robust in the fully asynchronous setting. However, this protocol is not practical. It requires a polynomial number of rounds, which makes it unacceptable in practice. Note the huge gap between the upper and lower bounds for fully asynchronous zero-knowledge proofs for NP. The upper bound has a polynomial number of rounds, whereas the lower bound has 4 rounds.

Other researchers have concentrated on presenting efficient concurrent zero-knowledge protocols for NP with weaker compromises on the asynchronisity of the environment. Crescenzo and Ostrovsky [4] presented a concurrent zero-knowledge argument for NP with a preprocessing phase. They removed the (α, β) constraint of [9], and the only requirement is that there must be a separating point in time between all preprocessing of all concurrent proofs, and all bodies of all proofs. Namely, the first body of any proof may start only after all preprocessing phases in all the proofs have completed. Dåmgard [5] and Canetti et. al. [3] have further reduced the limits on asynchronisity. They require that prior to the beginning of the proofs, all verifiers have deposited a public key in a public database. In [5] it is required that the public key is valid, i.e., that the verifier must know the secret key associated with it. In [3] this requirement is relaxed. The verifier only has to have a deposited string in the public database. Thus, these proofs are efficient, and make only the following compromise over full asynchronisity: all verifiers must be previously registered before they can engage in a proof. A verifier that has not been registered cannot join until all proofs are completed and then it registers itself before any new proof begins.

1.2 This work

Returning to the fully asynchronous environment, the question remains: how many rounds are required for a fully asynchronous concurrent zero-knowledge proof for all languages in NP? In this work, we provide a significant improvement of the upper bound. In particular, we present a concurrent (black-box) zero-knowledge argument for all languages in NP in $\omega(\log^2 k)$ rounds. This improves over the polynomial bound of $O(k^\varepsilon)$ rounds in [25].

The argument we provide relies on the existence of bit commitment for polynomially bounded receiver and committer (see further in Section 2.3 below). Using [1], such schemes can be based on the existence of collision-intractable hash functions.

Our zero-knowledge argument for all languages in NP is a simple modification of the protocol of Richardson and Kilian: we change the parameters to make the preamble short enough, and use a round-efficient zero-knowledge (or witness indistinguishable) proof for NP in the body of the proof. However, our simulator for this proof and analysis of the simulator is completely different from the simulator and analysis in [25]. The simulator in [25] rewinds the verifier according to the adversarial schedule as it is being revealed with time. Interestingly, we propose a rewinding schedule which is oblivious to any of the adversary's actions. The rewinding is done at specific points in time, regardless of the content of messages and regardless of the schedule of the proofs as determined by the verifier. Nevertheless, we are able to show that the simulator manages to simulate the interaction well with probability almost 1.

1.3 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 2.2 below). The prover will be computationally bounded (i.e., we will build a zero-knowledge argument).

1.4 Guide to the paper

In Section 2 we present some definitions and the tools we are using. In Section 4 we present the concurrent zero-knowledge argument for NP. In Section 5 we provide a simulator for the interaction between the prover and the adversarial verifier. In Section 6 we analyze the simulator with respect to a static schedule. Namely, the schedule may be the worst possible, but it is not modified during the rewinds of the simulator. In Section 7 we show that the simulator works as well also with respect to schedules that change dynamically during the simulation. Thus, our proof is concurrent zero-knowledge in the asynchronous setting.

2 Preliminaries

2.1 Zero-knowledge proofs

Let us recall the concept of interactive proofs, as presented by [19]. For formal definitions and motivating discussions the reader is referred to [19].

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 [1] 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 [19, 24, 15, 18]. The reader is referred to [18] for more details and motivation.

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

In this paper, we concentrate on computational zero-knowledge. In the sequel we will say *zero-knowledge* meaning *computational zero-knowledge*.

2.1.1 Concurrent zero knowledge

Following [8], 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 $(P,\mathcal{A})(x)$.

Definition 2.3 *We say that a proof or argument system (P, V) for a language L is (computational)* concurrent zero-knowledge *if there exists a probabilistic polynomial time oracle machine S (the simulator) such that for any probabilistic polynomial time adversary $\mathcal{A}$, the distributions $(P, A)(x)$ and $S^{\mathcal{A}}(x)$ are computational indistinguishable over the strings that belong to the language L.*

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

2.2 Blackbox simulation

The initial definition of zero-knowledge [18] required that for any probabilistic polynomial time verifier $\hat{V}$, a simulator $S_{\hat{V}}$ exists that could simulate $\hat{V}$'s view. Oren [24] 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.

