Resettable Zero-Knowledge*

Ran Canetti[†] Oded Goldreich[‡] Shafi Goldwasser[§] Silvio Micali[¶]

June 7, 2000

Abstract

We introduce the notion of Resettable Zero-Knowledge (rZK), a new security measure for cryptographic protocols which strengthens the classical notion of zero-knowledge. In essence, an rZK protocol is one that remains zero knowledge even if an adversary can interact with the prover many times, each time resetting the prover to its initial state and forcing it to use the same random tape. All known examples of zero-knowledge proofs and arguments are trivially breakable in this setting. Moreover, by definition, all zero-knowledge proofs of knowledge are breakable in this setting. Under general complexity assumptions, which hold for example if the Discrete Logarithm Problem is hard, we construct:

- Resettable Zero-Knowledge proof-systems for NP with non-constant number of rounds.
- Five-round Resettable Witness-Indistinguishable proof-systems for NP.
- Four-round Resettable Zero-Knowledge arguments for NP in the *public key model*: where verifiers have fixed, public keys associated with them.

In addition to shedding new light on what makes zero knowledge possible (by constructing ZK protocols that use randomness in a dramatically weaker way than before), rZK has great relevance to applications. Firstly, rZK protocols are closed under parallel and concurrent execution and thus are guaranteed to be secure when implemented in fully asynchronous networks, even if an adversary schedules the arrival of every message sent so as to foil security. Secondly, rZK protocols enlarge the range of physical ways in which provers of ZK protocols can be securely implemented, including devices which cannot reliably toss coins on line, nor keep state between invocations. (For instance, because ordinary smart cards with secure hardware are resettable, they could not be used to implement securely the provers of classical ZK protocols, but can now be used to implement securely the provers of rZK protocols.)

Keywords: Zero-Knowledge, Concurrent Zero-Knowledge, Public-Key Cryptography, Witness-Indistinguishable Proofs, Smart Cards, Identification Schemes, Commitment Schemes, Discrete Logarithm Problem.

*A subset of this work is included in patent application [31].

[†]IBM Research, Yorktown Height NY 10598; canetti@watson.ibm.com

[‡]Dept. of Computer Science, Weizmann Institute of Science, Rehovot, ISRAEL; oded@wisdom.weizmann.ac.il.
Supported by MINERVA Foundation, Germany.
[§]Laboratory for Computer Science, MIT, Cambridge, MA02139; shafi@theory.lcs.mit.edu

[¶]Laboratory for Computer Science, MIT, Cambridge, MA02139; silvio@theory.lcs.mit.edu

Contents

1 Introduction

The notion of a zero-knowledge interactive proof was put forward and first exemplified by Goldwasser, Micali and Rackoff [32]. The generality of this notion was demonstrated by Goldreich, Micali and Wigderson [28], who showed that any NP-statement can be proven in zero-knowledge, provided that commitment schemes exist.[1] Subsequently, related notions have been proposed; in particular, zero-knowledge arguments [9], witness indistinguishability [19], and zero knowledge proofs of knowledge [32, 43, 18, 1]. By now, zero-knowledge is the accepted way to define and prove security of various cryptographic tasks; in particular, as proposed by Fiat and Shamir [20], it provides the basis for many proofs of identity.

A basic question about zero-knowledge. A zero-knowledge proof of a non-trivial language is possible only if the Prover tosses coins.[2] But:

Is zero-knowledge possible when the prover uses the same *coins in more than one execution?*

For zero-knowledge proofs of knowledge (and thus for all proofs of identity à la Fiat-Shamir [20]), *by definition*, the answer is NO: if the verifier can force the prover to use the same coins for a polynomial number of executions, then even the honest verifier can easily extract the very same secret which the prover is claiming knowledge of.[3]

For zero-knowledge proofs (of language membership), the answer also appeared to be negative: all known examples of zero-knowledge proofs (including the 3-Coloring protocol of [28]) are trivially breakable if the prover is "reset" (to his initial state) and forced to use the same coins in future interactions, even if these interactions are with the honest verifier.

Example. For instance, to prove that $z = x^2 \bmod n$ is quadratic residue mod n, in [32] the following basic protocol is repeated: the prover randomly chooses $r \in Z_n^*$ and sends $r^2 \bmod n$ to the verifier; the verifier sends a random bit b to the prover; and the prover sends back r if $b = 0$, and $xr \bmod n$ if $b = 1$. Assume now that the prover is forced to execute twice with the same coins r the basic protocol. Then, by sending $b = 0$ in the first execution and $b = 1$ in the second execution, the verifier learns both r and xr and thus trivially extract x, a square root of z mod n.

A New Notion. In this paper we extend the classical notion of zero-knowledge by introducing the notion of *Resettable Zero-Knowledge* (rZK for short).[4] In essence, a rZK proof is a zero-knowledge proof in which a verifier learns nothing (except for the verity of a given statement) even if he can interact with the prover polynomially many times (in an "interleaved manner"), each time restarting the prover with the same configuration and coin tosses.

In other words, a polynomial-time verifier learns nothing extra even if it can "clone" the prover, with the same initial configuration and random tape, as many times as it pleases, and then interact with these clones in any order and manner it wants. In particular, it can start a second interaction in the middle of a first one, and thus choose to send a message in the second interaction as a function of messages received in the first. We stress that, in each of these *interleaved* interactions, the prover (i.e., each prover clone) is not aware of any other interaction, nor of having been cloned.

[1] Or, equivalently [40, 35], that one-way functions exist.

[2] Zero-knowledge proofs in which the prover is deterministic exist only for $\mathcal{BPP}$ languages (cf., [29]).

[3] For instance, in [20] it suffices to repeat the protocol twice with the same prover-coins to be able to extract the prover's secret.

[4] In a preliminary version of this work [24], the same notion was called *rewind zero-knowledge* and *interleaved zero-knowledge*.

Resetability can be incorporated in the various variants of zero knowledge. In particular in this work we will pay close attention to *Resettable Zero-Knowledge proofs*, *Resettable Zero-Knowledge arguments*, and *Resettable Witness-Indistinguishable Proofs* (rWI for short).

Informally, in all of the above cases (i.e., ZK proofs, arguments, and WI proofs) the security requirement is maintained even if the prover is forced to use the same coin tosses in repeated and interleaved executions.

The Importance of the New Notion. Resettable zero knowledge sheds new light on what it is that make secure protocol possible. In particular, such protocols make a much weaker use of randomness than previously believed necessary. Moreover, resettable zero knowledge is a powerful abstraction which yields both theoretical and practical results in a variety of settings: In particular,

- rZK increases the number of physical ways in which zero-knowledge proofs may be implemented, while guaranteeing that security is preserved.

  As we have said, previous notions of zero knowledge were insecure whenever an attacker could reset the device implementing the prover to its initial conditions (which include his random tape). For example, this class of implementations includes ordinary smart cards. In fact, without a built-in power supply or without a re-writable memory that is not only tamper-proof, but also non-volatile, these cards can be reset by disconnecting and reconnecting the power supply.

- rZK proofs, rWI proofs and rZK arguments are guaranteed to preserve security when executed *concurrently* in an asynchronous network like the Internet.

- rZK proofs, rWI proofs and rZK arguments provide much more secure identification (ID) schemes; that is, ID schemes that preserve security under circumstances as above.

New Results. We show that, under standard complexity assumptions, Resettable Zero-Knowledge exists. Let us quickly state our assumptions and main results.

ASSUMPTIONS. All our protocols are based on the existence of certain types of commitment schemes. Some of these schemes may be implemented under traditional complexity assumptions, such as the hardness of the Discrete Log Problem (DLP), and for some we use stronger assumptions such that the existence of strong trapdoor claw-free pairs of permutations.[5] For the purposes of the current write-up, we renounce to some generality, and rely directly on two forms of the DLP assumption: Informally, denoting by $DLP(k)$ the task of solving DLP for instances of length k, we have

Strong DLP Assumption: $DLP(k)$ is not solvable in time 2^{k^ϵ}, for some $\epsilon > 0$.

Weak DLP Assumption: DLP is not solvable in polynomial time.

MAIN RESULTS. We prove the following theorems:

Theorem 1: *Under the weak DLP assumption, there is a (non-constant round)* rZK *proof for NP.*

[5] "Strong" refers to those in which the claw-free property should hold also with respect to sub-exponential-size circuits (i.e., circuits of size 2^{n^ϵ}, where n is the input length and $\epsilon > 0$ is fixed), rather than only with respect to polynomial-size circuits, and "trapdoor" refers to the fact that these pairs that can be generated along with auxiliary information which allows to form (random) claws.

Theorem 2: *Under the weak DLP assumption, there is a constant-round* rWI *proof for NP.*

Theorem 3: *Under the strong DLP assumption, there is a constant-round* rZK *argument for NP in the Public-Key Model.*

By the public-key model, we mean that a verifier has a public key that has been registered —i.e., fixed— prior to his interaction with the prover. We stress that we only assume that public-keys can be registered in the sense that it has been posted. Registration does not have to include any interaction with a trusted system manager that may verify properties of the registered public-key. We also stress that the prover does not need a public key.[6] (As we shall point out later on, this quite standard model of fixing a key before interaction starts can be further relaxed.) For a more detailed discussion of this model see Section 6.1.

1.1 Resettable vs. Concurrent ZK

A WEAKER NOTION. In the past few years, considerable attention has been devoted to *concurrent* zero-knowledge (cZK) protocols. In essence, these are ZK proofs that withstand malicious verifiers who can interact several times with the prover, in an "interleaved way," about the same theorem. In each interaction, however, the prover will use a "fresh" random tape. (This model was first considered in [17].)

Concurrent ZK is a weaker notion than resettable ZK, because in a rZK protocol, a malicious verifier may not only interact several times with the prover in an interleaved way, but also enforce that, in each such interaction, the prover has the same initial configuration (and thus uses the same random tape).

SOME PRIOR cZK PROTOCOLS. Concurrent ZK protocols have been suggested by Dwork, Naor and Sahai [15], assuming that a certain level of synchronization is guaranteed: the so-called *timing assumption*. Under this assumption, (1) there are a-priori known bounds on the delays of messages with respect to some ideal global clock, and (2) each party uses a local clock whose rate is within a constant factor of the rate of the ideal clock. Under the timing assumption (and some standard intractability assumption), constant-round, ZK arguments for NP were presented in [15]. In a later paper, Dwork and Sahai [14] show how the push up the use of the timing assumption to a pre-processing protocol, to be executed before the concurrent executions of protocols. More recent protocols by Richardson and Kilian [41] and Kilian and Petrank [38] do not use the timing assumption, however their protocols are not constant-round. We stress that none of these concurrent ZK protocols is rZK.

1.1.1 rZK vs. cZK in the standard model

In the standard (non public-key) model, we construct our resettable ZK protocols based on concurrent ZK ones, and in particular the cZK protocol of Richardson and Kilian [41]. (Therefore, in this model, our constructions do not result in better cZK protocols.)

CONSTRUCTIONS OF rZK PROOF SYSTEMS. We actually present two constructions of rZK protocols for NP: one "by reduction" and a "direct" one. Our first construction consists of two steps. In

[6] Note that the fact that only the verifier requires a public key is especially suitable when extending rZK proofs to rZK proofs of identity. In the latter case, in fact, the verifier usually guards a resource and needs to identify the identity of the user (the prover) attempting to use the resource. In this scenario, it is reasonable to expect (the few) verifiers to have public key accessible by all users, and it useful that the (many) provers may implemented by cheap, resettable devices which do not have any registered public keys.

a first step, we provide a transformation mapping any cZK protocol satisfying a special condition (the *admissible* cZK *protocols*) into rZK protocols. In the second step, we show how to transform the cZK protocol of [41] into an admissible one.

Our direct construction also consists of two steps. In the first step, we provide a constant-round resettable witness-indistinguishable (rWI) protocol for NP (a step of independent interest). In the second step, we properly combine our rWI protocol with the cZK protocol of [41] so as to obtain an rZK protocol for NP. The combined protocol inherits the round complexity of the [41] protocol, and thus is not constant-round.

Lower bounds for rZK proof systems. Demonstrating limitations on the ability to construct cZK protocols, Kilian et.al. [39] show that four-round cZK protocols whose security is proved via black-box simulation exist only for languages in BPP. Rosen [42] has recently extended this result to seven-round protocols. Since any rZK protocol is also cZK, these lower bounds apply to rZK protocols as well.

1.1.2 rZK vs. cZK in the public-key model

In the public-key model, our rZK protocols are built in totally novel ways (i.e., are not based on prior cZK protocols), and indeed provide new implications for concurrent zero knowledge. In particular, Theorem 3 yields the following corollary.

Corollary 4: *Under the strong DLP assumption, there exists a constant-round, concurrent ZK arguments for NP in the public-key model.*

This result is important whenever ZK protocols are to be played over asynchronous networks (like the Internet), because in such networks it is easy for a malicious verifier to run many ZK protocols at once in an interleaved way (thus making concurrent executions an eminent threat), and because the number of rounds is an important resource for internet protocols.

Moreover, the above result is widely implementable, because the public-key model is ubiquitous whenever cryptography is used (specifically, it underlies any public-key encryption or digital signature scheme). As the public-key model is both simpler and more realistic than the timing assumptions of [15, 14], we believe that the constant-round cZK protocol of Corollary 4 is preferable to the constant-round one of [15, 14]. Indeed, even if one thinks of the public-key model as a form of preprocessing, Corollary 4 provides an alternative to Dwork and Sahai's protocol which is based on pre-processing with the timing assumption. For further comparison see Section 6.1.

Another constant-round cZK argument for NP (but not an rZK one!) has been independently provided by Damgård[11, 12], but his protocol relies on a stronger public-key model: one in which a trusted center generates the (secret key, public key) pairs (i.e the soundness of the protocols depends on the trusted center keeping the secret key confidential). Alternatively, this trusted center can be replaced by a pre-processing interactive protocol between users (setting up their public-keys) and certification authorities.

1.2 Implications of rZK for Proofs of Identity

Fiat and Shamir in [20] introduced a paradigm for ID schemes based on the notion of Zero Knowledge Proof of Knowledge. In essence, a prover identifies himself by convincing the verifier of knowing a given secret (e.g., in [20], of knowing a square root of a given square mod n). All subsequent ID schemes follow this paradigm, and are traditionally implemented by the prover being a smart card (as suggested in [20]). However, Zero Knowledge Proof of Knowledge are impossible in a resettable setting (i.e., they exist only in a trivial sense[7]), and thus *all* Fiat-Shamir like ID schemes fail to be secure whenever the prover is resettable.

Instead, an alternative paradigm emerges for constructing ID schemes so that the resulting schemes are secure when the identification is done by a device which can be reset to its initial state such as a smart card. The new paradigm consists of viewing the *ability to convince the verifier that a fixed input is in a "hard" NP-language* as a proof of identity, and employing an rZK proof to do so. Further elaboration on the notion and the construction of Resettable Proofs of Identity will appear in a separate paper.

Organization. Section 2 defines the notions of rZK and rWI. Section 3 provides a general method for transforming a certain class of proof systems designed for the concurrent setting into resettable ones. Sections 4 and 5 use the transformation of Section 3 to present rWI and rZK proof systems for NP, respectively. Sections 2 through 5 concentrate on the standard model for interactive proofs. Section 6 presents the public key model and describes our results in this model.

2 The Notions of rWI and rZK

2.1 Preliminaries

We shortly review some basic notions and point the reader to more comprehensive sources on these notions.

Interactive proof systems. Throughout this paper we consider interactive proof systems [32] in which the designated prover strategy can be implemented in probabilistic polynomial-time given an adequate auxiliary input. Specifically, we consider interactive proofs for languages in $\mathcal{NP}$ and thus the adequate auxiliary input is an NP-witness for the membership of the common input in the language. Also, whenever we talk of an interactive proof system, we mean one in which the error probability is a negligible function of the length of the common input (i.e., for every polynomial p and all sufficiently long x's, the error probability on common input x is smaller than $1/p(|x|)$). Actually, we may further restrict the meaning of the term 'interactive proof system' by requiring that inputs in the language are accepted with probability 1 (i.e., so-called *perfect completeness*).

Argument systems. Likewise, when we talk of computationally-sound proof systems (a.k.a arguments) [9] we mean ones with perfect completeness in which it is infeasible to cheat with non-negligible probability. Specifically, for every polynomial p and all sufficiently large inputs x not in the language, every circuit of size $p(|x|)$ (representing a cheating prover strategy) may convince the verifier to accept only with probability less than $1/p(|x|)$.

Zero-knowledge. We adopt the basic paradigm of the definition of zero-knowledge [32]: The output of every probabilistic polynomial-time adversary which interacts with the designated prover on a common input in the language, ought to be simulatable by a probabilistic polynomial-time machine (which interacts with nobody), called the simulator. We mention that the simulators in

[7] It can be shown that if, on input x, one can provide an rZK proof of knowledge of y so that (x, y) is in some polynomial-time recognizable relation, then it is possible given x to find such a y in probabilistic polynomial-time. Thus, such a proof of knowledge is useless, since by definition (of knowledge) anybody who gets input x knows such a y.

Sections 3 and 5 run in *strict* polynomial-time, whereas those in Section 6.1 run in *expected* poly nomial-time. (As Section 6 focuses on constant-round resettable zero-knowledge systems, *expected* polynomial-time simulation seems unavoidable: recall that it is not known whether constant-round zero-knowledge proofs for $\mathcal{NP}$ exists, when one insists on strictly polynomial-time simulators (rather than expected polynomial-time ones); See [25, 22].)

Witness indistinguishable proof systems [19]. Loosely speaking, these are proof systems in which the prover is a probabilistic polynomial-time machine with auxiliary input (typically, an NP-witness), having the property that interactions in which the prover uses different "legitimate" auxiliary-inputs are computationally indistinguishable from each other. Recall that any zero-knowledge proof system is also witness indistinguishable, and there are witness indistinguishable proof systems that are not zero-knowledge.

2.2 Definition of rWI and rZK

Given a specified prover P, a common input x and an auxiliary input y to P (e.g., y may be an NP-witness for x being in some NP-language), we consider polynomially-many interactions with the deterministic prover strategy $P_{x,y,\omega}$ determined by uniformly selecting and fixing P's coins, denoted ω. That is, ω is uniformly selected and fixed once and for all, and the adversary may invoke and interact with many instances of $P_{x,y,\omega}$. An interaction with an instance of $P_{x,y,\omega}$ is called a session. It is stressed that $P_{x,y,\omega}$'s actions in each session are oblivious of other sessions (since $P_{x,y,\omega}$ mimics the "single session strategy" P); nonetheless, the actions of the adversary may depend on other sessions.

We consider two variants of the model, and prove their equivalence. In the basic variant, a session must be terminated (either completed or aborted) before a new session can be initiated by the adversary. In the interleaving variant, this restriction is not made and so the adversary may concurrently initiate and interact with $P_{x,y,\omega}$ in many sessions. A suitable formalism must be introduce in order to support these concurrent executions. For simplicity, say that the adversary prepend a session-ID to each message it sends, and a distinct copy of $P_{x,y,\omega}$ handles all messages prepended by each fixed ID. Note that in both variants, the adversary may repeat in the current session the same messages sent in a prior session, resulting in an identical prefix of an interaction (since the prover's randomness is fixed). Furthermore, by deviating in the next message, the adversary may obtain two different continuations of the same prefix of an interaction. Viewed in other terms, the adversary may "effectively rewind" the prover to any point in a prior interaction, and carry-on a new continuation (of this interaction prefix) from this point.

