A Tutorial on Concurrent Zero Knowledge[*]

Rafael Pass[†]

February 12, 2019

Abstract

In this tutorial, we provide a brief overview of Concurrent Zero Knowledge and next present a simple proof of the existence of Concurrent Zero-knowledge arguments for $\mathcal{NP}$ based on one-way permutations.

[*]This tutorial appeared as a chapter in *Oded Goldreich, Providing Sound Foundations for Cryptography: On the work of Shafi Goldwasser and Silvio Micali, ACM 2019.*

[†]Cornell Tech, E-Mail: `rafael@cs.cornell.edu`. Supported in part by NSF Award CNS-1217821, AFOSR Award FA9550-15-1-0262, a Microsoft Faculty Fellowship, and a Google Faculty Research Award.

1 Introduction

Following the seminal works of Dolev, Dwork and Naor [DDN00] and Feige and Shamir [FS90] from the early 90's, *concurrent security* of cryptographic protocols has been an active area of research. In this tutorial, we focus on concurrent security of *zero-knowledge proof systems.* Zero-knowledge ($\mathcal{ZK}$) proofs, introduced by Goldwasser, Micali and Rackoff [GMR89] are paradoxical constructs that allow one player P (called the Prover) to convince another player V (called the Verifier) of the validity of a mathematical statement $x \in L$, while providing *zero additional knowledge* to the Verifier. This is formalized by requiring the existence of an efficient (i.e., polynomial-time) *simulator* Sim that can indistinguishably emulate the view of any malicious Verifier V^* in its interaction with the Prover P; thus, anything the Verifier V^* learns in a "real" interaction with the Prover, could have been generated by the Verifier "on-its-own", and as a consequence, the Verifier did not learn anything new.

Soon after their conception, zero-knowledge proofs for all of $\mathcal{NP}$ were demonstrated by Goldreich, Micali and Wigderson [GMW91]; subsequently, Brassard , Crépeau and Yung [BCY91], Feige and Shamir [FS90] and Goldreich and Kahan [GK96a] demonstrated the existence of *constant-round zero-knowledge protocols* with negligible soundness error for all of $\mathcal{NP}$.

Beyond being fascinating in their own right, $\mathcal{ZK}$ proofs and arguments (i.e., proofs that only are computationally sound) have numerous cryptographic applications and are one of the most fundamental cryptographic building blocks. As such (and as we shall also discuss below), techniques developed in the context of $\mathcal{ZK}$ often extend to more general types of interactions (most notably, general secure computations [Yao86, GMW87, BGW88].)

Concurrent $\mathcal{ZK}$. The notion of *Concurrent $\mathcal{ZK}$*, first introduced and achieved 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 Verifier that participates in multiple concurrent executions—called *sessions*—of a $\mathcal{ZK}$ proof. The same $\mathcal{ZK}$ protocol is used in all the sessions, but the adversarial Verifier is communicating with multiple *independent* instances of the Prover. At first sight it may seem like every $\mathcal{ZK}$ protocol also remains $\mathcal{ZK}$ in such a setting, but as shown by Feige and Shamir [FS90] and Goldreich and Krawczyk [GK96b], this turns out to be false: there are $\mathcal{ZK}$ arguments that reveal the whole witness being used in the proof if a Verifier performs a coordinated attack on just two simultaneous protocols!

Roughly speaking (following [FS90]), one can come up with a $\mathcal{ZK}$ protocol where the Verifier can select between two modes of operation: In Mode 1, the Verifier requests to hear a standard $\mathcal{ZK}$ proof of the statement $x \in L$, whereas in Mode 2, the Verifier may instead attempt to prove x to the Prover using the same type of $\mathcal{ZK}$ proof, and if the proof succeeds, the Prover simply reveals the witness w to x (and otherwise aborts). It is not hard to see that such a protocol is $\mathcal{ZK}$ in isolation, assuming L has *unique* witnesses: In Mode 1, this follows by definition, and in Mode 2, this follows from the fact that the Prover only gives the witness w to the Verifier if the Verifier already knows it! (Actually, the $\mathcal{ZK}$ protocol employed needs to be a so-called *proof of knowledge* [FS90, BG92] to ensure that the Verifier must convince the Prover that it actually knows w before the Prover hands it out.) On the other hand, a malicious Verifier participating in two concurrent executions can use Mode 1 in the first session, and Mode 2 in the second session, and then simply forward the Mode 1 proof provided by the Prover in session 1 as its Mode 2 proof in the session 2, and thereby get the Prover to reveal the witness in the session 2. So, by participating in two sessions of a $\mathcal{ZK}$ proof, the malicious Verifier learns a witness for x (even thought it may not have known one before the interaction). In fact, by adding dummy message, one can obtain a protocol that no longer is zero-knowledge even when two instances of the protocol are repeated in *parallel* (i.e., the two instances proceed in a lockstep fashion).

What makes Concurrent $\mathcal{ZK}$ hard? Of course, the above construction is clearly artificial—it was designed to break down under concurrent executions. One could have hoped that more "natural" constructions of $\mathcal{ZK}$ protocols retain their $\mathcal{ZK}$ property under concurrent sessions. Indeed, the constant-round protocols of [FS90, GK96a] are known to preserve their zero-knowledge property under *parallel* composition (i.e., when we have an unbounded number of parallel sessions) [FS90, Gol02].

However, it is still unknown whether these protocol remain zero-knowledge under *concurrent* executions (where the Verifier may decide the scheduling of the messages in the different sessions). Even though concrete attacks are not known against these protocols, we also do not know how to prove them secure. The problem is that the standard simulation method fails in the concurrent setting. For concreteness, consider the constant-round $\mathcal{ZK}$ protocol of Feige and Shamir [FS90] (we are using this protocol as our example as the ideas underlying it will be useful to us in the sequel). Roughly speaking, the protocol for proving a statement $x \in L$ proceeds in two stages:

- In Stage 1, the Verifier samples a *different* statement $\tilde{x}$ and witness $\tilde{w}$ from some hard-on-the average language and next proves to the Prover that it knows a witness to the statement $\tilde{x}$ using a, so-called, *Witness Hiding* [FS90] proof system which does not reveal the witness $\tilde{w}$.
- In Stage 2, the Prover next provides a proof that it either knows a witness for the true statement x, or that it knows a "fake" witness to the other statement $\tilde{x}$; this second stage proof needs to be *Witness Indistinguishable* [FS90] so that it does not reveal whether the Prover is using a witness for x or $\tilde{x}$.

Each of these subprotocols (for Stage 1 and 2) can be implemented in just 3 communication rounds using Blum's Hamiltonicity protocol [Blu86].[1] We will refer to the Stage 1 messages as $(\alpha_1, \alpha_2, \alpha_3)$ and depict them using (single) arrows, and to simply our illustrations, refer to the whole Stage 2 protocol as α_4 and depict it as a double arrow; see Figure 1 for an illustration.

