On Constant-Round Concurrent Zero-Knowledge

Rafael Pass and Muthuramakrishnan Venkitasubramaniam

Cornell University,
{rafael,vmuthu}@cs.cornell.edu

Abstract. Loosely speaking, an interactive proof is said to be *zero-knowledge* if the view of every "efficient" verifier can be "efficiently" simulated. An outstanding open question regarding zero-knowledge is whether constant-round concurrent zero-knowledge proofs exists for non-trivial languages. We answer this question to the affirmative when modeling "efficient adversaries" as probabilistic quasi-polynomial time machines (instead of the traditional notion of probabilistic polynomial-time machines).

1 Introduction

Zero-knowledge interactive proofs [14] 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. This is formalized by requiring that the view of every "efficient" adversary verifier V^* interacting with the honest prover P be simulated by an "efficient" machine S (a.k.a. the *simulator*). The idea behind this definition is that whatever V^* might have learned from interacting with P, he could have actually learned by himself (by running the simulator S). As "efficient" adversaries normally are modelled as probabilistic polynomial-time machines ($\mathcal{PPT}$), the traditional definition of $\mathcal{ZK}$ models both the verifier and the simulator as $\mathcal{PPT}$ machines. In this paper, we investigate alternative models of efficient adversaries—in particular, as in [21], we model adversaries as probabilistic quasi-polynomial time machines ($\mathcal{PQT}$).

Concurrency and $\mathcal{ZK}$. The notion of concurrent $\mathcal{ZK}$, first introduced and achieved, by Dwork, Naor and Sahai [8] considers the execution of zero-knowledge proofs in an asynchronous setting and concurrent setting. More precisely, we consider a single adversary mounting a coordinated attack by acting as a verifier in many concurrent executions. Concurrent zero-knowledge 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 protocols have been obtained based on different set-up assumptions (e.g., [9] [6] [4]). On the other hand, in the "plain" model without any set-up Canetti, Kilian, Petrank and Rosen [5] (building on earlier works by [17] [26]) show that concurrent $\mathcal{ZK}$ proofs for non-trivial languages, with so called "black-box" simulators, require at least $\Omega(\frac{\log n}{\log\log n})$ number of communication rounds. Richardson and Kilian [25] constructed the first concurrent zero-knowledge argument in the standard model. Their protocol which uses a black-box simulator requires $O(n^{\epsilon})$ number of rounds. Kilian and Petrank [16] later obtained a round complexity of $\tilde{O}(\log^2 n)$, and finally Prabhakaran, Rosen and Sahai [23] essentially closed the gap by obtaining a round complexity of $\tilde{O}(\log n)$.

All of the above results rely on the traditional modeling of adversaries as $\mathcal{PPT}$ machines. Thus, it is feasible that there exists some super-polynomial, but "well-behaved", model of adversaries that admits constant-round concurrent $\mathcal{ZK}$ proofs.

Concurrent $\mathcal{ZK}$ w.r.t super-polynomial adversaries. The lower bound of [17] shows that only languages decidable in probabilistic subexponential-time have 4-round concurrent black-box zero-knowledge arguments w.r.t to probabilistic subexponential-time adversaries. On the other hand, [21] constructs constant-round concurrent zero-knowledge arguments w.r.t $\mathcal{PQT}$ *verifiers* (and consequently also simulators); however the soundness condition of those argument systems only holds w.r.t. $\mathcal{PPT}$ adversaries—in fact, the simulator succeeds in its simulation by breaking the soundness condition of the argument system. Additionally, it is noted in [21] that there exist 3-round concurrent $\mathcal{ZK}$ proofs w.r.t. exponential-time adversaries (as any witness indistinguishable proof is also zero-knowledge with respect to exponential-time verifiers). Finally, [25] claimed that a constant-round version of their protocol remains secure w.r.t $\mathcal{PQT}$ adversaries, when considering a "benign" type of concurrent adversary (which never sends any invalid messages and has a fixed—i.e., non-adaptively chosen—scheduling), but as far as we know a proof of this has never appeared.

Thus, the above results leave open the question of whether there exist $r(n)$-round concurrent black-box zero-knowledge proofs w.r.t super-polynomial, but sub-exponential, adversaries, as long as $4 < r(n) < \log n$. In particular,

> *Does there exists constant-round concurrent zero-knowledge arguments w.r.t. $\mathcal{PQT}$ (or even sub-exponential time) adversaries*?

1.1 Our results

Our main result answers the above question in the affirmative. Let $\mathcal{PQT}$ denote the class of probabilistic quasi-polynomial time machines, i.e., randomized machines that run in time $n^{poly(\log(n))}$. Let $\omega(\mathcal{PQT})$ denote the class of *probabilistic super quasi-polynomial time* machines, i.e. randomized machines that run in time $n^{\omega(poly(\log(n)))}$.

Theorem 1 (Main Theorem). *Assume the existence of claw-free permutations w.r.t $\mathcal{PQT}$. Then, every language in $\mathcal{NP}$ has an $O(1)$-round perfect concurrent black-box $\mathcal{ZK}$ argument w.r.t $\mathcal{PQT}$.*

In addition, we show:

Theorem 2. *Assume the existence of one-way functions that are secure w.r.t $\omega(\mathcal{PQT})$ and collision-resistant hash function that are secure w.r.t $\mathcal{PQT}$. Then, every language in $\mathcal{NP}$ has an $O(1)$-round concurrent computational black-box $\mathcal{ZK}$ proof w.r.t $\mathcal{PQT}$.*

Theorem 3. *Assume the existence of one-way function that are secure w.r.t $\omega(\mathcal{PQT})$. Then, every language in $\mathcal{NP}$ has an $O(1)$-round concurrent computational black-box $\mathcal{ZK}$ arguments w.r.t $\mathcal{PQT}$.*

Theorem 4. *There exists an $O(1)$-round concurrent perfect $\mathcal{ZK}$ proof w.r.t $\mathcal{PQT}$ for* Graph Non-Isomorphism *and* Quadratic Non-Residuosity

We emphasize that in the above theorems, "$\mathcal{ZK}$ proofs and arguments w.r.t $\mathcal{PQT}$" refer to proofs/ arguments where both the soundness condition and the $\mathcal{ZK}$ condition holds w.r.t to $\mathcal{PQT}$ adversaries; in particular, for the $\mathcal{ZK}$ property we also require that the distinguishability gap is smaller than the inverse of any quasi-polynomial function.

