

Concurrent Zero Knowledge, Revisited

Rafael Pass* Wei-Lung Dustin Tseng* Muthuramakrishnan Venkitasubramaniam*

January 2, 2013

Abstract

We provide a more general and, in our eyes, simpler variant of Prabhakaran, Rosen and Sahai's (FOCS '02) analysis of the concurrent zero-knowledge simulation technique of Kilian and Petrank (STOC '01).

Keywords: zero-knowledge, concurrency, oblivious simulation.

*Cornell University, E-Mail: {rafael, wdtseng, vmuthu}@cs.cornell.edu. Pass was supported in part by a Alfred P. Sloan Fellowship, Microsoft New Faculty Fellowship, NSF CAREER Award CCF-0746990, AFOSR YIP Award FA9550-10-1-0093, and DARPA and AFRL under contract FA8750-11-2-0211. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US government.

1 Introduction

Following the seminal works of [DDN00, FS90] from the early 90’s, concurrent security of cryptographic protocols has been an active area of research. Yet, it is still not well-understood when and where concurrent security can be achieved. One potential reason for this might be the complexity of traditional analyses. In this work we focus on generalizing and (in our eyes) simplifying analyses of concurrent security in one of the most basic settings, namely that of *zero-knowledge proofs*.

Zero-knowledge (ZK) interactive proofs [GMR89] are paradoxical constructs that allow one player (called the Prover) to convince another player (called the Verifier) of the validity of a mathematical statement $x \in L$, while providing *zero additional knowledge* to the Verifier. Beyond being fascinating in their own right, ZK proofs have numerous cryptographic applications and are one of the most fundamental cryptographic building blocks. As such, techniques developed in the context of ZK often extend to more general types of interactions.

The notion of concurrent zero knowledge, first introduced and achieved in the paper by Dwork, Naor and Sahai [DNS04], considers the execution of zero-knowledge proofs in an asynchronous and concurrent setting. More precisely, we consider a single adversary mounting a coordinated attack by acting as a verifier in many concurrent executions (called sessions). Concurrent ZK proofs are significantly harder to construct and analyze. Since the original protocols by Dwork, Naor and Sahai (which relied on so called “timing assumptions”), various other concurrent ZK protocols have been obtained based on different set-up assumptions (e.g., [DS98] [Dam00] [CGGM00]). In the standard model without set-up assumptions (the focus of our work), Canetti, Kilian, Petrank and Rosen [CKPR01] (building on earlier works by [KPR98] [Ros00]) show that concurrent ZK proofs for non-trivial languages, with so called “black-box” simulators, require at least $\tilde{\Omega}(\log n)$ number of communication rounds. On the other hand, Richardson and Kilian [RK99] constructed the first concurrent ZK argument in the standard model without any extra set-up assumptions. Their protocol, which uses a black-box simulator, requires $O(n^\epsilon)$ number of rounds. (See also the work of Canetti, Goldreich, Goldwasser and Micali [CGGM00] for a somewhat different and more detailed analysis of this protocol.) Kilian and Petrank [KP01] then introduced a new *oblivious* zero-knowledge simulator. Using this simulation technique they obtained a simpler and cleaner analysis, and additionally improved the round complexity to $\tilde{O}(\log^2 n)$. Finally, the work of Prabhakaran, Rosen and Sahai [PRS02] further simplifies and improves the analysis of the oblivious simulator, obtaining an essentially optimal round complexity of $\tilde{O}(\log n)$.

Despite these simplifications and improvements, the analysis of concurrent zero-knowledge protocols remains quite complex. Furthermore, the different analyses are tailored to different types of protocols. In particular, the most refined analysis from [PRS02] considers *committed-verifier* protocols, where the verifier commits to its messages in advance; more specifically, as far as we know, the analysis has only been applied to generalizations of the Goldreich-Kahan ZK protocol [GK96]. For instance, no generalizations of the Feige-Shamir ZK protocol [FS90] have been analyzed using it; apart from theoretical interests, the Feige-Shamir ZK protocol is noteworthy due to its efficient instantiations via “sigma protocols” [CDS94].

In this work, we focus on simplifying and generalizing current analysis techniques for concurrent ZK. More precisely, we provide a variant of Prabhakaran, Rosen and Sahai’s (PRS) analysis [PRS02] of the Kilian-Petrank (KP) zero-knowledge simulator [KP01]. Our contribution is twofold:

- In our eyes, this analysis is simpler and more flexible than the original PRS analysis. In particular, the analysis also directly applies to more efficient variants of the KP-simulator,

resulting in concurrent ZK protocols with “tight” [GMW91, Gol01], and even “precise” [MP06] simulations (i.e., simulations where the running-time of the simulator is close to the running-time of the malicious verifier, in an execution-by-execution manner). Such results were already established in [PPS⁺08], but required a more elaborate analysis (building upon [PRS02]).

- Our analysis applies to a broad range of protocols, and in particular to “Feige-Shamir”-type protocols. As a consequence, we establish a simple $\omega(\log n)$ -round concurrent zero-knowledge argument of knowledge for NP based on one-way functions. The same protocol construction also yields a poly(n)-round concurrent statistical ZK argument of knowledge for NP, based on one-way functions (concurrent statistical ZK arguments were first constructed in [GMOS07] using a more complex protocol). Furthermore, in a subsequent work, Lin et. al. [LPTV10] rely on our analysis to construct concurrent *non-malleable* zero-knowledge proofs for NP; our analysis is helpful in this context since their protocol is not of the committed-verifier type.

Previous techniques. Kilian and Petrank’s (KP) ingenious simulation technique relies on a static—and oblivious—rewinding schedule; namely, the simulator rewinds the adversarial verifier after some fixed number of messages, independent of the content of the messages and the interleaving schedule of the sessions. The crux of their analysis is to show that using this rewinding schedule, every session is “successfully rewound” at least once with high probability; in a successful rewind, the simulator can extract a “trapdoor” that will allow it to complete the simulation. To bound the failure probability, they rely on a subtle computation of conditional probabilities.

The elegant work of Prabhakaran, Rosen and Sahai (PRS) [PRS02, Ros03, PS02], on the other hand, directly analyze the probability space of the simulator, i.e., count the random tapes of the simulator; this makes the analysis both simpler and sharper. The idea is to show that each “bad” random tape (that produces a failed simulation) can be mapped into super-polynomially many *distinct* “good” tapes. This is done by identifying random tape segments, called *rewinding intervals*, that can be “swapped” among each other in order to turn a bad tape into a good one¹. The crux of their proof is then to count how many such “swappings” actually generate new and distinct random tapes. However, complications arise since swappings performed on different rewinding intervals may overlap and even remove other possible rewinding intervals. A bit more precisely, the PRS analysis focuses only on “disjoint” rewinding intervals, but performs a computation based on the “multiplicity” on those intervals. A count with multiplicity is needed because the number of disjoint rewind intervals in general could not be guaranteed to be sufficiently large, at least in the case of $\omega(\log n)$ round protocols. (As we shall see, in our analysis, we are able to swap also non-disjoint rewinding intervals; as a result, we can avoid the count with multiplicities.)

Additionally, to enable this counting argument, the PRS analysis bounds the failure probability of a “hybrid” simulator (which has access to the witnesses of input statements). To show that the real simulator is indistinguishable from the hybrid simulator, committed-verifier protocols are used; this is required to ensure that when changing the hybrid simulator (which uses the actual witness) to the real simulator (which doesn’t know the actual witness), indistinguishability holds despite the rewinds performed by the simulator. Intuitively, the committed-verifier property ensures that the rewinds are “harmless”.

¹Here we use the terminologies from Rosen’s thesis [Ros03].

Our techniques. We show how to directly analyze the failure probability of the *actual* simulator (as opposed to a hybrid one), while (in our eyes) simplifying the counting argument. Our key step is to identify a *stronger* notion of rewinding intervals, which we call *composable blocks*. Just like rewinding intervals, properties of *composable* blocks guarantee that a “swap” will generate a new good random tape; moreover, these same properties are closed under *composition*—namely the swapping of one such block leaves other *composable* blocks intact, even if these *composable* blocks are not disjoint. By this new composition property, it is enough to identify K *composable* blocks to conclude that the simulation fails with probability less than 2^{-K} .