The idea for why this protocol is $\mathcal{ZK}$ is that (a) clearly, the Verifier cannot learn anything from the Stage 1 protocol, as here it is actually the Verifier who provides a proof to the Prover, and (b) since the Verifier first proves to the Prover that is knows a "fake" witness $\tilde{w}$, it can indistinguishably simulate Stage 2 on its own using this fake witness (due to the fact that the Stage 2 protocol is witness indistinguishable). A bit more precisely, to provide the actual simulation, the simulator will need to "extract" out the fake witness from the Stage 1 proof, and can later use this fake witness to complete the simulation of Stage 2.

In more detail, to simulate the view of a malicious Verifier V^*, the simulator honestly emulates the first 3 rounds $(\alpha_1, \alpha_2, \alpha_3)$ of the protocol, and then "rewinds" the Verifier, resending different second messages α'_2 until it gets a second accepting third message α'_3 from the Verifier in order to extract out the fake witness which can be used to complete the simulation. (Technically, the proof-of-knowledge property of the Stage 1 protocol we here rely on is called "special-soundness"

[1] As we shall see shortly, the reason why we are relying on Blum's protocol as opposed to, say, [GMW87], is that Blum's protocol satisfies a strong proof-of-knowledge property which will simplify the analysis.

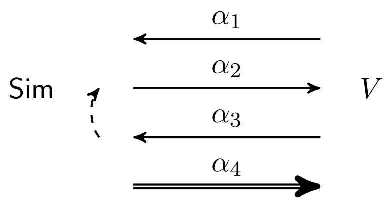

Figure 1: A standard $\mathcal{ZK}$ simulation.

[CDS94]; it stipulates that a valid witness $\tilde{w}$ for $\tilde{x}$ can be computed in polynomial time from any two accepting proof transcripts $(\alpha_1, \alpha_2, \alpha_3)$, $(\alpha_1, \alpha_2', \alpha_3')$ for the statement $\tilde{x}$ with the same first message α_1 but different second messages $\alpha_2 \neq \alpha_2'$. Blum's Hamiltonicity protocol [Blu86] satisfies this this property.) We refer to the second and third message pair (α_2, α_3) as a slot, and rewinding this slot is the key tool that enables simulation; see Figure 1.

This method no longer works in the concurrent setting. More precisely, a concurrent Verifier V^* may *nest* the concurrent sessions—putting session 1 *inside* the slot for session 2, and session 2 inside the slot for session 3, etc, and may generate its randomness for the different sessions as some function of the prefix of the execution up to this point. Then:

- Simulating the "innermost" session (i.e. session 1) will require running the Verifier twice (just as in the stand-alone simulation);
- simulating session 2 (which includes session 1 inside it) requires running the simulation of session 1 twice, since every time we rewind session 2, session 1 restarts with new randomness;
- simulation session 3 (which includes session 2 and 2 inside it) requires running the simulation of session 2 twice (since every time we rewind session 3, session 2 restarts with new randomness), and which in turn requires running the simulation of session 1 *four* times;
- and so on and so forth.

Thus, if we have n sessions, the running-time becomes *exponential* in n (and the simulator can no longer be a polynomial-time algorithm). See Figure 2 for an illustration for a simulation with just two sessions.

To overcome this exponential blow-up, we instead need to come up with new protocols and analyses. These protocols are significantly harder to construct and analyze than "stand-alone" $\mathcal{ZK}$ protocols [GMR89, GMW91, FS90, GK96a].

Benign Schedulings and Set-up Assumptions. To overcome the above obstacle, the original protocol by Dwork, Naor and Sahai [DNS04] relied on so-called "timing assumptions": informally speaking, the timing model assumes that every party has a local clock, that all these local clocks are roughly synchronized, and that all parties know a (pessimistic) upper-bound Δ on the time it takes to deliver a message on the network. In such a timing model, the Prover can use delays and time-outs to prevent "bad schedulings". Improved Concurrent $\mathcal{ZK}$ protocols in the timing model were presented in [Gol02, PTV10]; the idea behind these works is to identify more expressive classes of schedulings that can be handled and next to use timing contraints to restrict the attacker to those scheduling. For instance, the work of Goldreich [Gol02] demonstrates that the original constant-round stand-alone $\mathcal{ZK}$ protocol of Goldreich-Kahan [GK96a] remains $\mathcal{ZK}$ under the more "benign"

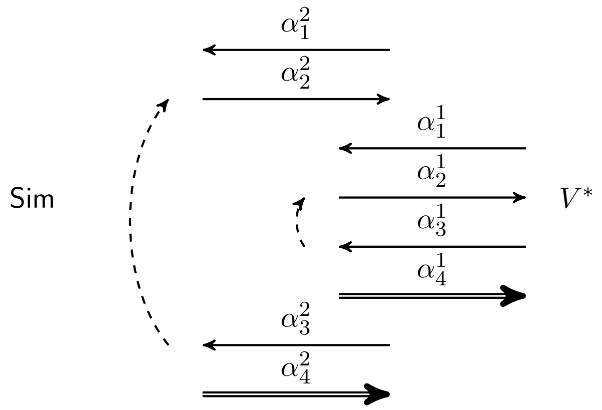

Figure 2: A simulation for 2 nested concurrent sessions.

schedulings of *parallel composition* and *bounded-simultaneity* (where we have only a *constant* number of sessions running at the same time), and next uses timing constraints to ensure that a combination of the simulation techniques used for those special cases of schedulings suffice to get a concurrent $\mathcal{ZK}$ protocol in the timing model. Whereas the protocols of [DNS04, Gol02] required imposing delays/slowdowns that were longer than the upper bound on the message delivery time, Δ, [PTV10] showed that the slowdown can be significantly smaller than Δ; moreover, the slowdown can be done adaptively so that only sessions that are slow anyways get "penalized" with delays.

Various other concurrent $\mathcal{ZK}$ protocols were also obtained based on different set-up assumptions (e.g., [DS98, Dam00, CGGM00b]). In this tutorial, however, our focus will be on the "standard model" without any set-up assumption.

Black-box Impossibilities. In the standard model (without any timing assumptions), Canetti, Kilian, Petrank and Rosen [CKPR01] (building on earlier works by [KPR98, Ros00, GK96b]) showed that concurrent $\mathcal{ZK}$ protocols for non-trivial languages, with so called "black-box" simulators (i.e., simulators that simply use the Verifier as a black-box but may rewind it), require at least $\tilde{\Omega}(\log n)$ number of communication rounds, where n is the length of the instance being proved; see also [CPT12] for a simplified (and generalized) analysis of the impossibility result from [CKPR01]. Thus, if we restrict to black-box $\mathcal{ZK}$, the original constant-round $\mathcal{ZK}$ protocols (e.g., [GK96a, FS90]) cannot be concurrently secure (but it is still open whether non-black-box techniques can be used to prove security of them).