Indeed, all known proofs of zero-knowledge construct black-box simulations. There is no way known to make use of a verifier's internal state, nor to customize simulators based on the description of $\hat{V}$ other than by using it as a black box.[1] Thus, given the current state of the art, an impossibility result for black-box zero-knowledge seems to preclude a positive result for the older definitions of zero-knowledge.

2.3 Bit commitments

We include a short and informal presentation of commitment schemes. For more details and motivation, see [13]. 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

[1] As one slight exception, [20] proves security against space-bounded verifiers by considering the internal state of the verifiers. However, these techniques do not seem applicable to more standard classes of verifiers.

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.

We work in the argument framework of Brassard, Chaum and Crépeau [1]. In this paradigm, all parties are assumed to be computationally bounded. It is shown in [1] how to commit to bits with statistical security, based on the intractability of certain number-theoretic problems. Dåmgard, Pedersen and Pfitzmann [6] give a protocol for efficiently committing to and revealing strings of bits with statistical security, relying only on the existence of collision-intractable hash functions. This scheme is quite practical and we adopt it for the verifiers in our protocol. For the prover, we use a commitment scheme whose binding is information theoretic and security is computational. Such schemes can be construced from any one-way function, see [23]. For simplicity, we will simply speak of committing to and revealing bits when referring to the protocols of [6] for the verifier and [23] for the prover.

2.4 Witness Indistinguishability

Witness indistinguishable proofs were presented in [12]. The motivation was to provide a cryptographic mechanism whose notion of security is similar though weaker than zero-knowledge, it is meaningful and useful for cryptographic protocols, and the security is preserved in an asynchronous composition. A witness indistinguishable proof is a proof for a language in NP such that the prover is using some witness to convince the verifier that the input is in the language, yet, the view of the verifier in case the prover uses witness w_1 or witness w_2 is polynomial time indistinguishable. Thus, the verifier gets no knowledge on which witness was used in the proof. The formal definition follows. For further discussion and motivation the reader is referred to [12].

We say that a relation R is polynomial time if there exists a machine that given (x, w) 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 2.4 *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 [12] that witness indistinguishability is preserved with asynchronous composition of proofs.

2.5 The complexity parameters

In this paper, we simplify the discussion by using a single security parameter k. Our proof has $\omega(\log^2 k)$ rounds and the security is preserved with a polynomial (in k) number of concurrent proofs. It is possible to separate the number k of allowed concurrent proofs from the security parameter. If we know that the number of proofs to be run concurrently is substantially smaller than the security parameter, we can relate the number of rounds to the number of proofs and not to the security parameter. We leave this (and similar) extensions to future versions of this paper.

3 Main result

Our main result is the existence of poly-logarithmic round concurrent (and black box) zero-knowledge arguments for NP. We state this explicitly in the following theorem.

Theorem 3.1 *Assume there is a secure bit commitment scheme, and constant round witness indistinguishable arguments for all languages in NP. Let k be a complexity parameter bounding the size of the input, all parties are polynomial time in k, and the concurrent proof may contain a polynomial (in k) number of proofs concurrently. Then there exists a zero-knowledge argument for all languages in NP which is: computational, black-box, concurrent, and has a number of rounds $t(k)$ which is any function that is asymptotically greater than $\log^2 k$, i.e., $t(k) = \omega(k)$.*

This theorem is proven in the rest of this paper. First we present the protocol, next we present the simulator, and last, we analyze the simulator.

4 The zero-knowledge proof

We use a zero-knowledge argument for NP which is similar to the one suggested by Richardson and Kilian [25], following the ideas presented by Feige, Lapidot and Shamir [10]. The argument proof in [25] for a theorem T in NP consists of a proof-preamble of k^ε rounds and a proof-body being any "standard" zero-knowledge proof for a modified NP theorem T'. We modify the proof parameters to use only a poly-logarithmic number of rounds in the preamble rather than the polynomial number used in [25]. We then use a round-efficient zero-knowledge proof in the main body. The body of the proof can be any (low error) standard round-efficient zero-knowledge (or witness indistinguishable) proof for the languages in NP (see for example [11, 2, 14]). An important property of these known protocols, is that the prover need not be computationally unbounded. It is enough that the prover has a witness for the NP theorem T' that must be proven, and then the prover runs in polynomial time. All these zero-knowledge proofs are also witness indistinguishable, which is enough for us. Feige and Shamir showed that witness indistinguishability is preserved also in the asynchronous setting [12].