The interleaved variant of our model seems related to the model of concurrent zero-knowledge. In both models an adversary conducts polynomially-many interleaved interactions with the prover. In our case these interactions are all with respect to the same common input, and more importantly with respect to the same prover's random coins (i.e., they are all with copies of the same $P_{x,y,\omega}$, where ω is random). In contrast, in the concurrent zero-knowledge model, each interaction is with respect to an independent sequence of prover's coin tosses (while the common input may differ and may be the same). That is, in the concurrent zero-knowledge model, one may interact only once with each P_{x_j,y_j,ω_j}, where the ω_j's are random and independent of one another. Intuitively, interacting with copies of the prover that share the same coin sequence ω seem far more advantageous to the adversary than interacting with copies which have each its independent coin tosses ω_j. However, in order to show that resettable zero-knowledge implies concurrent zero-knowledge, we augment the former model a little so to allow polynomially-many interaction with respect to each of a set of polynomially-many independent choices of prover's coin sequence. That is, we allow the adversary to interact polynomially-many times with each of polynomially-many P_{x_i,y_i,ω_j}'s, where the ω_j's are random and independent of one another.

2.2.1 The actual definition

In the actual definition we use a different formalism than the one presented informally above. That is, instead of prepending each message to P_{x_i,y_i,ω_j} with a session ID, we prepend each message by the full transcript of all messages exchanged so far. That is, we adopt the following convention.

Convention: *Given an interactive pair of (deterministic) machines, (A, B), we construct a modified pair, (A', B'), so that for $t = 1, 2, ...$*

$$\begin{aligned} A'(\alpha_1, \beta_1, ..., \alpha_{t-1}, \beta_{t-1}) &= (\alpha_1, \beta_1, ..., \alpha_{t-1}, \beta_{t-1}, A(\beta_1, ..., \beta_{t-1})) \\ & \quad \text{provided that } \alpha_i = A(\beta_1, ..., \beta_{i-1}), \text{ for } i = 1, ..., t-1 \\ B'(\alpha_1, \beta_1, ..., \alpha_{t-1}, \beta_{t-1}, \alpha_t) &= (\alpha_1, \beta_1, ..., \alpha_{t-1}, \beta_{t-1}, \alpha_t, B(\alpha_1, ..., \alpha_{t-1})) \\ & \quad \text{provided that } \beta_i = B(\alpha_1, ..., \alpha_{i-1}), \text{ for } i = 1, ..., t-1 \end{aligned}$$

In case the corresponding condition does not hold, the modified machine outputs a special symbol indicating detection of cheating. Probabilistic machine are handled similarly (just view the random-tape of the machine as part of it). Same for initial (common and auxiliary) inputs. We stress that the modified machines are memoryless (they respond to each message based solely on the message and their initial inputs), whereas the original machines respond to each message based on their initial inputs and the sequence of all messages they have received so far.

In the traditional context of zero-knowledge, the above transformation adds power to the adversary, since each machine just checks *partial properness* of the history presented to it – its own previous messages. That is, A' checks that $\alpha_i = A(\beta_1, ..., \beta_{i-1})$, but it does not (and in general cannot) check that $\beta_i = B(\alpha_1, ..., \alpha_{i-1})$ since it does not know B (which by the convention regarding probabilistic machines and inputs may depend also on "hidden variables" – the random-tape and/or the auxiliary input to B). However, in the context of resettable zero-knowledge this transformation does not add power: Indeed, the transformation allows an adversary to pick a different (possible) continuation to an interaction, but this is allowed anyhow in the resettable model. In the following definition, we assume that P is a machine resulting from the modification above. Also, without loss of generality we use the standard convention where the "cheating verifier", V^*, is deterministic.

Definition 1 (rZK and rWI - standard model): *An interactive proof system (P, V) for a language L is said to be* resettable zero-knowledge *if for every probabilistic polynomial-time adversary V^* there exists a probabilistic polynomial-time simulator M^* so that the following two distribution ensembles are computational indistinguishable: Let each distribution be indexed by a sequence of common inputs $\overline{x} = x_1, ..., x_{\text{poly}(n)} \in L \cap \{0,1\}^n$ and a corresponding sequence of prover's auxiliary-inputs $\overline{y} = y_1, ..., y_{\text{poly}(n)}$,*

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

1. *Randomly select and fix $t = \text{poly}(n)$ random-tapes, $\omega_1, ..., \omega_t$, for P, resulting in deterministic strategies $P^{(i,j)} = P_{x_i,y_i,\omega_j}$ defined by $P_{x_i,y_i,\omega_j}(\alpha) = P(x_i, y_i, \omega_j, \alpha)$, for $i, j \in \{1, ..., t\}$. Each $P^{(i,j)}$ is called an* incarnation of *P.*
2. *Machine V^* is allowed to run polynomially-many sessions with the $P^{(i,j)}$'s.*

- *In the general model (i.e., the* interleaving version) *we allow V^* to send arbitrary messages to each of the $P^{(i,j)}$, and obtain the responses of $P^{(i,j)}$ to such messages.*
- *In the* sequential (or non-interleaving) version *V^* is required to complete its current interaction with the current copy of $P^{(i,j)}$ before starting a new interaction with any $P^{(i',j')}$, regardless if $(i,j) = (i',j')$ or not. Thus, the activity of V^* proceeds in rounds. In each round it selects one of the $P^{(i,j)}$'s and conducts a complete interaction with it.*

3. *Once V^* decides it is done interacting with the $P^{(i,j)}$'s, it* (i.e., V^*) *produces an output based on its view of these interactions. Let us denote this output by $\langle P(\overline{y}), V^* \rangle(\overline{x})$.*

`Distribution 2:` *The output of $M^*(\overline{x})$.*

In case there exists a universal probabilistic polynomial-time machine, M, so that M^* can be implemented by letting M have oracle-access to V^*, we say that P is resettable zero-knowledge via a black-box simulation.[8]

An interactive proof system (P, V) for L is said to be resettable witness indistinguishable (rWI) *if every two distribution ensembles of Type 1 that are indexed by the same sequence of inputs $\overline{x} = x_1, ..., x_{\mathrm{poly}(n)} \in L \cap \{0,1\}^n$, (but possibly different sequences of prover's auxiliary-inputs, $\mathrm{aux}^{(1)}(\overline{x}) = y_1^{(1)}, ..., y_{\mathrm{poly}(n)}^{(1)}$ and $\mathrm{aux}^{(2)}(\overline{x}) = y_1^{(2)}, ..., y_{\mathrm{poly}(n)}^{(2)}$), are computationally indistinguishable. That is, we require that $\{\langle P(\mathrm{aux}^{(1)}(\overline{x})), V^* \rangle(\overline{x})\}_{\overline{x}}$ and $\{\langle P(\mathrm{aux}^{(2)}(\overline{x})), V^* \rangle(\overline{x})\}_{\overline{x}}$ are computationally indistinguishable.*

Comments on the Definition:

Several previously investigated aspects of zero-knowledge can be cast as special cases of the above general definition. For example, *sequential composition* of zero-knowledge protocols coincides with the special case where V^* must complete each session before starting another, and to run against a different incarnation of the prover in each session so that the prover uses different coin tosses in every session. More importantly, *Concurrent zero-knowledge* coincides exactly with rZK, except that in each session V^* runs against a different incarnation of the prover, so that the prover uses different coin tosses in every session. Thus, *every resettable zero-knowledge protocol is concurrent zero-knowledge.*

2.2.2 Relationship among the variants

Below we refer to four variants of the above definition, depending on two parameters:

1. *Sequential versus interleaving*: This aspect is explicitly considered in Definition 1.

2. *Single versus multiple incarnations*: Definition 1 refers to multiple incarnations, and the single-incarnation variant is obtained by postulating above that $t \equiv 1$ (or, equivalently, allowing V^* to interact only with $P^{(1,1)}$).

Sequential versus interleaving. As stated above, the restricted non-interleaved model is actually as powerful as the general (interleaved) model. That is, any prover strategy that is resettable zero-knowledge in the non-interleaved model is also resettable zero-knowledge in general (i.e., is rZK in the interleaved model). This holds both when allowing a single incarnation or many incarnations. In fact, a stronger result holds:

[8] Recall that the existence of black-box simulators implies auxiliary-input zero-knowledge (cf. [29, 26]).

Theorem 2 *Let P be any prover strategy. Then for every probabilistic polynomial-time V^* for the interleaved model, there exists a probabilistic polynomial-time W^* in the non-interleaved model so that $\langle P(\overline{y}), W^* \rangle(\overline{x})$ is distributed identically to $\langle P(\overline{y}), V^* \rangle(\overline{x})$. Furthermore, W^* uses V^* as a black-box, and if V^* interacts with a single incarnation of P then so does W^*.*

So, in particular, a (zero-knowledge) simulator guaranteed for W^* will do also for V^*, and the black-box feature will be preserved. Furthermore, resettable witness indistinguishable in the sequential model imply rWI in the general (interleaved) model.

Proof Sketch: Using V^* as a black-box and interacting with instances of P in a non-interleaved manner, W^* emulates interleaved interactions of V^* with P. The emulation proceeds round by round. In order to emulate the next communication round (i.e., a message sent by the interleaving adversary followed by a respond by some copy of $P_{x,y,\omega}$), the (non-interleaving) adversary W^* initiates a *new session* of the protocol, and conducts the prior interaction relating to the session that the interleaving adversary wishes to extend. Details follow.

Recall that by our conventions, each message sent in an interaction contains the full transcript of prior messages exchanged during that session. Thus, given a verifier-message, we can recover all prior verifier-messages sent in the corresponding session.[9] For simplicity, we first assume that V^* interacts with a single incarnation of P (i.e., a single $P_{x,y,\omega}$ rather than polynomially-many such P_{x_i,y_i,ω_j}'s).

Suppose that the sequence of messages emulated so far is $\beta_1, ..., \beta_t$ and the message to be emulated is $\beta_{t+1} = (\beta_{i_1}, \alpha_{i_1}, ..., \beta_{i_j}, \alpha_{i_j})$. That is, $\beta_{i_{j+1}} \stackrel{\mathrm{def}}{=} \beta_{t+1}$ is the $j+1^{\mathrm{st}}$ verifier-message in the current session that V^* wishes to extend, and the previous verifier-messages in that session are $\beta_{i_1}, ..., \beta_{i_j}$. Then the non-interleaving adversary, W^*, initiates a *new* session with $P_{x,y,\omega}$, and proceeds in $j+1$ steps so that in the k^{th} step it sends β_{i_k} and obtains the response of $P_{x,y,\omega}$. The non-interleaving adversary W^* forward to V^* (only) the last response of $P_{x,y,\omega}$ (i.e., the response of $P_{x,y,\omega}$ to $\beta_{i_{j+1}}$). Finally, W^* aborts the current session with $P_{x,y,\omega}$ (or, actually, to fit the exact definition of the sequential model, it completes the interaction with this session arbitrarily).[10]

Note that the emulation of each message-exchange between V^* and $P_{x,y,\omega}$ (in the interleaved model) is performed by W^* by initiating and conducting a brand new session with $P_{x,y,\omega}$ (in the sequential model). Thus, if V^* (interleavingly) interacts with s sessions of $P_{x,y,\omega}$ then W^* will (sequentially) interact with $r \cdot s$ sessions, where r is the number of message-exchanges in the protocol (P, V).

The argument extends easily to the general case in which V^* (interleavingly) interacts with polynomially-many P_{x_i,y_i,ω_j}'s. All that is required is for W^* to initiate a new session with the corresponding P_{x_i,y_i,ω_j} (i.e., the one to which the current message of V^* was directed). ■

Single versus multiple incarnations. As stated above, it is our intuition that interacting with multiple incarnations of P is less advantageous to the adversary than interacting (many times) with the same incarnation. *This intuition holds for all natural results presented in this paper*: as in the proof of Theorem 2, the argument for the of security for the single-incarnation case extends easily to the multiple-incarnation case. Unfortunately, a clean result analogous to Theorem 2 is false:

Proposition 3 *There exists a protocol that is resettable zero-knowledge in the single-incarnation model, but is not resettable zero-knowledge in the multiple-incarnation model.*

[9] Note that this holds also in case the alternative convention of specifying a session-ID is adopted. In such a case, one recovers the prior messages corresponding to the current session from the sequence of all messages exchanged.

[10] Indeed, the current session of $P_{x,y,\omega}$ may be "unhappy" with this completion, but (by definition) this information cannot be passed to other sessions of $P_{x,y,\omega}$.

Proof Sketch: We adapt an argument of Goldreich and Krawczyk [26], introducing a prover P that behaves as follows:

- In case the common input x is of even parity, the prover sends the $|x|$-bit long prefix of its random-tape (i.e., ω), and halts.
- In case the common input x is of odd parity, the prover compares the message received from the verifier to the $|x|$-bit long prefix of its random-tape (i.e., ω). If equality holds then the prover reveals to the verifier some hard to compute function of x and/or its auxiliary input (and halts). Otherwise, it halts without sending anything.

It can be easily verified that P is resettable zero-knowledge in the single incarnation model: for x of even parity, the simulator merely outputs a (sequence of repeats of a) uniformly chosen $|x|$-bit long string; whereas for x of odd parity it outputs nothing. In contrast, P is NOT resettable zero-knowledge in the multiple incarnation model: an adversary interacting with $P_{0x',y,\omega}$ and $P_{1x',y,\omega}$, where ω is uniformly selected and x' is of even parity, obtains "knowledge" (and/or y), by first obtaining the $|0x'|$-bit long prefix of ω from $P_{0x',y,\omega}$ and then sending it to $P_{1x',y,\omega}$. ■

Summary and simplified notation. In view of the results above, we analyze the protocols presented in the rest of this paper only with respect to the sequential multiple-incarnation model. In all cases, we first present the analysis of the single-incarnation (sequential) model, and then (easily) extend it to the multiple-incarnation model. Since we shall be using the sequential variant, we can drop the conventions of dealing with many sessions (which were introduced in Section 2.2.1). These conventions were introduced only for the interleaving model, since there an indication must be provided as to which session the current message belongs. Such an indication is unnecessary for the sequential model.

3 Constructing rWI and rZK Protocols: A general paradigm

This section presents a general methodology for constructing rWI and rZK proof systems. This is done as follows. First we present a transformation from a certain class of proof systems, called admissible proof systems, into proof systems in the resettable model. Next, we define a slight strengthening of the concurrent model, called the hybrid model. We show that if the original proof system is admissible and WI (respectively ZK) in the hybrid model then the transformed proof system is rWI (respectively rZK).

It turns out that in the single-incarnation case the same transformation turns any admissible proof-system that is WI (ZK) in the *concurrent* model (rather than in the hybrid model) into an rWI (rZK) proof-system. The proof of this fact is somewhat simpler than the proof for the multiple-incarnation case and can serve as a "warm-up" for that proof. This simpler proof appears in Appendix A.

The next two sections demonstrate how to transform known constructions of concurrent WI and ZK proof systems for $\mathcal{NP}$ (specifically, the constructions of Goldreich and Kahan [25] and Richardson and Kilian [41]) into admissible ones that are WI and ZK in the hybrid model, obtaining:

Theorem 4 *Suppose that there exists a* two-round *perfectly-hiding commitment scheme. Then the following holds:*

1. *Every language in $\mathcal{NP}$ has a* constant-round *resettable witness indistinguishable interactive proof system.*

2. *Every language in $\mathcal{NP}$ has a resettable zero-knowledge interactive proof system. Furthermore, rZK holds via black-box simulation.*

The Class of Admissible Protocols Intuitively, we consider protocols (P, V) in which the first verifier-message "essentially determines" all its subsequent messages. What we mean by "essentially determine" is that the only freedom retained by the verifier is either to abort (or act so that the prover aborts) or to send a practically predetermined message. For clarification, consider the special case (which actually suffices for our applications), in which the first verifier-message is a sequence of commitments that are revealed (i.e., decommitted) in subsequent verifier steps. In such a case, the verifier's freedom in subsequent steps is confined to either send an illegal decommitment (which is viewed as aborting) or properly decommit to the predetermined value. (See Appendix B for a more detail definition of commitment schemes.)

Although the above intuitive formulation suffices for our main results (i.e., deriving the conclusion of Theorem 4 under the standard DLP assumption), we wish to relax it for greater generality. We syntactically partition each subsequent message of the verifier into two parts: a *main part* and an *authenticator*. In the special case considered above (of the first verifier-message being a commitment), the main part (of a subsequent verifier-message) is the revealed value and the authenticator is the extra decommitment information that establishes the validity of this value. The relaxation is that the main part (in this case the revealed value) must be determined by the first verifier message (i.e., the commitment), but the authenticator (i.e., the decommitment information) may vary. Note that this corresponds to the standard definition of commitment schemes that require that the commitment binds the sender to a unique revealed value, but the decommitment information may vary. (We comment that in some implementations, like the one based on DLP, the proper decommitment information is unique too.) The above relaxed form suffices, provided that the prover's subsequent actions merely depend on whether the authenticator is valid (otherwise it aborts), and in case the authenticator is valid the action depends only on the main part of the message. Note that this fits the usual use of commitment schemes within protocols.

Let us first set some useful convention regarding the presentation of protocols in the concurrent and resettable settings. The first message in a session is always sent by the verifier and specifies an incarnation of P. The second message is sent by the prover, and is called the prover initialization message. The third message, sent by the verifier, is called the determining message of the session. (Recall that by our convention the determining message includes the previous two messages.) This terminology will become self-explanatory below.

Definition 5 (admissible proof-systems): *A proof-system (P, V) is called* admissible *if the following requirements hold:*

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

2. *Each verifier message (other than the first one) is first received by P_1 and is interpreted as consisting of two parts, called* main *and* authenticator. *P_1 decides*[11] *whether to accept the*

[11] The above phrase postulates a deterministic decision, which suffices for our applications. We may allow the decision to be probabilistic; In such a case we require that the decision is via bounded-away probabilities (which, without loss of generality, means that the prover either rejects or accepts with negligible probability). The analysis of our transformation holds also in this case. A more relaxed (and natural) definition allows the prover's decision

message or to abort.[12] *If P_1 accepts, it forwards the* main part *of the message to P_2, who generates the next prover message.*

3. *Let V^* be an arbitrary* (deterministic) *polynomial-size circuit representing a possible strategy for the verifier in the interactive proof (P, V). Then, except with negligible probability, V^* is unable to generate two different messages for some round ℓ that specify the same session determining message in their corresponding prefixes, and such that P_1 accepts both.*

The hybrid model. Recall that the difference between the concurrent model and the resettable model is that in the resettable model the "cheating verifier" V^* can invoke many incarnations of the prover with the same random input w, whereas in the concurrent model any two incarnations of the prover have independently chosen random inputs. The hybrid model is defined for admissible protocols (where the random input of the prover is of the form $w = w^{(1)}, w^{(2)}$) and provides the following intermediate power to V^*. Here V^* can invoke many incarnations of the prover with the same value of $w^{(1)}$; but any two incarnations of the prover must have independently chosen values for $w^{(2)}$. The hybrid verifier may be regarded as closer in spirit to the verifier in the concurrent setting (than to the verifier in the resettable setting), since the first message usually only contains initialization information for the session (and in particular is independent of the input). Specifically, in the proof systems considered in this work the first prover message consists only on initialization parameters for a (perfectly secret) commitment scheme.

More formally, In admissible proof systems an incarnation of the prover is identified via three indices: $P^{(i,j,k)} = P_{x_i, y_i, w_{j,k}}$, where $w_{j,k} = w_j^{(1)}, w_k^{(2)}$. That is, i specifies the input, j specifies the random input to P_1 and k specifies the random input to P_2.