In essence, our proof will consist of two simple steps: First, we establish *local* properties of a *composable* block (namely that a swap generates one new good random tape, and that swappings are *composable*); then, we count the number of *composable* blocks on a bad random tape; as we shall see, each round in the protocol gives rise to a new *composable* block. As such, our analysis conveys a strong intuition of how “each additional round of the protocol halves the simulator’s failing probability”. However, we emphasize that our techniques do not improve the “quantity” of the count (e.g., does not improve upon the round-complexity of the PRS protocol).

To employ this new notion of *composable* blocks, we consider and analyze a “lazy” variant of the KP simulator. Intuitively, the lazy KP simulator is identical to the KP simulator but only makes use of information gathered in its rewinds after some delay. The lazy KP simulator can only fail more often than the original KP simulator (and thus our analysis indirectly also applies to the KP simulator); yet, considering this “weaker” simulator enables our way of directly analyzing the failure probability of the simulation. In a sense, much like making a stronger inductive hypothesis can simplify the inductive step, our stronger notion of *composable* blocks and our weaker lazy KP simulator enable and simplify the analysis. (We note that the PRS analysis also seems to apply to the lazy KP simulator, although this was not made use of in [PRS02].)

After directly bounding the failure probability of the real simulator, we provide a simple hybrid argument to show that the output of the simulator is indistinguishable from the view of the verifier. The base case of this hybrid argument considers only a “straight-line” (i.e., a non-rewinding) execution, and as such the analysis directly applies also to committed-verifier protocols.

Overview. We define concurrent ZK and give some preliminaries in Section 2. We construct and analyze computational and statistical concurrent black-box ZK arguments of knowledge in Section 3 (our main theorems). For completeness, we also provide a brief overview of the PRS analysis in Appendix A.

2 Preliminaries

We assume familiarity with indistinguishability and interactive proofs. $[n]$ denotes the set $\{1, \dots, n\}$.

2.1 Black-box Concurrent Zero-Knowledge

Let (P, V) be an interactive proof for a language L . An m -session *concurrent adversarial verifier* V^* is a probabilistic polynomial time machine that, on common input x and auxiliary input z , interacts with $m(|x|)$ independent copies of P concurrently (called *sessions*). There are no restrictions on how V^* schedules the messages among the different sessions, and V^* may choose to abort some sessions but not others. Let $\text{View}_{V^*}^P(x, z)$ be the random variable that denotes the *view* of $V^*(x, z)$

in an interaction with P (this includes the random coins of V^* and the messages received by V^*). A *black-box simulator* S is a probabilistic polynomial-time machine that is given black-box access to V^* (written as S^{V^*}). Roughly speaking, we require that for every instance $x \in L$, and every auxiliary input z , $S^{V^*(x,z)}(x)$ can generate the view of $V^*(x, z)$ in an interaction with P . Since we provide V^* with an auxiliary input, we can without loss of generality restrict our attention to deterministic V^* (as V^* can always receive its random coins as auxiliary advice).

Definition 1 (Black-Box Concurrent Zero-Knowledge [DNS04]). Let (P, V) be an interactive protocol for a language L . Π is *black-box concurrent zero-knowledge* if for all polynomials m , there exists a black-box simulator S_m such that for every common input x and auxiliary input z , and every deterministic m -session concurrent adversary V^* , $S_m^{V^*(x,z)}(x)$ runs in time polynomial in $|x|$. Furthermore, the ensembles $\{\text{View}_{V^*}^P(x, z)\}_{x \in L, z \in \{0,1\}^*}$ and $\{S_m^{V^*(x,z)}(x)\}_{x \in L, z \in \{0,1\}^*}$ are computationally indistinguishable (as a function of $|x|$).

2.2 Other Primitives

Witness-indistinguishable (WI) Proofs [FS90]. Roughly speaking, an interactive proof is *witness indistinguishable* if the verifier’s view is “independent” of the witness used by the prover for proving the statement.

Definition 2 (Witness-indistinguishability). Let (P, V) be an interactive proof system for a language $L \in \text{NP}$ with witness relation R_L . We say that (P, V) is *witness-indistinguishable* for R_L if for every probabilistic polynomial-time adversarial V^* and for every two sequences of witnesses $\{w_x^1\}_{x \in L}$ and $\{w_x^2\}_{x \in L}$ satisfying $w_x^1, w_x^2 \in R_L(x)$, the following two probability ensembles are computationally indistinguishable as a function of n :

$$\{\text{View}_{V^*}^{P(w_x^1)}(x, z)\}_{n \in \mathbb{N}, x \in L \cap \{0,1\}^n, z \in \{0,1\}^*} \approx \{\text{View}_{V^*}^{P(w_x^2)}(x, z)\}_{n \in \mathbb{N}, x \in L \cap \{0,1\}^n, z \in \{0,1\}^*}$$

Proofs and arguments of knowledge (POK, AOK) [FS90, BG92]. An interactive proof (resp. argument) is a proof (resp. argument) of knowledge if the prover convinces the verifier that it *possesses*, or can *feasibly compute*, a witness for the statement proved.

Definition 3 (Proofs and arguments of knowledge [BG92]). An interactive protocol $\Pi = \langle P, V \rangle$ is a *proof of knowledge* (resp. *argument of knowledge*) of language L with respect to witness relation R_L if Π is indeed an interactive proof (resp. argument) for L . Additionally, there exists a polynomial q , a negligible function ν , and a probabilistic oracle machine E , such that for every interactive machine P^* (resp. for every polynomially-sized machine P^*) and every $x \in L$, the following holds:

If $\Pr[\langle P^*, V \rangle(x) = 1] > \nu(|x|)$, then on input x and oracle access to $P^*(x)$, machine E outputs a string from the $R_L(x)$ within an expected number of steps bounded by

$$\frac{q(|x|)}{\Pr[\langle P^*, V \rangle(x) = 1] - \nu(|x|)}$$

The machine E is called the knowledge extractor.

Special-Sound (SS) Proofs [CDS94]. Special-sound proofs are proofs of knowledge with a very rigid and useful structure.

Definition 4 (Special soundness). A 4-round interactive proof (P, V) for language $L \in \text{NP}$ with witness relation R_L is *special sound* with respect to R_L if (P, V) is public-coin (i.e., verifier messages are segments of its random tape), and on input x , all verifier messages have length $g(|x|) \geq |x|$.

Moreover, there exists a deterministic polynomial-time extraction procedure X such that on input x , with all but negligible probability in $|x|$ over the choice of a uniform $\rho \in \{0, 1\}^{g(|x|)}$, for all $\alpha, \beta, \beta', \gamma, \gamma'$ such that $\beta \neq \beta'$, and $(\rho, \alpha, \beta, \gamma)$ and $(\rho, \alpha, \beta', \gamma')$ are both accepting transcripts of (P, V) on input x , $X(x, (\rho, \alpha, \beta, \gamma), (\rho, \alpha, \beta', \gamma'))$ outputs a witness $w \in R_L(x)$.

2.3 Known Protocols

In our construction of concurrent zero-knowledge arguments we use:

- 4-round computational WI and SS proofs based on one-way functions. This can be instantiated with a parallel repetition of the Blum Hamiltonicity protocol [Blu86] with 2-round statistically binding commitments constructed from one-way functions ([Nao91, HILL99]).
- 4-round computational WI-AOK or $\text{poly}(n)$ -round statistical WI-AOK based on one-way functions. Again, this can be instantiated with the Blum Hamiltonicity protocol with the help of 2-round statistically binding commitments ([Nao91, HILL99], this actually gives a POK) or statistically hiding commitments ([HNO⁺09]) from one-way functions.

3 Black-Box Concurrent Zero-Knowledge Arguments of Knowledge

In this section we re-prove the following theorem.