Feasibility of Concurrent $\mathcal{ZK}$. Richardson and Kilian [RK99] constructed the first concurrent $\mathcal{ZK}$ argument in the standard model without any 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 [CGGM00b] for a somewhat different and more detailed analysis of this protocol. Subsequent works by Kilian and Petrank [KP01], Prabhakaran, Rosen and Sahai [PRS02] improved the round-complexity to $\tilde{O}(\log n)$; see also [PTV14] for a simplified and generalized analysis of such more round-efficient concurrent $\mathcal{ZK}$ proofs.

The key idea for overcoming the above-mentioned blow-up in the running-time of the simulator is to construct a protocol with *many sequential* slots, such that only one of the slots needs to be rewound to ensure that the rest of the session can be simulated. Then, intuitively, the above-mentioned nesting attack can no longer be performed—an attacker would need to nest sessions within *all* of the slots, but since it can only start polynomially many sessions, the nesting depth can never become too big and thus, intuitively, the running-time of the simulation remains polynomial. Formalizing this, however, turned out to be quite complex and subtle.

Towards Constant-round Concurrent $\mathcal{ZK}$. The question of whether *constant-round* concurrent $\mathcal{ZK}$ protocols exist still remains an intriguing problem: A breakthrough result in this direction was obtained by Barak in 2001 [Bar01]; Barak presented a constant-round $\mathcal{ZK}$ protocol for $\mathcal{NP}$ (based on standard cryptographic hardness assumptions) which remains secure under an *a priori bounded* number of concurrent instances—a.k.a. *bounded concurrency*. More precisely, for any m (polynomial in the security parameter), Barak demonstrates the existence of a $\mathcal{ZK}$ protocol which remains secure as long as the number of concurrent sessions is bounded by m. (On the flipside, however, the communication complexity of his protocol grows linearly with m.) Intriguingly, the black-box impossibility results of [CKPR01] for constant-round concurrent $\mathcal{ZK}$ actually applies also to bounded-concurrency, and indeed Barak develops a new *non-black-box simulation* [Bar01] to obtain his result. Note that bounded-concurrent $\mathcal{ZK}$ is different from concurrent $\mathcal{ZK}$ in that for the latter, we require the same protocol to be secure under *any* polynomial number of concurrent sessions.

Towards getting a constant-round concurrent $\mathcal{ZK}$ protocol, in [CLP13, CLP15], the existence of constant-round concurrent $\mathcal{ZK}$ arguments were shown assuming the existence of certain types of "delegation of computation schemes" for $\mathcal{P}$ (as well as standard cryptographic hardness assumptions—namely collision-resistant hash functions and one-way permutations); additionally, in [CLP15] it was shown that such delegation schemes can be based on the existence of *indistinguishability obfuscation* (iO) [BGI$^+$01, GGH$^+$16] (as well as one-way permutations). Although iO is an extremely intriguing concept in its own right, constructions of iO under standard assumptions are still not known, and thus the question of basing constant-round concurrent $\mathcal{ZK}$ on "standard" assumptions still remains open.

Public-coins v.s. Private-coins. Whereas the original ZK protocols of [GMR89, GMW91, Blu86] are *public-coin*—i.e., the Verifier's messages are its random coin-tosses—all of the aforementioned parallel or concurrent $\mathcal{ZK}$ protocols use private coins. Indeed, Goldreich and Krawczyk [GK96b] showed that only trivial languages can have constant-round public-coin (stand-alone) *black-box* $\mathcal{ZK}$ protocols with negligible soundness error, let alone the question of parallel composition. Their result implies that (unless $\mathcal{NP} \subseteq \mathcal{BPP}$), the constant-round $\mathcal{ZK}$ protocols of e.g., [GMW91, Blu86] with constant soundness error cannot be black-box $\mathcal{ZK}$ under parallel repetition (as this would yield a constant-round black-box $\mathcal{ZK}$ protocol with negligible soundness error). More recently, Pass, Tseng and Wikström [?] showed that *no* public-coin protocol (even those with a polynomial number of rounds) for a non-trivial language can be black-box $\mathcal{ZK}$ under parallel composition.

These black-box barriers can be overcome: Pass, Rosen and Tseng [PRT13] show the existence of a constant-round public-coin $\mathcal{ZK}$ protocol for $\mathcal{NP}$ (with negligible soundness error) which remains secure under (unbounded) parallel composition with a non-black-box simulator (based on standard cryptographic hardness assumptions), and Goyal [Goy13] demonstrates a (polynomial-round) public-coin protocol that remains secure even under (unbounded) concurrent composition.

Concurrency Beyond $\mathcal{ZK}$: Secure Computation and Black-box Impossibilites. As one may expect, techniques developed for concurrent $\mathcal{ZK}$ enable reasoning about concurrent security of other types of cryptographic protocols:

- [Lin03, PR03, Pas04] show how to extend Barak's simulation technique to develop general secure computation protocols [Yao86, GMW87, BGW88] that remain secure under bounded concurrency.
- [CLP10] shows how to get general secure computation protocols satisfying a relaxed notion of concurrent "super-polynomial-time" (SPS) security [Pas03, PS04, BS05, CLP10] based on standard assumptions using simulation techniques similar to those employed by Richardson and Kilian [RK99]. It is known that concurrent secure computation satisfying the standard notion of "polynonomial-time simulation" is impossible [CF01, Lin04] and thus going for a relaxed notion of security such as SPS is needed here. (See also [GLP$^+$15] for a protocol with improved round-complexity).

But perhaps more surprisingly, techniques developed for establishing the *feasibility* of concurrent $\mathcal{ZK}$ protocols turned out to also be useful for developing *impossibility* results for seemingly unrelated tasks:

- As we showed in [Pas11], concurrent simulation techniques are important also when trying to show *black-box separations*: [Pas11] shows that black-box security reductions cannot be used to base the security of several "paradoxical" protocols (such as e.g., Schnorr's identification scheme, commitment schemes secure against selective openings, Chaum Blind Signatures, etc.) on "standard assumptions". As a black-box reduction may invoke multiple (concurrent) sessions of the adversary, dealing with concurrency (and nested sessions) is a key technical challenge in establishing such impossibility results.

And conversely, techniques developed to establish black-box impossibility results for concurrent $\mathcal{ZK}$ turned out to be useful in establishing the feasibility of other primitives:

- In [COPV13], it was shown that black-box impossibility results for concurrent $\mathcal{ZK}$ due to [CKPR01, CPT12] can be used to develop so-called "resettably-sound" $\mathcal{ZK}$ protocols [BGGL01] based on minimal assumptions.