Definition 6 (hZK and hWI): *A* hybrid cheating verifier *V^* works against admissible proof systems as described above. That is, V^* proceeds as in Distribution 1 of Definition 1 with the exception that no two sessions started by V^* may interact with incarnations $P^{(i,j,k)}$ and $P^{(i',j',k')}$ such that $k = k'$. An admissible proof system is* hZK *(resp.,* hWI*) if it satisfies Definition 1 with respect to hybrid cheating verifiers.*

3.1 The transformation

We are now ready to present our transformation from admissible proof systems to resettable ones:

Construction 7 *Given an admissible proof system (P, V), where $P = (P_1, P_2)$, and a collection $\{f\}$ of pseudorandom functions (see [23]), we define a new proof system $(\mathbf{P}, \mathbf{V})$ as follows.*

The new verifier *is identical to V.*

The new prover: *The new prover's randomness is viewed as a pair $(w^{(1)}, f)$, where $w^{(1)} \in \{0,1\}^{\mathrm{poly}(n)}$ is of length adequate for the random-tape of P_1, and $f : \{0,1\}^{\leq \mathrm{poly}(n)} \to \{0,1\}^{\mathrm{poly}(n)}$ is a description of a function taken from an ensemble of pseudorandom functions. For convenience we describe the new prover, $\mathbf{P}$, as a pair $\mathbf{P} = \mathbf{P}_1, \mathbf{P}_2$. $\mathbf{P}_1$ is identical to P_1 with*

to depend also on the first part of its random-tape. However, in this case the validity of verifier's messages is not universally verifiable (but rather verifiable only by the prover). We were not able to analyze our transformation for the latter class.

[12] The definition can be further extended by allowing P_1 to consider the main part of all prior verifier-messages. This requires to further specify in the next item what is meant by a properly authenticated generated by the oracle machine (rather than in an interaction). However, the current definition suffices for our purposes.

random-tape $w^{(1)}$; $\mathbf{P}_2$ emulates the actions of P_2 with random tape that is determined by applying f to the determining message and the input. That is, upon receiving the determining message, denoted `msg`*, $\mathbf{P}_2$ sets $w^{(2)} = f(x, \mathtt{msg})$ and runs P_2 with random input $w^{(2)}$. From this step on, $\mathbf{P}_2$ emulates the actions of P using $(w^{(1)}, w^{(2)})$ as P's random-tape.*

Intuitively, Construction 7 "takes care" of the fact that in the resettable model the random-tape of $\mathbf{P}_2$ is fixed in all the sessions of an incarnation of $\mathbf{P}$. The construction does not modify $\mathbf{P}_1$, and in particular does not solve potential problems that may occur when $\mathbf{P}_1$ uses the same $w^{(1)}$ in different incarnations of the prover. However, this is where the hybrid model becomes useful, since in this model the cheating verifier is *prohibited* from using the same $w^{(1)}$ in different incarnations.

In other words, in the single-incarnation case Construction 7 is sufficient for turning any admissible proof-system that is WI (ZK) in the *concurrent* model into an rWI (rZK) proof-system. The hybrid model is not necessary in this case. The proof of this fact is somewhat simpler than the proof for the multiple-incarnation case (presented below) and can serve as a "warm-up" for that proof. This simpler proof appears in Appendix A. We now turn to stating and proving the adequacy of Construction 7 for the general (i.e., multiple-incarnation) case.

Theorem 8 *Suppose that (P, V) is admissible, and let $\mathbf{P}$ be the prover strategy obtained from P by applying Construction 7, assuming that pseudorandom functions exist. Then for every probabilistic polynomial-time cheating verifier V^* (as in Definition 1 there exists a probabilistic polynomial-time hybrid cheating verifier W^* so that $\langle P(\overline{y}), W^*\rangle(\overline{x})$ is computationally indistinguishable from $\langle \mathbf{P}(\overline{y}), V^*\rangle(\overline{x})$.*

Corollary 9 *If a proof system (P, V) is* hWI *then $(\mathbf{P}, \mathbf{V})$ is* rWI*. Similarly, if (P, V) is* hZK *then $(\mathbf{P}, \mathbf{V})$ is* rZK.

Proof of Theorem 8 (sketch): Our analysis refers to a mental experiment in which $\mathbf{P}$ utilizes a truly random function rather than a pseudorandom one. As usual, the corresponding views of the verifier V^* in the two cases (i.e., random versus pseudorandom function) are computationally indistinguishable. From this point on, we identify the random-tape of P with a truly random function.

A first consequence of the above is that in the hybrid model there is essentially no difference between the actions of P and of $\mathbf{P}$.[13] For clarity, we state and prove Theorem 8 with respect to W^* interacting with P rather than with $\mathbf{P}$. Recall that $\langle \mathbf{P}(\overline{y}), V^*\rangle(\overline{x})$ denotes the view (or output) of V^* after interacting with $\mathbf{P}$ on various inputs under the resettable model. Similarly, $\langle P(\overline{y}), W^*\rangle(\overline{x})$ denotes the view (or output) of W^* after interacting with P on various inputs under the hybrid model.

We construct a hybrid-model adversary, W^*, that interacts with incarnations of P, denoted $P^{(i,j,k)}$'s (as in Def. 6). To satisfy Definition 6, this W^* will invoke each $P^{(i,j,k)}$ at most once, and furthermore if it invokes $P^{(i,j,k)}$ then it will not invoke any other $P^{(i',j',k)}$. Essentially, W^* serves as a "mediator" between adversary V^* and the prover P. That is, W^* runs V^*; whenever V^* starts a new session whose determining message is different from all previous ones, W^* merely relays the messages of this session between V^* and P. When V^* "replays" an existing session s (i.e., V^* starts a new session whose determining message is identical to that of an existing session s) W^* responds to V^* using the answers of P in session s, *without interacting with P*. Finally W^* outputs whatever V^* outputs.

[13] More precisely, a verifier that can distinguish between P and $\mathbf{P}$ with non-negligible probability breaks the "admissibility" of (P, V).

The construction of W^*. Working in the hybrid model, W^* handles the messages of V^* as follows:

1. *V^* initiates a new session with some $\mathbf{P}^{(i,j)}$*: In this case W^* initiates a new session with $P^{(i,j,k)}$, where k is a new index not used so far. Next it obtains the prover initialization message, and forwards $\mathtt{msg}$ to V^*.

   We stress that a session with $P^{(i,j,k)}$ may be invoked even if a session with some $P^{(i,j,k')}$, with $k' < k$, was invoked before. In the latter case, since $r_1 = r_1^{(j)}$ is identical in both sessions, the prover initialization message obtained from $P^{(i,j,k)}$ is identical to the prover initialization message obtained previously from $P^{(i,j,k')}$.

2. *V^* sends a new determining message to $\mathbf{P}^{(i,j)}$*: That is, we refer to the case where V^* sends a determining message in the current session, and assume that this message is different from all determining messages sent in prior sessions with $\mathbf{P}^{(i,j)}$. Let $\mathtt{msg}'$ denote the message sent by V^*. Then W^* sends $\mathtt{msg}'$ to one of the sessions of the form $P^{(i,j,\cdot)}$ that still awaits a determining message, obtains the response, and forwards it to V^*. It designates this session (with $\mathbf{P}^{(i,j)}$) as the active session of $(i, j, \mathtt{msg}')$, and stores the prover's response.

   (All subsequent sessions of V^* with $\mathbf{P}^{(i,j)}$ in which the determining message equals $\mathtt{msg}'$ will be "served" by the single session of W^* designated as the active session of $(i, j, \mathtt{msg}')$.)

3. *V^* repeats a first-message to $\mathbf{P}^{(i,j)}$*: That is, we refer to the case where the current message sent by V^* is the determining message in the current session, and assume that this message equals a determining message, $\mathtt{msg}'$, sent in a prior session of V^* with $\mathbf{P}^{(i,j)}$. In this case, W^* retrieves from its storage P's answer in the active session of $(i, j, \mathtt{msg}')$, and forwards it to V^*.

   We stress that W^* does not communicate with any session of P in this case. (Note that if W^* were to send the same message $\mathtt{msg}'$ to two sessions of the form $P^{(i,j,\cdot)}$ then the responses could have differed, whereas V^* expects to see exactly the same answer in sessions in which it sends the same $\mathtt{msg}'$.)

4. *V^* sends a valid message to $\mathbf{P}^{(i,j)}$*: That is, we refer to the case where V^* sends a message in the current session with $\mathbf{P}^{(i,j)}$, and assume that this message is accepted; that is, P_1 accepts it as valid as per Definition 5. (In this case, the message is essentially determined by the determining message in that session.)[14]

   We distinguish two cases, depending on whether this is the first time that a valid verifier-message of the current round was sent in a session of V^* with $\mathbf{P}^{(i,j)}$ in which the determining message equals $\mathtt{msg}'$, where $\mathtt{msg}'$ is the determining message sent by V^* in the current session. Let $\xi > 1$ denote the index of the current message sent by V^*.

   (a) *The current session is the first session of V^* with $\mathbf{P}^{(i,j)}$ in which the determining message equals $\mathtt{msg}'$ and the ξ^{th} verifier-message is valid*: In this case W^* forwards the current message to the active session of $(i, j, \mathtt{msg}')$, obtains P's response, stores it, and forwards it to V^*.

   (b) *The current session is* NOT *the first session of V^* with $\mathbf{P}^{(i,j)}$ in which the determining message equals $\mathtt{msg}'$ and the ξ^{th} verifier-message is valid*: In this case W^* does not

[14] We stress that by the standard definition of commitment schemes it is universally verifiable whether the current message of V^* is valid or not (i.e., this depends only on the current and the determining messages, and on all prover-messages in the current session).

communicate with any session of P. Instead, it merely retrieve the corresponding prover response from its storage, and forwards it to V^*. Note that the corresponding answer is stored in the history of the active session of $(i, j, \mathtt{msg}')$.

(Note that by Definition 5, it is infeasible for V^* to send, in two sessions starting with any fixed verifier-message, valid messages for the same round that differ in their main part. Thus, the responses of $\mathbf{P}^{(i,j)}$ to valid ξ^{th} messages, in sessions starting with any determining message, are identical. It follows that V^* will be content with the identical responses supplied to it by W^*.)

5. *V^* sends an invalid message to* $\mathbf{P}^{(i,j)}$: That is, we refer to the case where V^* sends a message in the current session with $\mathbf{P}^{(i,j)}$, and assume that this message is invalid. In this case, W^* just forwards P's standard `abort` message to V^*.

   We stress that W^* does NOT forward the invalid message of V^* to any session of P, most importantly not to an active session. This allows W^* to handle a corresponding valid message that may be sent by V^* in a future session.

6. *V^* terminates*: When V^* sends a termination message, which includes its output, W^* just outputs this message and halts.

We stress that W^* is defined to operate in the hybrid model. That is, in every session it invokes with P, a different incarnation is used, and furthermore for every k the adversary W^* holds at most one session with an incarnation of the form $P^{(\cdot,\cdot,k)}$. So the second part of P's random-tape in this session is independent from the random-tape in all other sessions. In contrast, V^* that operates in the (stronger) resettable model may invoke each incarnation of $\mathbf{P}$ many times, and so the tape r_2 as determined (by the same incarnation of $\mathbf{P}$) in these sessions is identical. Nevertheless, we claim that the output of W^* is computationally indistinguishable from the output of V^*. The key observations justifying this claim refer to the actions of $\mathbf{P}$ in the various sessions invoked by V^*:

- In sessions having different determining messages, the second parts of the random-tape (i.e., the r_2 part) are independent. Same for sessions in which a different incarnation $\mathbf{P}^{(i,j)}$ is used. This is because $\mathbf{P}$ determines r_2 by applying a random function on the the triplet $(x_i, r_1^{(j)}, \mathtt{msg}')$, where $\mathtt{msg}'$ is the determining message.

  (Indeed, if $i \neq i'$ (resp., $j \neq j'$) then $x_i \neq x_{i'}$ (resp., $r_1^{(j)} \neq r_1^{(j')}$, with overwhelmingly high probability).)

- In sessions having the same common-input, the same r_1, and the same determining message, the actions of $\mathbf{P}$ are essentially determined by the determining message. This is because in this case $\mathbf{P}$ determines the same r_2, and the only freedom of V^* is practically to choose at each message whether to send a predetermined (by the determining message) value or to abort. Thus, the transcripts of all these sessions correspond to various augmented prefixes of one predetermined transcript, where each prefix is either the complete transcript or a strict prefix of it augmented by an `abort` message.

The corresponding transcripts (of imaginary sessions with $\mathbf{P}$) are generated by W^* by merely copying from real sessions it conducts with P. Each set of $\mathbf{P}^{(i,j)}$-sessions sharing the same determining message, is generated from a single (distinct) session with P (called the active session of that message). The way in which W^* handles invalid messages of V^* guarantees that it never aborts an active session, and so such a session can always be extended (up-to completion) to allow the generation of all $\mathbf{P}^{(i,j)}$-sessions sharing that determining message. We stress again that W^* does not need to (and in fact does not) abort a session in order to produce $\mathbf{P}$'s abort message; it merely determines whether $\mathbf{P}$ aborts (and, if so, generates the standard `abort` message by itself). ■

4 rWI proof systems for NP

Part 1 of Theorem 4 is proved by applying Construction 7 to an admissible (as per Definition 5) proof system for $\mathcal{NP}$ that is constant-round and witness-indistinguishable in the hybrid model (of Definition 6). Thus, we need to assert the existence of such a proof system.

Proposition 10 *Suppose that there exists a* two-round *perfectly-hiding commitment scheme. Then every language in $\mathcal{NP}$ has a 5-round admissible proof system that is* hWI.

Proof Sketch: It suffices to present a proof system for some NP-complete problem (we use Graph 3-Colorability). We comment that most of the known zero-knowledge proofs systems are either not admissible (e.g., typically, they do not satisfy the third requirement in Definition 5) or are not witness-indistinguishable in the hybrid model.[15] Fortunately, as we show below, the (5-round) zero-knowledge proof system of [25] is both admissible and witness-indistinguishable in the hybrid model. On an abstract level, the proof system of [25] is as follows.

Common input: A graph $G = (V, E)$, where $V = [n] \stackrel{\text{def}}{=} \{1, ..., n\}$, claimed to be 3-colorable.

Prover's auxiliary input: A 3-coloring $\phi : [n] \to \{1, 2, 3\}$ of G.

(V1) The verifier commits to a sequence of $t \stackrel{\text{def}}{=} n \cdot |E|$ uniformly and independently chosen edges. The commitment is done using a perfectly-hiding commitment scheme,[16] so that the prover gets *no information* on the committed values, while it is infeasible for the verifier to "de-commit" in two different ways (i.e., the scheme is computationally-binding).

(P1) The prover commits to $t \cdot n$ values corresponding to the colors of all vertices under t random relabeling of the coloring ϕ. The commitments are done using an ordinary commitment scheme, providing computational-secrecy and perfect-binding.

(V2) The verifier reveals the sequence of t edges to which it has committed to in Step (V1). It also provides the necessary information required to determine the correctness of the revealed values (i.e., "de-commit").

(P2) In case the values revealed (plus the "de-commitment") in Step (V2) match the commitments sent in Step (V1), and in case all queries are edges, the prover reveals the corresponding colors and provides the corresponding "de-commitment".

[15] For example, in the zero-knowledge proof system of Goldreich, Micali and Wigderson [28], the prover starts by committing itself to a (random) coloring of the graph, and the verifier asks it to reveal the colors of a pair of adjacent vertices. In case the prover's commitment is via unidirectional communication, the proof system is trivially admissible (since the prover uses randomness only in its first message, and the verifier sends a single message), but is not witness-indistinguishable in the hybrid model (since the verifier can obtain a full coloring of the graph by invoking the prover many times on the same r_1. In case the prover's commitment is via a two-round commitment scheme (cf. [40], the proof system is not admissible (since the verifier has total freedom in selecting the edges).

[16] See discussion following this abstract presentation.

(V3) In case the values revealed (plus the "de-commitment") in Step (P2) match the commitments sent in Step (P1), and in case they look as part of legal 3-colorings (i.e., each corresponding pair is a pair of different elements from the set $\{1, 2, 3\}$), the verifier accepts. Otherwise it rejects.

There is one problem, however, with the above presentation. In Step (V1) we have assumed the existence of a 1-round (i.e., uni-directional communication) perfectly-hiding commitment scheme. However, any perfectly-hiding commitment scheme requires at least two rounds of communication (i.e., a message sent from the commitment-receiver to the commitment-sender followed by a message from the sender to the receiver).[17] Thus, we need to integrate such (two-round) commitment schemes in the above proof system. We stress that doing so means that the prover's initial randomization is interpreted as a pair (r_1, r_2), where r_1 is randomness required by the receiver's strategy in the two-round (perfectly-hiding) commitment scheme, and r_2 is the randomization used for implementing Step (P1).

The reader may easily verify that the resulting proof system is indeed admissible. Furthermore, as shown in [25], the proof system is indeed a 5-round zero-knowledge proof system for Graph 3-Colorability. Thus, it follows that the proof system is witness-indistinguishable in the concurrent model (cf. [17]). However, we need to show that it is witness-indistinguishable also in the hybrid model. The extra power of the adversary in the latter model is to invoke sessions with the same (random) value of r_1. However, the randomness of r_1 is only used to establish the (computational) binding property of the verifier's commitment, and this property continues to hold also when the sender commits to several values using the same receiver message.[18] Thus, the above proof system is witness-indistinguishable in the hybrid model, and the proposition follows. ■

Remarks:

1. In fact, the "modification" to the proof system of [25] does not modify any protocol messages; it only specifies the separation of the prover P into P_1 and P_2 as in Definition 5. Essentially, P_1 will play the role of the receiver in the (unconditionally binding) commitment scheme in the [25] protocol; the other functions of P are played by P_2.

2. The resulting proof system is probably *not* rZK; in fact, it is probably not even cZK. This follows from recent work of A. Rosen (priv. comm.). Specifically, extending [26, 39], Rosen shows that no language outside $\mathcal{BPP}$ can have a 7-round proof system that is concurrent zero-knowledge via black-box simulation.

An alternative proof of Proposition 10. An alternative approach for constructing proof systems as required by Proposition 10 is to start with a non-interactive zero-knowledge proof system (cf., [6, 16]). The idea is to employ "coin tossing into the well" (cf., [4]). First, the verifier commits to a sequence of random bits using a perfectly-hiding (two-round) commitment scheme.

[17] The lower bound refers to commitment schemes in which the computationally-hiding requirement should hold w.r.t (non-uniform) polynomial-size circuits. (Such circuits may just incorporate two valid decommitments for the same 1-message commitment.) Note that the standard zero-knowledge condition is itself somewhat non-uniform (as it refers to any verifier's input), and so the commitment scheme used by the verifier must be computationally-binding w.r.t. non-uniform polynomial-size circuits. (Such non-uniform complexity assumptions are employed in all work on zero-knowledge, with the exception of a fully-uniform treatment (cf. [21]).)

[18] See an analogous discussion in the proof of Proposition 11.

Next, the prover sends a corresponding sequence of bits that it selects uniformly. Then, the verifier de-commits and a reference-string for the non-interactive zero-knowledge proof is defined (as usual in "coin tossing into the well"), and finally the prover sends such a (non-interactive) proof (relative to that reference-string). Recall that the latter may require the prover to toss additional coins. The reader may easily verify that the resulting proof system constitutes a 5-round admissible proof system. Again, witness-indistinguishability in the hybrid model is established by analyzing the role of the randomization in the prover's very first message.