Theorem 1. *Assume the existence of one-way functions. Then every language in NP has a $\omega(\log n)$ -round concurrent black-box ZK argument of knowledge, and a $\text{poly}(n)$ -round concurrent black-box statistical-ZK argument of knowledge.*

3.1 The Protocol

Our concurrent ZK protocol ConcZKArg (also used in [PV08]) is a slight variant of the precise ZK protocol of [MP06], which in turn is a generalization of the Feige-Shamir protocol [FS90]. The protocol for language L proceeds in three stages, given a security parameter n , a common input statement $x \in \{0, 1\}^n$, and a “round-parameter” $k \in \omega(\log n)$:

Stage Init: The verifier picks two random strings $r_1, r_2 \in \{0, 1\}^n$ and sends their images $c_1 = f(r_1)$, $c_2 = f(r_2)$ through a one-way function f to the prover. The verifier then provides, in parallel, k instances of a 4-round computationally-WI and SS proof of knowledge of the NP statement “ c_1 or c_2 is in the image set of f ” (a witness here would be a pre-image of c_1 or c_2). The first two (out of four) messages of each SS-POK are exchanged in this stage. The end of Stage Init is called the START of the protocol.

Protocol ConcZKArg:

Common Input: an instance x of a language L with witness relation R_L .

Auxiliary Input for Prover: a witness w , such that $(x, w) \in R_L(x)$.

Stage Init:

V uniformly chooses $r_1, r_2 \in \{0, 1\}^n$.

$V \rightarrow P$: $c_1 = f(r_1), c_2 = f(r_2)$ for a one-way function f .

$V \leftrightarrow P$: Exchange in parallel (interactively) the first two messages $\vec{\alpha}_1, \dots, \vec{\alpha}_k$ of k copies of 4-round computational-WI and SS proofs on common input (c_1, c_2) with respect to the witness relation:

$$R_f(c_1, c_2) = \{r : f(r) = c_1 \text{ or } f(r) = c_2\}$$

Note that V acts as the prover in these SS-POK's.

We say the protocol has reached START (of Stage 1) if all messages in Stage Init are exchanged.

Stage 1: For $j = 1$ to k do the following (called a *slot*)

$P \rightarrow V$: The second last message β_j of the j^{th} SS-POK.

$V \rightarrow P$: The last message γ_j of the j^{th} SS-POK.

We say the protocol has reached END (of Stage 1) if all k SS-POK are accepted.

Stage 2:

$P \leftrightarrow V$: a 4-round computational-WI (or $\text{poly}(n)$ -round statistical WI) argument of knowledge from P to V on common input (c_1, c_2, x) with respect to the witness relation:

$$R_{f \vee L}(c_1, c_2, x) = \{(r, w) : r \in R_f(c_1, c_2) \text{ or } w \in R_L(x)\}$$

Figure 1: Concurrent ZK argument of knowledge for NP with round parameter k .

Stage 1: k message exchanges occur in Stage 1. In the j^{th} iteration, the prover sends $\beta_j \in \{0, 1\}^n$, a random second last message of the j^{th} SS-POK, and the verifier replies with the last message γ_j of the SS-POK. These k iterations are called *slots*. A slot is *convincing* if the verifier produces an accepting proof. If there is ever an *unconvincing* slot, the prover aborts the whole session. The end of Stage 1 (after k convincing slots) is called the END of the protocol.

Stage 2: The prover provides a 4-round computational-WI (resp. $\text{poly}(n)$ -round statistical-WI) argument of knowledge of the statement “ $x \in L$, or one of c_1 or c_2 is in the image set of f ”.

Completeness and soundness/proof of knowledge follows directly from the proof of Feige and Shamir [FS90]; in fact, the protocol is an instantiation of theirs. Intuitively, to cheat in the protocol a prover must “know” an inverse to c_1 or c_2 (since Stage 2 is an argument of knowledge), which requires inverting the one-way function f . A formal description of protocol ConcZKArg is shown in Figure 1.

3.2 The “Lazy KP” ZK simulator

We show that whenever k is super logarithmic (i.e. $k = \omega(\log n)$), our protocol is concurrent ZK. This requires us to construct a simulator $\text{Sim} = \text{Sim}^{V^*(x,z)}(x)$ that given input instance $x \in L$ and black-box access to $V^*(x, z)$, outputs a view that is indistinguishable from the real view of $V^*(x, z)$. On a very high-level, the simulation follows that of Richardson and Kilian [RK99]. The simulator simulates Stage Init and Stage 1 of the protocol by following the honest prover strategy, and attempts to rewind one of the *slots* (i.e. the last two messages of the special-sound proofs provided by V^*). If the simulator obtains two *matching* convincing slots, i.e., the slots are from the same round of the protocol and share the same initial transcript, the special-soundness property allows the simulator to compute a *fake* witness r such that $f(r) = c_1$ or c_2 . This fake witness can then be used to simulate Stage 2 of the protocol. Towards this goal, we let Sim be an *oblivious* black-box simulator similar to [KP01].

Description of Sim. Let n be the security parameter, m be a bound on the number of concurrent sessions invoked by V^* and T be the total number of messages exchanged, bounded by $O(mk)$, a polynomial in n . Keep in mind that during black-box simulation, we assume without loss of generality that V^* is deterministic; therefore the view of V^* is just the transcript of its interaction with the honest prover.

In order to extract a fake witness from V^* , Sim follows an oblivious rewinding schedule based only on the number of messages exchanged so far, just like in [KP01] and [PRS02]. During the oblivious simulation, Sim keeps a repository of all messages generated by V^* among all rewinds; whenever Sim encounters Stage 2 of the protocol, Sim looks for matching convincing slots in this repository to compute the required fake witness. More precisely, Sim uses the recursive procedure *lazy-rewind* described below.

At a high level, $\text{lazy-rewind}(t, \mathcal{V}, \mathcal{T}) \rightarrow (\mathcal{V}', \mathcal{T}')$ attempts to recursively simulate $V^*(x, z)$ for t messages starting from a partial view \mathcal{V} of V^* , with the help of a repository of messages generated by V^* during rewinds, \mathcal{T} (formally just a set of all simulator query and verifier message pairs). If *lazy-rewind* is successful, it outputs a longer view \mathcal{V}' of V^* (that contains exactly t more verifier messages than \mathcal{V}), and an updated repository \mathcal{T}' including verifier messages that *lazy-rewind* gathered from various rewinds (and most likely contains more verifier messages than what is recorded in \mathcal{V}'). Sim simply outputs the view produced by $\text{lazy-rewind}(T, \mathcal{V} = \emptyset, \mathcal{T} = \emptyset)$, i.e., *lazy-rewind* starting from the empty view and an empty repository.

Description of lazy-rewind(t, s, h). At the base case of the recursion ($t = 1$), *lazy-rewind* receives a message from V^* and produces a prover response. *lazy-rewind* behaves identically to an honest prover to generate Stage Init and Stage 1 messages. Whenever a session reaches END, *lazy-rewind* will attempt to compute a fake witness r for the session ($f(r) = c_1$ or c_2) by searching \mathcal{T} for matching convincing slots. If this is successful, the fake witness r is used to generate prover messages in Stage 2 of this session (i.e. the WI-POK). Otherwise, *lazy-rewind* outputs \perp , which in turn causes Sim to output \perp as well.² In the end, *lazy-rewind* outputs the updated view \mathcal{V}' of V^* (the input view appended with the newly exchanged pair of messages), and the updated repository \mathcal{T}' (the input repository inserted with the newly exchanged pair of messages).

²We distinguish between legitimate failures, i.e., Sim may abort just like a prover should V^* fail to follow the protocol, and simulation failures, i.e., Sim outputs \perp if it fails to compute a fake witness r .

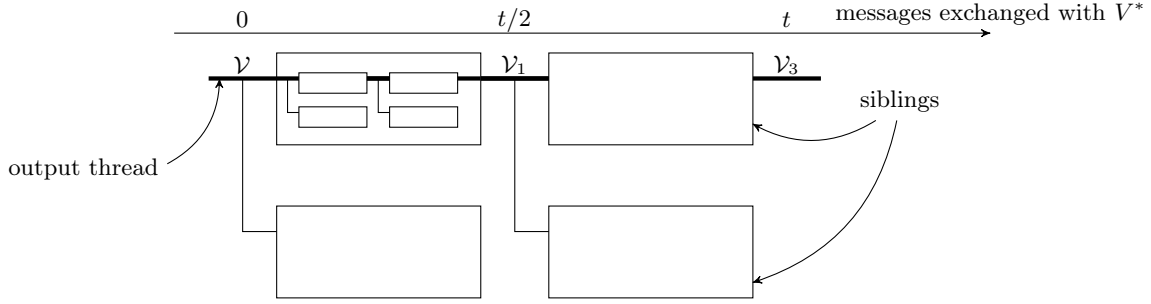


Figure 2: A pictorial representation of the rewinding schedule of `lazy-rewind`. The boxes represent *blocks*, and the lines represent *threads*. If this is the top level call (i.e., `lazy-rewind(T, ∅, ∅)`), then the thicker thread is the *output thread*, whose view is the output of `Sim`.