A note on expected running-time. In contrast to earlier work on concurrent zero-knowledge (e.g. [25, 16, 23]), our simulators run in *expected* $\mathcal{PQT}$. This is inherent: by the work of Barak-Lindell [1] it follows that only languages decidable in $\mathcal{PQT}$ have constant-round $\mathcal{ZK}$ protocols w.r.t $\mathcal{PQT}$ if requiring a strict $\mathcal{PQT}$ simulator (let alone the question of concurrency). In particular, this shows that none of the previous simulation techniques can be extended to get constant-round protocols w.r.t $\mathcal{PQT}$ (at least when requiring that the output of the simulation is also indistinguishable for $\mathcal{PQT}$).[1]

Additional results. Finally, we mention that our techniques apply also to concurrent $\mathcal{ZK}$ proofs w.r.t $\mathcal{PPT}$. As a result we obtain the first concurrent *perfect* $\mathcal{ZK}$ arguments/proofs w.r.t $\mathcal{PPT}$.

Theorem 5. *Assume the existence of claw-free permutations (w.r.t $\mathcal{PPT}$). Then, every language in $\mathcal{NP}$ has an $O(n^\epsilon)$-round perfect concurrent black-box $\mathcal{ZK}$ argument w.r.t $\mathcal{PPT}$, for every $\epsilon > 0$.*

Theorem 6. *For every $\epsilon > 0$, there exists a $O(n^\epsilon)$-round concurrent perfect $\mathcal{ZK}$ proof for* Graph Non-Isomorphism *and* Quadratic Non-Residuosity.

As an additional contribution, we believe that both our protocols and their analysis provides the simplest proof of the existence of concurrent $\mathcal{ZK}$ proofs (w.r.t $\mathcal{PPT}$).[2]

PQT v.s. PPT: What is right model for adversarial computation? Recall that to show that $\mathcal{ZK}$ is closed under sequential composition, the original definition of $\mathcal{ZK}$ was extended to consider *non-uniform* $\mathcal{PPT}$ adversaries

[1] On the other hand, it might still be plausible that the technique of [25] can be extended to give constant-round protocols w.r.t $\mathcal{PQT}$, when allowing the indistinguishability gap to be a polynomial (or even some *fixed* quasi-polynomial) function.

[2] In a related work [24], joint with Dustin Tseng we provide a simple proof for existence of concurrent $\mathcal{ZK}$ proofs with logarithmic round complexity.

[13]—in other words, in the context of $\mathcal{ZK}$ the notion of non-uniform $\mathcal{PPT}$ (for modeling adversaries) is more robust than simply $\mathcal{PPT}$. Additionally, security is guaranteed w.r.t a stronger class of adversaries. Of course, the extra price to pay is that all hardness assumptions now must hold also with respect to non-uniform $\mathcal{PPT}$.

In this paper we show that by considering an even stronger class of adversaries—namely $\mathcal{PQT}$—we get a notion that is even more robust; in particular, it is now possible to get constant-round concurrent $\mathcal{ZK}$ protocols. Again, this requires us to rely on hardness assumptions against $\mathcal{PQT}$, but this seems like a weak strengthening of traditional hardness assumptions (especially since the known attacks on traditional conjectured hard functions require subexponential time).

A note on plausible deniability. The notion of $\mathcal{ZK}$ is traditionally associated with *plausible deniability*—i.e., that the interaction leaves "no trace" which the verifier can use later to convince that the interaction took place. Intuitively, this holds since the verifier could have executed the simulator (on its self) to generate its view of the interaction. We mention, however, that since the traditional definition of $\mathcal{ZK}$ allows the simulator to have an arbitrary (polynomial) overhead with respect to the verifier (who's view it is supposed to simulate), the deniability guarantee offered by traditional $\mathcal{ZK}$ proofs is weak: consider for instance a verifier with a running-time of $t = 2^{40}$ computational steps, and a simulator with running-time, say, t^3; although 2^{40} is very feasible, 2^{120} seems like a stretch! The example is not hypothetic—the "tightest" concurrent $\mathcal{ZK}$ protocols [16, 23] indeed have a running-time of t^2 not counting the time need to emulate the verifier. Additionally, as demonstrated in [18], the traditional notion of $\mathcal{ZK}$ does not guarantee that the running-time of the simulator is (even polynomially) related to the running-time of the verifier in the view it is outputting, but rather the *worst-case* running-time of the verifier; this makes deniability even harder to argue.[3]

Nevertheless, in this respect, $\mathcal{ZK}$ w.r.t $\mathcal{PQT}$ provides even worse guarantees (as the overhead is now allowed to be quasi-polynomial).

1.2 Our techniques

The concurrent $\mathcal{ZK}$ protocols of Richardson and Kilian (RK) [25], Kilian and Petrank (KP) [16] and Prabhakaran, Rosen and Sahai(PRS) [23] rely on the same principal idea: provide the simulator with multiple possibilities (called "slots") to rewind the verifier. If a rewinding is successful, the simulator obtains a trapdoor that allows it to complete the execution that has been rewound. The RK simulator is "adaptive" and dynamically decides when and where to rewind, while making sure there are not too many recursive rewinding (which would result in a large running-time). On a high-level this is done by recursively

[3] In a recent work [19], joint with Pandey, Sahai and Tseng we also show how to obtain precise concurrent $\mathcal{ZK}$ proofs. Precise zero knowledge guarantees that the view of any verifier V can be simulated in time closely related to the *actual* (as opposed to the worst-case) time spent by V in the generated view.

invoking the simulator, but ensuring that the number of levels of the recursion stays small (in fact, constant). On the other hand the KP (and PRS) simulator is "oblivious"; the simulator has a fixed rewinding scheduling, thereby ensuring a fixed (and bounded) running-time. The core of the argument is then to show that every execution has a slot that is rewound at least once.

Our approach is based on the approach taken by RK. As RK, we consider an adaptive simulator that makes recursive calls to itself, while ensuring that the depth of the recursion stays small. Our actual simulation procedure is, however, quite different. On a high-level, our approach will perform a straight-line simulation until a "good" slot has been found, and then continue rewinding that slot until a trapdoor has been found. Thus, in contrast to the previous approach, we can not bound the worst-case running-time of our simulator, instead we are forced to bound the expected running-time of the simulator.

The benefit of our approach is that 1) it enables us to achieve perfect simulation, and 2) our analysis works no matter how many slots we have and what the depth of recursion is. In fact, we can achieve both of these properties while still guaranteeing the same expected running-time as RK—namely $O(m^{O(\log_r m)})$, where r is the number of slots. As a consequence, when applied to constant-round protocols (and considering a logarithmic recursive depth) we get a quasi-polynomial running time. As already mentioned, for this application, it is inherent to have an *expected* quasi-polynomial running-time.

1.3 Open questions

We have demonstrated that constant-round concurrent $\mathcal{ZK}$ is possible w.r.t $\mathcal{PQT}$ adversaries. Our protocol currently uses 10 communication rounds[4]. A natural open question is to either improve the round-complexity or to strengthen the 4-round lower bound of [17]. Another question is to investigate the possibility of using an even weaker (but still super-polynomial) model of computation. Rosen [26] shows that only languages in probabilistic sub quasi-polynomial time have 7-round concurrent black-box zero-knowledge arguments when adversaries are modelled as probabilistic sub quasi-polynomial time machines; thus, such protocols would require more than 7-rounds.