5 rZK proof systems for NP

Part 2 of Theorem 4 is proved by applying Construction 7 to an admissible (as per Definition 5) proof system for $\mathcal{NP}$ that is zero-knowledge in the hybrid model (of Definition 6). Thus, we need to assert the existence of such a proof system.

Proposition 11 *Suppose that there exists a* two-round *perfectly-hiding commitment scheme. Then every language in $\mathcal{NP}$ has an admissible proof system that is zero-knowledge in the hybrid model.*

The rest of this section is dedicated to proving Proposition 11. The desired proof-system is obtained by properly modifying the cZK proof system of [41]. While the modification is simple, proving that it suffices is non-trivial. We start by sketching their proof system (a more detailed description of the modified scheme is to follow.)

The Richardson-Kilian Protocol. In essence, given a common input x (allegedly a member of an NP-complete language L), their proof system consists of two stages.

The *first stage* is independent of x. At its start, the verifier commits to k random bit sequences, $r_1, ..., r_k \in \{0,1\}^n$, where n is the security parameter and k is a parameter of the proof. We fix k to be polynomial in the security parameter.[19] This initial commitment is then followed by k *iterations*. In iteration i, the prover commits to a random bit sequence, s_i, and the verifier decommits to the corresponding r_i, thereby pinning down the i^{th} *coin-toss* $r_i \oplus s_i$, the bit-by-bit exclusive or of s_i and r_i. Note that $r_i \oplus s_i$ string is known only to the prover.

In *the second stage*, the prover provides a witness indistinguishable (WI) proof of the following statement: either $x \in L$ or one of the k coin-tosses is the all-zero string (i.e., $r_i = s_i$ for some i).

Intuitively, since the latter case is unlikely to happen, the protocol constitutes a proof system for the language. However, the latter case is the key to the simulatability of the protocol: whenever the simulator may force $r_i = s_i$ for some i, it can simulate the rest of the protocol (and specifically Stage 2) by merely running the WI proof system with r_i (and thus s_i) as a witness. By the WI property, such a run will be indistinguishable from a run in which an NP-witness for the common input being in the language is used.

Our Modification. The above proof system is cZK, but it is not admissible. To obtain a proof system $(\mathcal{P}, \mathcal{V})$, that is admissible and secure in the hybrid model, we require the verifier to send its first message in the rWI proof system of Stage 2 together with his initial commitment message of phase 1. more precisely, the statement S to be proven in Stage 2 is "$x \in L \vee \exists i \text{ s.t.} r_i = s_i$. This

[19] Recall that, by our convention, the verifier commitment message is in fact the third message in the proof system. In the first message the verifier initiates the session; next the prover chooses and sends, in the second message, parameters for the (perfectly secret) commitment scheme used by the verifier in the third message. Indeed, the verifier commitment message corresponds to the *determining message* defined in the previous section.

"NP statement" is transformed in a standard fashion in an instance of 3 colorability, that is, in a graph G, and this G will be the common input of our rWI proof system. Therefore, because (the structure and) the size of S is known in advance (i.e., it is independent of the particular execution of Stage 1 and thus of the particular values of the strings s_i and r_i), so is the number of nodes in G. Consequently, because the verifier of our rWI proof system should commit to choosing random edges in G, such commitment can be made right away (i.e., together with the verifier's commitment at the start of Stage 1), based solely of the number of nodes of G. The verifier simply commits to a random pair of nodes, and when such a pair (u, v) will not correspond to an actual edge of G, it will simply be ignored.

Because the verifier of our rWI proof system proceeds deterministically after sending its first message, our modification guarantees that, barring negligible probability events (e.g. the probability that one could de-commit in two different ways), the verifier's first message (in Stage 1) of an execution of the modified protocol uniquely determines the rest of its messages (both in Stage 1 and Stage 2), unless the verifier decides to abort the execution in the middle.

The next sub-section provides a more complete description of this proof-system.

5.1 The Proof-System $(\mathcal{P}, \mathcal{V})$

The implementation of the protocol uses two complementary types of commitment schemes: The prover's commitments are via a perfectly-binding commitment scheme (which is only computationally-hiding), whereas the verifier's commitments are via a perfectly-hiding commitment scheme (which is only computationally-binding). For simplicity of presentation, we will use a one-round scheme based on any one-way permutations[20] for the first type, and a two-round scheme based on claw-free pairs[21] for the second type. The protocol is visually summarized in Figure 1.

Common Input: x supposedly in the language $L \in \mathcal{NP}$, and a security parameter n.[22]

Prover's Auxiliary Input: an NP-witness w for $x \in L$.

Prover's Randomness consists of two parts $w = w^{(1)}, w^{(2)}$. (Recall that $w^{(2)}$ is later used in Construction 7 to define a pseudorandom function $f : \{0,1\}^{\leq \text{poly}(n)} \to \{0,1\}^{\text{poly}(n)}$.)

Stage 1: This stage has little effect on the actual interaction between the prover and the verifier, yet it provides a "trapdoor" for the simulation.

1. The verifier sends a `new-session`(x) message to the prover. This message indicates that the verifier wishes to start a session for proving that $x \in L$.
2. The prover uses $w^{(1)}$ to determine its first message in the two-round perfectly-hiding commitment scheme.[23]

[20] Specifically, given a one-way permutation f with a hard-core b (e.g., see [27]), one commits to bit σ by selecting uniformly a string x, and sending the value $f(x), b(x) \oplus \sigma$. Decommitment is done by providing (σ and) x.

[21] Specifically, given a family of claw-free pairs, $\{(f_a^0, f_a^1) : a \in I \subseteq \{0,1\}^*\}$ (e.g., see [22]), the sender commits to bit σ as follows. The receiver first selects at random an index $a \in I$ and sends it to the sender, which uniformly selects x in the domain of f_a^σ, and sends the value $f_a^\sigma(x)$. Decommitment is done by providing (σ and) x.

[22] For simplicity we equate the "security governing" parameters such as the the length of strings committed to in Stage 1, the security parameters used in the pseudorandom function and in the commitment schemes, etc.

[23] Here and in the sequel, whenever a party fails to provide a message as instructed the other party halts (detecting an obvious cheating attempt).

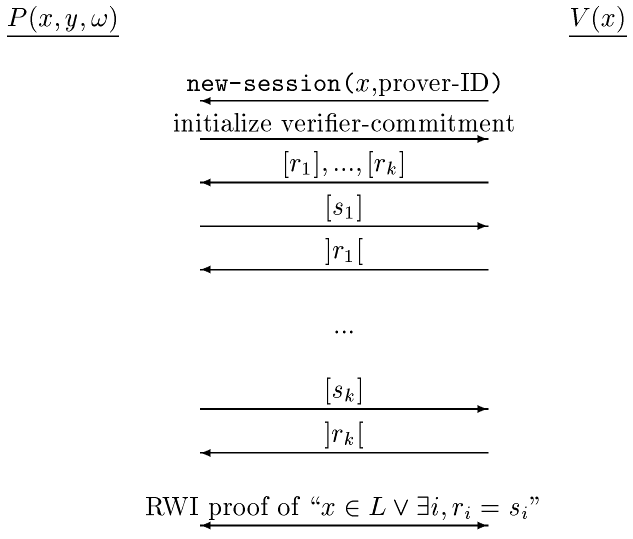

Figure 1: A sketch of the hZK protocol for language L. Here "$[a]$" and "$]a[$" denote commitment and decommitment to a, respectively. The prover sets sets $\omega = \omega_1, \omega_2$, and uses ω_1 for the first message and ω_2 for the rest. Recall that, when transformed to the resettable setting, the value $\rho = f_{\omega_2}(x, [r_1], ..., [r_k])$ is used instead of using ω_2 directly.

3. The verifier commits to k uniformly selected n-bit strings $r_1, ..., r_k \in \{0,1\}^n$, using the perfectly-hiding commitment scheme whose first message was just sent by the prover. Denote by $\overline{\beta} = \beta_1, ..., \beta_k$ the sequence of k commitments sent by the verifier. Note that $\overline{\beta}$ reveals no information about $r_1, ..., r_k$.

   In addition, the verifier executes the first round of the rZK WI proof from Section 4, using the same commitment scheme whose parameters were just sent by the prover.
4. For $i = 1, ..., k$, the following two-round interaction goes on. First the prover commits (in a perfectly-hiding way) to a random k-bit string, denoted s_i, and next the verifier decommits to β_i by providing r_i along with the randomness used in forming β_i from r_i. The prover's choice (i.e., s_i) as well as the randomization used in its commitment are determined by $w^{(2)}$. (Recall that, when transforming this protocol to the resettable model, the randomness for this step, as well as for the prover's actions in Step 2, is determined by applying f to $(x, \bar{\beta}, i)$.) We stress that s_i is uniquely determined by the string, denoted α_i, sent by the prover.

Stage 2: The prover provides a resettable witness indistinguishable proof that *either* $x \in L$ *or* $r_i = s_i$*, for some* i. The NP-witness used by the prover is w, and the witness indistinguishable proof is the one presented in Section 4. Specifically, we reduce the NP-statement *either* $x \in L$ *or there exists an* i *and an* s *so that* α_i *is a valid commitment to* s *and* $r_i = s$ to Graph 3-Colorability. (Letting G be the graph resulting from this reduction, then G does not directly depend on w, but w is efficiently transformed into a 3-coloring of G.)

Completeness and soundness of $(\mathcal{P}, \mathcal{V})$ are straightforward. The rest of this section is dedicated to proving that $(\mathcal{P}, \mathcal{V})$ is zero-knowledge in the hybrid model. To simplify that proof, we make the following assumptions on the model, but stress that none of them restricts the generality of the result.

5.2 Without Loss of Generality

Recall that the verifier V^* is taken to be deterministic. We also assume that V^* does not generate invalid messages (e.g., invalid decommitments). In addition, we make the following simplifying assumptions. (Yet other simplifying assumptions are made when presenting the actual proof below.)

A fixed bound on the number of sessions. Our simulator works assuming that a malicious (resetting) verifier V^* opens at most K sessions in any of its runs, where K depends on the security parameter n via some *fixed* polynomial known in advance. The simulator built here can be extended in standard ways to handle situations where the value of K is no a-priori known or the number of sessions depends on the execution itself. (For instance, keep running the simulator with exponentially increasing values of K, until a successful simulation is generated.) Actually, without loss of generality, but with greater resulting simplicity, we shall assume that V^* always opens exactly K sessions in each run.

Ignoring Stage 2. To prove that $(\mathcal{P}, \mathcal{V})$ is hZK, we must exhibit a simulator $\mathcal{S}$ that generates, with the right distribution, the view of a malicious verifier interacting with an honest prover. Such a verifier will reset the prover K times, and thus execute K different sessions of the protocol. Because the protocol consists of two stages, in a session the verifier may obtain a sub-view of Stage 2. Thus, it may choose its messages in some session (better said, its initial-commitment message of Stage 1 in some session) in a way that also depends on its sub-views of Stage 2 of other sessions. Nonetheless, we shall construct our simulator $\mathcal{S}$ assuming that Stage 2 does not exist. The reason this can be done without loss of generality is that, even though operating in the more difficult resetting model, our simulator $\mathcal{S}$ satisfies the following "forcing" property: *whenever $\mathcal{S}$ completes a Stage 1 view of some session in which the malicious verifier has properly decommitted in all k iterations, then, with overwhelming probability, $\mathcal{S}$ has already forced "$s_i = r_i$ for some i" in that session.* Therefore, simulating the subsequent Stage 2 of that session against the malicious verifier is so trivial that it does not even need any "rewinding" of the verifier: $\mathcal{S}$ just executes with the malicious verifier the honest prover program of our hWI protocol on common input "$x \in L \vee \exists i, r_i = s_i$" and private input "$\exists i, r_i = s_i$". Witness indistinguishability in the hybrid model guarantees that the so generated view of the malicious resetting verifier is computationally indistinguishable from that the same verifier may have with the real prover (who would instead use his witness of $x \in L$ as private input).

Therefore, it suffices and it is simpler to construct $\mathcal{S}$, with the above "forcing" property, by imagining that the protocol actually "consists of Stage 1 alone." Figuratively, we imagine that at the end of Stage 1 (i.e., if the verifier has properly decommitted in all k iterations of Stage 1), we replace Stage 2 by the verifier asking the prover if it has an NP-witness for "$x \in L$ or $s_i = r_i$ for some i" and by the prover providing a truthful answer (which will always be YES whenever he has a witness for $x \in L$). Accordingly, the simulator is *required* to answer truthfully too.

5.3 The High-Level Strategy of the Simulator

At a high level, this strategy is similar to that of the simulator of the cZK protocol of [41]; but significant differences exist in the actual simulation. (In fact, our proof also provides a detailed alternative to the proof of the underlying cZK protocol.)

Call a session *solved* if, in it, $s_i = r_i$ for some i, *unsolved* otherwise; and say that the simulator *solves* a session in iteration i if it forces $s_i = r_i$. As we said, our simulator $\mathcal{S}$ should, with high probability, solve every session s (in some iteration i_s). But: how can it do this?

Recall that a malicious verifier V^* starts the first (of K) session by committing to all its iteration values, $r_1, \ldots, r_k$. Upon receiving this commitment message, the simulator (like the prover) does not know r_1, nor the other r_i's. Thus it tries, by means of a "look ahead," to figure out what r_1 might be. To this end, it sends V^* a commitment, α, to a random value s_1 (most probably, different from r_1). Assume V^* responds to α with its next message of session 1, that is, by decommitting to r_1, then $\mathcal{S}$ has succeeded in discovering r_1. In this case it "rewinds" V^* up to the point in which V^* sent its initial commitment to $r_1, \ldots, r_k$, and, instead of α, this time sends a commitment to r_1, thus forcing $s_1 = r_1$ and "solving" the session.

Assume, however, that V^* responds to α by opening a new session s', thus sending a commitment to $r'_1, \ldots, r'_k$. Then, (at least) the following two choices are available to the simulator:

1. It may stop the look-ahead and leave session 1 unsolved at iteration 1. (That is, it may stick with its answer α and hope to solve session s in some future iteration.)

2. It may insist on solving session 1 at iteration 1. (That is, it may choose random prover values in all other sessions, until V^* decommits to r_1 in session 1.)

Similar choices arise relative to other sessions and other iterations. Clearly, a simulator always opting for type-1 choices cannot work properly. Nor can a simulator always opting for type-2 choices. (In particular, it fails to work whenever it interacts with the malicious verifier V^* that decommits to r_1 in session 1 only when it has completed all iterations of all other sessions and has verified in Stage 2 that the proper NP-witness exists for all of them.) Therefore, simulator $\mathcal{S}$ will adopt an "in-between" strategy.

Specifically, $\mathcal{S}$ uses three main procedures. Procedure `Simulate` tries to generate a simulated run of V^*. Each time a new prover's message is needed, `Simulate` calls procedure `NextProverMsg`, which returns the next prover's message to be used in the simulated run. When `NextProverMsg` encounters an unsolved session in an iteration $j < k$, it calls procedure `Solve`. Procedure `Solve` tries to solve session s at iteration $j+1$ by means of a *bounded look ahead.* That is, before committing to prover's string s_{j+1}, `Simulate` tries to find the value r_{j+1} that V^* might decommit to in iteration $j+1$ of s, so as to choose $s_{j+1} = r_{j+1}$. Such look-ahead is no different than the main simulation: it is done via a recursive call to `Simulate`. At each call, however, a properly initialized counter ℓ is decreased by 1, and when $\ell = 0$ no more recursive calls are made, and the simulator always commits to random prover values s'_i for all iterations i of other sessions s' (unless it happens to know the right r'_i), hoping to solve s at iteration $j+1$. If it fails, it abandons all hope to solve s in iteration $j+1$, hoping instead to solve it at a future iteration. (In this case the simulator sticks with a commitment, α, to a randomly selected value s_i, and proceeds with the rest of the simulation). Below we show that the running time of the simulator is $n^{O(\log_k K)}$, and that, as long as $k = \Omega(\log_k K)$, the above strategy solves all sessions with high probability.

5.4 The Simulator $\mathcal{S}$

Basic Notation

- Say that a message is an *iteration-j (verifier) message* if it is a *valid* decommitment in iteration j of some session i. The initial commitment message of a session is called an *iteration*-0 *message.*

- Sessions are addressed via their *session identifier.* A session identifier s includes the first three messages in the session; these include the verifier's `new-session(`x`,prover ID)` message, the prover's initialization message, and the verifier's commitment to $r_1, \ldots, r_k$.

- A session in an execution prefix is *completed* if, within this execution prefix, the verifier has decommitted properly in all its k iterations. A session is *solved* if for some j the j^{th} prover's commitment in this session is to a value equal to the j^{th} value decommitted to by the verifier in the same session. A black-box simulator is called *truthful* if for every query h made by the simulator and for every completed session in h, that session is solved.

VARIABLES

- ℓ, a local variable bounding the number of recursive calls of `Simulate`.

  Initially, $\ell = L = 2\lceil \log_{\frac{k}{2^{14}}} K \rceil$.

  Comment: *In each call, the value of ℓ is reduced by one; when $\ell = 0$ no more recursive calls are made. This parameter will be used to bound the running time of* `NextProverMsg` *(and the entire simulation).*

- h, a local variable holding the current (prefix of) the execution history of V^*.

  Initially, h holds the common input x.

- S, a global variable holding a set of triplets (s, j, r).

  Initially, S is empty.

  Comment: *S consists of "identified decommitment values". If $(s, j, r) \in S$, then s is a session identifier, j is a numeral of an iteration within session s, and r is the (presumably unique) value that V^* can decommit to in the jth iteration of session s. A variable is "global" if it is accessible to all procedures at all levels of the recursion.*

- SS, a global variable holding the set of solved sessions.

  Initially, SS is empty.

Simulator $\mathcal{S}$

1. Let $h \leftarrow$ `Simulate`(L, h).
2. If the last message in h is a `halt` message of the verifier then output h. Otherwise (the last message in h is `fail`) output `fail`.

Procedure `Simulate`(ℓ, h)**:** Here ℓ is the level of the recursion and h is a prefix of a transcript at a point where V^* is expected to provide the next message. Keep iterating the following instructions:

1. Extend the transcript by one verifier message:

   (1) Set `v-msg` $\leftarrow V^*(h)$ and $h \leftarrow (h,$ `v-msg`$)$.
   Comment: *Recall that* `v-msg` *can be either a* `halt` *message, or a* `new-session` *message, or an iteration-j message for some session s.*

   (2) If `v-msg` $=$ `halt` (i.e., verifier terminates), then return h.

   (3) If `v-msg` is a decommitment in iteration j of session s, then set $S \leftarrow S \cup \{(s, j, r)\}$, where r is the decommitted value.

2. Extend the transcript by one prover message:

   (1) Set (`p-msg`) $\leftarrow$ `NextProverMsg`(ℓ, h) and $h \leftarrow (h,$ `p-msg`$)$.

   (2) If `p-msg` $=$ `fail` (i.e., failure of `NextProverMsg`) then return h. Otherwise, continue the simulation by going back to Step 1.

Procedure `NextProverMsg`(ℓ, h)**:** Extract the last message in h, which is a verifier message denoted `v-msg`, and proceed as follows.

1. *New-Session message*: If `v-msg` is a verifier's message asking to initiate a new session then act as follows. If this is the first session to be initiated (in h) then emulate the prover's answer by selecting a random first receiver-message for the commitment scheme in use. Otherwise, answer with the same prover's message provided in the previous initializations.