When $t > 1$, `lazy-rewind`($t, \mathcal{V}, \mathcal{T}$) proceeds roughly as follows: It first recursively simulates V^* for $t/2$ messages twice starting from the partial view \mathcal{V} . Then, continuing from one of those simulations, `lazy-rewind` recursively simulates V^* for another $t/2$ messages, twice. More formally, `lazy-rewind`($t, \mathcal{V}, \mathcal{T}$) calls itself four times as follows:

1. $(\mathcal{V}_1, \mathcal{T}_1) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T})$.
2. $(\mathcal{V}_2, \mathcal{T}_2) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T})$. Merge \mathcal{T}_1 and \mathcal{T}_2 into a larger repository of messages \mathcal{T}' .
3. $(\mathcal{V}_3, \mathcal{T}_3) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}_1, \mathcal{T}')$.
4. $(\mathcal{V}_4, \mathcal{T}_4) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}_1, \mathcal{T}')$. Merge \mathcal{T}_3 and \mathcal{T}_4 into a larger repository of messages \mathcal{T}'' .
5. Output $(\mathcal{V}_3, \mathcal{T}'')$.

Because the first two recursive calls to `lazy-rewind` (respectively the last two calls) have identical inputs (they differ only because they use different segments of `Sim`'s random tape), they are called *sibling* calls. See Figure 2 for an illustration of the rewinding schedule, and Figure 3 for a pseudo-code description.

Let us describe some terminology that is useful for the analysis `Sim` and `lazy-rewind`. Because `Sim` follows an oblivious rewinding schedule, it always makes a fixed set of calls to `lazy-rewind` at fixed moments in the simulation, and it always “connects” these calls of `lazy-rewind` in a fixed way to generate partial views of V^* . Intuitively, a *thread* is one of these fixed connections.

Definition 5 (Threads). A *thread* is a sequence of 0's and 1's; from the beginning of the simulation, this sequence specifies, whenever a pair of sibling calls are encountered, whether to follow the first or second sibling call of `lazy-rewind`, respectively. (A sequence may terminate prematurely to specify a “partial” thread.) The thread $00 \cdots 0$ (of sufficient length) is the thread that follows the first sibling calls to the end of the simulation, and is called the *output thread* because the view of V^* generated on this thread is the output of `Sim`.

Given an execution of `Sim` (on an input $x \in L$ and a random tape), a *block* intuitively refers to the “location” (in the static rewinding schedule) of a call to `lazy-rewind`, as well as the actual simulation performed by the call.

lazy-rewind($t, \mathcal{V}, \mathcal{T}$):

1. **Base Case:** $t = 1$. Exchange one pair of messages with V^* .

- (a) If the next scheduled message is from an aborted session where V^* has deviated from the protocol (e.g., there has been an unconvincing slot), return $(\mathcal{V}||\text{abort}, \mathcal{T})$ (i.e. do nothing).
- (b) If the next scheduled message is a Stage Init or Stage 1 prover message for session i , compute a message p following the honest prover strategy. Let v be the response of V^* ; if v deviates from the protocol (e.g., v is an unconvincing last message of a SS-POK) abort session i .
- (c) If the next scheduled message is a Stage 2 prover message for session i , use the computed fake witness to compute the prover messages p for the WI – AOK, and let v be the verifier response. Note that a fake witness must have already been computed to reach this point in the simulation; see next bullet.
- (d) After exchanging a pair of messages p and v , if we reach the END of a session, attempt to compute a fake witness of the session using the special-soundness property and previous messages stored in the repository \mathcal{T} (in particular are looking for matching convincing slots for session i). If lazy-rewind fails to compute a fake witness, output \perp .
- (e) Output $(\mathcal{V}||p||v, \mathcal{T} \cup \{\mathcal{V}||p||v\})$, i.e., extend the input partial view with the message pair (p, v) and enlarge the input repository with the new message generated by V^* .

2. **Recursive step**

Simulate the first $t/2$ messages twice

(a) $(\mathcal{V}_1, \mathcal{T}_1) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T})$

(b) $(\mathcal{V}_2, \mathcal{T}_2) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T})$

Simulate the second $t/2$ messages twice

(c) $(\mathcal{V}_3, \mathcal{T}_3) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T}_1 \cup \mathcal{T}_2)$

(d) $(\mathcal{V}_4, \mathcal{T}_4) \leftarrow \text{lazy-rewind}(t/2, \mathcal{V}, \mathcal{T}_1 \cup \mathcal{T}_2)$

(e) output $(\mathcal{V}_3, h_3 \cup h_4)$

Figure 3: The recursive procedure used by Sim—the “lazy” KP simulator.

Definition 6 (Blocks). Given an execution of Sim , a *block* B is a pair $B = (B_{\text{loc}}, B_{\text{content}})$, where B_{loc} specifies the location of a call of *lazy-rewind* and B_{content} specifies the inputs and randomness of the same call. Formally B_{loc} is a partial thread (that leads to and includes the *lazy-rewind* call), and B_{content} is just the inputs and random tape used by the *lazy-rewind* call, i.e., $(t, \mathcal{V}, \mathcal{T}, r)$. We say a block C is *contained* in block B if the recursive call of *lazy-rewind* corresponding to block C is nested inside the recursive call of *lazy-rewind* corresponding to block B .

Due to the recursive nature of *lazy-rewind*, every block would contain four “smaller” blocks; of these four blocks, we call the first pair (respectively the second pair) *sibling blocks*, as they correspond to sibling calls of *lazy-rewind*. Finally, we say a block contains a thread if the thread “passes through” the block.

Definition 7 (Threads in a Block). Given an execution of Sim , we say a block B contains a thread h if B_{loc} is a prefix of h .

Since *lazy-rewind* does not update the message repository \mathcal{T} between sibling recursive calls (sibling blocks) we call it *lazy*. This departs from previous works such as [KP01, PRS02], and is crucial for our analysis. We have also changed how blocks are threaded together from [KP01, PRS02]. In *lazy-rewind*, the second pair of recursive calls are continued from the first recursive call of the first pair (i.e. continued from view \mathcal{V}_1). This is similar to the precise simulation of [MP06] and [PPS⁺08]. This choice is inconsequential for our analysis, but will be useful later when we discuss precision in Section 3.6); [KP01] and [PRS02], in contrast, continue the recursive calls from the view \mathcal{V}_2 . See Figure 2 for an illustration of blocks, threads and siblings in an execution of *lazy-rewind*.

3.3 Proof Overview

In order to prove the correctness of the simulation, we need to show that for every adversarial verifier V^* , the simulator runs in polynomial time and the output distribution is “correct”. The running time of Sim can be bound just as in [KP01, PRS02]. Sim spends a maximum of $\text{poly}(n)$ time on responding to each verifier message. It follows from the recursive structure of the simulator that the number of messages exchanged is doubled for each level of the recursion; since we have a recursive depth of $\log_2 T$, the running time of the simulator is bounded by $\text{poly}(n) \cdot T \cdot 2^{\log_2 T} = \text{poly}(n) \cdot T^2 = \text{poly}(n)$.

Intuitively, the correctness of the output view follows from the fact that Sim chooses Stage Init and Stage 1 messages honestly, and that the protocol used in Stage 2 is witness indistinguishable (this requires a proof later since Sim performs rewinds). Therefore, as long as Sim gets stuck (outputs \perp) with negligible probability, taken over the random tapes of Sim (the random tape of V^* is fixed during black box simulation), the output distribution is correct. Towards this goal we will show that the probability of getting stuck at any point in the simulation is negligible.