The preamble of our proof consists of $2m$ rounds, where $m = \omega(\log^2 k)$ for the security parameter k. Namely, m is asymptotically strictly larger than $\log^2 k$. The main body consists of a constant round zero-knowledge proof for NP. Thus, the number of rounds is dominated by the preamble.

Let us concentrate now on the preamble, which is the main tool in making the zero-knowledge proof 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 will each pick m strings in $\{0,1\}^k$ denoted $p_1, p_2, \ldots, p_m$ and $v_1, v_2, \ldots, v_m$ respectively. (Recall that k is the security parameter.) The prover P will then prove that either T is true or for some i, $1 \leq i \leq m$, $v_i = p_i$. We denote this modified theorem T'. For each i, $1 \leq i \leq m$, 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 6 below.

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

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

$V \rightarrow P$: Reveal v_1
$P \rightarrow V$: Commit to p_2
$\ldots$
$V \rightarrow P$: Reveal v_i
$P \rightarrow V$: Commit to p_{i+1}
$\ldots$
$V \rightarrow P$: 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 he can never tell whether it holds that $p_i = v_i$ for some $1 \leq i \leq m$.

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 an 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 original proof 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 Sketch: It is easy to see that the completeness property is not harmed by the change. Regarding soundness, the advantage a prover P^* may get is by managing to set $p_i = v_i$ for one of the rounds. We need to show that that cannot happen too often. Here, the security

of the verifier's bit commitment is not enough. In order to make sure that the prover cannot cheat, we must require that the verifier's commitment is non-malleable [7]. In order to cheat, the prover does not need to know committed bit. It just needs to produce a commitment such that after the verifier opens its commitment to a certain string, the prover may open its commitment to the same string. Preveting this is exactly the issue in the non-malleability study, and one may use non-malleable commitment schemes as in [7] to make sure that the soundness property is preserved. We choose the following manner to get non-malleability and keep the scheme efficient. The verifier commits using information theoretic secrecy. Thus, the committed value of the prover cannot depend on the committed value (but with negligible probability). Next, the prover commmits with an information theoretic binding scheme. Thus, the committed value binds the prover before it gets to see the verifier openning its commitment. Using these two schemes, the soundness holds.

We remark that the problem is not symmetric. Namely, we do not need non-malleable commitment scheme for the prover. the reason is that the prover never opens its commitments, thus, the verifier can only act upon the knowledge it gets from the commit stage. This gives the verifier nothing by the security property of the commitment scheme. □

5 The simulator

We provide a black-box simulator as explained in Section 2.2 above. Namely, the adversarial verifier V^* is given as a black box and the simulator interacts with it. We assume that by the time the simulator gets to the main body, it has a witness to the modified theorem T'. 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.) In particular, the simulator will rewind the verifier at several points in their interaction. If the verifier reveals v_i before a rewind, 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 first round of the interaction), then unless the rewind goes beyond the first round of the 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 round 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 it has a witness to the theorem T' that has to be proven). We stress that the rest of the simulation requires no further rewinding. The main good feature of the preamble is exactly this. It is enough to rewind once in any of the m rounds of the preamble, and the proof is solved. Of-course, if during a predetermined rewind the simulator can solve more than one proof by setting p_j's of other proofs to values of v_j's that were discovered during the first run of rewound interval, then it solves all the proofs it can.

Note that rewinding one step in one proof may render irrelevant 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 [8] and proved for a set of protocols in [22]. We

employ a different strategy of rewinding. We specify a fixed rewinding timing 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.

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 on in the first round. 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 may abort 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.

5.1 The adversarial scheduler uses round slots

We begin by simplifying our view of the adversarial schedule. Recall that we are running k proofs, each has $2m$ rounds. W.l.o.g., we assume that the real prover (and so also the simulator) always answers immediately. Namely, the adversarial verifier may delay its answers as it pleases, but the prover answers at once. Thus, we get that the adversarial verifier V^* may schedule an overall number of $k \cdot m$ pairs of rounds in the preambles. We are only interested in the scheduling of the preambles. We do not care how the body of the proof is scheduled and whether it will be rewound. We will not need to rewind the bodies: the simulator will behave like the real prover in the bodies.

We first remark that We do not care how much actual time passes between one pair of rounds to the next. Our schedule will repeat each of these times a polynomial number of times, so the simulation time will be a polynomial times the sum of the pauses, which must be polynomial since the adversarial verifier is a polynomial time machine. The output delays are those determined by the verifier in the last run of the rewinds. Meaning that they will be distributed like in the real interaction.