2. *Iteration-j message in session s:*

   (1) *If $s \in SS$:*

      1. If $j = k$ (last message in a session) then return (`yes`).
      2. If $j < k$ then return (α), where α is a commitment to a random k-bit string.

   (2) *If $s \notin SS$:*

      1. If $j = k$ then return (`fail`).
      2. Else, if $(s, j+1, r) \in S$ for some r then add s to the set SS of solved sessions, and return (α) where α is a commitment to r.
         Comment: *In this case we actually solve session s.*
      3. *Else (i.e., if $j < k$ and $(s, j+1, r) \notin S$:*
         1. If $\ell > 0$ then set $S' \leftarrow$ `Solve`$(\ell - 1, h, s, j)$. If S' equals $(s, j+1, r)$ for some r, then add s to SS and return (α), where α is a commitment to r. Otherwise, return (α), where α is a commitment to a random k-bit string.
            Comment: *The values of s and j are implicit in h, but we provide them explicitly for clarity of exposition.*
         2. If $\ell = 0$ then return (α), where α is a commitment to a random k-bit string.

Procedure `Solve`(ℓ, h, s, j)**:** Set $i \leftarrow 1$. Next:

1. Make up to $128K$ attempts to solve session s at iteration j. That is, as long as $i \leq 128K$ do:

   (1) Run `Simulate`(ℓ, h).

   (2) If the global variable S contains a triple (s, j, r) for some r then return $\{(s, j, r)\}$. Otherwise set $i \leftarrow i + 1$ and go back to Step 1 (i.e., continue to the next attempt).

2. Return $\emptyset$.

Comments:

1. *Note that procedure* `Solve` *never updates h, even though* `Simulate` *returns an updated value for h. In particular, all the attempts (i.e., all the invocations of* `Simulate`*) start with the same value of h.*

2. *Note that, in principle, it suffices to solve any one of the sessions in the inputs of the invocations of* `Solve` *that recursively called the current invocation. To simplify the analysis, we do not take advantage of this additional "leeway" for the simulation.*

3. *It may seem at first glance that if an attempt stops due to the fact that V^* halts then the simulator can stop at this point and output the current history h. Doing so, however, would skew the probability of the output of the simulation. (For instance, doing so would raise the*

probability of "short executions", namely executions where V^* halts after a smaller number of steps.) To avoid such skew, we end the simulation only when V^* halts within the main procedure.

5.5 $\mathcal{S}$ is Poly-Time

We first show that the simulator runs in (worst case) polynomial time. Say that an invocation of `Simulate` (resp., an invocation of `NextProverMsg` or of `Solve`) is at level ℓ if it is called with first parameter ℓ. Let $T_{\mathrm{NPM}}(\ell, n)$ (resp., $T_{\mathrm{SOLVE}}(\ell, n)$) denote the (worst case over all inputs) running time of `NextProverMsg` (resp., `Solve`) at level ℓ and with security parameter n. We assume for simplicity that the verifier, V^*, runs in worst-case polynomial time (in n). Recall that $L = 2\lceil \log_{\frac{k}{2^{14}}} K \rceil$. The reader may verify the following facts:

1. The running time of the main procedure is at most $\mathrm{poly}(n) \cdot T_{\mathrm{NPM}}(L, n)$;
2. $T_{\mathrm{NPM}}(\ell, n) < T_{\mathrm{SOLVE}}(\ell - 1, n) + \mathrm{poly}(n)$;
3. $T_{\mathrm{SOLVE}}(\ell, n) \leq \mathrm{poly}(n) \cdot T_{\mathrm{NPM}}(\ell, n)$;
4. $T_{\mathrm{SOLVE}}(0, n) = \mathrm{poly}(n)$.

Thus, $T_{\mathrm{NPM}}(L) \leq n^{O(L)}$ and the running time of the main procedure is $\mathrm{poly}(n)$ (as long as $K = k^{O(1)}$).

5.6 The Output of $\mathcal{S}$ is correctly distributed

To prove that $\mathcal{S}$ does its job, we show that it satisfies two main properties:

1. Conditioned on the event that $\mathcal{S}$ does not output `fail`, we have $[\mathcal{S}, V^*](x) \approx [P, V^*](x)$; and
2. $\mathcal{S}$ outputs `fail` with negligible probability.

Property 1 is easily verified. (Essentially, any verifier that violates property 1 can be turned into an adversary that breaks the computational secrecy of the prover's commitments. We omit further details.) Let us then focus on proving property 2, more specifically that $\mathcal{S}$ fails with probability at most $2^{-O(n)}$. The heart of the proof is showing that, when invoked in a situation that is not too unfavorable, procedure `Solve` succeeds in obtaining the desired value (i.e., the value decommitted by V^* in the corresponding iteration) with overwhelming probability. Once this is formalized and shown, the rest of the proof is simpler.

The following three types of events can cause an "unfavorable situation" where `Solve`(l, h, s, j) may fail. First, the probability of V^* ever decommitting in iteration j of session s may be small. Second, V^* may open and complete too many new sessions before it decommits in iteration j of session s. Third, the transcript h may contain existing sessions that are closed to completion but not yet solved.

Demonstrating that each one of these three bad events does not affect the simulation too much is done using different techniques. For the first bad event, we argue that a session where many of its iterations have small probability of completing will almost never complete. For the second type of event we argue that if V^* opens and completes "too many" new sessions before decommitting in iteration j of session s, then it must be the case that during other iterations of this session only few new sessions are opened. For the third bad event, we show that it never occurs; this, however, requires that k, the number of iterations, be non-constant. (Note that this requirement on k is made regardless of the restriction on k imposed by the requirement that the simulator runs in polynomial time. In particular, we require k to be non-constant even if the simulator is allowed to run in quasi-polynomial time.) We remark that much of the complexity of the proof is due to the fact that the schedule of opening and completing sessions is determined by V^* "on the fly". If V^* were limited to a fixed and known schedule the proof would would be considerably simpler.

The rest of the proof is organized as follows. We start with setting some notation to be used throughout the proof. Next we prove three claims bounding the probabilities of the bad events sketched above. Next, we show (in Claim 5.4) that under certain conditions each invocation of `Solve` succeeds with overwhelming probability. Finally, we demonstrate Property 2 based on Claim 5.4.

Notation:

Simulator states, executions, and attempts. A state σ of the simulator at some point during its run describes all the current memory and control information about the current point in the run. (Our convention is that a state does *not* specify the random choices to be used by the simulator in a continuation of a run from the state.) An execution describes a specific run of the simulator. some state. It is determined by the input x and by the random choices of the simulator made during this execution. Given $r \in \{0,1\}^*$, we use the term "execution r" to denote "the execution with random choices r". (Formally, an execution r is a sequence of states, where each state follows from the previous one by the code of the simulator and the (deterministic) behavior of V^*.) We also consider "execution-segments" that start at some state (not necessarily the start state of the simulator) and end at some state (not necessarily a halting state).

Successful attempts. Say that an attempt within an invocation of $\texttt{Solve}(\ell, h, s, j)$ (within some execution of the simulator) is successful if during this attempt an entry (s, j, r) is added to S. An invocation of `Solve` is successful if one of its attempts is successful.

Definition of q_σ. Let σ be some state of the simulator at the point where an invocation $\texttt{Solve}(\ell, h, s, j)$ is called, and consider the following bad event: An invocation of $\texttt{Simulate}(\ell, h)$ within an attempt of this invocation of `Solve` returns a transcript in which V^* halts, and a triple (s, j, r) was not added to S during this attempt. Let q_σ denote the probability of this event. Here the probability is taken over the random choices of the simulator, starting from state σ.[24]

Definition of μ_σ. Consider an execution r of `Simulate` starting at state σ. Let $h_{\sigma,r}$ denote the value returned by `Simulate` in this execution. (Recall that this value holds a transcript, or history of a run of V^*.) Let $\mu_{\sigma,r}$ denote the number of sessions in $h_{\sigma,r}$ that are opened after `Simulate` is called and are completed before $h_{\sigma,r}$ ends. Let μ_σ denote the expected value of $\mu_{\sigma,r}$ when r is chosen at random. (Loosely speaking, μ_σ denotes the expected number of sessions that are opened and completed within the "main thread" of a run of `Simulate`. Sessions that are opened in recursive invocations of `Simulate` (i.e., during "look-aheads") are not counted.) Since we have a strict bound K on the number of sessions in an execution of V^* we have that $\mu_\sigma \leq K$ for all σ.

[24] The above definition implicitly assumes that all the attempts of the invocation of `Solve` start at the same state. This is somewhat imprecise, since each attempt has a slightly different start state. specifically, the sets S and SS may be different for each attempt. We ignore this slight imprecision and return to it at the end of the analysis. (For concreteness, define q_σ with respect to the first attempt made in the said invocation of `Solve`.)

Safe sessions, states, and executions. Consider a state σ of the simulator at the point where either an invocation `Simulate`(ℓ, h) is called or an invocation `Solve`(ℓ, h, s, j) is called. A session s' is called an **old session** for σ if the verifier's commitment message in session s' is already sent in h. An old session s' is called m-safe for σ if session s' is already solved by the simulator (i.e., $s' \in SS$), or V^* has not yet decommitted in at least $m+1$ iterations of session s'. State σ is called m-**safe** if all the old sessions for it (and, in particular, session s) are m-safe. An execution of `Solve` starting from state σ is m-**safe** if σ is m-**safe**. An execution of `Simulate` is called m-**safe throughout** if *all* the states during this execution, in which `Simulate` is recursively called, are m-**safe**. An attempt of `Solve` is m-**safe throughout** if the corresponding execution of `Simulate` is m-safe throughout.

Observe that in a session that is m-safe and not solved for some state σ there are at least m iterations where the prover did not yet send its commitment. Also, an attempt of `Solve` that starts from state σ and is m-safe throughout for some $m \geq 0$ never fails due to failure of `NextProverMsg` (in Step 22 of `Simulate`) on a session that is old for σ.

Next we prove three claims that bound the probability of bad events in executions of `Solve` and `Simulate`. The claims correspond to the three possibilities of failure, sketched above. The first claim is used to bound the number of attempts that may fail due to old sessions:

Claim 5.1 *Let $\ell \geq 0$ and consider an $(\ell+1)$-safe execution of* `Solve`(ℓ, S, h, s, j). *Then at most $K-1$ attempts in this execution are not ℓ-safe throughout.*

Proof: Assume that an attempt in the given execution of `Solve` is not ℓ-safe throughout. This means that some old session s' is not ℓ-safe at some state during this attempt. For that to happen, V^* must have properly decommitted in iteration $k-\ell$ of session s'. However, in this case the entry $(s', k-\ell, r)$ is added to S in this attempt. Since the current execution of `Solve` is $(\ell+1)$-safe, we have that the prover's commitment for iteration $k-\ell$ in session s' is not yet sent in h. Consequently, in future attempts session s' will remain ℓ-safe. (Either V^* will not decommit in iteration $k-\ell$ or session s' will be solved and added to SS.)

Altogether, there are at most $K-1$ old sessions. Thus at most $K-1$ attempts are prevented from being ℓ-safe throughout. ■

For the next two claims, consider an execution of `Simulate` at level ℓ. The second claim (Claim 5.2) says, essentially, that any new session that reaches the point where $2k/3$ of its iterations are completed must have many iterations where the corresponding invocations of `Solve`, at level $\ell-1$, have low probability of failure due to the fact that V^* halts. (We remark that the "cut-off" point of $2k/3$ iterations is rather arbitrary. It is chosen so that there is a substantial number of iterations both before and after the cut-off point.) The third claim (Claim 5.3) says that any new session that reaches the point where $2k/3$ of its iterations are completed must have many iterations where the corresponding values of μ are sufficiently small.

More precisely, consider an execution r of `Simulate` at level ℓ, starting from state σ. Recall that $h_{\sigma,r}$ denotes the value returned by `Simulate` in this execution. Let $\sigma_{i,j,r}$ denote the state of the simulator at the point where `Solve` is invoked (at level $\ell-1$) for the jth iteration at the ith new session in $h_{\sigma,r}$.[25] Recall that $\mu_{\sigma,r}$ denotes the number of sessions in $h_{\sigma,r}$ that are opened after the

[25] We remark that state $\sigma_{i,j,r}$ is a bit "over-specified", in the sense that only a prefix of r may be needed to specify $\sigma_{i,j,r}$. Nonetheless, we stress that the state $\sigma_{i,j,r}$ does not specify any random choices to be used in a continuation of the run. In particular, the values $q_{\sigma_{i,j,r}}$ and $\mu_{\sigma_{i,j,r}}$ are defined, as usual, with respect to random continuations of $\sigma_{i,j,r}$.

execution of `Simulate` started and completed before this execution returns, and that μ_σ denotes the expected value of $\mu_{\sigma,r}$ when r is chosen at random. Define the following bad events: (Again, the constants are somewhat arbitrary. They are set to match the conditions of Claim 5.4.)

Qbad executions. An execution r of `Simulate`, starting from state σ, is i-Qbad if $q_{\sigma_{i,j,r}} > \frac{31}{32}$ for at least $k/12$ iterations j out of the first $2k/3$ iterations in the ith new session in $h_{\sigma,r}$ to complete $2k/3$ iterations. An execution is Qbad if it is i-Qbad for some i.

Mbad executions. An execution r of `Simulate` at level ℓ, starting from state σ, is i-Mbad if $\mu_{\sigma_{i,j,r}} > \frac{1}{64}(k/2^{14})^{\ell-1}$ for at least $k/2$ iterations j out of the first $2k/3$ iterations in the ith new session to complete $2k/3$ iterations in $h_{\sigma,r}$. An execution is Mbad if it is i-Mbad for some i.

Observe that in an execution r of `Simulate` at level ℓ, that is not Qbad and not Mbad, any session that completes $2k/3$ iterations has at least one iteration j where both $q_{\sigma_{i,j,r}} \le \frac{31}{32}$ and $\mu_{\sigma_{i,j,r}} \le \frac{1}{64}(k/2^{14})^{\ell-1}$. (In fact, at least $k/12$ such iterations exist.) This fact will be useful in proving the induction step of Claim 5.4 below.

Long executions. An execution r of `Simulate`, starting from state σ, is ℓ-long if $\mu_{\sigma,r} > (\frac{k}{2^{14}})^\ell$. An execution that is not ℓ-long is called ℓ-short.

Claim 5.2 *Let σ be a state of the simulator at the point where an execution of* `Simulate` *is called. Then an execution of* `Simulate` *starting from state σ is Qbad with probability at most $K \cdot 2^{-O(k)}$.*

Proof: Let β_i denote the probability that an execution is i-Qbad. We show that $\beta_i \le 2^{-O(k)}$ for every i. Consider an i-Qbad execution r, and let r_j denote the segment of r that is used by the simulator from state $\sigma_{i,j-1,r}$ until state $\sigma_{i,j,r}$ is reached. Note that $r_1, ..., r_k$ are disjoint and contained in r. Since the execution is i-Qbad, we have $q_{\sigma_{i,j,r}} > \frac{31}{32}$ for at least $k/12$ iterations j out of the first $2k/3$ iterations in the ith session. Thus, we have:

$$\begin{aligned}
\beta_i &\le \sum_{B \subset \{1..\frac{2k}{3}\}, |B| < \frac{k}{12}} \Pr_r(r \text{ is } i\text{-Qbad} \mid q_{\sigma_{i,j,r}} > \tfrac{31}{32} \text{ for } j \in B) \\
&\le \sum_{B \subset \{1..\frac{2k}{3}\}, |B| < \frac{k}{12}} \Pr_r\left(\bigwedge_{j \in B} \text{iteration } j \text{ completed} \mid q_{\sigma_{i,j,r}} > \tfrac{31}{32} \text{ for } j \in B\right) \\
&\le \sum_{B \subset \{1..\frac{2k}{3}\}, |B| < \frac{k}{12}} \prod_{j \in B} \Pr_{r_j}(\text{iteration } j \text{ completed} \mid q_{\sigma_{i,j,r}} > \tfrac{31}{32}) \\
&\le \binom{2k/3}{k/12} \cdot (1/32)^{k/12} \le 2^{-O(k)}
\end{aligned}$$

(The third inequality follows from the fact that the simulator uses independent random choices for each message sent to V^*.) It follows that an attempt is Qbad with probability at most $\sum_{i=1..K} \beta_i \le K \cdot 2^{-O(k)}$. ■

Claim 5.3 *Consider a state σ at the point where an invocation of* `Simulate` *at level ℓ is called, and assume that $\mu_\sigma \le \frac{1}{64}(k/2^{14})^\ell$. Then, a random ℓ-short execution of* `Simulate` *starting from state σ is Mbad with probability at most $K2^{-O(k)}$.*

Proof: Let $h_{\sigma,r}^{i,j}$ denote the segment of $h_{\sigma,r}$ that starts when V^* decommits in iteration $j-1$ in the ith session opened in execution r, and ends when V^* either halts or decommits in iteration j of this session. Let $m_{\sigma,r}^{i,j}$ denote the number of sessions that are opened and completed within the segment $h_{\sigma,r}^{i,j}$. Observe that for any execution r and session i we have $\sum_{j=1}^{k} m_{\sigma,r}^{i,j} \leq \mu_{\sigma,r}$. (This holds since for all r the segments $\{h_{\sigma,r}^{i,j}\}_{j=1..k}$ are disjoint and contained in $h_{\sigma,r}$.) It holds that in each execution r there are at most $k/4$ iterations j with $m_{\sigma,r}^{i,j} > \frac{4}{k}\mu_{\sigma,r}$. Since in ℓ-short executions we have $\mu_{\sigma,r} \leq (k/2^{14})^{\ell}$, it follows that in such executions there are at least $3k/4$ iterations j with:

$$m_{\sigma,r}^{i,j} \leq \frac{4}{k}\mu_{\sigma,r} \leq \frac{4}{k}(\frac{k}{2^{14}})^{\ell} = \frac{1}{2^{12}}(\frac{k}{2^{14}})^{\ell-1}.$$

It follows that in Mbad executions there are at least $k/4$ iterations j among the first $2k/3$ iterations, such that $m_{\sigma,r}^{i,j} \leq \frac{1}{2^{12}}(\frac{k}{2^{14}})^{\ell-1}$ and $\mu_{\sigma_{i,j,r}} \geq \frac{1}{64}(k/2^{14})^{\ell-1}$. Call such iterations unlikely. Notice that an unlikely iteration occurs with probability at most 2^{-6}: $m_{\sigma,r}^{i,j}$ describes a value drawn at random from a distribution with expectancy at least $\mu_{\sigma_{i,j,r}}$. It thus follows from Markov Inequality that $m_{\sigma,r}^{i,j}$ is less than 2^6 times $\mu_{\sigma_{i,j,r}}$ only with probability 2^{-6}.[26] Moreover, unlikely iterations occur independently of each other, since they use disjoint segments of the random input r. More precisely, recall that r_j denotes the segment of r that is used by the simulator from state $\sigma_{i,j-1,r}$ until state $\sigma_{i,j,r}$ is reached. Let γ_i denote the probability that an ℓ-short execution r is i-Mbad. We have:

$$\begin{aligned}
\gamma_i &\leq \sum_{B\subset\{1..\frac{2k}{3}\},|B|<\frac{k}{4}} \Pr_r(\text{iterations } j \in B \text{ are unlikely}) \\
&\leq \sum_{B\subset\{1..\frac{2k}{3}\},|B|<\frac{k}{4}} \prod_{j\in B} \Pr_{r_j}(\text{iteration } j \text{ is unlikely}) \\
&\leq \binom{2k/3}{k/4} \cdot (2^{-6})^{k/6} \leq 2^{-O(k)}
\end{aligned}$$

It follows that an ℓ-short execution is Mbad with probability at most $\sum_{i=1..K}\gamma_i \leq K \cdot 2^{-O(k)}$. ■

We proceed to state and prove the main technical claim used to prove Lemma 5.5:

Claim 5.4 *Let σ be a state of the simulator at the point where some invocation of* `Solve` *at level ℓ is called, and assume that:*

- $q_\sigma \leq \frac{31}{32}$
- $\mu_\sigma < \frac{1}{64}(\frac{k}{2^{14}})^{\ell}$
- *State σ is $(\ell+1)$-safe.*

Then there exists a constant c such that a random execution of `Solve` *starting from state σ fails with probability 2^{-cK}. Again, the probability is taken over the random choices of the simulator starting from state σ.*

Proof: The proof proceeds by induction on ℓ:

[26] The above argument implicitly assumes that all the attempts of the invocation of `Solve` start at state $\sigma_{i,j,r}$. This is somewhat imprecise, since each attempt has a slightly different start state. specifically, the sets S and SS may be different for each attempt. We ignore this slight imprecision and return to it at the end of the analysis.