Recall that Sim can only get stuck on a particular thread when the simulation reaches the END of some session and could not extract the fake witness. Following the approach of [PRS02], we show that the probability of getting stuck on *any session* and *any thread* is negligible. Since there are only polynomially many sessions and threads, the main theorem follows by the union bound.

Fix any thread h and session i ; from now on we refer to it as the “main” thread and the “main” session, and call all other threads and sessions “auxiliary”. We say a random tape of Sim is *bad* if Sim gets stuck at the END of main session i on the main thread thread h ; all other random tapes are called *good* (including those that got stuck on an auxiliary session or thread). The high-level idea, just like in [PRS02], is to show that for every *bad* random tape, there exists super-polynomially

many **good** random tapes. Furthermore, the **good** tapes corresponding to any two **bad** tapes are disjoint. Hence the probability of a tape being **bad** is negligible. From here on, **START** and **END** refer to those on the main session and thread unless otherwise noted.

Here is how we generate **good** random tapes from **bad** ones. Recall that on a **bad** tape, the simulator reaches **END** without extracting a “fake witness”. Hence, all slots on the main thread are convincing (or else we would never reach **END**), but no corresponding convincing slots are on an auxiliary thread prior to **END** (since otherwise **Sim** would have extracted a witness). Intuitively, to generate a **good** tape from a **bad** one we just need to “swap” a convincing slot from the main thread into an auxiliary thread. After the swapping, should the simulation reaches **END** of the main session on the main thread, the newly formed convincing slot on the auxiliary thread, together with the corresponding convincing slot on the main thread, will allow **Sim** to compute the fake witness. Hence the simulation may continue on without getting stuck. So far we have not deviated from the analysis of [PRS02].

To actually “swap” convincing slots, we modify the random tape of **Sim**. The basic operation that we perform on the random tape is to exchange the randomness used by sibling blocks (i.e., the segments of the random tape used to simulate these blocks). Since sibling blocks are identical modulo randomness, swapping the random tape between siblings swaps the simulation result in the two blocks *exactly*. (In the rest of the paper, we use the convention that after swapping a block B with its sibling B' , the “new block B ” refers to the block in the old location of B' with the same content as the “old block B ”, i.e., $(B'_{\text{loc}}, B_{\text{content}})$.) Note that this “exact swap” property is made possible by the lazy nature of **Sim**; the same property does not hold for the KP simulator where the second sibling benefits from fake witnesses extracted during the execution of the first sibling.

Intuitively, we call a block on the main thread **composable** if it satisfies the following properties:

Goodness. Swapping a composable block with its sibling produces a **good** random tape.

Composability. The above swap leaves other composable blocks on the main thread composable.

Reversibility. Given the random tape obtained after swapping a composable block, there is a procedure **undo** that reverses the swap. This ensures that the resulting **good** tape is unique.

Consider K composable blocks with an ordering such that each swap will leave the successive composable blocks still composable. Then, we can generate $2^K - 1$ **good** random tapes by choosing to swap each block or not in the ordering. By a simple counting argument, we will show that for any **bad** tape, there are $k - 2 \log_2 T$ composable blocks with an ordering, therefore generating $2^{k-2 \log_2 T}$ distinct **good** tapes. We then use the **undo** procedure to show that different **bad** tapes generate different **good** tapes. Thus, if $k \in \omega(\log_2 T) = \omega(\log n)$, the probability of having a **bad** tape is negligible.

3.4 The Actual Proof

Formally, **Sim** may output \perp for two reasons. Firstly, it may reach **END** without encountering two matching convincing slots after the **START** of the session; we call this a *rewinding failure*. Secondly, **Sim** may not be able to compute a fake witness even though it has access to matching special-sound transcripts; we call this a *special-sound failure*. Special-sound failures are easy to upper bound; see Claim 8. As mentioned, the main part of the proof is bounding the probability of rewinding failures.

3.4.1 Composable Blocks

We first define the notion of **composable** blocks and show that they satisfy the three properties of *goodness*, *composability* and *reversibility*. Let us fix a particular main session and main thread, and formally define a random tape to be **bad** if **Sim** encounters a *rewinding failure* in the main session on the main thread; otherwise a random tape is **good**. From here on **START** and **END** refers to those of the main session and main thread, unless otherwise noted.

Definition 8 (Composable Block). Consider an execution of **Sim** with any random tape (not necessarily **bad**). A block B , with sibling B' , is called a *composable block* (with respect to the main thread and session), if it satisfies the following conditions:

Main block condition: B contains the main thread h , a convincing slot of the main session (not necessarily on the main thread) and does not contain **START** (of the main session on the main thread). The last condition is equivalent to saying that the prefix of B contains **START**.

Sibling condition: B' does not contain any **END** (of the main session on the main thread).

Tracing condition: The simulation after **START** but before B contains only convincing slots on the main thread h , and contains no convincing slots on the auxiliary threads.

As we will soon see, the **Main block condition** and the **Sibling condition** implies goodness and composability, while the **Tracing condition** enables the **undo** procedure, which implies reversibility. We also define an ordering relation $>$ on **composable** blocks.

Definition 9. Let C and B be two blocks on a common thread. We write $C > B$ iff

- C and B are disjoint, and C occurs before B (Case 1 in Figure 4), or
- C and B are not disjoint, and C is a larger block that contains B (Case 2 in Figure 4)

Note that given two blocks on the same thread, if they are not disjoint, then one must contain another. Thus $>$ is a total order on any set of blocks that share a common thread.

Finally, we define a deterministic **undo** function on random tapes in order to achieve reversibility:

- Given a random tape τ' , execute **lazy-rewind** with the tape τ' . Call a block that does not contain the main thread *special* if it contains a convincing slot of the main session.
- Let D be the first special block after **START**; that is, any other special block E after **START** satisfies $D > E$. Swap the parts of τ' used by D and its sibling, and output the new random tape.

Claim 2. Let τ be a random tape (not necessarily **bad**). Let B be a *composable* block with sibling B' when **lazy-rewind** is executed with random tape τ , and let \mathcal{V} be the common prefix of B and B' . Furthermore, let τ' be the random tape obtained after swapping the blocks B and B' . Then:

1. [Goodness]: τ' is a *good* random tape.
2. [Composability]: Any *composable* block C on τ with $C > B$ is still *composable* on τ' .
3. [Reversibility]: $\text{undo}(\tau') = \tau$.

Proof. Recall that after the swapping, blocks B and B' are exchanged in the simulation.

Goodness When *lazy-rewind* is executed with τ' , B' will now be on the main thread (see Figure 4).

Recall that B' does not contain any *END* of the main session (*sibling condition*). Thus, if the *END* of the main session ever occurs on the main thread, it will occur after both B and B' are executed. In that case, both the convincing slot in B (which is now in an auxiliary thread) and the corresponding convincing slot on the main thread (which must be there before *END* occurs) together forms a matching pair of convincing slots that occurs after *START*. Moreover, this pair of convincing slots occur before *END*. Thus τ' is a good tape.

Composability Given a composable block $C > B$ with sibling C' on τ , we have two cases as shown in Figure 4. In case 1, when C is disjoint from B , the swapping of B and B' does not change the simulation inside C , C' , and between *START* and C . Respectively, this leaves the *main block condition*, *sibling condition*, and *tracing condition* of C intact on τ' . On the other hand, in case 2 where C contains B , the swapping of B and B' again leaves the simulation inside C' and between *START* and C unchanged, keeping the *sibling condition* and *tracing condition* intact. In addition, since C still contains B under τ' , and B in turn contains a convincing slot, the *main block condition* still holds as well (other parts of C may have changed). In both cases, C continues to be a composable block on τ' .