As we hope to have conveyed, understanding concurrent $\mathcal{ZK}$ is important beyond just $\mathcal{ZK}$—whether it is to study concurrent security of more general secure computations protocols, or to establish black-box impossibility results for other tasks. Furthermore, the question of whether constant-round concurrent $\mathcal{ZK}$ exists is linked to other intriguing open questions in the context of delegation of computation and program obfuscation.

A Simple Concurrent $\mathcal{ZK}$. Despite improvements, simplifications and generalizations, the analyses of concurrent $\mathcal{ZK}$ protocols remain very complex and subtle. In the remainder of this tutorial, we aim to present a concurrent $\mathcal{ZK}$ protocol with a simple analysis: we do not try to minimize rounds, or assumptions—the protocol requires $O(n^\epsilon)$ rounds (just as the original work by Richardson and Kilian), and relies on the existence of one-way permutations—but our hope is that this analysis may make it feasible to teach the wonders of concurrent $\mathcal{ZK}$ in a graduate class on Cryptography.

Our analysis follows simulation techniques from [CLP10, Pas11] (and is also closely related to a technique from [DGS09]) developed for concurrent simulation of more general interaction, but as far as we know, these simulation techniques were not previously brought back to concurrent $\mathcal{ZK}$.

A Personal Note: *$\mathcal{ZK}$ proofs are, in my opinion, one of the deepest, intriguing and most surprising concepts in computer science. The fact that one can convince someone of the validity of some statement without revealing anything else beyond it, just seems impossible. Yet $\mathcal{ZK}$ proofs enable it! It was this notion that made me fall in love with Cryptography: I had decided to go back to graduate school and Johan Håstad handed me the paper* "Resettable Zero-Knowledge" *by Canetti, Goldreich, Goldwasser and Micali [CGGM00a], and said "this is a paper by some of my friends, it may be fun". At this point, I had no background in crypto and had never read a research paper. So "fun" it was not. It took me months to get even the most basic understanding of this paper—a core technical component was a detailed analysis of Richardson and Kilian's concurrent zero-knowledge protocol—but, even though I didn't understand the details, I had become obsessed by $\mathcal{ZK}$ and even more so by the notion of Concurrent Zero-knowledge: How could it be that something gave "zero knowledge" when executed in isolation, but no longer did so if one provided many concurrent proofs. How can 0+0 not be 0? Over a decade later, I am still as obsessed with this notion, and it is an honor to contribute a piece on Concurrent Zero-Knowledge in this tribute to Shafi's and Silvio's work.*

2 Preliminaries

We assume familiarity with probability ensembles, indistinguishability and interactive proofs [GMR89] and arguments [BCC88]; recall that in interactive proof, soundness holds with respect to all computationally *unbounded* malicious provers, whereas in an interactive argument, soundness only needs to hold with respect to computationally bounded (i.e., non-uniform polynomial-time) provers. .

2.1 Black-box Concurrent Zero-Knowledge

Let (P, V) be an interactive proof/argument 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—called *sessions*—of some prover $P(x, w)$. 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 $\mathrm{View}_{V^*}^{P(x,w)}(x, z)$ be the random variable that denotes the *view* of $V^*(x, z)$ in an interaction with $P(x, w)$ (this includes the random coins of V^* and the messages received by V^*). A *black-box simulator* S is a probabilistic expected 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, the simulator $S^{V^*(x,z)}(x)$ (having only access to $V^*(x, z)$, but *not* the prover $P(x, w)$) can generate the view of $V^*(x, z)$ in an interaction with $P(x, w)$. 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 proof/argument for a language $L \in \mathcal{NP}$ with witness relation R_L. (P, V) 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 following ensembles are computationally indistinguishable:

- $\left\{\text{View}_{V^*}^{P(x,w)}(x,z)\right\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$
- $\left\{S_m^{V^*(x,z)}(x)\right\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$

It is worth noting that the definition of black-box concurrent $\mathcal{ZK}$ allows for a *different* simulator for every polynomial bound m on the number of sessions (whereas the standard definition of black-box $\mathcal{ZK}$ does not need a different simulator for each polynomial that bounds the verifier's running time). The reason for this is that we need to allow the simulator to run in polynomial time in m even just to read all the messages sent by an m-session verifier.

2.2 Other Primitives

Witness-indistinguishable ($\mathcal{WI}$) 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 $L \in \mathcal{NP}$ with witness relation R_L. We say that (P, V) is *witness-indistinguishable ($\mathcal{WI}$)* 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:

- $\left\{\text{View}_{V^*}^{P(w_x^1)}(x,z)\right\}_{x\in L, z\in\{0,1\}^*}$
- $\left\{View_{V^*}^{P(w_x^2)}(x,z)\right\}_{x\in L, z\in\{0,1\}^*}$

If, further, the above probability ensembles are identically distributed, we say that (P, V) is *perfectly witness indistinguishable.*

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. Given two interactive machine, A,B, let $\langle A, B\rangle(x)$ be a random variable denoting the output of B in an interaction with A given the common input x.

Definition 3 (Proofs and arguments of knowledge [BG92])**.** An interactive protocol (P, V) is a *proof of knowledge* (resp. *argument of knowledge*) of language L with respect to witness relation R_L if (P, V) is an interactive proof (resp. argument) for L, and 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 $R_L(x)$ 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 Proofs [CDS94]. Special-sound proofs are proofs of knowledge with a very rigid and useful structure.

Definition 4 (Special soundness)**.** A 3-round interactive proof (P, V) for language $L \in \mathcal{NP}$ with witness relation R_L is *special sound* with respect to R_L if:

- (P, V) is public-coin (i.e., the verifier message is its random tape), and the length of the verifier message (a.k.a. the "challenge") on input x is $|x|$.[2]
- there exists a deterministic polynomial-time extraction procedure E such that for any $x \in L$, all α, β, β', γ, γ' such that $\beta \neq \beta'$, and (α, β, γ) and $(\alpha, \beta', \gamma')$ are both accepting transcripts of (P, V) on input x, the extractor $E(x, (\alpha, \beta, \gamma), (\alpha, \beta', \gamma'))$ outputs a witness $w \in R_L(x)$ for x.

2.3 Known Protocols

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

- A $\mathcal{WI}$ special-sound proof for $\mathcal{NP}$; this can be instantiated by a parallel repetition of the Blum's Hamiltonicity protocol [Blu86] based on one-way permutations.
- For every $\epsilon > 0$, an $O(n^\epsilon)$-round perfectly $\mathcal{WI}$ argument of knowledge for $\mathcal{NP}$. This can be instantiated with a variant of Blum's Hamiltonicity protocol using an $O(n^\epsilon)$-round perfectly hiding (as opposed to perfectly binding) commitment, which also can be based on one-way permutations [NOVY98].