1.4 Organization

Definitions are found in Section 2. The proof of main theorem is contained in Sections 3 and 4. We give proof sketches for the remaining theorems in Section 5.

2 Definitions and Notations

We assume familiarity with the basic notions of an Interactive Turing Machine (ITM for brevity) and a protocol (in essence a pair of ITMs. Briefly, a protocol

[4] To obtain a 10 round protocol, we require non-interactive commitment schemes, which can be constructed from one-way-permutations. If we assume only existence of one-way functions, we get a 11-round protocol.

is pair of ITMs computing in turns. A round ends with the active machine either halting - in which case the protocol halts - or by sending a message m to the other machine, which becomes active with m as a special input.
We let $\mathcal{C}$ denote any class of functions.

2.1 Interactive Proofs and Arguments

Given a pair of interactive Turing machines, P and V, we denote by $\langle P, V\rangle(x)$ the random variable representing the (local) output of V when interacting with machine P on common input x, when the random input to each machine is uniformly and independently chosen.

Definition 1 ($T(\cdot)$-sound Interactive Proof System) *A pair of interactive machines $\langle P, V\rangle$ is called* $T(\cdot)$-sound interactive proof system *for a language L if machine V is polynomial-time and the following two conditions hold :*

- Completeness: *For every $x \in L$,* $\Pr[\langle P, V\rangle(x) = 1] = 1$
- Soundness: *For every $x \notin L$, and every interactive machine B,* $\Pr[\langle B, V\rangle(x) = 1] \leq \frac{1}{T(|x|)}$

In case that the soundness condition holds only with respect to a $T(n)$-bounded prover, the pair $\langle P, V\rangle$ is called an $T(\cdot)$-sound interactive argument.

$\langle P, V\rangle$ is an interactive proofs (interactive argument) w.r.t. $\mathcal{C}$ if for all $T(\cdot) \in \mathcal{C}$ the protocol is a $T(\cdot)$-sound interactive proof ($T(\cdot)$-sound interactive argument).

2.2 Indistinguishability

We rely on a generalization of the notion of indistinguishability [27], which considers $T(n)$-bounded distinguishers and require the indistinguishability gap to be smaller than $\frac{1}{\mathrm{poly}(T(n))}$.

Definition 2 (Strong $T(\cdot)$-indistinguishability[21]) *Let X and Y be countable sets. Two ensembles $\{A_{x,y}\}_{x\in X, y\in Y}$ and $\{B_{x,y}\}_{x\in X, y\in Y}$ are said to be indistinguishable in time $T(\cdot)$ over $x \in X$, if for every probabilistic "distinguishing" algorithm D with running time $T(\cdot)$ in its first input, and every $x \in X, y \in Y$ it holds that:*

$$|\Pr[a \leftarrow A_{x,y} : D(x, y, a) = 1] - \Pr[b \leftarrow B_{x,y} : D(x, y, b) = 1]| < \frac{1}{poly(T(|x|))}$$

Definition 3 (Computational indistinguishability w.r.t $\mathcal{C}$) *Let X and Y be countable sets. Two ensembles $\{A_{x,y}\}_{x\in X, y\in Y}$ and $\{B_{x,y}\}_{x\in X, y\in Y}$ are said to be indistinguishable w.r.t $\mathcal{C}$ over $x \in X$, if A, B are $q(\cdot)$-indistinguishable for every function $q(\cdot) \in \mathcal{C}$.*

2.3 Witness Indistinguishability

An interactive proof is said to be *witness indistinguishable* ($\mathcal{WI}$) if the verifier's view is "computationally independent" of the witness used by the prover for proving the statement—i.e. the view of the Verifier in the interaction with a prover using witness w_1 or w_2 for two different witnesses are indistinguishable.

Definition 4 (Witness-indistinguishability w.r.t $\mathcal{C}$) *Let $\langle P, V\rangle$ be an interactive proof system for a language $L \in \mathcal{NP}$. We say that $\langle P, V\rangle$ is* $\mathcal{C}$-witness-indistinguishable *for R_L, if for every probabilistic polynomial-time interactive machine V^* and for every two sequences $\{w_x^1\}_{x\in L}$ and $\{w_x^2\}_{x\in L}$, such that $w_x^1, w_x^2 \in R_L(x)$ for every $x \in L$, the probability ensembles $\{\mathrm{VIEW}_2[P(x, w_x^1) \leftrightarrow V^*(x,z)]\}_{x\in L, z\in\{0,1\}^*}$ and $\{\mathrm{VIEW}_2[P(x, w_x^2) \leftrightarrow V^*(x,z)]\}_{x\in L, z\in\{0,1\}^*}$ are computationally indistinguishable w.r.t $\mathcal{C}$ over $x \in L$.*

We say that the proof system is perfectly witness indistinguishable (Perfect-$\mathcal{WI}$) if the corresponding views are identically distributed.

2.4 Black-box concurrent zero-knowledge

Let $\langle P, V\rangle$ be an interactive proof for a language L. Consider a concurrent adversary verifier V^* that, given an input instance $x \in L$ interacts with m independent copies of P concurrently, without any restrictions over the scheduling of the messages in the different interactions with P. Let $\Big\{\mathrm{VIEW}_2[P(x,y) \leftrightarrow V^*(x,z)]\Big\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$ denote the random variable describing the view of the adversary V^* on common input x and auxiliary input z, in an interaction with P.

Definition 5 (Black-box concurrent zero-knowledge w.r.t $\mathcal{C}$:) *Let $\langle P, V\rangle$ be an interactive proof system for a language L. We say that $\langle P, V\rangle$ is* black-box concurrent zero-knowledge w.r.t $\mathcal{C}$ *if for every functions $q, m \in \mathcal{C}$, there exists a probabilistic algorithm $S_{q,m}$, such that for every concurrent non-uniform adversary V^* that on common input x and auxiliary input z has a running-time bounded by $q(|x|)$ and opens up $m(|x|)$ executions, $S_{q,m}(x,z)$ runs in time polynomial in $|x|$. Furthermore, the ensembles $\Big\{S_{q,m}(x,z)\Big\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$ and $\Big\{\mathrm{VIEW}_2[P(x,y) \leftrightarrow V^*(x,z)]\Big\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$ are computationally indistinguishable w.r.t $\mathcal{C}$ over $x \in L$.*

2.5 Other primitives

We informally define the other primitives we use in the construction of our protocols.

Special-sound proofs: A 3-round public-coin interactive proof for the language $L \in \mathcal{NP}$ with witness relation R_L is special-sound with respect to R_L, if for any two transcripts (α, β, γ) and $(\alpha', \beta', \gamma')$ such that the initial messages α, α' are the same but the challenges β, β' are different, there is a deterministic procedure to extract the witness from the two transcripts that runs in polynomial time. Special-sound $\mathcal{WI}$ proofs for languages in $\mathcal{NP}$ can be based on the existence of non-interactive commitment schemes, which in turn can be based on one-way permutations. Assuming only one-way functions, 4-round special-sound $\mathcal{WI}$ proofs for NP exists[5]. For simplicity, we use 3-round special-sound proofs in our protocol though our proof works also with 4-round proofs.