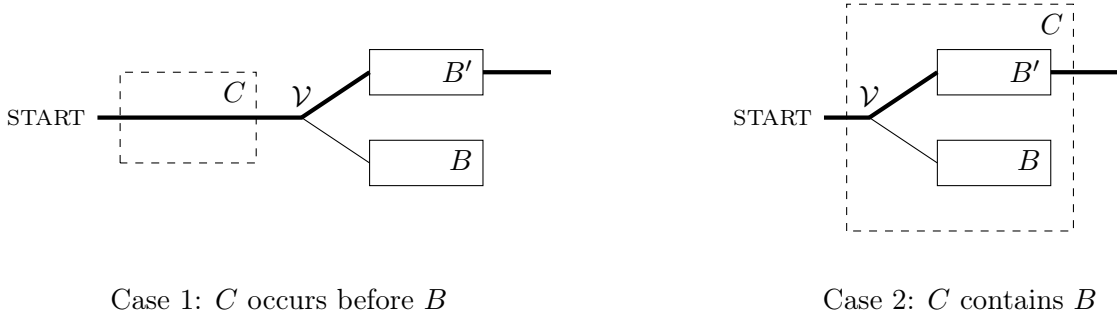


Figure 4: Two possible block diagrams after the swapping procedure in Claim 2 (B and B' is swapped). The main thread is shown in a thick line, and a composable block $C > B$, drawn with dashed lines, is shown in two possible configurations.

Reversibility Finally, we need to show that $\text{undo}(\tau') = \tau$. After the swap (executing with random tape τ'), block B no longer contains the main thread and contains a convincing slot; it is therefore a special block. Next we show that any block $C > B$ is not special. Either C occurs strictly before B , or it contains B (in this case C also contains B'). In the first case, block C is unchanged during the swap, and therefore is not special because it does not contain a convincing slot (*tracing condition*). In the second case, since C contains B' and therefore the main thread, it is not special. Thus, *undo* will always locate B as the first special block and perform the correct inverse swapping to recover τ .³ \square

³ Note that we here rely on the “exact” swapping of sibling blocks (a consequence of the lazy property of *Sim*). Suppose that sibling blocks are not symmetric and that the second sibling uses information obtained in the first sibling to compute fake witnesses. Then, if the *END* of an auxiliary session occurs before the convincing slot in B' , it

The next claim demonstrates how to compose multiple composable blocks.

Claim 3. *Let τ be a bad random tape, $\mathcal{B} = \{B_1, \dots, B_p\}$ be a set of composable blocks for τ . Then, we can generate a set of good random tapes, $S(\tau, \mathcal{B})$, by swapping the various composable blocks in \mathcal{B} , so that the following holds:*

1. $|S(\tau, \mathcal{B})| \geq 2^p - 1$.
2. For any bad tape $\tau' \neq \tau$ and any set of composable blocks \mathcal{B}' for τ' , $S(\tau, \mathcal{B}) \cap S(\tau', \mathcal{B}') = \emptyset$.

Proof. Since all composable blocks lie on the main thread, there is a total ordering of the blocks. Without loss of generality, let $B_1 > B_2 \dots > B_p$. Consider any non-empty subsequence of $1, \dots, p$, say u_1, \dots, u_q . There are $2^p - 1$ such sequences. Let $\tau_{u_1 \dots u_q}$ be the random tapes obtained from τ by swapping the blocks B_{u_i} with its sibling, in the order of $i = q, q-1, \dots, 1$.

From Claim 2, it follows that $\tau_{u_1 \dots u_q}$ is a good random tape. We further note that given $\tau_{u_1 \dots u_q}$, we can recover the blocks B_{u_1}, \dots, B_{u_q} by repeatedly applying `undo` until we reach a bad tape (it will always be τ). Therefore given two different subsequences, u_1, \dots, u_q and $v_1, \dots, v_{q'}$, we must have $\tau_{u_1 \dots u_q} \neq \tau_{v_1 \dots v_{q'}}$ in order for `undo` to recover a different set of swapped blocks. Thus, we obtain $2^p - 1$ distinct good random tapes.

Similarly, take any $\alpha \in S(\tau, \mathcal{B})$ and $\beta \in S(\tau', \mathcal{B}')$ (good tapes produced by swapping from τ and τ' , respectively). Applying `undo` repeatedly on α until the result is a bad tape will result in τ , while applying the same procedure on β will give τ' . If $\tau \neq \tau'$, then we must have $\alpha \neq \beta$. \square

Corollary 4. *Suppose every bad random tape has p composable blocks. Then, the probability of a random tape being bad is at most $1/2^p$*

3.4.2 Number of Composable Blocks

We now proceed to count the number of composable blocks. First we introduce the notion of *minimal containing blocks* (this is identical to minimal rewinding intervals as defined by [PRS02]). For each slot, its *minimal containing block* is the minimal block on the main thread that contains the slot. Claims 5 and 6 below together show that there are at least $k - 2 \log T$ composable blocks when we run `Sim` with a bad tape. Claim 5, which counts the number of minimal containing blocks, is identical to [PRS02]; we include it here for completeness.

Claim 5. *In an execution of `Sim` with a bad random tape, there are at least k minimal containing blocks.*

Proof. As observed earlier, on a bad tape there will be k convincing slots of the main session on the main thread (in order to reach `END`). We merely need to show that for each slot, its respective minimal containing blocks are distinct. Suppose that two slots share the same minimal containing block of length t . Since slots on the same thread are disjoint, we reach a contradiction as one of the slots must be properly contained in one of the two smaller blocks of size $t/2$. \square

Claim 6. *Consider an execution of `Sim` with a bad random tape τ . If there are k' minimal containing block, then there are at least $k' - 2 \log T$ composable blocks.*

may now output \perp after the swapping (since it has lost the information collected in B after the swap). In this case, block B would not exist when executing `Sim` with random tape τ' , and `undo` would fail.

Proof. Let B be a minimal containing block that does not contain START or END. Since START (or END) can only be in at most $\log T$ different blocks on the main thread (since that is the recursion depth), we conclude that there are at least $k' - 2 \log T$ such blocks.⁴ It remains to show that B is a composable block. Let B' be the sibling of B .

The main block condition of composable blocks follows directly, while the tracing condition on the main thread actually holds for the whole simulation from START to END, since τ is a bad random tape. Thus, we only need to show that the sibling condition is satisfied, i.e. B' does not contain END. Assume to the contrary that B' does contain END. Since B and B' are siblings with a common starting point and B contains a slot of the main session, B' must contain that same slot in a convincing manner in order to reach END. On the other hand, B does not contain END. Thus B' will be executed before the main thread reaches END (if at all), and this convincing slot will allow Sim to compute the witness of the main session by the same argument in Claim 2. This contradicts the fact that τ is a bad tape. \square

3.4.3 Concluding the Proof

We first show that Sim gets stuck with negligible probability, and then use it in Claim 9 to conclude that the output distribution of Sim^{V^*} is computationally (resp. statistically) indistinguishable from the real view of V^* .

Claim 7. *Sim encounters rewinding failures with negligible probability.*

Proof. As mentioned before, since there are only polynomially many sessions and threads, it suffices to show that the probability of the simulator getting stuck on any fixed thread and session is negligible. The union bound then shows that Sim overall gets stuck with negligible probability

For any fixed thread and session, combining Claim 3, 5 and 6 shows that a random tape is bad with probability at most

$$\frac{1}{2^{k-2\log T}}$$

This is negligible in n since T is polynomial in n and $k = \omega(\log n)$. \square

Claim 8. *Sim encounters special-sound failures with negligible probability.*

Proof. Suppose for the sake of contradiction that Sim encounters special-sound failures with non-negligible probability. Consider an unbounded adversarial prover P^* that forwards the prover messages of the prefix of the SS-POK (Stage Init), in a random session and random thread from an execution of Sim^{V^*} , to an outside honest verifier V_{SS} of the SS-POK (essentially we are forwarding messages between V^* , who acts as the prover of the SS-POK in Stage Init, to the outside honest verifier V_{SS}). Since an execution of Sim^{V^*} only has polynomially many instances of Stage Init, P^* would contradict the special-soundness property with non-negligible probability (i.e., produce a prefix of the SS-POK where it is possible for the witness-computing procedure to fail, even when supplied with two different completions of the prefix).

The actual “forwarding” procedure of P^* has a subtlety due to the rewinding nature of Sim. In the middle of forwarding the prefix of a SS-POK from V^* to V_{SS} , Sim may decide to rewind V^* partially to an earlier point in the SS-POK proof. In a naive forwarding scheme, this would require

⁴This is the same counting argument used in [PRS02] to count minimal rewinding intervals without START or END.