Both of these primitives can, for instance, be based on the hardness of the discrete logarithm problem [Gol01].

3 Black-Box Concurrent Zero-Knowledge Arguments of Knowledge

In this section, we prove the following theorem.

Theorem 1. *For any $\epsilon > 0$, assume the existence of a $\mathcal{WI}$ special-sound proof for $\mathcal{NP}$, and an $O(n^\epsilon)$-round perfectly-$\mathcal{WI}$ argument of knowledge for $\mathcal{NP}$. Then, there exists an $O(n^\epsilon)$-round concurrent black-box $\mathcal{ZK}$ argument of knowledge for $\mathcal{NP}$.*

[2] For most applications, including ours, it suffices that that the length of the challenge is $\omega(\log|x|)$, but for notational simplicity, we simply require the length of the challenge to be $|x|$.

3.1 The Protocol

Our concurrent $\mathcal{ZK}$ protocol ConcZKArg (also used in [PV08, PTV14]) is a slight variant of the precise $\mathcal{ZK}$ protocol of [MP06], which in turn is a generalization of the Feige-Shamir protocol [FS90]. Given a common input statement $x \in \{0,1\}^n$, a "round-parameter" $k = n^\epsilon$, the protocol for language L proceeds in three stages,

Init Stage: The verifier V picks two random strings $r_1, r_2 \in \{0,1\}^n$ and sends their images $c_1 = f(r_1)$ and $c_2 = f(r_2)$ under a one-way function f to the prover. Next, V then initiates k repetitions of a $\mathcal{WI}$ special-sound proof of the $\mathcal{NP}$ statement "c_1 or c_2 is in the image set of f" (a witness here would be a pre-image of either c_1 or c_2), and sends the prover P the first messages $(\alpha_1, \ldots, \alpha_k)$ for each of these k instances.

Stage 1: k message exchanges occur in Stage 1. In the j'th iteration, the prover P sends $\beta_j \in \{0,1\}^n$, a random "challenge" for the j'th special-sound proof, and V replies with the third message γ_j of the special-sound proof. These k iterations are referred to as slots. A slot is *convincing* if V produces an accepting proof. If there is ever an *unconvincing* slot, P aborts the whole session.

Stage 2: The prover provides a perfectly-$\mathcal{WI}$ argument of knowledge of the statement "$x \in L$, or either 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 (due to the $\mathcal{WI}$ property of Stage 1). A formal description of protocol ConcZKArg is shown in Figure 3.

3.2 The Simulator Algorithm

We will show that the protocol is black-box concurrent $\mathcal{ZK}$ when $k = n^\epsilon$. To simplify notation, let $\tilde{n} = n^{\epsilon/2}$, and thus the number of slots $k = \tilde{n}^2$. We construct a simulator $\mathsf{Sim} = \mathsf{Sim}^{V^*(x,z)}(x)$ that given as input an instance $x \in L$ and black-box access to $V^*(x,z)$, outputs a view that is statistically close from the "real view" of $V^*(x,z)$ in a multi-session interaction with $P(x,w)$, for any $w \in R_L(x)$.

On a high-level, the simulation follows that of Richardson and Kilian [RK99]. The simulator simulates the Init Stage 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 manages to successfully rewind some slot (i.e., obtain two accepting responses to the slot), it can use the special-soundness extractor to *extract* 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 by straight-forward emulation. The crux of the simulation is to provide a method for rewinding slots that ensures the following two properties:

Property 1: Whenever the simulator reaches Stage 2 of the protocol in any of the concurrent sessions, at least one of the slots for that session has been "successfully rewound" (and thus the simulator has a fake witness that can be used to complete Stage 2). We refer to such a session as being solved.

Protocol ConcZKArg:

Common Input: an instance $x \in \{0,1\}^n$ of a language L with witness relation R_L.

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

Init Stage:

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

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

$V \to P$: the first messages $\alpha_1, \dots, \alpha_k$ for k $\mathcal{WI}$ special-sound proof of the statement (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 special-sound proofs.

Stage 1: For $j = 1$ to k repeat the following slots:

$P \to V$: The second message (a.k.a. the "challenge") β_j of the j'th special-sound proof

$V \to P$: The last message (a.k.a. the "response") γ_j of the j'th special-sound proof

Stage 2:

$P \leftrightarrow V$: a perfectly-$\mathcal{WI}$ argument of knowledge from P to V of the statement (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 3: Concurrent $\mathcal{ZK}$ argument of knowledge for $\mathcal{NP}$ with round parameter $k(\cdot)$.

Property 2: The rewindings can be done in a way that does not "blow-up" the running-time of the simulation. In particular, to ensure Property 1, the simulator will have to *recursively* rewind the verifier, and will need to carefully select which slot to rewind to ensure to ensure timely termination.

Description of Sim. Given some statement x, let $n = |x|$, and let $m = m(n)$ be an upper bound on the number of concurrent sessions invoked by V^* and $T = T(n)$ be a bound on the total number of messages exchanged to be exchanged with V^*; note that $T = mk = \text{poly}(n)$. Recall that by the definition of black-box simulation, we need only consider deterministic malicious verifiers V^*; therefore the view of $V^*(x, z)$ is just the transcript of its interaction with the honest prover.

As mentioned above, our simulator Sim starts by honestly simulating the Init Stage and Stage 1 for V^* (i.e., by replying just like the honest prover). We say that a slot (of one of the sessions) "opens" when V^* receives a " challenge" β_j from Sim (i.e., when it receives the first message of the slot), and that the slot "closes" when V^* sends back its response to Sim. Formally, the opening of a slot is a partial view v of V^* immediately after which the slot opens. In the sequel, we identify a slot simply by its opening (i.e., the partial view after which it opens). Analogously, the closing of a slot s is a partial view v immediately after which s closes.

By definition, a slot can never close before opening. But what makes our life complicated is that V^* may send lots of other messages (and start other sessions) before responding to a slot (i.e., it can nest sessions as in Figure 2). Thus, we may never see the closing of a slot unless we can simulate all the messages V^* expects to see before closing the slot.

Once a slot closes, we would like to "rewind" it by sending a new slot opening (i.e., a new challenge) and waiting for the slot to close again (so that we can solve the session). But this requires simulating all the messages within the slot again: we do this by *recursively* invoking the simulator. The problem, of course, is that if the recursive depth (i.e., the number of nested recursive calls) becomes large, the running time of the simulation will blow up. Our goal is to ensure that the recursive (i.e., nesting) depth is some *constant* D: this will intuitively ensure that the expected running time will be $\text{poly}(n)^D$ (as the expected number of rewindings for each slot is 1 and there are at most $\text{poly}(n)$ slots).