Base case: $\ell = 0$. An execution of `Solve` is not successful only if none of its $128K$ attempts is successful. We bound the probability that this event happens. We first assert the following fact:

Base-Case Fact: The invocation of `Simulate`$(0, h)$ within a random attempt that is 0-safe throughout returns due to failure of `NextProverMsg`(in Step 22 of `Simulate`) with probability at most $1/64$.

Proof: Executions of `Simulate` that are 0-safe throughout never return due to failure of `NextProverMsg` on an old session. The probability that an execution of `Simulate` returns due to failure of `NextProverMsg` on a new session is at most the probability that V^* starts and completes a new session during this execution. However, since $\mu_\sigma < 1/64 \cdot (k/2^{14})^0 = 1/64$, it follows from Markov Inequality that V^* starts and completes even a single new session during this execution with probability at most $1/64$. □

We are now ready to prove the base case. Say that an attempt is unfortunate if one of the two events occur:
(i) The invocation of `Simulate` returns due to the fact that V^* halts (i.e., in Step 12), and a triple (s, j, r) was not added to S during this attempt.
(ii) The invocation of `Simulate` returns due to failure of `NextProverMsg` on a new session.
By the premise of the lemma, the probability of event (i) is at most $q_\sigma < \frac{31}{32}$. By our Base-Case Fact, the probability of event (ii) is at most $1/64$. Altogether, the expected number of unfortunate attempts is at most $\frac{63}{64} \cdot 128K = 126K$. Since the attempts are independent of each other, it follows from Chernoff Inequality that the probability that more than $126.5K$ attempts are unfortunate is at most 2^{-cK} for some $c > 0$. In other words, except with probability 2^{-cK}, there are at least $1.5K$ attempts that are either successful or where `Simulate` returns due to failure of `NextProverMsg` on an old session. However, it follows from Claim 5.1 that at most K attempts are not 0-safe throughout, thus in at most K attempts `Simulate` returns due to failure of `NextProverMsg` on an old session. It follows that, except with probability 2^{-cK}, at least $\frac{K}{2}$ attempts succeed. (In fact, a single successful attempt would suffice.)

Induction step: As in the base case, we bound the probability that none of the $128K$ attempts in an execution of `Solve` (now at level ℓ) is successful. We first show:

Induction-Step Fact: Let σ be a state where an invocation of `Simulate` at level ℓ is called. Assume that Claim 5.4 holds for all $\ell' < \ell$ and that $\mu_\sigma < \frac{1}{64}(\frac{k}{2^{14}})^\ell$. Then a random execution of `Simulate`, starting from state σ, that is ℓ-short and ℓ-safe throughout, returns due to failure of `NextProverMsg` with probability at most $K2^{-\mathrm{poly}(k)}$.

Proof: Recall that an execution of `Simulate` that is ℓ-safe throughout never returns due to failure of `NextProverMsg` on an old session. We bound the probability that an ℓ-safe throughout execution of `Simulate` returns due to failure of `NextProverMsg` on a new session. In fact, we show a slightly stronger result than stated in the claim: Except with probability $1/64 + K2^{-\mathrm{poly}(k)}$, all the new sessions in this execution remain $\frac{k}{3}$-safe (i.e., they do not go beyond their $\frac{2k}{3}$-th iteration without being solved).

Say than an execution of `Simulate` at level ℓ is bad if it is either Qbad or Mbad. Recall that a random execution of `Simulate` at level ℓ is Qbad with probability at most $K2^{-\mathrm{poly}(k)}$ (Claim 5.2), and is Mbad given that it is ℓ-short with probability at most $K2^{-\mathrm{poly}(k)}$ (Claim 5.3). It follows that a random execution of `Simulate` is bad with probability at most $K2^{-\mathrm{poly}(k)}$. We show that an execution of `Simulate` that is not bad and ℓ-safe throughout fails in Step 22 of `Solve` on a new session with probability at most $K \cdot 2^{O(K)}$. Specifically, we demonstrate the following inductive claim. Order the new sessions in an execution of `Simulate` according to the order in which they complete $2k/3$ iterations. That is, let s_i denote the ith session to complete $2k/3$ iterations. Let Z_i denote the event that, at the point where session s_i has completed $2k/3$ iterations, there are new sessions that are not $k/3$-safe. We show by induction on i that there exists a constant $c > 0$ such that $\Pr(Z_i) \le i \cdot 2^{-cK}$. (In other words, we show that, except for probability at most $i \cdot 2^{-cK}$, all the first i new sessions to complete $2k/3$ iterations are solved. (c is the constant guaranteed by Lemma 5.4 for level $\ell - 1$.)

We proceed to prove the induction step. (The base case is treated as a special case of a step.) Let $i \ge 1$, and let r be a random execution of an attempt that is ℓ-safe throughout, not Qbad and not Mbad. Consider the invocations of `Solve` at level $\ell - 1$ that are associated with the first $2k/3$ iterations of the ith session. Say that an invocation is `good` if the three conditions of Lemma 5.4 are satisfied with respect to this invocation. (That is, let σ' denote the state at the onset of a good invocation; then, $q_{\sigma'} < \frac{31}{32}$, $\mu_{\sigma'} < \frac{1}{64}(\frac{1}{2^{14}})^{\ell-1}$, and σ' is ℓ-safe.) We want to show that at least one of these invocation of `Solve` is good. Recall that, since the execution is neither Qbad not Mbad, there is at least one such invocation of `Solve` for which the first two conditions are met. It remains to show that this invocation is also ℓ-safe. This is argued as follows. Since the attempt is ℓ-safe throughout, all the old sessions are ℓ-safe. From the hypothesis of the induction on i we have that, except for probability $(i-1)2^{-cK}$, all the new sessions are $k/3$-safe, hence also ℓ-safe.[27]

We conclude that, except for probability $(i-1)2^{-cK}$, at least one out of the first $2k/3$ invocation of `Solve` associated with the ith session is good. Applying the hypothesis of the induction on ℓ, we get this invocation of `Solve` fails with probability at most 2^{-cK}. Consequently, the ith session is not solved by the time $2k/3$ of its iterations are completed with probability at most $2^{-cK} + (i-1)2^{-cK} = i2^{-cK}$. This completes the step of the induction on i, and the proof of our Induction-Step Fact. □

We are now ready to complete the step of the induction on ℓ (and the proof of Lemma 5.4) based on our Induction-Step Fact. The argument is very similar to that of the base case. We repeat it here for completeness, at the price of some repetition.

Recall the definition of unfortunate attempts. By the premise of the lemma, the probability of event (i) is at most $q_\sigma < \frac{31}{32}$. By our Induction-Step Fact, the probability that `Simulate` returns due to failure of `NextProverMsg` in an ℓ-short execution is at most $K2^{-\text{poly}(k)} = 2^{-\text{poly}(n)}$. By the premise of Claim 5.4, $\mu_\sigma < \frac{1}{64}(\frac{1}{2^{14}})^{\ell}$. It follows from Markov Inequality that an execution of `Simulate` within an attempt of `Solve` at level ℓ is ℓ-long with probability at most $1/64$. Consequently, the probability of event (ii) is at most $1/64 + 2^{-\text{poly}(n)}$. Altogether, the expected number of unfortunate attempts is at most $(\frac{63}{64} + 2^{-\text{poly}(n)})128K = 126K + 2^{-\text{poly}(n)}$. Since the attempts are independent of each other, it follows from Chernoff Inequality that the probability that more than $126.5K$ attempts are unfortunate is at most 2^{-cK} for some $c > 0$. In other words, except with probability 2^{-cK}, there are at least $1.5K$ attempts that are either successful or where `Simulate` returns due to failure of `NextProverMsg` on an old session. However, it follows from Claim 5.1 that at most K attempts are not ℓ-safe throughout, thus in at most K attempts `Simulate` returns due to failure of `NextProverMsg` on an old session. It follows that, except with probability 2^{-cK}, at least $\frac{K}{2}$ attempts succeed. This completes the proof of Lemma 5.4. ■

We are finally ready to prove the main lemma:

Lemma 5.5 *The main procedure fails with probability at most* $2^{-cK} = 2^{-\text{poly}(n)}$.

Proof:

[27]Recall that both k and K are polynomial in the security parameter n. Consequently $\ell \le L = O(1)$, and $k > 3\ell$ for large enough n.

The main procedure consists of a single invocation of `Simulate` at level L, and fails if this invocation returns due to failure of `NextProverMsg`. We prove the lemma by applying our Induction-Step Fact to this invocation.[28] Specifically, all the requirements of our Induction-Step Fact are met:

- Claim 5.4 holds for all $\ell < L$.
- We have that V^* opens at most K sessions in an execution. Let σ_0 be the state of the simulator at the point where `Simulate` is invoked within the main procedure. Thus, for $K > 64$ we have $\mu_{\sigma_0} < \frac{K^2}{64} = \frac{1}{64}(\frac{k}{2^{14}})^L$. Furthermore, any execution of `Simulate` within the main procedure is L-short.
- There are no old sessions for state σ_0. Consequently, any execution of `Simulate` within the main procedure is L-safe throughout.

We conclude that a random execution of `Simulate` within the main procedure returns due to failure of `NextProverMsg` with probability at most $2^{-\mathrm{poly}(n)}$. ■

On the effect of imperfect commitments. In the definition and argumentation on the quantities q_σ and μ_σ we have so far ignored the following fact. Different attempts within an invocation of `Solve` start with somewhat different states of the simulator. Specifically, the sets S and SS may differ from attempt to attempt (and also in the continuation of a simulation once `Solve` returns). Consequently, when invoked by `Simulate` in different attempts, procedure `NextProverMsg` may try to "solve" different sessions in Steps 2(2)2 and 2(2)31.

As long as the prover's commitments are assumed to be perfectly secret these differences in the behavior of the simulator is transparent to V^*, and the above analysis is precise. We conclude the analysis by noting that the above analysis holds even when the prover's commitments are only computationally secret. Specifically, we claim that the differences between the values of q_σ (and similarly μ_σ) in different attempts of an execution of `Solve` are negligible. More precisely, consider an execution of `Solve` starting from state σ, and let σ_v denote the state at the point where the vth attempt begins. Then for all $v, v' = 1...128K$ the differences $q_{\sigma_v} - q_{\sigma_{v'}}$ and $\mu_{\sigma_v} - \mu_{\sigma_{v'}}$ must be negligible in the security parameter n. (If this were not the case then we could use this fact to break the computational secrecy of the prover's commitment.) Consequently, it can be verified that the analysis (and in particular Claims 5.2 and 5.3) still holds.[29]

6 rZK in the Public-Key Model

Thus far in the paper no set-up assumptions have been made in the model. This is indeed the "simplest" model used for two-party and multi-party computation. Another model, used routinely in the context of providing privacy and/or authenticity of messages (i.e public-key encryption and digital signatures), is the *public-key* model, which relies on a set-up stage in which public-keys are registered. In the work presented in this section, the public-key model is used for tasks totally unrelated to privacy and authenticity, but rather for proving security of protocols whose participants

[28] Though our Induction-Step Fact is stated within the induction step of Claim 5.4, it applies also to the invocation of `Simulate` within the "main procedure."

[29] The proof also ignores the fact that the verifier's commitments are only computationally binding. This "abstraction" can be removed in standard ways: Assume that V^*, in a run with the simulator, decommits to one of its commitments in two different valid ways with some probability $p(n)$. Then, it is easy to construct an algorithms that breaks the computational binding property of the commitment scheme in use with probability $\mathrm{poly}(p(n))$.

hold public-keys. (A similar use was independently suggested by Damgård [11, 12]. See discussion below.)

6.1 The Public Key Model

In the mildest form of the public-key model, users are assumed to have deposited a public-key in a public file that is accessible by all users at all times. In fact, it is only necessary for the verifiers in our protocol to have public-keys. Access to this file may be implementable by either providing access to several identical servers, or by providing users with certificates for their deposited public-keys. The sole assumption is that entries in the public-file were deposited before any interaction among the users takes place. But no assumption is made on whether or not the public key deposited are unique or "non-sensical" or "bad" (e.g., for which no corresponding secret key exist or are known) public keys.

We use such a public-file simply for limiting the number of different identities that a potential adversary may assume – it may indeed try to impersonate any registered user, but it cannot act on behalf of a non-registered user.

We analyze our solutions in an "idealized" setting where the registration to the public file is complete before any interaction starts. A more realistic public-key model allows users to register at all times. Note, however, that formally speaking such flexibility requires some restriction as otherwise it will coincide exactly with the the case in which no set-up stage or special model is used. We thus suggest two intermediate augmentations of the public-key models in which we can obtain our result (We note that others are possible but we defer discussion of those for another paper.)

One augmentation is to enforce a time lag between when a public-key is registered and the first time it will be used in an actual protocol. Namely, a prover will not interact with a verifier unless the verifier's public-key was registered a sufficiently long time before the interaction starts, where "sufficiently long" is chosen so as to ensure that whatever sessions were in progress before registration occurred have terminated by the first time the key registered will be used. This implies that users need be able to distinguish between some predetermined large delay (that all newly registered public-keys must undergo before being used) and a small delay (that upper bounds the communication delays in actual interaction).

Making such a distinction is quite reasonable in practice (e.g., say that a user in nowadays internet may start using its key a couple of days after registration, whereas each internet session is assumed to be completable within a couple of hours). Notice that, unlike usage of timing in [15], our usage of timing here *does* NOT *affect typical interactions*, which can be and actually are completed much faster than the conservative upper bound (of message delay) being used. In contrast, in [15] each user delays each critical message by an amount of time that upper bounds normal transmission delay. This means that all communication is delayed by this upper bound. Thus, in their case, this *always* causes *significant* delays: in fact the upper bound should be conservative enough so to guarantee that communication by honest users are rarely rejected.

The second augmentation of the public-key model possible to require newly registered public-keys to be used only after authorization by a trusted "switchboard", which go through an interactive protocol with the new user and then issue a certificate that will allow it to act as a verifier. We stress that users that register *at set-up time* are not required to interact with a server (or a switchboard): they merely deposit their public-key via a one-sided communication. This alternative seems better suited to the smart-card application discussed in the introduction.

Moreover, the fact that registration is only required of verifiers is nicely suited to smart-card applications in which the provers are played by the smart-cards and the verifiers by service providers. In such applications service providers are much fewer in number, and are anyhow required to undergo more complex authorization procedures (than the smart-card users).

Definition 12 resettable zero-knowledge in the public-key model *is defined similarly to* rZK *in the standard model (Definition 1), with the following exceptions:*

- *Before any interaction begins, the verifier generates a* public file *that contains identities of potential verifier-incarnations, together with information associated with each identity. (In fact, the identities of verifier-incarnations are only implicit in the file, or in other words they are understood as the numeral of the entry in the file.) The algorithm for generating the file is part of the description of the (honest) verifier.*
- *The first verifier message in any interaction should contain an identity that appears in the public file. In other words, the completeness and and soundness requirements are made only in case that the first verifier message contains an entry in the public file.*
- *The* rZK *requirement remains unchanged, with the exception that the verifier V^* generates an arbitrary public file before any interaction begins. It is stressed that in all the sessions the prover has access to the same instance of the public file.*

A more imposing model (i.e., assuming stronger set-up assumptions), which is still quite reasonable in practice, augments the public-key model by allowing ("validating") interaction between users and system manager at deposit time. In general, the preprocessing model postulates that before any interaction among users takes place, the users have to interact with a system manager that provides them with suitable certificates (in case it did not detect cheating at this stage). In particular, one may use the preprocessing stage in order to verify that the user knows a secret-key for the public-key it wishes to have certified.

6.2 Overview

We start with an overview of our results and techniques in the public-key model.

Theorem 13 *Under the strong DLP assumption, there exist* constant-round *resettable zero-knowledge arguments for $\mathcal{NP}$ in the public-key model.*

Recall that *arguments* (a.k.a computationally-sound proofs)[9] are a weaker notion than interactive proofs [32]: it is infeasible rather than impossible to fool the verifier to accept wrong statements with non-negligible probability.

Since concurrent zero-knowledge are a special case of resettable zero-knowledge, we obtain:

Corollary 14 *Under the strong DLP assumption, there exist* constant-round *concurrent zero-knowledge arguments for $\mathcal{NP}$ in the public-key model.*

6.2.1 Techniques

Several techniques used by our construction are worth singling out. First, in all messages of the prover which require randomization, the prover will use, instead of fair coins, the result of applying a pseudo random function [23] to the prover's input and the sequence of messages exchanged in the interaction thus far. (In fact, it suffices to apply the pseudo random function only to the input plus some critical parts of the communication.) This ensures that on the same prefix of an interaction, the verifier will always get the same response from the prover. Thus, the verifier will not be able to collect different responses of the prover to the same questions – a capability which can lead to breaking the security of the protocol, and is an obvious attack strategy for a verifier who can run several executions of the protocol each time resetting the prover to the same initial state and same random tape.

Second, the public-key i which the verifier deposited in the public-file is used to specify a perfectly hiding (and computationally binding) commitment scheme $Comm_i$ for the prover to use during the protocol when he encrypts the coloring of graph which he attempts to show is 3-colorable (Recall that in a top level, the prover is trying to convince the verifier that a graph is 3-colorable). The first phase of the protocol is a sub-protocol in which the verifier convinces the prover that he (the verifier) knows the secret key that matches the public key i. The knowledge of such secret key enables decommitting values committed to using $Comm_i$ in more than one way. One must be careful that this sub-protocol will not leak too much knowledge about this secret-key as otherwise the soundness of the global protocol will be compromised. We did not construct a full fledged zero-knowledge proof of knowledge sub-protocol for this task as we do not know of the existence of one which runs in constant rounds and will maintain its soundness even when its verifier can be reset (note: the verifier in the sub-protocol is actually the resettable prover in the global protocol). Instead we use a constant round proof of knowledge which can be simulated in sub-exponential time, and argue that such simulation is sufficient to prove global soundness as otherwise our assumption that commitment scheme secure against sub-exponential time exists will be violated.