A second remark is about the possibility that the adversarial verifier schedules messages in parallel. In the sequel, we do not explicitly consider parallel pairs of rounds. If the adversary sends more than one verifier's message to the prover in parallel, then the prover answers all of them in parallel. Thus, we get less then $k \cdot m$ pairs of actual rounds run. In the analysis we will analyze the probability that "something bad" happens within a specific proof, ignoring the rest of the proofs that run with it. Thus, it will not matter if this proof is run in parallel to other proofs. Note that rounds of the same proof cannot run in parallel, since the order within a proof is guaranteed to be preserved in the concurrent setting. Parallel repetitions will reduce the number of pairs of rounds and that may only make the simulation more efficient. We will not explicitly discuss parallel repetitions in the sequel.

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 km rounds, with the only constraint that within a proof the order of rounds is preserved. We consider only the rounds of the preambles and completely ignore rounds of the proof-bodies that happen while the concurrent preambles run. In fact, we think of this schedule as assigning rounds of the various (preambles of) proofs to km "slots" of rounds. We consider the km slots by their order in time, and specify the rewinding strategy with respect to these slots, regardless of how the adversary assigns rounds to these slots. For example, we may let the simulator rewind the verifier to the first slot after running the second 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 $(j \rightarrow i)$, for $1 \leq i < j \leq km$, means that after reading the verifier's message of round slot j, the simulator rewinds the simulation back to 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 $j-1$.

5.2 Specification of the rewind timing

We use recursion to specify the rewinding timing. At the top level of the recursion, the simulator is running all the round slots $1..mk$. 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 $mk/2$ 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. Finally, at the bottom level, we are left with one or two round slots. 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).

Let us explain this rewinding schedule with an example. Suppose the number of round slots, mk, is 8 (just for the sake of this example), then the round slots are run by the simulator in 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: $(2 \rightarrow 1)$, $(4 \rightarrow 3)$, $(4 \rightarrow 1)$, $(2 \rightarrow 1)$, $(4 \rightarrow 3)$, $(6 \rightarrow 5)$, $(8 \rightarrow 7)$, $(8 \rightarrow 5)$, $(6 \rightarrow 5)$, $(8 \rightarrow 7)$. At the top recursion level, we execute $(4 \rightarrow 1)$ and $(8 \rightarrow 5)$, which means rewinding the first and the second half of the round slots. Each of the two times we run rounds 1 to 4, we rewind the first and the second half of that, getting $(2 \rightarrow 1)$ and $(4 \rightarrow 3)$ performed twice. The same goes for the rewinding of the second half with round slots 5 to 8. The recursion ends here, since we are left with two round slots in the second level of recursion.

6 Analysis of the simulator with respect to a static schedule

To simplify the presentation of the analysis of our simulator, we start in this section by showing that the simulator works well for a static schedule. Namely, 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 Section 7 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.

We first note that the number of rounds run in the recursion is at most $(mk)^2$ and thus, the simulator runs in polynomial time. Our goal is to show that with very high probability the simulator will manage to obtain a witness for T' 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 (recursive) manner we defined the rewinding schedule, the rewind intervals are either disjoint or contained within each other. So for any two rewind intervals $(j \rightarrow i)$ and $(\ell \rightarrow k)$ if $i < \ell \leq j$, then it holds that k must be greater or equal to i. In the above case, in which the rewind interval $(\ell \rightarrow k)$ is contained within the rewind interval $(j \rightarrow i)$ we will say that the rewind $(j \rightarrow i)$ *dominates* the rewind $(\ell \rightarrow k)$.

Definition 6.1 *We say that a rewind* $(\ell \rightarrow k)$ dominates *the rewind* $(j \rightarrow i)$ *if* $k \leq i < j \leq \ell$.

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 above rewinding timing lets the simulator get "good" runs with very high probability, no matter what schedule is chosen for the messages in the proofs. During the simulation, we do not need to rewind bodies of proofs, though, of-course a rewinding of a proof body that happens while rewinding a preamble of another proof does not hurt the simulation.