On a high-level, we achieve this goal by carefully selecting which slots to rewind (intuitively, ones that is "light" to simulate). The malicious verifier V^* may abort in the rewinding, in which case we simply rewind the slot again. Furthermore, although the slot was "light" in the initial simulation, V^* may change its scheduling in the rewinding to make the slot "heavy"! Whenever, this happens, we artificially abort the rewinding and restart with a new one. Sim continues rewinding in this fashion until the session gets solved.

More precisely, Sim honestly emulates the Init Stage and Stage 1 for V^* until a slot s closes for which the following property holds:

- Between the time when the slot s opened, and the time that it closed, the *number of other slots* that opened is "small", where "small" will be defined shortly based on the recursive depth of the simulator.

Whenever this happens, Sim rewinds V^* back until the point where s opened, and recursively invokes itself to simulate the messages within the slot s one more time; additionally, if the number of slots s' that V^* opens up in this rewinding (i.e., within slot s) no longer is "small", the rewinding is cancelled. Sim continues rewinding V^* until it gets another accepting closing of the slot s, and can now use the special-soundness extractor to recover a fake witness to use in Stage 2. (We remark that in contrast to the simulation technique of [RK99], we do *not* decide what slot to rewind based on the *number of sessions* that start within the slot, but rather, following [DGS09, CLP10, Pas11], decide what slot to rewind based on the total *number of slots* within the slot (regardless of sessions).)

It remains to specify what "small" means. Note that the recursive depth of the simulation corresponds to the number of "nested rewindings" in the simulation. Recall that we want to make sure that the maximal depth (i.e., maximal number of nested rewindings) becomes a *constant* so that the running-time of the simulation stays polynomial. To do this, the definition of "small" will need to vary based on the recursive depth d of the simulation. Given the (partial) view τ of V^*:

- We say that a prefix ρ of τ is d-good if the number of slots that open in τ *after* ρ is less than $\frac{T}{\tilde{n}^d}$ (recall that T is an upper bound on the total number of messages);
- We say that a slot s is d-good in τ if s (i.e, the opening of s) is a d-good prefix of τ. (In other words, a slot s is d-good if the number of new slots that opened since its opening is less than $\frac{T}{\tilde{n}^d}$).

Now, at recursive level d, whenever a slot s closes, we will only rewind it if it is $(d+1)$-good; thus, when we are "deeper" in the recursion (i.e., at a higher recursive depth), we will only rewind slots that have fewer slots inside them (and this ensures that the recursive depth of the simulation is a constant). In more detail, the simulation proceeds as follows.

- On recursive level $d \geq 0$, starting from a view $\mathcal{V}$, Sim honestly emulates the prover strategy for V^*, until a slot s that opened inside the view $\mathcal{V}$ closes *and* the slot is $(d+1)$-good for the current view v. Whenever this happens, it rewinds V^* back to the point when s opened, and invokes itself recursively at level $d+1$ to simulate the slot once more. If the slot closes after this "rewinding", Sim applies the special soundness extractor X to extract a fake witness; if the extractor outputs a valid witness r (to the statement c_1, c_2 currently proved by V^*), the pair (c_1, c_2, r) is stored. If the simulation in the rewinding fails (the condition under which it fails will be defined shortly), Sim simply attempts another rewinding of the slot, and continues doing so until it encounters a closing of the slot.

- At each recursive level $d \geq 1$ (i.e., on all recursive levels except the first one), if V^* aborts in $\mathcal{V}$, or $\mathcal{V}$ is not a d-good prefix of the current view v (i.e., if the number of new openings of slots becomes $\frac{T}{\tilde{n}^d}$), the recursive procedure halts outputting a fail symbol $\perp$ (returning to the earlier recursive call); this ensures that all rewindings are "cut-off" if V^* attempts to open more slots in the rewinding.

- Finally, whenever V^* is expecting to hear a Stage 2 proof for session j for a statement (c_1^j, c_2^j, x), Sim checks whether a "fake witness" r for (c_1^j, c_2^j, x) has been extracted; if so, it honestly completes Stage 2 using this witness, and otherwise halts outputting fail.

3.3 A Formal Description of Sim

We proceed to a formal description of the procedure Sim, and analyze its running-time and success probability. $\mathsf{Sim} = \mathsf{Sim}^{V^*(x,z)}(x)$ starts by invoking the recursively-defined procedure SIM (which we assume has oracle access to $V^* = V^*(x,z)$), described in Figure 4, on input $(x, 0, \emptyset)$.

Let us start by showing that the running time of Sim is bounded in expectation.

Proposition 2. *There exists some polynomial $t(\cdot)$ such that for every $x \in L$, $\mathsf{Sim}(x)$ runs in expected time bounded by $t(|x|)$.*

Proof. Intuitively, the proposition is based on the following observations:

- The maximal recursive depth is bounded by a constant $D = \lceil \log_{\tilde{n}} T \rceil$, as on level D, SIM returns $\perp$ if it encounters $\frac{T}{\tilde{n}^D} \leq 1$ new slots, so no new recursive calls can be made at level D.

- The expected number of rewindings to solve each slot is 1, as by the *perfect* witness indistinguishability property of Stage 2, the rewinding of a slot is simulated using exactly the same distribution as the original simulation of the slot.

- Since the total number of slots at each recursive level is bounded by T, the expected number of recursive calls at each level is bounded by T, from which we can conclude that expected running time of the simulator is bounded by $\mathrm{poly}(T^D)$.

We proceed to a formal proof. To simplify the analysis, let us consider a slight variant of Sim that never gets "stuck"—instead of ever halting outputting fail, let us assume that Sim has access to a witness w for x, which it can use in Stage 2 if Sim is ever is required to provide a witness for

```
PROCEDURE SIM$(x, d, \mathcal{V})$:
On input a statement $x$, the recursive level $d$ and the partial view $\mathcal{V}$ of $V^*$, proceeds as follows.
Let $v = \mathcal{V}$. Repeat the following:
  • If $V^*(v)$ is expecting to hear any Init Stage or Stage 1 message, honestly generate it and
    append it to $v$.
  • If $d > 0$ and $v$ is the closing of the slot opened at $\mathcal{V}$, return $v$.
  • If $d > 0$ and the partial view $v$ is not $d$-good, or if $V^*(v)$ aborts (i.e., sends an invalid
    message or simply terminates), return $\bot$.
  • If $v$ is the closing of a slot $s$ that opened after $\mathcal{V}$ and that is $(d+1)$-good for $v$, repeat:
      – $v' = \mathsf{SIM}(x, d+1, s)$
    until $v' \neq \bot$.
    Next, apply the special soundness extractor $X$ on the transcripts corresponding to the
    special-soundness proofs in the two views $v, v'$. If $X$ succeeds in finding a witness $r$ for the
    statement $(c_1, c_2)$ proved, store $(c_1, c_2, r)$.
  • If $V^*(v)$ is expecting to hear a Stage 2 proof for a statement $(c_1, c_2, x)$, check if a tuple pair
    $(c_1, c_2, r)$ has been stored. If so, use the "fake witness" $r$ to honestly provide the Stage 2
    proof (one message at a time), and append the prover message $v$; otherwise halt outputting
    fail.
  • Finally, if $d = 0$ and $V^*(v)$ aborts (i.e., sends an invalid message or simply terminates),
    return $v$.
```

Figure 4: Pseudo-code for the recursive simulation strategy employed by Sim.

a statement (c_1, c_2, x) for which it has not recovered a fake witness. Clearly this change can only increase Sim's running-time.