Proofs of knowledge: Informally an interactive proof is a proof of knowledge if the prover convinces the verifier not only of the validity of a statement, but also that it possesses a witness for the statement. If we consider computationally bounded provers, we only get a "computationally convincing" notion of a proof of knowledge (a.k.a *arguments of knowledge*)

3 Our Protocol and Simulator

3.1 Description of the protocol

Our concurrent $\mathcal{ZK}$ protocol (also used in [24]) is a slight variant of the precise $\mathcal{ZK}$ protocol of [20], which in turn is a modification of the Feige-Shamir protocol [10]. The protocol proceeds in the following two stages, on a common input statement $x \in \{0,1\}^*$ and security parameter n,

1. In Stage 1, the Verifier picks two random strings $s_1, s_2 \in \{0,1\}^n$, and sends their image $c_1 = f(r_1)$, $c_2 = f(r_2)$ through a one-way function f to the Prover. The Verifier sends $\alpha_1, \ldots, \alpha_r$, the first messages of r invocations of a $\mathcal{WI}$ special-sound proof of the fact that c_1 and c_2 have been constructed properly (i.e., that they are in the image set of f). This is followed by r iterations so that in the j^{th} iteration, the Prover sends $\beta_j \leftarrow \{0,1\}^{n^2}$, a random second message for the j^{th} proof and the Verifier sends the third message γ_j for the j^{th} proof.
2. In Stage 2, the Prover provides a WI proof of knowledge of the fact that either x is in the language, or (at least) one of c_1 and c_2 are in the image set of f.

More precisely, let $f : \{0,1\}^n \rightarrow \{0,1\}^n$ be a one-way function and let the witness relation $R_{L'}$, where $((x_1, x_2), (y_1, y_2)) \in R_{L'}$ if $f(x_1) = y_1$ or $f(x_2) = y_2$, characterize the language L'. Let the language $L \in \mathcal{NP}$. Protocol ConcZKArg for proving that $x \in L$ is depicted in Figure 1.

The soundness and the completeness of the protocol follows directly from the proof of Feige and Shamir [10]; in fact, the protocol is an instantiation of theirs.

[5] A 4-round protocol is special sound if a witness can be extracted from any two transcripts $(\tau, \alpha, \beta, \gamma)$ and $(\tau', \alpha', \beta', \gamma')$ such that $\tau = \tau$,$\alpha = \alpha'$ and $\beta \neq \beta'$.

(Intuitively, to cheat in the protocol a prover must "know" an inverse to either c_1 or c_2, which requires inverting the one-way function f.).

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 1:
V uniformly chooses $r_1, r_2 \in \{0,1\}^n$.
V $\rightarrow$ P: $c_1 = f(r_1), c_2 = f(r_2)$. r first messages $\alpha_1, \ldots, \alpha_r$ for $\mathcal{WI}$ special-sound proof of the statement. (called the start message)
either there exists a value r_1 s.t $c_1 = f(r_1)$
or there exists a value r_2 s.t $c_2 = f(r_2)$
The proof of knowledge is with respect to the witness relation R'_L
For $j = 1$ to r do
P $\rightarrow$ V: Second message $\beta_j \leftarrow \{0,1\}^{n^2}$ for j^{th} $\mathcal{WI}$ special-sound proof. (called the opening of slot j)
V $\rightarrow$ P: Third message γ_j for j^{th} $\mathcal{WI}$ special-sound proof. (called the closing of slot j)
Stage 2:
P $\leftrightarrow$ V: a perfect-$\mathcal{WI}$ argument of knowledge of the statement
either there exists values r'_1, r'_2 s.t either $c_1 = f(r'_1)$ or $c_2 = f(r'_2)$.
or $x \in L$
The argument of knowledge is with respect to the witness relation $R_{L \vee L'}(c_1, c_2, x) = \{(r'_1, r'_2, w) | (r'_1, r'_2) \in R_{L'}(c_1, c_2) \vee w \in R_L(x)\}$.

Fig. 1. Concurrent Perfect $\mathcal{ZK}$ argument for $\mathcal{NP}$

3.2 Description of the simulator

On a very high-level the simulation follows that of Feige and Shamir [10]: the simulator will attempt to rewind one of the special-sound proofs—each such proof, i.e. the challenge(β) and the response(γ) is called a slot. If the simulator gets two accepting proof transcripts, the special-soundness property allows the simulator to extract a "fake" witness r_i such that $c_i = f(r_i)$. This witness can later be used in the second phase of the protocol. We call an execution "solved" if a witness is extracted. More precisely, our simulation is defined recursively in the following manner.

On the recursive level ℓ, the simulator feeds random Stage 1 messages to V^* (Step 3). Whenever a slot s closes, S decides whether or not to rewind s depending on the number of new executions that started between the opening and the closing of s. If the number of executions is "small" (where small is defined based on the level ℓ), S begins rewinding the slot, i.e. S sends a new challenge β for slot s and recursively invokes itself on recursive level $\ell + 1$, and continues executing until one of the following happens:

1. *S is "stuck" at Stage 2 of an unsolved execution that started at level $\ell + 1$:* S halts and outputs fail.
2. *The* closing *message γ for slot s occurs*: S extracts a "fake" witness using the special-sound property and continues its simulation (on level ℓ).
3. *V^* aborts or starts "too many" executions:* S restarts its rewinding using a new challenge β for s. We show that S in expectation restarts $O(1)$ times because of this. (Intuitively this follows since during the execution at level ℓ, S only starts rewinding if V^* did not abort and only opened a "small" number of executions).
4. *S gets "stuck" at Stage 2 of an unsolved execution that started at level ℓ:* Again, S restarts its rewinding. We show that this case can happen at most $m - 1$ times, where m is the total number of executions.
5. *S gets "stuck" at Stage 2 of an unsolved execution that started at level $\ell' < \ell$:* S returns the view to level ℓ'.

In the unlikely event that S asks the same challenge β twice, S performs a brute-force search for the witness. Furthermore, to simplify the analysis of the running-time, the simulation is cut-off if it runs "too long" and S extracts witnesses for each execution using brute-force search.