A proof Π may be solved via a rewind $(j \rightarrow i)$ if the there are at least two rounds of Π appearing within the round slots $i, i+1, \ldots, j$, and the proof Π does not begin or end during the round slots $i+1, i+2, \ldots, j$. The proof is actually solved with 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: Round a and Round $a+1$, $(2 \leq a \leq m-2)$ within the rewind interval. Thus, in the second run of these rounds, we can set p_a in the preamble to equal v_a, and solve the proof. The reason we require that the first round of the proof Π does not appear within the rewind interval is that if we rewind beyond the first round of the proof, then V^* gets to run its first round again, 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 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 we will noticeably fail in the simulation, 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 $(j \rightarrow i)$ that may solve the proof, and there exists a larger rewind $(\ell \rightarrow k)$ that dominates it, then the existence of $(\ell \rightarrow k)$ does not "ruin" the ability of $(j \rightarrow i)$ to solve the proof. This is true since in both runs of the rounds $\ell, \ell+1 \ldots, k$ in the dominating rewind interval we rewind $(j \rightarrow i)$. So even if the rewind $(j \rightarrow i)$ 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 $(\ell \rightarrow k)$, then sometimes it may also be solved by several rewinds that dominate $(\ell \rightarrow k)$. However, we will be interested only in the smallest rewind interval that may solve a proof. Minimality is expressed in Condition

(4) of the following definition. This minimality property will be used in the proof of the dynamic schedule, in Section 7 below.

Definition 6.2 *We say that a rewind* $(\ell \to k)$ may solve *a proof* Π *if the following four conditions hold:*

1. *At least two rounds of the preamble of* Π *take place during round slots* $k, k+1, \ldots, \ell$,
2. *the first round of* Π *takes place at a round slot* $i < k$,
3. *the last round of* Π *takes place at a round slot* $j > \ell$, *and*
4. *any rewind* $(b \to a)$ *that is dominated by rewind* $(\ell \to k)$ *does not satisfy condition 1 with respect to* Π.

Note that a rewind that may solve a proof contains exactly two rounds of that proof. It contains at least two by Condition (1) of the definition. If it contains more than 2, then there must be a dominated rewind interval that contains at least two such rounds, thus, the dominating rewind interval does not satisfy the minimality condition (4). We now claim that for each proof, there are "many" rewinds that may solve it.

Lemma 6.3 *For any schedule of* k *copies of the proof (in the* mk *round slots), if a preamble of a specific proof* Π *completes in round slot* ℓ, *then there are at least* $\left\lceil \frac{m}{\log(mk)+1} \right\rceil - 2$ *rewind intervals that complete by round* ℓ *and that may solve* Π.

Proof: We first show that there are at least $\left\lceil \frac{m}{\log(mk)+1} \right\rceil$ rewind intervals that satisfy Conditions (1) and (4) in Definition 6.2 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 $\left\lceil \frac{m}{\log(mk)+1} \right\rceil - 2$ as required. Clearly, any good interval must end by round ℓ, since the preamble terminates at round ℓ.

Fix a proof Π and any schedule of the rounds for all the proofs. We denote a rewind interval *good* if it satisfies Conditions (1) and (4) in Definition 6.2 above (with respect to Π). Consider the rewinds by the height of the recursion. At the top level, i.e., recursion height $\lceil \log(mk) \rceil$, we have mk 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 almost[2] two 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 in each recursion invocation. The split of these rounds of Π is not necessarily 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 at 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 $h \geq \log(mk)$ 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 6.3.

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

Claim 6.4 *For any schedule of k copies of the proof (in mk 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 at 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 rewind interval 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.) There are r_1 rounds that go into the first rewind interval, and r_2 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 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 6.4. □

As mentioned above, this also concludes the proof of Lemma 6.3 since for any proof Π there are m rounds at recursion level $\lceil \log(mk) \rceil$, and since only two of them may contain the first or last round of the preamble. □

6.1 Why the rewinding works

We would like to claim now that the simulator will be able to solve each proof during its preamble and before it is required to simulate the main body of the proof. By Lemma 6.3, for each of the k proofs, there are at least $\left\lceil \frac{m}{\log(mk)} \right\rceil - 2$ rewind intervals that may solve it. Of-course, it is enough that for each proof there is one rewind that solves it during the preamble. If we have one such rewind, the simulator can properly simulate each proof and all of them together no matter what the schedule is.

However, there is a delicate point to consider here. It is not always the case that a proof is **solved** in a rewind that **may 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). Let the number of these rounds be a and $a+1$. The proof is solved in this rewind unless the following event happens: the verifier **does not** reveal the committed value v_a in the first run, but **does** reveal the committed value v_a in the second run. All three other alternatives (i.e., the verifier reveals the committed values in both runs, or does not reveal the committed value in both runs, or reveals the committed value only in the first run) allow the simulator solve the proof Π in this rewind. If the verifier reveals the committed value in the first run, 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, then the proof Π is also solved, since the prover does not answer any of the following rounds of the proof Π, and the simulator may easily "simulate" that.