Note that the recursive depth is bounded by $D = \lceil \log_{\tilde{n}} T \rceil$, which is a constant (since T is polynomial in n and thus also in $\tilde{n}$). Secondly, at each recursive level d, there are at most T possible points from which we can rewind. As we shall argue, from each of these points (i.e., partial views), the expected number of rewindings is bounded by 1. Recall that for every view $\mathcal{V}$, the execution of $\mathsf{SIM}(x, d, \mathcal{V})$, Sim only starts "rewinding" a slot s if 1) the slot s opened after $\mathcal{V}$, 2) the slot s closes in the current view v (which extends $\mathcal{V}$), and 3) the slot s is $(d+1)$-good for v. Furthermore, in each of the rewindings, the simulated view of the adversary on the recursive level $d+1$ (i.e., in the execution of $\mathsf{SIM}(x, d+1, s)$) is *identically distributed* to its view in the execution on level d; note that we here rely on the the assumption that Sim never gets "stuck", and the fact that the Stage 2 proof is *perfectly* witness indistinguishable. Thus, the probability that the slot s becomes $(d+1)$-good for some view v' in the recursive call on level $d+1$ (i.e., that the rewinding is successful) is at least the probability that the slot was $(d+1)$-good on level d.[3] Since Sim rewinds the slot until it gets another accepting closings, the expected number of rewindings from each partial view

[3]The probability might actually be larger, since on level d we might also abort if the current view is no longer d-good.

is thus at most 1.

So, for any recursive level d, and any view $\mathcal{V}$, in the execution of $\mathsf{SIM}(x, d, \mathcal{V})$, the expected number of rewindings (i.e., recursive invocations of $\mathsf{SIM}(x, d+1, \mathcal{V}')$ for some view $\mathcal{V}'$) is bounded by T. It follows using a standard induction that for each recursive level $d \leq D$, and every view $\mathcal{V}$, the expected number of messages sent by $\mathsf{SIM}(x, d, \mathcal{V})$ (and its recursive sub-routine calls) to V^* is bounded by T^{D+1-d}—note that we here rely on the fact that the upper-bound on the expected number of rewindings inside $\mathsf{SIM}(x, d+1, \mathcal{V}')$ is *independent* of the starting view $\mathcal{V}'$, and thus the expectations can be multiplied. □

Let us now argue that Sim generates a view that is statistically close to the real view. First, note that if we consider a variant $\tilde{\mathsf{Sim}}$ of Sim that (1) never halts outputting fail, and (2) always uses the real witnesses for x in Stage 2, then the view output by $\tilde{\mathsf{Sim}}$ is identically distributed to a real view: This directly follows from the fact that $\tilde{\mathsf{Sim}}$ honestly emulates Init Stage, Stage 1 and Stage 2 messages, and only uses rewindings to learn a "fake witness", which is not even used by $\tilde{\mathsf{Sim}}$.

Next, consider a variant Sim' of Sim that proceeds just as Sim but never fails and instead uses a real witness in Stage 2 for any session for which it fails to extract a fake witness; for all other sessions (i.e., those for which a fake witness is extracted), it still uses the fake witness in Stage 2. It follows directly from the perfect witness indistinguishability property of Stage 2 that the view output by Sim' is identically distributed to the view output by $\tilde{\mathsf{Sim}}$ (and thus also a real view).

Finally, note that Sim and Sim' behave identically except in the event that Sim outputs fail. Below, we show that Sim outputs fail only with negligible probability which concludes the proof of the correctness of the simulation.

Proposition 3. *There exists a negligible function μ such that for all $x \in L$, the probability that* $\mathsf{Sim}(x)$ *outputs* fail *is bounded by $\mu(|x|)$.*

Proof. Intuitively, the proposition is based on the following observations:

- Whenever the simulator reaches Stage 2 of some session j, it is the case that $\tilde{n}^2$ slots for that session have closed. Since the maximal recursive depth is some constant D, at least $\tilde{n}^2/D > \tilde{n}$ (for sufficiently large n) of these slots closed in one invokation of SIM on some particular recursive depth $\tilde{d}$.
- As there can be at most a total of $M = \frac{T}{n^{\tilde{d}}}$ slots that opened up in that invokation of SIM (or else SIM would abort returning $\perp$), we are guaranteed that there is at least one slot for session j that has less than $\frac{M}{\tilde{n}}$ slots inside it, and this slot must thus be $(\tilde{d}+1)$-good and will be rewound.
- Since a slot is rewound until it closes again, we are guaranteed that a witness can be extracted for session j as long as the special-soundness extractor does not fail to extract a witness. But the special-soundness extractor only fails if the challenge (i.e., slot opening) in the two transcript we feed him are the same. This happens with negligible probability as the length of the challenge is n and the expected running time of the simulator is polynomial (as shown in Proposition 2).

We proceed to a formal proof. Let us consider the following two events:

- Let E_1 denote the event that Sim is required to provide a Stage 2 proof for some instance (c_1, c_2, x) without having previously "rewound" at least one slot for a proof of (c_1, c_2).

- Let E_2 denote the event that the special soundness extractor X fails to output a valid witness in the execution by Sim.

Note that if neither E_1 nor E_2 happen, there always exists some slot that is rewound for which the special-soundness extractor succeeds, which means that Sim can never fail.

We show below that the these events can happen only with negligible probability, and thus by a union bound, the probability that either of them happens is also negligible, which concludes the proof.

Claim 3.1. *For sufficiently large $x \in L$, the probability that E_1 happens in the execution of $\mathsf{Sim}(x)$ is 0.*

Proof. Assume for contradiction that Sim reaches Stage 2 of some session and is required to provide a proof for (c_1, c_2, x), yet none of the slots for (c_1, c_2) were rewound. Fix some random tape for Sim for which this happens—in the sequel of the proof, we will be considering the execution of Sim with this fixed random tape. Let $\tilde{v}, \tilde{d}$ be the view and recursive level for which this happened. To reach Stage 2, Sim must thus have previously encountered $k = \tilde{n}^2$ slots for (c_1, c_2). These slots may not necessarily have opened on recursive level $\tilde{d}$, but may instead have opened on some earlier recursive level $d < \tilde{d}$—formally, we say that a slot s opened up on recursive level d if the opening of the slot was generated by $\mathsf{SIM}(x, d, v)$ in the execution by Sim (with the fixed random tape), where v is a prefix of $\tilde{v}$. Since the recursive depth of Sim is bounded by some constant $D = \lceil \log_{\tilde{n}} T \rceil$, there nevertheless must exist some recursive level d such that at least $k/D = \tilde{n}^2/D$ of those slots opened on recursive level d. For sufficiently large $n, \tilde{n}^2/D > \tilde{n}$ and thus there is exists more than $\tilde{n}$ such slots. Additionally, by the recursive construction of the simulator, there exist a *single* partial view v such that all those $\tilde{n}$ slots opened within the execution of $\mathsf{SIM}(x, d, v)$.