The basic idea behind the simulation is similar to [25]: if we define "small" appropriately we can ensure that some slot of every execution is rewound and the expected running time is bounded. A first approach would be to ensure that at recursive level l at most $\frac{m}{r^\ell}$ executions start, and define "small" to be $\frac{m}{r^{\ell+1}}$, where m is the number of executions and r is the number of slots. Then, for every execution that started at level ℓ and completed r slots, S is guaranteed to rewind at least one slot. Furthermore, if we show that the expected number of rewindings of each slot is $O(m)$, then the expected running time of the simulator is at most $poly(m^{\log_r m})$; letting $r = 2$, the running time becomes $poly(m^{\log_2 m})$. However, to make sure that the simulator does not output fail, our analysis requires the simulator to be able to rewind at least two slots—in fact, we require that once the simulator reaches the last slot, it has already performed one rewinding. To ensure this, we make sure that at level ℓ, there are at most $\frac{m}{(r-1)^\ell}$ executions and define "small" to be $\frac{m}{(r-1)^{\ell+1}}$; now letting $r = 3$ we get a running-time of $poly(m^{\log_r m})$.

A formal description of our simulator can be found in Figure 4.2. We rely on the following notation.

- $\mathsf{d} = \lceil \log_{r-1} m \rceil$ will denote the maximum depth of recursion.
- slot (i, j) will denote slot j of execution i.
- A partial view h is defined to be good w.r.t (s, l), if in h, V^* does not abort on s and does not open more that $(r-1)^{\mathsf{d}-l}$ new executions after the opening of the s.
- W is a repository that stores the witness for each execution. The update W command extracts a witness from two transcripts of a slot (using the special-sound property). If the two transcripts are identical (i.e. the openings of the slot are the same), the simulator performs a brute-force search to extract a "fake" witness r_i s.t. $c_i = f(r_i)$ for $i \in \{1, 2\}$.

– R is a repository that stores the transcripts of slots of unsolved executions. Transcripts are stored in R when the simulator gets stuck in a rewinding (cases 4 and 5 mentioned in the high-level description).

4 Analysis of the Simulator

To prove correctness of the simulator, we show that the output of the simulator is correctly distributed and its expected running-time is bounded. We first prove in Claim 1 that the simulator never outputs fail. Using Claim 1, we show in Proposition 1 that the output distribution of the simulator is correct. In Proposition 2, we show that the expected running time of the simulation is at most $poly(m^d r^d)$. Throughout this proof we assume without loss of generality the adversary verifier V^* is deterministic (as it can always get its random coins as part of the auxiliary input).

4.1 Simulation never fails

Claim 1 *For every $x \in L$, $S^{V^*}(x, z)$ never outputs* fail.

Proof: Recall that $S^{V^*}(x, z)$ outputs fail only if $\text{SOLVE}_d^{V^*}(x, 0, , ,)$ outputs fail. Furthermore, SOLVE outputs fail at recursive level ℓ only if it reaches Stage 2 of an unsolved execution that started at level ℓ (i.e. only in Step 3 of SOLVE). Note that at recursive level ℓ, at most $(r-1)^{\mathsf{d}-\ell}$ executions are opened up. Hence, for all executions that start and complete $r-1$ slots at level ℓ, there is some slot, inside which have fewer than $(r-1)^{\mathsf{d}-(\ell+1)}$ executions opened; SOLVE must have rewound that slot "completely"—i.e. executed Step 5.d to obtain m good views without returning to a lower recursive level. Below, we show that whenever SOLVE rewinds a slot completely a witness is extracted and thus the proof of the claim follows.

Assume for contradiction that SOLVE fails to extract a witness after rewinding a particular slot. Let level ℓ and slot j of execution i be the first time this happens. This means at the end of Step 5.d, m good views are obtained and none of them contained a second transcript for slot j. Furthermore, in each such view, SOLVE got stuck only on unsolved executions that started at level ℓ (since otherwise SOLVE would have returned the view to the lower level). We now show that SOLVE can get stuck on the (at most $m-1$) other executions that started on level ℓ at most once; this contradicts the fact that m good views were obtained.

For every execution i' that SOLVE gets stuck on, both the opening and the closing of the last slot occurs inside the rewinding of slot (i, j); otherwise, SOLVE would have rewound one of the $r-1$ slots that occurred before the opening of slot (i, j) and by our assumption that l, i, j was the first "failed" slot, extracted a witness. Furthermore, the transcript of this slot enables SOLVE to never get stuck on execution i' again, since next time the last slot of execution i' closes a witness for that execution will be extracted. ∎

4.2 Indistinguishability of the simulation

Proposition 1 *The ensembles* $\{\mathsf{VIEW}_2[P(x,w) \leftrightarrow V^*(x,z)]\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$ *and* $\{S^{V^*}(x,z)\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$ *are identical.*

$\underline{\mathrm{SOLVE}_d^{V^*}(x,\ell,h_{initial},s,\mathsf{W},\mathsf{R})}$:

Let $h \leftarrow h_{initial}$.

Repeat forever:

1. If v is a Stage 2 verifier message of some execution, continue.
2. If V^* aborts or the number of executions that started after $h_{initial}$ in h exceeds $(r-1)^{\mathsf{d}-\ell}$, return h.
3. If the next scheduled message is a Stage 2 prover message for execution i and $\mathsf{W}(i) \neq \perp$, then use $\mathsf{W}(i)$ to complete the $\mathcal{WI}$ proof of knowledge; if $\mathsf{W}(i) = \perp$ and start message of execution i is in $h_{initial}$ return h, otherwise halt with output **fail**.
4. If the next scheduled message is a Stage 1 prover message for slot s', pick a random message $\beta \leftarrow \{0,1\}^{n^2}$. Append β to h. Let $v \leftarrow V^*(h)$.
5. Otherwise, if v is the $\mathsf{closing}$ message for $s' = \mathsf{slot}$ (i',j'), then update W with v (using R) and proceed as follows.
   (a) If $s = s'$, then return h.
   (b) Otherwise, if execution i' starts in $h_{initial}$, then return h.
   (c) Otherwise, if $\mathsf{W}(i') \neq \perp$ or the number of executions started inside s' exceeds $(r-1)^{\mathsf{d}-(\ell+1)}$, then continue.
   (d) Otherwise, let h' be the prefix of the history h where the prover message for s' is generated. Set $\mathsf{R}' \leftarrow \phi$.
   Repeat m times:
      i. Repeat $h^* \leftarrow \mathrm{SOLVE}_d^{V^*}(x,\ell+1,h',s',\mathsf{W},\mathsf{R}')$ until h^* is "good" w.r.t $(s',\ell+1)$.
      ii. If h^* contains an accepting proof transcript for slot s', extract witness for execution i' from h and h^* and update W.
      iii. Otherwise, if the last message in h^* is the $\mathsf{closing}$ message for the last slot of an execution that started in $h_{initial}$ return h^*.
      iv. Otherwise, add h^* to R'.

$\underline{S^{V^*}(x,z)}$:

1. Let $\mathsf{d} \leftarrow \lceil \log_{r-1} m \rceil$. Run $\mathrm{SOLVE}_d^{V^*}(x,0,,,,)$ and output whatever SOLVE outputs, with the following exception. If in the execution of $\mathrm{SOLVE}_d^{V^*}(x,0,,,,)$, it queries V^* more that 2^n times, proceed as follows: Let h denote the view reached in the "main-line" simulation (i.e., in the top-level of the recursion). Continue the simulation in a "straight-line" fashion from h by using a brute-force search to find a "fake" witness each time Stage 2 of an execution i is reached.