We stress that the following naive solution would not work here. One may want to abort this proof if either in the first or in the second run V^* does not reveal the committed value. This solution does not work, since it increases the probability of aborting Π above the probability of aborting Π in the real proof. 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. Of-course the verifier doesn't "know" that it has been rewound, so it cannot make an effort to abort the first run and behave well on the second run. However, when we solve a proof, the second run 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. The probability that the bad verifier can tell that a proof has been solved is, thus, negligible. 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. 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. The simulation succeeds if 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 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 $2^{\lceil \log(km) \rceil} \leq 2km$, 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.

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

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

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.

To summarize, with probability almost 1, up to a negligible fraction, the simulator solves all proofs within their preamble and thus, can finish the simulation successfully. Also, The schedule of the various proofs is independent of how many rewinds have been run, since the adversarial verifier does not know that it is being rewound. Also, assuming that the bit commitment that the prover is using is secure, the schedule does not depend on whether the simulator has managed to solve the proofs. Thus, the schedule of the proofs is indistinguishable from the schedule in the real interaction. Finally, the actual content of the preamble is similar to the real interaction (except for proofs being solved, which is indistinguishable in polynomial time by the security of the bit commitment), and the content of the proof bodies in the simulations is indistinguishable from the proof bodies in the real interaction since the simulator actually behaves as a prover in the bodies. by the witness indistinguishability property of the proof body, the fact that the simulator uses different witnesses from the real prover is polynomial time indistinguishable. Feige and Shamir showed that witness indistinguishability is preserved also in the asynchronous setting [12]. Thus, the transcript of the interaction is indistinguishable in polynomial time from the output of the simulation.

7 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 mk 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 7.1 *(Dynamic analogue of Definition 6.2:) 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 there are no rewind intervals dominated by ρ that contain two rounds of Π.*

Definition 7.2 *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 during 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, 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. Similarly to Lemma 6.3, the following lemma holds.

Lemma 7.3 *In any schedule of k copies of the proof (in mk round slots), if a preamble of a specific proof Π completes in round ℓ, then there are at least $\left\lceil \frac{m}{\log(mk)+1} \right\rceil - 2$ rewind intervals that completed before round ℓ with a second good run with respect to Π.*

Proof: similar to the proof of Lemma 7.3. Proof omitted.

By Lemma 7.3, before a preamble may complete, the history must contain at least $a \stackrel{\text{def}}{=} \left\lceil \frac{m}{\log(mk)+1} \right\rceil - 2$ 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 7.4 *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 completion of a preamble of a specific proof Π, 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 7.3, 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 7.1).

We may think of the adversarial verifier as running the following stochastic experiment, which we denote *the sequential experiment*. It runs through a series of 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). 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. 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. This is the case since for a preamble to complete 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 $b \leq 2mk$.

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

Claim 7.5 *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 $\ell \geq 0$, the probability that the adversary goes from winning ℓ tests to winning $\ell+1$ 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 ℓ 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 $\ell+1$ 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 7.5. □

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 $a = \left\lceil \frac{m}{\log(mk)+1} \right\rceil - 2 = \omega(\log k)$ (and $b \leq (2mk)^2$), so we get that the above is a negligible fraction in k. Since we have at most mk 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 7.4. □

Using Lemma 7.4, 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.

8 Acknowledgment

We thank Uri Feige, Oded Goldreich, and Yuval Rabani for helpful discussions.

References

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

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

[3] R. Canetti, O. Goldreich, S. Goldwasser and S. Micali. Resettable zero-knowledge. In *Proc. 32nd Annual ACM Symposium on Theory of Computing* May 2000.

[4] 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.

[5] 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.

[6] 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.

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

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

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

[10] 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.

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

[12] 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.

[13] O. Goldreich. Foundation of Cryptography - Fragments of a Book . Available from the *Electronic Colloquium on Computational Complexity (ECCC)* `http://www.eccc.uni-trier.de/eccc/`, February 1995.

[14] 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.

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

[16] 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.

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

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

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

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

[21] J. Kilian and E. Petrank. "Improved Lower Bounds for Concurrent Zero-Knowledge", Manuscript, April 2000.

[22] 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.

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

[24] 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.

[25] R. Ransom and J. Kilian. Non-Synchronized Composition of Zero-Knowledge Proofs. Manuscript.