Since the total number of slots that can open up during the execution of $\mathsf{SIM}(x, d, v)$ is bounded by $M = \frac{T}{\tilde{n}^d}$ (for $d = 0$, this follows by the definition of T; and for $d > 0$, this follows since by definition of SIM, the simulation at recursive level d is cancelled if more than $\frac{T}{\tilde{n}^d}$ slots open), there exists at least 1 slot that contains less than $\frac{M}{\tilde{n}} = \frac{T}{\tilde{n}^{d+1}}$ slots; this slot is thus $(d+1)$-good and would have been rewound, which is a contradiction. □

Claim 3.2. *There exists some negligible function $\mu(\cdot)$, such that for every $x \in L$, the probability that E_2 happens in the execution of $\mathsf{Sim}(x)$ is bounded by $\mu(|x|)$.*

Proof. Assume for contradiction that there exists some polynomial $p(\cdot)$ such that E_2 happens in the execution of $\mathsf{Sim}(x)$ with probability $\frac{1}{p(|x|)}$ for infinitely many x. Recall that by Proposition 2, the expected running time of Sim is bounded by some polynomial $t(\cdot)$. By the Markov inequality, it follows follows that the probability that Sim's running time exceeds $t'(|x|) = t(|x|) \cdot 2p(|x|)$ steps is at most $\frac{1}{2p(|x|)}$. Thus, by the union bound, we have that the probability that E_2 happens while Sim takes less than $t'(|x|)$ steps is at least $\frac{1}{2p(|x|)}$ (i.e., inverse polynomial).

Next, note that the special-soundness extractor can only fail to extract a witness if the simulator sent the same verifier challenge in the two views v, v'. Since the length of the verifier challenges is $|x|$, the probability that this happens for any given pair v, v' is $2^{-|x|}$. Consequently, it follows by a union bound that E_2 can happen with probability at most $t'(|x|)2^{-|x|}$ (i.e., with negligible probability) when Sim takes at most $t'(|x|)$ steps, which is a contradiction. □

□

4 Acknowledgements

I am extremely grateful to Oded Goldreich for his great comments on the presentation of this tutorial. I am also grateful to Naomi Ephraim, Cody Freitag, and Andrew Morgan for their useful feedback. Finally, I am indebted to Alon Rosen for endless hours of discussions about concurrent $\mathcal{ZK}$ in 2004; my understanding of this notion was born out of these discussions.

References

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS '01*, volume 0, pages 106–115, 2001.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[BCY91] Gilles Brassard, Claude Crépeau, and Moti Yung. Constant-round perfect zero-knowledge computationally convincing protocols. *Theoretical Computer Science*, 84(1):23–52, July 1991.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO '92*, pages 390–420, 1992.

[BGGL01] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. In *FOCS '02*, pages 116–125, 2001.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology CRYPTO 2001*, pages 1–18. Springer, 2001.

[BGW88] Michael Ben–Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88*, pages 1–10, 1988.

[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.

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *FOCS '05*, pages 543–552, 2005.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO '01*, pages 19–40, 2001.

[CGGM00a] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 235–244, 2000.

[CGGM00b] 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.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *FOCS*, pages 541–550, 2010.

[CLP13] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero knowledge from p-certificates. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 50–59, 2013.

[CLP15] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero-knowledge from indistinguishability obfuscation. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 287–307, 2015.

[COPV13] Kai-Min Chung, Rafail Ostrovsky, Rafael Pass, and Ivan Visconti. Simultaneous resettability from one-way functions. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 60–69, 2013.

[CPT12] Kai-Min Chung, Rafael Pass, and Wei-Lung Dustin Tseng. The knowledge tightness of parallel zero-knowledge. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 512–529, 2012.

[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.

[DGS09] Yi Deng, Vipul Goyal, and Amit Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *FOCS*, pages 251–260, 2009.

[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.

[GGH$^+$16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM J. Comput.*, 45(3):882–929, 2016.

[GK96a] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.

[GK96b] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.

[GLP$^+$15] Vipul Goyal, Huijia Lin, Omkant Pandey, Rafael Pass, and Amit Sahai. Round-efficient concurrently composable secure computation via a robust extraction lemma. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, pages 260–289, 2015.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *J. ACM*, 38(3):690–728, 1991.

[Gol01] Oded Goldreich. *Foundations of Cryptography — Basic Tools.* Cambridge University Press, 2001.

[Gol02] Oded Goldreich. Concurrent zero-knowledge with timing, revisited. In *STOC '02*, pages 332–340, 2002.

[Goy13] Vipul Goyal. Non-black-box simulation in the fully concurrent setting. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 221–230, 2013.

[KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in poly-loalgorithm 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.

[Lin03] Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 683–692, 2003.

[Lin04] Yehuda Lindell. Lower bounds for concurrent self composition. In *TCC '04*, pages 203–222, 2004.

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In *STOC '06*, pages 306–315, 2006.

[NOVY98] Moni Naor, Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Perfect zero-knowledge arguments for *NP* using any one-way permutation. *J. Cryptology*, 11(2):87–108, 1998.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *EUROCRYPT*, pages 160–176, 2003.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *STOC '04*, pages 232–241, New York, NY, USA, 2004. ACM.

[Pas11] Rafael Pass. Limits of provable security from standard assumptions. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 109–118, 2011.

[PR03] Rafael Pass and Alon Rosen. Bounded-concurrent secure two-party computation in a constant number of rounds. In *FOCS '03*, pages 404–, 2003.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithic round-complexity. In *FOCS '02*, pages 366–375, 2002.

[PRT13] Rafael Pass, Alon Rosen, and Wei-Lung Dustin Tseng. Public-coin parallel zero-knowledge for NP. *J. Cryptology*, 26(1):1–10, 2013.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *STOC*, pages 242–251, 2004.

[PTV10] Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Eye for an eye: Efficient concurrent zero-knowledge in the timing model. In *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, pages 518–534, 2010.

[PTV14] Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Concurrent zero knowledge, revisited. *J. Cryptology*, 27(1):45–66, 2014.

[PV08] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. 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.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.