Fig. 2. Description of Simulator

Proof: Consider the following hybrid simulator $\tilde{S}^{V^*}$ that receives the real witness w to the statement x. $\tilde{S}^{V^*}$ on input x,w, and z proceeds just like S^{V^*}

in order to generate the prover messages in Stage 1, but proceeds as the honest prover using the witness w in order to generate messages in Stage 2 (instead of using the "fake" witness as S^{V^*} would have). Using the same proof as in Claim 1, we can show that $\tilde{S}^{V^*}(x,(w,z))$ never outputs fail. Furthermore, as the prover messages in Stage 1 are chosen uniformly and $\tilde{S}^{V^*}$ behaves like an honest prover in Stage 2. Therefore, we get:

Claim 2 *The ensembles* $\{\mathsf{VIEW}_2[P(x,w) \leftrightarrow V^*(x,z)]\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$ *and* $\{\tilde{S}^{V^*}(x,(w,z))\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$ *are identical.*

To show the proposition, it suffices to show that output distributions of $\tilde{S}^{V^*}$ and S^{V^*} are identical. This follows from the perfect-$\mathcal{WI}$ property of Stage 2 of the protocols, since the only difference between the simulators $\tilde{S}^{V^*}$ and S^{V^*} is the choice of witness used. For completeness, we provide a proof below.

Claim 3 *The ensemble* $\{\tilde{S}^{V^*}(x,(w,z))\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$ *is identical to* $\{S^{V^*}(x,z)\}_{x\in L, w\in R_L(x), z\in\{0,1\}^*}$

Proof: To prove the claim we will rely on the fact that the running time of the simulator is bounded. This holds since S stops executing SOLVE whenever it performs more than 2^n queries and continues the simulation in a straight-line fashion, extracting "fake" witnesses using brute-force search. Assume, for contradiction, that the claim is false, i.e. there exists a deterministic verifier V^* (we assume w.l.o.g that V^* is deterministic, as its random-tape can be fixed) such that the ensembles are not identical.

We consider several hybrid simulators, S_i for $i = 0$ to N, where N is an upper-bound on the running time of the simulator. S_i receives the real witness w to the statement x and behaves exactly like S, with the exception that Stage 2 messages in the first i proofs are generated using the honest prover strategy (and the witness w). By construction, $S_0 = \tilde{S}$ and $S_N = S$. Since, by assumption, the outputs of S_1 and S_N are not identically distributed, there must exist some j such that the output of S_j and S_{j+1} are different. Furthermore, since S_j proceeds exactly as S_{j+1} in the first j executions, and also the same in Stage 1 of the $j+1$'th execution, there exists a partial view v—which defines an instance $x' \in L \vee L'$ for Stage 2 of the $j+1$'th execution—such that outputs of S_j and S_{j+1} are not identical also conditioned on the event that S_j and S_{j+1} feed V^* the view v. Since the only only difference between the view of V^* in S_j and S_{j+1} is the choice of the witness used for the statement x' used in Stage 2 of the $j+1$'the execution, we contradict the perfect-$\mathcal{WI}$ property of Stage 2. ∎

∎

4.3 Running-time of S

We consider the hybrid simulator $\tilde{S}^{V^*}$ constructed in proof of Proposition 1. It follows by the same proof as in Claim 3 that the running time distributions of $\tilde{S}$ and S are identical. Therefore, it suffices to analyze the expected running time of $\tilde{S}$.

Proposition 2 *For all $x \in L, z \in \{0,1\}*$, and all V^* such that $V^*(x,z)$ opens up at most m executions,* $E[\mathsf{time}_{\tilde{S}^{V^*}(x,z)}] \leq poly(m^{\mathsf{d}} r^{\mathsf{d}})$

Proof: Recall that $\tilde{S}^{V^*}(x,z)$ starts running SOLVE, but in the event that SOLVE uses more than 2^n queries to V^*, it instead continues in a straight-line simulation using a brute-force search. By linearity of expectation, the expected running time of S is

$$poly(E[\# \text{ queries made to } V^* \text{ by SOLVE }]) \\ + E[\text{time spent in straight-line simulation}]$$

In Claim 4 below, we show that expected time spent in straight-line simulation is negligible. In Claim 5 we show that the expected number of queries made by $SOLVE$ to V^* is at most $m^{2(\mathsf{d}+1-\ell)}(2r)^{\mathsf{d}+1-\ell}$. The proof of the proposition follows.

Claim 4 *The expected time spent by $\tilde{S}^{V^*}$ in straight-line simulation is negligible.*

Proof: The straight-line simulation takes at most $poly(2^n)$ steps since it takes $O(2^n)$ steps to extract a "fake" witness. Recall that, SOLVE runs the brute-force search only if it picks the same challenge (β) twice. Since, SOLVE is cut-off after 2^n steps, it can pick at most 2^n challenges. Therefore, by the union bound, the probability that it obtains the same challenge twice is at most $\frac{2^n}{2^{n^2}}$. Thus, the expected time spent by S^{V^*} in straight-line simulation is at most $\frac{2^n}{2^{n^2}} poly(2^n)$, which is negligible. ■

Claim 5 *For all $x \in L, h, s, \mathsf{W}, \mathsf{R}$, $\ell \leq d$ such that $SOLVE_d^{V^*}(x, \ell, h, s, \mathsf{W}, \mathsf{R})$ never outputs* fails, $E[\#$ *queries by* $SOLVE_d^{V^*}(x, \ell, h, s, \mathsf{W}, \mathsf{R})] \leq m^{2(\mathsf{d}+1-\ell)}(2r)^{\mathsf{d}+1-\ell}$

Proof: We prove the claim by induction on ℓ. To simplify notation let $\alpha(\ell) = m^{2(\mathsf{d}+1-\ell)}(2r)^{\mathsf{d}+1-\ell}$. When $\ell = \mathsf{d}$ the claim follows since SOLVE does not perform any recursive calls and the number of queries made by SOLVE can be at most the total number of messages, which is mr.

Assume the claim is true for $\ell = \ell' + 1$. We show that it holds also for $\ell = \ell'$. Consider some fixed $x \in L$, h, s, W, R such that $\text{SOLVE}_d^{V^*}(x, \ell', h, s, \mathsf{W}, \mathsf{R})$ never outputs fails. We show that

$$E[\# \text{ queries by } \text{SOLVE}_d^{V^*}(x, \ell', h, s, \mathsf{W}, \mathsf{R})] \leq m^{2(d+1-\ell')} r^{d+1-\ell'} \\ = \alpha(\ell') = m^2(2r)\alpha(\ell'+1)$$

Towards this goal we introduce some additional notation. Given a view $\hat{h}$ extending the view h,

- Let $q_{\hat{s}}^{\ell'}(\hat{h})$ denote the probability that the view $\hat{h}$ occurs in the "main-line" execution of
  $\text{SOLVE}_d^{V^*}(x, \ell', h, s, \mathsf{W}, \mathsf{R})$ (i.e., starting on level ℓ) and that slot $\hat{s}$ opens immediately after $\hat{h}$.