V_{SS} to be rewound as well to generate fresh verifier messages (which cannot be done). Fortunately, since the prefix of our SS protocol has only 2 messages (non-interactive), such a rewinding cannot occur. \square

Claim 9. *If the argument of knowledge in Stage 2 is WI (resp. statistical WI), then the ensembles $\{\text{View}_{V^*}^P(x, z)\}_{x \in L, z \in \{0,1\}^*}$ and $\{\text{Sim}(x, z)\}_{x \in L, z \in \{0,1\}^*}$ are computationally (resp. statistically) indistinguishable (as a function of $|x|$).*

Proof. We consider polynomially many intermediate hybrids Sim_i , $0 \leq i \leq m + 1$, that receive the real witnesses to the statements x_1, \dots, x_m . Sim_i proceeds as Sim until the i^{th} Stage 2 proof on the output thread, after which Sim_i continues in a straight-line simulation with V^* using the real witnesses for Stage 2 proofs. (We note that a similar type of “cut-off” simulator was used in [BPS06].) Sim_i will output \perp , however, should Sim encounter a rewinding or special-sound failure during the i^{th} proof. Clearly, Sim_0 generates $\text{View}_{V^*}^P(x, z)$ and Sim_{m+1} generates $\text{Sim}(x, z)$. Thus, it is enough to show that for all i , the output of Sim_i and Sim_{i+1} , are computationally (resp. statistically) indistinguishable.

We introduce yet another hybrid Sim'_i that proceeds as Sim_i except that it utilizes the extracted fake witness for the i^{th} proof. Sim'_i and Sim_{i+1} differ only in that Sim_{i+1} has more chances to output \perp . But by Claim 7 and 8, both actually outputs \perp with negligible probability, and therefore they are statistically close. On the other hand, Sim_i and Sim'_i differ only in the i^{th} proof, which both simulators run in a straight-line fashion, without rewinds. Therefore they are computationally (resp. statistically) indistinguishable by the WI property of the Stage 2 proof. Thus, the output of Sim_i and Sim_{i+1} are indeed computationally (resp. statistically) indistinguishable. \square

Claim 9 completes the proof of Theorem 1.

Remark. Since we have shown that our lazy simulator is a concurrent zero-knowledge simulator, it follows directly that the KP simulator is also a concurrent zero-knowledge simulator: because the KP simulator receives more information than the lazy simulator at any point during the simulation (i.e. a bigger history repository h), the probability that the KP simulator outputs \perp is no more than the probability that the lazy simulator outputs \perp . Thus, the same argument presented in Claim 9 can be applied also to the KP simulator.

3.5 Improving the Running Time of the Simulator

A faster simulator gives tighter “knowledge security” [GMW91, Gol01]. In this section, we bound the running time Sim by bounding the number of queries the Sim makes to V^* . Recall that T is the maximum number of queries needed in a straight-line execution of V^* . This means the recursive depth of $\text{lazy-rewind}(T, \cdot, \cdot)$ invoked by Sim is at most $\log_2 T$. Since lazy-rewind doubles the number of queries per recursive depth, Sim makes at most $T2^{\log_2 T} = T^2$ queries.

We can reduce the number of queries by following the approach of [PPS⁺08]. Currently, when lazy-rewind is asked to simulate t messages (a block of size t), it divides the t messages into two halves and recursively calls itself on each half (creating blocks of size $t/2$). One approach would be for lazy-rewind to divide each block into smaller sub-blocks, thus reducing the recursive depth. For example, an illustration of dividing a block into 3 equal parts is shown in Figure 5 (this means $\text{lazy-rewind}(t, \cdot, \cdot)$ would make 6 recursive calls of the form $\text{lazy-rewind}(t/3, \cdot, \cdot)$). Suppose we divide

each block into g equal sized sub-blocks; we call g the *splitting factor*. Then it immediately follows that the recursive depth of $\text{lazy-rewind}(T, \cdot, \cdot)$ becomes $\log_g T$, and the number of queries made by Sim is reduced to at most $T2^{\log_g T}$.

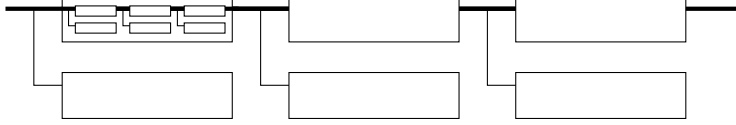


Figure 5: A pictorial representation of a rewinding schedule with splitting factor 3.

Now that the Sim is making less queries overall, can it still successfully generate a view of V^* ? It is easy to see that the combinatorial properties of **composable** blocks do not change with this generalized rewinding scheme. Therefore, we only need to count the number of **composable** blocks in this new rewinding schedule to bound Sim 's failure probability. As in Section 3.4.2, we start by counting the number of minimal containing blocks. The following two claims mirror Claim 5 and 6:

Claim 10. *In an execution of Sim with splitting factor g on a bad random tape, there are at least $k/(g-1)$ minimal containing blocks.*

Proof. Recall that in an execution with a **bad** tape, there are k convincing slots of the main session on the main thread (in order to reach **END**). Since each slot has its respective minimal containing block, and any block can be the minimal containing block for at most $g-1$ slots (by the pigeon hole principle), there are at least $k/(g-1)$ minimal containing blocks. \square

Claim 11. *Consider an execution of Sim with splitting factor g on bad random tape τ . If there are k' minimal containing block, then there are at least $k' - 2\log_g T$ composable blocks.*

Proof. As shown in Claim 6, it still holds that any minimal containing block that does not contain **START** or **END** is a **composable** block. Since **START** (or **END**) can only be in at most $\log_g T$ different blocks on the main thread (since that is the recursion depth), we conclude that there are at least $k' - 2\log_g T$ composable blocks. \square

Following the claims in Section 3.4.3, we conclude that Sim with splitting factor g is still a valid zero-knowledge simulator as long as

$$\frac{1}{2^{k/(g-1) - \log_g T}}$$

is negligible in the security parameter n ; this holds whenever $k/g \in \omega(\log n)$. In particular, for any $\varepsilon > 0$, if we set $g = 2^{1/\varepsilon}$ and $k = \omega(\log n)$, then protocol **ConcZKArg** remains secure and $\omega(\log n)$ -rounds, and Sim makes at most $T^{1+\varepsilon}$ queries to V^* where T is the maximum number of queries needed in a straight-line interaction with V^* .

3.6 Achieving a Precise Simulation

Precise zero-knowledge, introduced by Micali and Pass [MP06], considers an “execution-by-execution” notion of knowledge-tightness; it requires that the view of any adversarial verifier V^* be simulated in time closely related to the *actual* running time (as opposed to the worst-case running time) spent

by V^* in the view generated. More formally, a zero-knowledge simulator has precision $p(t)$ if for all views \mathcal{V} of V^* generated by the simulator, if V^* takes time t in the view \mathcal{V} , then the simulator must have generated the view in time $p(t)$. See [MP06] for more detailed discussions and definitions.

Pandey et. al. [PPS⁺08] recently extended precise ZK to the concurrent setting. The crux of [PPS⁺08] is a slightly modified KP simulator:

1. The KP simulator is modified to obviously rewind based on time, i.e., the number of Turing machine *steps* taken by V^* , instead of the number of *messages* sent by V^* .
2. The KP simulator is modified to output the view of V^* on the “first” thread that it simulates (i.e., outputs the view in the first sibling block instead of the second).

In fact, the second modification could not be directly analyzed with the techniques of [PPS⁺08] (based on the PRS analysis). Instead, [PPS⁺08] ask that sibling blocks be simulated in parallel (instead of one after another); this requires subtle modifications to the PRS analysis, and the addition of a doubling trick to guess the running time of V^* so that the simulator knows how many recursive levels to simulated in parallel.