Third, our construction uses actually two secure commitment schemes which interact in a novel way, One commitment scheme is with security parameter K and the other with a smaller security parameter k. We assume that , for some $\epsilon > 0$, the security of the first commitment scheme (with security parameter K) is maintained against adversaries running in time 2^{K^ϵ},[30] and that instances of the second scheme (with security parameter k) can be broken in time 2^k. Then setting $k = K^\epsilon/2$ guarantees both security of the second scheme as well as "non-malleability" (cf. [13]) of the first scheme in presence of the second one. The reason for the latter fact is that breaking the second scheme can be incorporated into an adversary attacking the first scheme without significantly effecting its running-time: Such an adversary is allowed running-time 2^{K^ϵ} which dominates the time $2^k = 2^{K^\epsilon/2}$ required for breaking the second scheme. This "telescopic" usage of intractability assumptions can be generalized to a case in which we have a lower and upper bound on the complexity of some problem; specifically, we need a lower bound $L(n)$ on the average-case of solving n-bit long instances, and an upper-bound $U(n) \gg L(n)$ on the corresponding worst-case complexity. Suppose that we can choose polynomially-related security parameters k and K so that $L(k)$ is infeasible and $U(k) \ll L(K)$ (i.e., $L(k)$ is infeasible and $U(k) \ll L(\text{poly}(k))$). Then the above reasoning still holds. (Above we used $L(n) = 2^{n^\epsilon}$ and $U(n) = 2^n$.)

Outline of the Protocol. The common input is a 3-colorable input graph G. The input of the prover is a 3-coloring of this graph and the input to the verifier is the secret-key that matches his public-key.

1. prover looks up public-key i of the verifier (no interaction required) which specifies a commitment scheme $Comm_i$ (which is perfectly private and computationally binding and has security parameter K)

[30] The strong DLP assumption is used to guarantee security against adversaries running in time 2^{K^ϵ} (rather than in polynomial-time).

2. verifier runs a sub-protocol in which he convinces the prover that he knows the matching secret key to i

3. verifier commits to a sequence of edges $e_1, ...e_n$ of the graph G using 2nd commitment scheme $Comm'$ (which is also perfectly private and computationally binding but its security parameter is k)..

4. prover commits independently to n copies $EG_1, ..., EG_n$ of the graph G, each copy is colored with a permutation of the 3-coloring which the prover knows, using the commitment scheme $Comm_i$ specified by i.

5. verifier decommits edges e_1,..., e_n.

6. prover decommits for each EG_i the colors of the end points of the edge e_i

7. the verifier accepts if indeed all edges e_i were colored properly and rejects otherwise.

Almost constant-round rZK **under weaker assumptions.** We mention that using the weak DLP assumption (rather than the strong one), we obtain for every unbounded function $r : \mathsf{N} \to \mathsf{N}$, an $r(\cdot)$-round resettable zero-knowledge argument for $\mathcal{NP}$ in the public-key model. Again, such protocols are concurrent zero-knowledge (as a special case).

6.3 A constant-round rZK protocol

The main result of this section is a construction of constant-round computationally-sound resettable zero-knowledge proof systems. Here we use two-round perfect commitment schemes with some additional features (to be specified below). Such schemes exist assuming that DLP is hard for sub-exponential circuits. Thus, as a special case, we obtain:

Theorem 15 *Suppose that for some $\epsilon > 0$ and sufficiently large n's, any circuit of size 2^{n^ϵ} solves DLP correctly only on a negligible fraction of the inputs of length n. Then every language in $\mathcal{NP}$ has a* constant-round *resettable zero-knowledge computationally-sound proof system* in the public-key model. *Furthermore, the prescribed prover is resettable zero-knowledge via a black-box simulation.*

In the proof below, we only refer to the sequential single-incarnation variant of resettable zero-knowledge in the public-key model. The treatment extends easily to the sequential multiple-interaction model, and the equivalence (to the general interleaving variant), proven in Theorem 2, holds here too.

6.3.1 rZK for NP in the preprocessing model

We first present a resettable zero-knowledge protocol for a model allowing preprocessing (i.e., a model which has stronger set-up assumptions). The preprocessing will be used in order to guarantee that verifiers know "trapdoors" corresponding to "records" deposited by them in the public file.

The protocol uses two types of perfect commitment schemes; that is, secrecy of commitment holds in an information theoretic sense, whereas the binding property holds only in a computational sense. The two commitment schemes used have some extra features informally stated below. For a precise definition see Appendix A.

1. A two-round perfectly-hiding commitment scheme, denoted PC1, with two extra features:

- *The trapdoor feature*: It is possible to efficiently generate a receiver message (called the *index*) together with a trapdoor, so that knowledge of the trapdoor allows to decommit in any way.

  Note that the first message in a two-round commitment scheme is from the commitment-receiver to the commitment-sender. The trapdoor feature says that the receiver will be able to decommit to the sender's message in any way it wants (but as usual the sender, not knowing the trapdoor, will not be able to do so).

  In our solution we will "decouple the execution" of the two-round commitment scheme so that the first message (i.e., the index) will be sent in a preliminary stage (i.e., will be deposited in a public-file), and only the second message will be send in the actual protocol. We stress that the same index can and will be used for polynomially many commitments, and that the number of such commitments need not be a-priori known. (Note that both perfect secrecy and computational-binding continue to hold also under such "recycling" of the index.)

- *The strong computational-binding feature*: The computational-binding property holds also with respect to sub-exponential circuits. That is, there exists a constant $\epsilon > 0$ so that for sufficiently large security parameter K no sender strategy that is implementable by a circuit of size 2^{K^ϵ} can, given a random K-bit index, produce a single commitment together with two conflicting decommitments (i.e., to two different values) with probability greater than 2^{-K^ϵ}.

2. A constant-round perfectly-hiding commitment scheme, denoted PC2. Without loss of generality, we may assume that the binding property can be violated in exponential time. That is, when the commitment protocol is run on security parameter k, the sender may in time 2^k decommit any way it wants.

Indeed, any PC1 scheme yields a PC2 scheme. However, for sake of modularity we prefer the current presentation. We also note that for our application it is possible to further relax the requirement from PC2 so that secrecy may be demonstrated to hold at a latter stage (i.e., "a posteriori"); see [22, Sec. 4.8.2]. We comment that a PC1 scheme can be constructed under the assumption the DLP is hard for sub-exponential circuits; see details in Appendix A. More generally, one may use any pair of trapdoor claw-free permutations, provided the claw-free property holds w.r.t sub-exponential circuits.[31]

The protocol in the preprocessing model: The inputs to the protocol are as follows.

Security parameter: K. All objects (resp., actions taken) in the protocol have size poly(K) (resp., are implementable in poly(K)-time).

Common input: A graph $G = (V, E)$, where $V = [n] \stackrel{\text{def}}{=} \{1, ..., n\}$, claimed to be 3-colorable.

In addition, a public file containing a list of indices (i.e., receiver's message for PC1), generated by verifiers on security parameter K. Each verifier need only deposit a single index in the public file, which may be stored under its name. We consider also cheating verifiers who may deposit polynomially many such indices. We stress however that the number of entries in the public-file should be bounded by some fixed polynomial.

[31] In fact, it suffices to have collision-intractable family of hashing function, provided it carries trapdoors and is strong with respect to sub-exponential circuits.

At this point we assume that the verifier knows a trapdoor to any index it has deposited. This can be enforced by a preprocessing stage, say, via a zero-knowledge proof of knowledge.

Verifier's auxiliary input: A trapdoor, denoted $\mathrm{trap}(i)$, for some index i in the public file.

Prover's auxiliary input: A 3-coloring $\phi : [n] \to \{1,2,3\}$ of G.

Prover's initial randomization: The prover's random-tape is used to determine a pseudorandom function $f : \{0,1\}^{\mathrm{poly}(n)} \to \{0,1\}^{\mathrm{poly}(n)}$.

The protocol itself is an adaptation of the resettable witness indistinguishable proof system of Section 4, with Step (P1) being replaced (or rather implemented) by current Steps (1) and (3). Another important change is the replacement of former Step (V1) by current Step (2); the difference being that commitment via a standard commitment scheme (with perfect binding) is replaced by a commitment relative to a (perfect secrecy) scheme that is only computationally-binding.

(1) The verifier sends an index i to prover, who checks that it appears in the public-file. (Otherwise the prover aborts.)

Note that this step may be viewed as transcendental to the protocol, since it amount to the verifier telling the prover its identity. [Indeed, a cheating verifier may lie about its identity; we merely rely on the fact that somebody knows the trapdoor to the index i if indeed it is in the public file. Since we view the adversary as controlling the entire "world outside the prover" it really does not matter who knows the trapdoor.]

(2) This step is analogous to Step (V1) in the protocol of Section 4: The verifier commits to a sequence of $t \stackrel{\mathrm{def}}{=} n \cdot |E|$ uniformly and independently chosen edges. The commitment is done using the constant-round perfectly-hiding commitment scheme PC2, in which the verifier plays the role of the sender and the prover plays the role of the receiver. The scheme PC2 is invoked while setting the security parameter to $k = K^\epsilon/2$, where $\epsilon > 0$ is as specified in the strong binding feature of PC1. The randomization required for the actions of the receiver in PC2 is determined by applying the pseudorandom function f to $(G, \phi, \mathit{history})$, where *history* is the transcript of all messages received by the prover so far.

Thus, the prover gets *no information* on the committed edges, while it is infeasible for the verifier to "de-commit" in two different ways.

[The analysis makes heavy use of the setting of the security parameter $k = K^\epsilon/2$. On one hand, this setting guarantees that a quantity that is polynomial in K is also polynomial in k. On the other hand, time 2^k which suffices to violate the computational-binding property of PC2 when run on security parameter k, is insufficient to violate the strong computational-binding property of PC1 when run on security parameter K (since $2^k = 2^{K^\epsilon/2} \ll 2^{K^\epsilon}$).]

(3) This step is analogous to Step (P1) in the protocol of Section 4: The prover uses PC1 with index i in order to commit to a sequence of t random colorings. That is, the prover invokes t instances of protocol PC1 playing the sender in all of them, and acts as if it has received i (the index) in all these instances.

Recall that the prover wishes to commit to $t \cdot n$ values, the $(jn+v)^{\mathrm{th}}$ value being the color assigned to vertex v by the j^{th} random coloring (i.e., the j^{th} random relabeling of ϕ, selected among the six permutations of the colors $\{1,2,3\}$). All randomizations (i.e., the choice of the random coloring as well as randomization required by PC1) are determined by applying the pseudorandom function f to $(G, \phi, history)$, where *history* is the transcript of all messages received by the prover so far.

(4) The verifier decommits to the edge-sequence it has committed to in Step (2). That is, it reveals the sequence of t edges, as well as the necessary information required to determine the correctness of the revealed values. [This step is analogous to Step (V2).]

(5) In case the values revealed (plus the "de-commitment" information) in Step (4) match the commitments sent in Step (2), and in case all queries are edges, the prover reveals the corresponding colors and provides the corresponding de-commitment. [This step is analogous to Step (P2).]

(6) In case the values revealed (plus the "de-commitment") in Step (5) match the commitments sent in Step (3), and in case they look as part of legal 3-colorings (i.e., each corresponding pair is a pair of different elements from the set $\{1, 2, 3\}$), the verifier accepts. Otherwise it rejects. [This step is analogous to Step (V3).]

We note that, in the above description of the protocol, the verifier does not use the trapdoor (i.e., trap(i)). The fact that the verifier (or rather an adversary controlling all possible verifiers) knows the trapdoor will be used by the simulator which is rather straightforward: In contrast to standard constructions of simulators (cf., [32, 28]), the current simulator does not "rewind" the verifier. Instead, it simulates an execution of the protocol by emulating the actions of the prover in Steps (1)–(4), using some dummy sequence (rather than a sequence of colorings) in Step (3). However, when getting to Step (5), and in case the verifier has decommitted properly, the simulator uses trap(i) in order to decommit to the corresponding edge queries in a random legal-looking way (i.e., it decommits to a uniformly and independently chosen pair of distinct colors, for each such edge). This uses the trapdoor feature of PC1 and the hypothesis that the verifier (and so the simulator) knows this trapdoor. The above description corresponds to simulation of the first session with the prover. Subsequent sessions are simulated in the same way assuming that the execution of Steps (1)–(2) of the current session is different than in all previous sessions. Otherwise, we simulate Steps (3) and (5) by copying the values used in the previous session. A last issue to be addressed is the possibility that in two executions of the protocol the verifier may send the same messages in Step (2) but latter decommit in two different ways in Step (5), in which case the output of the simulator may be noticeably different from the output in real executions. Using the computational-binding property of the scheme PC2, we argue that this event may only occur with negligible probability. This establishes the resettable zero-knowledge property of the above protocol (in the preprocessing model).

Observe that the computational-binding property of PC1 allows computationally-unbounded provers to successfully fool the verifier, and hence the above protocol does not constitute an interactive proof. However, one can show that computationally-bounded provers can fool the verifier only with negligible probability, and so that the protocol is computationally-sound.

Proposition 16 (informal): *The above protocol is computationally sound.*

Proof Sketch: Intuitively, one would like to argue that the computational-binding property of PC1 does not allow to decommit to two different values in Step (5). The problem is that the prover commits to colors in Step (3) after obtaining the verifier's commitment to queries, and that the prover decommits only after the verifier decommits. How can we rule out the (intuitively unlikely) possibility that the verifier's decommitment allows the prover to decommit accordingly (in a way it could not have done before getting the verifier's decommitment)? Here we use the strong computational-binding property of PC1 (relative to security parameter K); that is, the fact that it holds also with respect to circuits of size $2^{K^\epsilon} = 2^{2k}$. We also use the fact that commitments with PC2 were done while setting the security parameter to k, and so we can decommit any way we want while using time 2^k. Thus, the binding property of PC1 has to be maintained in Step (5); i.e., it should be infeasible to decommit "at will" in Step (5) also after obtaining the decommitment of the verifier at Step (4).

In the actual proof we consider what happens in Step (5) when the prover interacts with an imaginary verifier that at Step (4) uniformly selects new queries and decommits according to these values. Observe that such an imaginary verifier can be implemented within time $\text{poly}(n) \cdot 2^k$. Thus, if we consider the mental experiment in which Steps (4)-(5) are repeated $T = 2^{k/3}$ times, after a single execution of Steps (1)-(3), then all proper decommits by the prover must be for the same value (or else the binding property of PC1 is violated in time $T \cdot \text{poly}(n) \cdot 2^k \ll 2^{2k}$). Furthermore, the above should hold for at least $1 - T^{-1}$ fraction of random executions of Steps (1)-(3). Thus, if we consider a computationally-bounded prover that fools the verifier, only a term of $O(2^{-k/3})$ in its success probability may be attributed to "ambiguous decommitment". The computational-soundness of the protocol follows by noting that $(1 - |E|^{-1})^t) \approx e^{-n}$ is an upper bound on the probability of fooling the verifier in case commitments are non-ambiguous. This establishes the computationally-soundness of the above protocol. ■

6.3.2 Back to the (bare) public-key model

Given the above, all that is needed in order to adapt the protocol to the public-key model is to replace the assumption that the verifier knows the trapdoor by a (zero-knowledge) proof-of-knowledge of this claim. We stress that the verifier in the above protocol will play the role of knowledge-prover, whereas the main prover will play the role of a knowledge-verifier. This protocol has to maintain its soundness also when the knowledge-verifier undergoes "rewinding". Furthermore, it should be constant-round. (We comment that we are not aware of a known protocol satisfying these strong requirements.) On the other hand, we don't need "full-fledged" zero-knowledge property; simulatability in sub-exponential time will suffice (as it is merely used for the computational-soundness property which is established based on the strong computational-binding property of PC1, which in turn accounts for such running times too). Thus, Step (1) in the above protocol is augmented by a constant-round proof-of-knowledge (POK) which proceeds as follows:

The parties: A knowledge-verifier, denoted KV, played by the main prover, and a knowledge-prover, denoted KP, played by the main verifier.

Inputs: Common input $i \in \{0,1\}^K$.

Furthermore, KP gets as auxiliary input the randomness used to generate i (equiv., to generate $(i, \text{trap}(i))$).

Goal: KP wants to prove that it knows $\text{trap}(i)$.

High level: We present a proof of knowledge (POK) of the relevant NP-witness; that is, POK of the randomness used to generate i. (Such knowledge yields knowledge of $\text{trap}(i)$.) The POK is via the standard reduction of this NP-relation to the NP-relation corresponding to Hamiltonicity (which is NP-Complete). We stress that the standard reduction comes with efficient transformation of NP-witnesses from the original relation to the target Hamiltonicity relation and vice versa. Thus, the auxiliary-input of KP allows to efficiently compute a Hamiltonian cycle in the target graph, and from any such Hamiltonian cycle one may efficiently retrieve trap(i).

The proof of knowledge (POK) of Hamiltonicity is based on Blum's proof system for this language, which is reproduced in Appendix B. An important property of Blum's *basic protocol* is that it is a "challenge–response" game in which the challenge consists of a single bit. Furthermore, responding correctly to both possible challenges allows to extract a Hamiltonian cycle (i.e., the knowledge claimed).[32] This property simplifies the knowledge extraction argument in case many copies are played in parallel: Ability to respond to any two different sequences of challenges yields a Hamiltonian cycle. Below we run the protocol k times in parallel, where $k = K^\epsilon/3$. The resulting protocol will have negligible knowledge-error[33] (i.e., error of 2^{-k}), and will be simulatable in time $\text{poly}(K) \cdot 2^k$. Furthermore, the simulation will be indistinguishable from the real interaction by any 2^{K^ϵ}-size circuits. As stated above, we are not concerned of the fact that the protocol may not be zero-knowledge (i.e., simulatable in $\text{poly}(K)$-time).

The protocol uses a perfectly-binding commitment scheme with strong computational-secrecy; that is, circuits of size 2^{K^ϵ} cannot distinguish commitments to two different known values (with distinguishing gap better than 2^{-K^ϵ}). Such a scheme can be constructed based on the DLP assumption utilized above.

(pok1) Using the perfectly-binding commitment scheme, KP commits to each of the entries of $k = K^\epsilon/3$ matrices, each generated as in Blum's basic protocol. (That is, each matrix is the adjacency matrix of a random isomorphic copy of the graph obtained from the reduction. In case the output of the reduction is a graph with N vertices, the commitment scheme is applied $k \cdot N^2$ times.) The commitment scheme is run with security parameter K.

(pok2) KV "randomly" selects a sequence $c = c_1 \cdots c_k \in \{0,1\}^k$ of k challenges. Actually, the sequence c is determined by applying the pseudorandom function f to the input (i.e., the index i) and the history so far (of the POK protocol).

(pok3) KP answers each of the k bit queries as in Blum's basic protocol. (That is, if $c_j = 0$ then KP decommits to all entries of the j^{th} matrix and also reveals the isomorphism; otherwise, KP decommits only to the entries corresponding to the Hamiltonian cycle. Note that the location of the latter entries is determined by applying the isomorphism to the original cycle.)

(pok4) KV accepts if and only if all answers are valid. Specifically, in case $c_j = 0$, KV checks that the revealed matrix is indeed isomorphic (via the provided isomorphism) to the matrix representing the reduced graph. In case $c_j = 1$, KV checks that all revealed entries are indeed 1's. (In both cases, for each revealed value, KV checks that the decommitment is valid.)

The weak zero-knowledge property is easy to establish. That is, we need and do show that the interaction with any (possibly dishonest but computationally-bounded) knowledge-verifier can be simulated in time $\text{poly}(k) \cdot 2^k$. This follows by merely using the standard simulator procedure (cf., [32, 28]), which merely selects a random string $c \in \{0,1\}^k$ and "simulates" Step (**pok1**) so

[32] This property holds also for other protocols for NP, but not for the 3-Colorability protocol of [28]. Any protocol having the property will do.