– Let $\Gamma_{\hat{s}}$ denote the set of views such that $q_{\hat{s}}^{\ell'}(\hat{h}) > 0$.

We bound the number of queries made by $\mathrm{SOLVE}_d^{V^*}(x, \ell', h, s, \mathsf{W}, \mathsf{R})$ as the sum of the queries SOLVE makes on level ℓ', and the queries made by recursive calls. The number of queries made by SOLVE on level ℓ' is at most the total number of messages in an execution, i.e. mr. The number of queries made on recursive calls is computed by summing the queries made by recursive calls on over every slot $\hat{s}$ and taking expectation over every view $\hat{h}$ (such that $q_{\hat{s}}^{\ell'}(\hat{h}) > 0$). More precisely,

$$E[\#\text{ queries by }\mathrm{SOLVE}_d^{V^*}(x, \ell', h, s, \mathsf{W}, \mathsf{R})] \leq mr + \sum_{\hat{s}} \sum_{\hat{h} \in \Gamma_{\hat{s}}} q_{\hat{s}}^{\ell'}(\hat{h}) E_{\hat{s}}(\hat{h})$$

where $E_{\hat{s}}(\hat{h})$ denotes the expected number of queries made by SOLVE from the view $\hat{h}$ on $\hat{s}$. There are two steps involved in computing $E_{\hat{s}}(\hat{h})$. The first step involves finding the expected number of times SOLVE is run on a slot and the second step using the induction hypothesis computing a bound for $E_{\hat{s}}(\hat{h})$.

Step 1: Given a view $\hat{h}$ from where slot $\hat{s}$ opens, let p^ℓ denote the probability that SOLVE rewinds slot $\hat{s}$ from $\hat{h}$, i.e. p^ℓ is the probability that in the simulation from $\hat{h}$ at level ℓ, V^* completes $\hat{s}$ with an accepting proof while opening fewer than $(r-1)^{\mathsf{d}-\ell'}$ new executions within the slot $\hat{s}$. Let y^ℓ denote the probability that when executing SOLVE at level ℓ from $\hat{h}$, V^* either aborts or opens more than $(r-1)^{\mathsf{d}-\ell'}$ new executions in slot $\hat{s}$. We clearly have that $p^\ell \leq 1 - y^\ell$ (note that equality does not necessarily hold since SOLVE might also return to a lower recursive level). Furthermore, it holds that $y^\ell = y^{\ell+1}$. This follows since SOLVE generates random Stage 1 messages, and uses the same (real) witness to generate Stage 2 messages, independent of the level of the recursion; additionally, since by Claim 4.1, SOLVE never halts outputting fail, we conclude that the view of V^* in the "main-line" simulation by SOLVE on level l is identically distributed to its view on level $l+1$.

Therefore, the expected number of times SOLVE recursively executes $\hat{s}$ at level $\ell+1$, before obtaining a good view, is at most $\frac{1}{1-y^{\ell+1}} = \frac{1}{1-y^\ell} \leq \frac{1}{p^\ell}$. Using linearity of expectation, the expected number of times SOLVE executes $\hat{s}$ before obtaining m good views is at most $\frac{m}{p^\ell}$. Since, SOLVE rewinds $\hat{s}$ from $\hat{h}$ only with probability p^ℓ, the expected number of recursive calls to level $\ell+1$ from $\hat{h}$ is at most $p^\ell \frac{m}{p^\ell} = m$.

Step 2: From the induction hypothesis, we know that the expected number of queries made by SOLVE at level $\ell'+1$ is at most $\alpha(\ell'+1)$. Therefore, if SOLVE is run u times on a slot, the expected total number of queries made by SOLVE is bounded by $u\alpha(\ell'+1)$. We conclude that

$$\begin{aligned}
E_{\hat{s}}(\hat{h}) &\leq \sum_{u \in \mathbf{N}} Pr[u \text{ recursive calls are made by SOLVE from } \hat{h}] u\alpha(\ell'+1) \\
&= \alpha(\ell'+1) \sum_{u \in \mathbf{N}} u \cdot Pr[u \text{ recursive calls are made by SOLVE from } \hat{h}] \\
&\leq m\alpha(\ell'+1)
\end{aligned}$$

Therefore, $E[\#$ queries by $\mathrm{SOLVE}_d^{V^*}(x, \ell', h, s, \mathsf{W}, \mathsf{R})] \leq$

$$mr + \sum_{\hat{s}} \sum_{\hat{h} \in \Gamma_{\hat{s}}} q_{\hat{s}}^{\ell'}(\hat{h}) E_{\hat{s}}(\hat{h}) \leq mr + \sum_{\hat{s}} m\alpha(\ell'+1) \sum_{\hat{h} \in \Gamma_{\hat{s}}} q_{\hat{s}}^{\ell'}(\hat{h})$$

$$\leq mr + \sum_{\hat{s}} m\alpha(\ell'+1) \leq mr + (mr)m\alpha(\ell'+1) \leq \alpha(\ell')$$

This completes the induction step and concludes the proof of Claim 2. ∎

∎

4.4 Concluding the proof of Theorem 1 (and Theorem 4)

Using $r = 3$, we get by Proposition 2 that the expected running-time of S is $poly(m^{log_2 m})$, and by Proposition 1 that its output is correctly distributed. This concludes the proof of Theorem 1. We also remark that the proof of Theorem 4 is directly obtained by instead relying on an n^{ϵ}-rounds version of the protocol.