Looking at the lazy KP simulator, it already outputs the view of V^* on the “first” simulation thread. Therefore, to make the lazy KP simulator precise, we *only* need to modify it to rewind based on time. In other words, simply let `lazy-rewind(t, \cdot, \cdot)` simulate V^* for t Turing machine steps instead of t messages. The observations in Section 3.5 then allow us to obtain (and expand to arguments of knowledge) the main theorems of [PPS⁺08], namely:⁵

Theorem 12. *For any integer functions k and g satisfying $k(n)/g(n) \in \omega(\log n)$, there exists a $O(k(n))$ -round concurrent zero-knowledge argument of knowledge for all of NP, based on one-way functions, with precision $p(t) \in O(t2^{\log_g t})$. In particular, for any $\varepsilon > 0$, there exists a $\omega(\log n)$ -round instantiation of the protocol with precision $p(t) \in O(t^{1+\varepsilon})$.*

4 Acknowledgements

We are very grateful to Manoj Prabhakaran, Alon Rosen and Amit Sahai for helpful discussions and comments. We are also deeply grateful to Oded Goldreich for his great comments about the presentation of our results.

References

- [BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO '92*, pages 390–420, 1992.
- [Blu86] M. Blum. How to prove a theorem so no one else can claim it. *Proc. of the International Congress of Mathematicians*, pages 1444–1451, 1986.
- [BPS06] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *FOCS'06*, pages 345–354, 2006.

⁵As in [MP06, PPS⁺08] we also need to appropriately pad the verifier messages to ensure that the simulator has enough time to generate its messages; see [MP06, PPS⁺08] for more details.

- [CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO '94*, pages 174–187, 1994.
- [CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *STOC '00*, pages 235–244, 2000.
- [CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires $\tilde{\omega}(\log n)$ rounds. In *STOC '01*, pages 570–579, 2001.
- [Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *EUROCRYPT '00*, pages 418–430, 2000.
- [DDN00] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.
- [DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.
- [DS98] Cynthia Dwork and Amit Sahai. Concurrent zero-knowledge: Reducing the need for timing constraints. In *CRYPTO '98*, pages 177–190, 1998.
- [FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC '90*, pages 416–426, 1990.
- [GK96] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.
- [GMOS07] Vipul Goyal, Ryan Moriarty, Rafail Ostrovsky, and Amit Sahai. Concurrent statistical zero-knowledge arguments for NP from one way functions. In *ASIACRYPT '07*, pages 444–459, 2007.
- [GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.
- [GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.
- [Gol01] Oded Goldreich. *Foundations of Cryptography — Basic Tools*. Cambridge University Press, 2001.
- [HILL99] Johan Håstad, Russell Impagliazzo, Leonid Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1999.
- [HNO⁺09] Iftach Haitner, Minh-Huyen Nguyen, Shien Jin Ong, Omer Reingold, and Salil P. Vadhan. Statistically hiding commitments and statistical zero-knowledge arguments from any one-way function. *SIAM Journal of Computing*, 39(3):1153–1218, 2009.
- [KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in poly-logalgorithm rounds. In *STOC '01*, pages 560–569, 2001.

- [KPR98] Joe Kilian, Erez Petrank, and Charles Rackoff. Lower bounds for zero knowledge on the internet. In *FOCS '98*, pages 484–492, 1998.
- [LPTV10] Huijia Lin, Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkatasubramanian. Concurrent non-malleable zero knowledge proofs. In *CRYPTO*, pages 429–446, 2010.
- [MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In *STOC '06*, pages 306–315, 2006.
- [Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.
- [PPS⁺08] Omkant Pandey, Rafael Pass, Amit Sahai, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkatasubramanian. Precise concurrent zero knowledge. In *EUROCRYPT '08*, pages 397–414, 2008.
- [PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS '02*, pages 366–375, 2002.
- [PS02] Manoj Prabhakaran and Amit Sahai. Concurrent zero knowledge proofs with logarithmic round complexity. Cryptology ePrint Archive, Report 2002/055, 2002. <http://eprint.iacr.org/2002/055>.
- [PV08] Rafael Pass and Muthuramakrishnan Venkatasubramanian. On constant-round concurrent zero-knowledge. In *TCC '08*, pages 553–570, 2008.
- [RK99] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *Eurocrypt '99*, pages 415–432, 1999.
- [Ros00] Alon Rosen. A note on the round-complexity of concurrent zero-knowledge. In *CRYPTO '00*, pages 451–468, 2000.
- [Ros03] Alon Rosen. *The round-complexity of black-box concurrent zero-knowledge*. PhD thesis, Weizmann Institute of Science, 2003.

A The PRS Analysis [PRS02]

On counting arguments. The PRS approach of mapping *bad* random tapes to *good* random tapes is different from the approach taken in this paper. In this section, we provide a brief overview of the PRS analysis.

Given a *bad* random tape, the PRS analysis deals with *minimal rewinding intervals*, defined to be minimal blocks that contain a slot, without containing `START` or `END`.⁶ Since minimal rewinding intervals are not “composable” when they overlap, the PRS analysis focuses on a (maximal) set of *disjoint* minimal rewinding intervals. To make up for lost intervals due to overlapping, the PRS analysis swaps each minimal rewinding interval not only with its sibling (as we do), but also with its “cousins”. See Figure 6 for an illustration of cousins blocks. Note that a block may have many cousins (but only one sibling). Moreover, swapping a block with its cousins may require an exchange of random tape segments outside the two blocks, and therefore produce changes in the simulation outside of the cousins

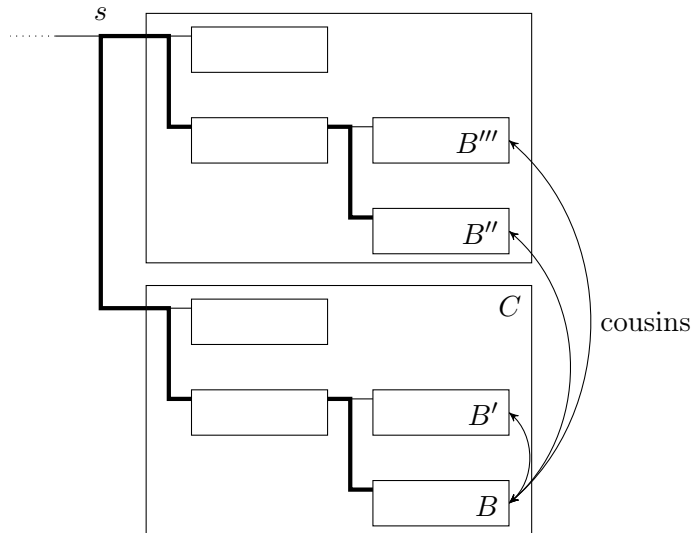


Figure 6: A pictorial representation of the original KP rewind schedule, extended from Figure 5 of [PRS02]. We show how a rewinding interval B is related to its sibling (B'), its parent (C), and its cousins (B', B'', B'''). To swap block B with its cousin B'' , one needs to exchange the randomness used on the two highlighted thread.

Next, the analysis needs to determine for each rewinding interval, how many cousins swaps will result in a new distinct random tape; this step is complicated because a large portion of the random tape maybe shuffled to perform a cousin swap, destroying other potential rewinding intervals. Each rewinding interval is thus assigned a weight corresponding to the number of available cousins. Finally, an analysis is used to lower-bound the sum of weights over the chosen (maximal) subset of disjoint rewinding intervals. (Recall that, in contrast, our analysis is local—we are only required to show that a *single* swap of a block with its sibling results in *one* new random tape.)

⁶Here we adopt some of our terminologies to explain the PRS analysis.

On the “hybrid simulator”. To enable the above counting argument, the PRS analysis bounds the failure probability of a “hybrid rewinding simulator”, which uses the witness of the input statement to produce a simulated view of V^* . More specifically, the hybrid simulator proceeds like the actual simulator to extract “fake witnesses”—and fails whenever extraction does not work—but uses the real witness to complete the Stage 2 proof. Next, the PRS analysis shows that the hybrid simulator is indistinguishable from the real simulator. This relies on the Stage 2 proof being committed-verifier, so that the adversarial verifier cannot extract the witness used in the Stage 2 proof (and thus distinguish the two simulators), even though there are many rewinds. In contrast, we do not use a hybrid argument regarding two rewinding simulators (see Claim 9), and so our analysis directly applies also to committed-verifier protocols.