5 Proving the other theorems

Due to lack of space, we provide only proof ideas for the remaining theorems. The complete proofs will be contained in the full version.
Proof idea of Theorem 2: To prove the theorem, we rely on a slight variant of the $\mathcal{ZK}$ proof of [18, 20] (which is an instantiation of the protocol of [23]); the protocol is described in Figure 3. We assume the existence of honest-verifier $\mathcal{ZK}$ proofs that are secure w.r.t $\omega(\mathcal{PQT})$. Such proofs exists if one-way functions that are secure w.r.t $\omega(\mathcal{PQT})$ exists. Furthermore, we require constant round statistically hiding commitments that are computationally binding w.r.t $\mathcal{PQT}$ adversaries. Such commitment schemes can be constructed from collision resistant hash functions that are secure w.r.t $\mathcal{PQT}$ [7, 15]. The simulator and the proof of indistinguishability is essentially similar to Section 3.2. However, to bound the running-time of the simulator we require the Stage 2 of the protocol to satisfy the honest-verifier $\mathcal{ZK}$ property w.r.t. $\omega(\mathcal{PQT})$.
Proof idea of Theorem 3: The protocol is obtained by using a computational $\mathcal{WI}$ protocol w.r.t $\mathcal{PQT}$ instead of the perfect $\mathcal{WI}$ protocol in Stage 2 described in Section 3.1, which can be constructed based on the existence of OWF secure for $\mathcal{PQT}$. The simulator and the analysis from Section 3.2 essentially works for this protocol too, except that to show indistinguishability we use the computational $\mathcal{WI}$ property of the protocol in Stage 2.
Proof idea of Theorems 5 and 6: Our constructions are essentially identical to the protocols in [18, 20]. On a high level, the protocols show how to recast the $\mathcal{ZK}$ protocols for Graph Non-Isomorphism and Quadratic Non-Residuosity into the Feige-Shamir paradigm, after which we can rely on the same proof as in the previous section. Finally, Theorem 6 is directly obtained by relying on an $r = n^{\epsilon}$-rounds version of the protocol.

PROTOCOL COMPZKPROOF

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 1:

V uniformly chooses $\overline{r} = r_1, r_2, ..., r_n \in \{0,1\}^n$, $s \in \{0,1\}^{poly(n)}$.

V $\rightarrow$ P: $c =$ COM$(\overline{r}; s)$, where COM is a statistically hiding commitment, which has the property that the commiter must communicate at least m bits in order to commit to m strings.

V $\rightarrow$ P: r first messages $\alpha_1, \ldots, \alpha_r$ for $\mathcal{WI}$ special-sound proofs of the statement. (called the start message)
there exists values $\overline{r}', s'$ s.t $c =$ COM$(\overline{r}'; s')$
The proof of knowledge is with respect to the witness relation $R'_L(c) = \{(v, s) | c = \text{COM}(v; s)\}$.

For $j = 1$ to r do

P $\rightarrow$ V: Second message $\beta_j \leftarrow \{0,1\}^{n^2}$ for j^{th} $\mathcal{WI}$ special-sound proof. (called the opening of slot j)

V $\rightarrow$ P: Third message γ_j for j^{th} $\mathcal{WI}$ special-sound proof. (called the closing of slot j)

Stage 2:

P $\leftrightarrow$ V: P and V engage in n parallel executions of the GMW's (3-round) Graph 3-Coloring protocol, where V uses the strings $r_1, .., r_n$ as its challenges:

1. P $\rightarrow$ V: n (random) first messages of the GMW proof system for the statement x.
2. V $\leftarrow$ P: V decommits to $\overline{r} = r_1, .., r_n$.
3. P $\rightarrow$ V: For $i = 1..n$, P computes the answer (i.e., the 3rd message of the GMW proof system) to the challenge r_i and sends all the answers to V.

Fig. 3. Computational $\mathcal{ZK}$ Proof for $\mathcal{NP}$

6 Acknowledgements

We are very grateful to both Joe Kilian and Alon Rosen for insightful and helpful conversations.

References

1. B. Barak and Y. Lindell. Strict Polynomial-Time in Simulation and Extraction. In *34th STOC*, pages 484–493, 2002.
2. J.D. Benaloh. Cryptographic Capsules: A disjunctive primitive for interactive protocols. In *Crypto86*, Springer LNCS 263, pages 213–222, 1987.
3. G. Brassard, D. Chaum and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *JCSS*, Vol. 37, No. 2, pages 156–189, 1988. Preliminary version by Brassard and Crépeau in *27th FOCS*, 1986.
4. R. Canetti, O. Goldreich, S. Goldwasser and S. Micali. Resettable Zero-Knowledge. In *32nd STOC*, pages 235–244, 2000.

5. R. Canetti, J. Kilian, E. Petrank and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires (almost) Logarithmically Many Rounds. *SIAM Jour. on Computing*, Vol. 32(1), pages 1–47, 2002.
6. I. Damgård. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. In *EuroCrypt2000*, LNCS 1807, pages 418–430, 2000.
7. I. Damgård, T. Pedersen and B. Pfitzmann. On the Existence of Statistically Hiding Bit Commitment Schemes and Fail-Stop Signatures. In *Crypto93*, Springer-Verlag LNCS Vol. 773, pages 250–265, 1993.
8. C. Dwork, M. Naor and A. Sahai. Concurrent Zero-Knowledge. In *30th STOC*, pages 409–418, 1998.
9. C. Dwork and A. Sahai. Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints. In *Crypto98*, Springer LNCS 1462, pages 442–457, 1998.
10. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Crypto86*, Springer LNCS 263, pages 181–187, 1987
11. O. Goldreich and A. Kahan. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Jour. of Cryptology*, Vol. 9, No. 2, pages 167–189, 1996.
12. O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *JACM*, Vol. 38(1), pp. 691–729, 1991.
13. O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Jour. of Cryptology*, Vol. 7, No. 1, pages 1–32, 1994.
14. S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Jour. on Computing*, Vol. 18(1), pp. 186–208, 1989.
15. S. Halevi and S. Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. In *Crypto96*, Springer LNCS 1109, pages 201–215, 1996.
16. J. Kilian and E. Petrank. Concurrent and Resettable Zero-Knowledge in Poly-logarithmic Rounds. In *33rd STOC*, pages 560–569, 2001.
17. J. Kilian, E. Petrank and C. Rackoff. Lower Bounds for Zero-Knowledge on the Internet. In *39th FOCS*, pages 484–492, 1998.
18. S. Micali, R. Pass. Local Zero Knowledge. In *STOC'06*.
19. O. Pandey, R. Pass, A. Sahai, D. Tseng and M. Venkitasubramaniam. Precise Concurrent Zero-Knowledge. *Manuscript*
20. R. Pass. A Precise Computational Approach to Knowledge. PhD thesis, MIT,2006.
21. R. Pass. Simulation in Quasi-Polynomial Time and Its Application to Protocol Composition. In *EuroCrypt2003*, Springer LNCS 2656, pages 160–176, 2003.
22. R. Pass and A. Rosen. Bounded-Concurrent Two-Party Computation in Constant Number of Rounds. In *44th FOCS*, pages 404–413, 2003
23. M. Prabhakaran, A. Rosen and A. Sahai. Concurrent Zero-Knowledge with Logarithmic Round Complexity. In *43rd FOCS*, pages 366–375, 2002.
24. R. Pass, D. Tseng and M. Venkitasubramaniam. Concurrent Zero Knowledge: A Simplified Proof. In Submission.
25. R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *EuroCrypt99*, Springer LNCS 1592, pages 415–431, 1999.
26. A. Rosen. A note on the round-complexity of Concurrent Zero-Knowledge. In *Crypto2000*, Springer LNCS 1880, pages 451–468, 2000.
27. S. Goldwasser, S. Micali. Probabilistic Encryption. JCSS 28(2), pages 270–299, 1984.