[33] Loosely speaking, the knowledge-error is the probability that the verifier may get convinced by a cheating prover who does not know a Hamiltonian cycle. For a precise definition, see Appendix B.

that it can answer the challenge c (but not any other challenge). The strong computational-secrecy of the commitment scheme (used with security parameter K) guarantees that the knowledge-verifier cannot guess c better than with probability approximately 2^{-k}, and so we will succeed with overwhelming probability after at most $k \cdot 2^k$ tries. Standard arguments will also show that the output of the simulator cannot be distinguish from the real interaction by circuits of size $2^{K^\epsilon - 1} > 2^{2k}$. Thus, this simulator can be plugged into the argument given above for computational-soundness in the case of preprocessing, and yield that the augmented protocol maintains computational-soundness: The potentially cheating prover in the main protocol induces a cheating knowledge-verifier, and what the simulation says is that in case the verifier (playing the knowledge-prover) follows the protocol then whatever the knowledge-verifier can compute after interacting with it, can also be computed with overhead of at most $\mathrm{poly}(k) \cdot 2^k$ on input the index i. Thus, we have

Proposition 17 (informal): *The modified main protocol is computationally sound.*

We now turn to establish the resettable zero-knowledge property of the entire protocol. *As a first step towards this goal*, we establish that the above sub-protocol is indeed a POK with knowledge-error 2^{-k} (see Def. 25 in Appendix B). In other words, we analyze a single execution of the sub-protocol, and thus we may assume that Step (**pok2**) is replaced by sending a truly random string c. This assumption is not valid when the sub-protocol is run many times, and this is why the simplified analysis provided here does not suffice. However, it does provide a good warm-up.

Without loss of generality, consider a deterministic cheating knowledge-prover, and let C be the message sent by it in Step (**pok1**). Consider the probability space of all 2^k possible challenges $c \in \{0,1\}^k$ that KV may send in Step (**pok2**). Say that a challenge $c \in \{0,1\}^k$ is *successful for this knowledge-prover* if its answer in Step (**pok3**) is accepted by KV in Step (**pok4**). The key observation is that given the knowledge-prover's answer to *any two different successful challenges* we can easily reconstruct the Hamiltonian cycle (and from it easily reconstruct the trapdoor).[34] To extract the Hamiltonian cycle we just invoke the knowledge-prover many times, each time it answers with the same Step (**pok1**) message but then we challenge it with a new randomly chosen c (i.e., chosen independently of all prior attempts). If we ever obtain its answer to two successful challenges then we are done. Denoting by p the probability that a uniformly chosen challenge is successful, we conclude that if $p > 2^{-k}$ then given oracle access to the knowledge-prover (played by the adversary) we can (with overwhelmingly high probability) find the trapdoor in time $\mathrm{poly}(k)/(p - 2^{-k})$. By a trivial modification, we obtain a knowledge extractor which for any $p > 0$ with overwhelming probability runs for time $\mathrm{poly}(k)/p$, and in case $p > 2^{-k}$ also retrieves the trapdoor.[35]

The above argument would have suffices if we were guaranteed that the adversary, when playing the role of KP, never repeats the same Step (**pok1**) message (in two different sessions of the entire protocol). Let use assume so and see how, under this unjustified assumption (which will be removed later), the resettable zero-knowledge property follows.

Consider a sequence of invocations (sessions) of the main protocol. The simulator will proceed by simulating one session after the other, where a single session is simulated as follows. The simulator starts by playing the role of KV in Step (1). In case KV rejects then the simulator complete the simulation of the current session by announcing that the prover aborts it. Note that this is exactly what would have happened in the real interaction. In case KV accepts, the simulator will use the knowledge-extractor described above in order to extract the trapdoor of the

[34] This is the case since each such pair of challenges differs at some location, and from the two answers to this location we may reconstruct the Hamiltonian cycle.

[35] This can be done by using a time-out mechanism invoked when $\mathrm{poly}(k) \cdot 2^k$ steps are completed, and observing that if $p > 2^{-k}$ then in fact $p \geq 2 \cdot 2^{-k}$ and so $(p - 2^{-k})^{-1} \leq 2/p$.

index i sent in Step (1). Here is where we use the assumption that the adversary does not repeat the same Step (**pok1**) message. The point is that the knowledge-extractor described above will try many different challenges for Step (**pok2**). Since the challenge is determined as a "random" function evaluated at a new point (here is where we use the "no repeat" clause), we may view this challenge as random. Thus, the above analysis applies. The conclusion is as follows. Suppose that the cheating verifier convinces KV with probability p, We distinguish three cases. In case $p = 0$, the simulator will always construct an aborting execution (just as in the real interaction). In case $p > 2^{-k}$, with probability $1 - p$ the simulator will construct an aborting execution (just as in the real interaction), and otherwise using time $\text{poly}(k)/p$ it finds the trapdoor of the index i sent in Step (1), which allows it to complete the simulation of Steps (2)–(6) just as done above (in the case of preprocessing). Note that the *expected* number of steps required for the simulation in this case is $(1-p) \cdot \text{poly}(k) + p \cdot (\text{poly}(k)/p) = \text{poly}(k)$. The only case left is the one where $p = 2^{-k}$. In this case, the simulator fails with probability p, which is negligible, and so its output is computationally indistinguishable from a real interaction. We stress that in all cases the simulator runs in expected time $\text{poly}(k)$.

Having concluded all these warm-ups, we are now ready to deal with reality. The difficulty occurs when the adversary uses the same index and same Step (**pok1**) message in two different sessions with the prover. Furthermore, suppose that in the first session it fails to convince KV played by the prover, but in the second session it succeeds. The problem (avoided by the assumptions above) is that we cannot use a different challenge (i.e., message for Step (**pok2**)) in the second session, since the challenge is determined already by the first session. Thus, the simulator cannot complete the simulation of the second session, unless it "rewinds" up to the first session in which the same Step (**pok1**) message is used.[36] This need to "rewind" sessions that were already completed may lead to exponential blow-ups as discussed by Dwork, Naor and Sahai [15]. What saves us here is that the number of times we possibly need to "rewind" is a-priori bounded by the total number of indices in the public file. (This is the key and only place where we use the assumption underlying the public-key model.)

The heart of the analysis – a sketch: Let us reproduce and further abstract the problem we need to analyze. Recall that we will consider only non-interleaving (i.e., sequential) adversaries. We are dealing with a game consisting of multiple (history dependent) iterations of the following steps, which depends on a random function f fixed once and for all.

(a) The verifier sends a pair (i, C), where i belongs to some fixed set I and C is arbitrary. This pair is determined by applying the verifier's strategy, V^*, to the history of all previous iterations (of these steps).

[Indeed, i corresponds to the index sent in Step (1), I to the public file, and C to the message of Step (**pok1**).]

(b) The prover determines a k-bit string, $c = f(i, C)$, by applying f to the pair (i, C).

[This corresponds to Step (**pok2**) of KV played by the prover.]

(c) The verifier either succeeds in which case some additional steps (of both prover and verifier) take place or it fails in which case the current execution is completed.

[36] We comment that in general, a simulator for resettable zero-knowledge may not proceed by generating the sessions one after the other without "rewinding" between different sessions.

[This corresponds to whether the verifier, playing KP, has provided a valid decommitment in Step (**pok3**), and to the continuation of the main protocol which takes place only in case the verifier has done so.]

We want to simulate an execution of this game, while having oracle access to the verifier's strategy (but without having access to the prover's strategy, which enables the further steps referred to in Step (c) above). Towards this goal we are allowed to consider corresponding executions with other random functions, $f', f'', \ldots$, and the rule is that whenever we have two different successes (i.e., with two different challenges c) for the same pair (i, C) we can complete the extra steps referred to in Step (c). [This corresponds to extracting the trapdoor of i, which allows the simulation of the rest of the steps in the current interaction of the main protocol.]

Thus, problems in simulating the above game occur only when we reach a successful Step (c). In such a case, in order to continue, we need a different success with respect to the same pair (i, C). In order to obtain such a different success, we will try to run related simulations of the game. Once we find two successes for the same pair (i, C), we say that i is covered, and we may proceed in the simulation temporarily suspended above. That is, a natural attempt at a simulation procedure is as follows. We simulate the iterations of the game one after the other, using a random function f selected by us. Actually, the random function f is defined iteratively – each time we need to evaluate f at a point in which it is undefined (i.e., on a new pair (i, C)) we randomly define f at this point. As long as the current iteration we simulate fails, we complete it with no problem. Similarly, if the current iteration is successful relative to the current pair (i, C) and i is already covered, then we can complete the execution. We only get into trouble if the current iteration is successful relative to (i, C) but i is not covered yet. One natural thing to do is to try to get i covered and then proceed. (Actually, as we shall see, covering any new element of I, not necessarily i, will do.)

Starting with all I uncovered, let us denote by p the probability that when we try to simulate the game a success occurs. Conditioned on such a success occurring, our goal is to cover some element of I within expected time $\text{poly}(k)/p$. Suppose we can do this. So in expected time $(1-p) \cdot \text{poly}(k) + p \cdot (\text{poly}(k)/p) = \text{poly}(k)$ we either completed a simulation of the entire game or got some $i \in I$ covered. In the first case, we are done. In the second case, we start again in an attempt to simulate the game, but this time we have already i covered. Thus, we get into trouble only if we reach a success relative to (i', C) with $i' \in I' \stackrel{\text{def}}{=} I \setminus \{i\}$. Again, we may denote by p' the probability that when we try to simulate the game a success occurs with respect to some $i' \in I'$. In such a case, we try to cover *some* element of I', and again the same analysis holds. We may proceed this way, in up to $|I|+1$ phases, where in each phase we either complete a random simulation of the game or we get a new element of I covered in each iteration. Eventually, we do complete a random simulation of the game (since there are more phases than new elements to cover). So, pending on our ability to cover new elements within time inversely proportional to the probability that we encounter a success relative to a yet uncovered element, each phase requires $\text{poly}(k)$ steps on the average. Thus, pending on the above, we can simulate the game within expected time $\text{poly}(k) \cdot |I| = \text{poly}(k)$ (by the hypothesis regarding I).

We now consider the task of covering a new element. Let us denote the set of currently uncovered elements by U. Let H denote the prefix of completed executions of the simulated game and let $(i, C) = V^*(H)$ be the current pair which is related to the current success, where $i \in U$. To get i covered we do the following:

1. Let H' be the maximal sequence of executions which does not contain (i, C) as a Step (a) message. Note that $H' = H$ in case the current pair (i, C) does not appear as a Step (a)

message in some (prior) execution in H.

2. Redefine $f'(i,C)$ uniformly at random, and try to extend H' (with respect to to the function f') just as we do in the main simulation (where we currently try to extend H with respect to to the function f). If during an attempt to extend H' we encounter a new (i.e., different than above) success with respect to the same pair (i,C) then i itself gets covered, and we have fulfilled our goal. Otherwise, we repeat the attempt to extend H' (with a new random choice for $f'(i,C)$) as long as we did not try more than $k \cdot 2^k$ times. In case all attempts fail, we abort the entire simulation.

   We will show that, for $p > 2^{-k}$, we will get a new element covered while making $(p - 2^{-k})^{-1}$ tries, on the average.

3. If during the current attempt to extend H' we encounter a success relative to some other pair $(i',C') \neq (i,C)$, where i' (possibly equals i) is also currently uncovered, then we abort the current extension of H' (and try a new one – again as long as $k \cdot 2^k$ tries are made).

Thus, we have

Proposition 18 (informal): *The above game can be simulated within expected* $\text{poly}(k)$*-time.*

The procedure has oracle access to the adversary V^*, and calls the procedure `Extend`.

(M1) Initialization: $U \leftarrow \emptyset$.
(M2) Repeat up to $|I|+1$ times
(M3) Initialization: $H \leftarrow x$ and f is totally undefined.
(M4) Let `answer` $\leftarrow$ $\texttt{Extend}^{f}_{V^*}(U,H)$.
(M5) If `answer` constitute a full simulation transcript then *halt with output* `answer`.
[Comment: Otherwise `answer` $= (H,(i,C),f(i,C))$, with $i \notin U$,
and $V^*(H,(i,C),f(i,C))$ is successful. Our aim now is to cover i]
(M6) Let H' be the maximal prefix of H satisfying $V^*(H') = V^*(H)$, and let $r = f(i,C)$.
(M7) Repeat up to $k \cdot 2^k$ times
(M8) Redefine $f(i,C)$ at random different than r:
That is, select r' uniformly in $\{0,1\}^k \setminus \{r\}$ and let $f(i,C) \leftarrow r'$.
(M9) Let `answer` $\leftarrow$ $\texttt{Extend}^{f}_{V^*}(U,H')$.
[Comment: `answer` is an extension of H'.]
(M10) If `answer` contains a success with respect to (i,C) then $U \leftarrow U \cup \{i\}$ and goto (M2).
[Comment: In this case we have two different successes w.r.t (i,C),
since $f(i,C) = r' \neq r$. Thus, i got covered.]
[Comment: Otherwise we proceed to the next iteration of (M7).]
(M11) End [of inner repeat loop]
(M12) In case all attempts have failed, *the procedure aborts with an error message.*
(M13) End [of outer repeat loop]

Figure 2: The main simulation procedure

A more precise description is provided in Figures 2 and 3, and the actual analysis presented below refers to this formal description. The main procedure (of Figure 2) attempts $|I|+1$ times to generate

The procedure is invoked with some set $U \subseteq I$, partial transcript H and partially defined function f. Specifically, it is either invoked with a trivial partial transcript (i.e., $H = x$) or with H so that $(i, C) \stackrel{\text{def}}{=} V^*(H)$ and $i \notin U$. In the latter case, f is defined on (i, C), and $V^*(H, (i, C), f(i, C))$ succeeds.

$\texttt{Extend}^f_{V^*}(U, H)$

(E1) Repeat till $V^*(H)$ halts

(E2) $(i', C') \leftarrow V^*(H)$ (assuming $V^*(H)$ does not halt).

(E3) If f is not defined on (i', C') then select r' uniformly in $\{0,1\}^k$ and let $f(i', C') \leftarrow r'$.

(E4) If $V^*(H, (i', C'), f(i', C'))$ fails then $H \leftarrow (H, (i', C'), f(i', C'), \perp)$ and goto (E1).
[Comment: Otherwise $V^*(H, (i', C'), f(i', C'))$ succeeds.]

(E5) If i' is covered (i.e., $i' \in I \setminus U$) then complete H as in Step (c) and goto (E1).
[Comment: Otherwise i' is not covered, and we return a partial transcript.]

(E6) $\texttt{return}(H, (i', C'), f'(i', C'))$.
[Comment: If $(i', C') = (i, C)$ we return a transcript containing a success w.r.t (i, C).]

(E7) End [of repeat loop]
[Comment: Reaching this point means completion of simulation.]

(E8) $\texttt{return}(H)$.

Figure 3: The Extend procedure

a full transcript, while constructing the random function, f, on the fly. Typically, each attempt which fails to generate a full transcript provides "progress" in the form of a new element being covered. The non-typical case, which (as we will show) occurs with negligible probability, is that neither happens. Another thing to be proven is that the expected number of times that the main procedure repeats (M8)–(M10) is inversely proportion to the probability that for uniformly chosen $r \in \{0,1\}^k$ it holds that $V^*(H', (i, C), r)$ succeeds, where H' and (i, C) are as defined in (M6). The extension of transcripts, either initial ones as in (M4) or partial ones as in (M9), is performed (in a straightforward manner) by the `Extend` procedure depicted in Figure 3. In particular, once `Extend` "gets into trouble" (reaches a success w.r.t (i', C') where i' is uncovered) it returns control to the main procedure. In case `Extend` is invoked in (M4), we next try to get i covered. In case `Extend` is invoked in (M9), if $(i', C') = (i, C)$ then we obtain a different success to the one obtained already, and consequently i gets covered.

Proposition 19 (Analysis of the main procedure): *We consider a single execution of the outer loop in the main procedure.*

1. *The total expected time spent in such an execution is* $\text{poly}(k)$.
2. *The probability that the the execution aborts with an error message is at most* $\text{poly}(k) \cdot 2^{-k}$.

Recall that, unless the execution aborts with an error message, it either completes a simulation of the game or provides a new covered element. Incorporating the abort event into the deviation of the simulator, we obtain a simulation of the game within expected time $|I| \cdot \text{poly}(k) = \text{poly}(k)$ and deviation $\text{poly}(k) \cdot 2^{-k}$.

Proof Sketch: The running-time of `Extract` is bounded by the running time of an execution of the real game, which in turn is polynomial in k. Thus, we may charge each invocation of `Extract` as unit cost. Our aim is to show that the expected charge accumulated in a single execution of the outer loop in the main procedure is poly(k).

For every partial transcript H (and every $U \subseteq I$), denote by p_H the probability that H appears as a prefix of a transcript generated by $\mathtt{Extend}_{V^*}(U,x)$. By disjointness of the events corresponding to prefixes of equal length we have $\sum_H p_H = \mathrm{poly}(k)$.

Let us call H' interesting if the following two conditions hold: (1) $V^*(H') = (i, C)$ with $i \in U$, and (2) for every prefix H'' of H', it holds that $V^*(H'') \neq V^*(H')$. For every interesting H', denote by $q_{H'}$ the probability that $\mathtt{Extend}_{V^*}(U,H')$ contains a success with respect to $V^*(H')$ and furthermore that this is the first success in the extension of H'. Note that $q_{H'}$ equals the probability that a single execution of the outer loop of the main procedure determines H' as a maximal prefix in (M6), conditioned on H' being a prefix of $\mathtt{Extend}_{V^*}(U,x)$. Thus, conditioned on the latter event, the inner loop is executed with probability $q_{H'}$. In case $q_{H'} > 2^{-k}$ (i.e., $q_{H'} \geq 2 \cdot 2^{-k}$), each iteration of the inner loop covers i with probability

$$\frac{2^k \cdot q_{H'} - 1}{2^k - 1} \;>\; q_{H'} - 2^{-k}$$

Thus, the expected number of iteration of the inner loop is less than $(q_{H'} - 2^{-k})^{-1} \leq 2/q_{H'}$. Furthermore, with probability at least $1 - 2^{-k}$, the inner loop is not repeated more than $2k/q_{H'}$ times. In case $q_{H'} = 2^{-k}$, the number of iteration of the inner loop equals $k \cdot 2^k = k/q_{H'}$. We conclude that the expected running time of a single iteration of the outer loop is at most

$$\sum_{H':q_{H'}=0} p_{H'} \cdot 1 + \sum_{H':q_{H'}>0} p_{H'} \cdot \left(q_{H'} \cdot \frac{O(k)}{q_{H'}} + (1 - q_{H'}) \cdot 1\right) \;=\; \mathrm{poly}(k)$$

and Part 1 of the proposition follows.

Part 2 follows easily by observing that the execution (of the outer loop) may be aborted only in two cases (relative to the current H' determined in (M6)). The first case is when $q_{H'} > 2^{-k}$, but (as mentioned above) in this case abort happens with probability at most $(1 - (q_{H'}/2))^{k2^k} < 2^{-k}$, since $k \cdot 2^k \geq 2k/q_{H'}$. The second case is when $q_{H'} = 2^{-k}$, but in this case we reach (M6) with probability $p_{H'} \cdot q_{H'}$. Summing over all H''s, the probability of abort is bounded above by

$$\sum_{H':q_{H'}=2^{-k}} p_{H'} \cdot q_{H'} + \sum_{H':q_{H'}>2^{-k}} p_{H'} \cdot q_{H'} \cdot 2^{-k} \;\leq\; \sum_{H'} p_{H'} \cdot 2^{-k} \;=\; \mathrm{poly}(k) \cdot 2^{-k}$$

and Part 2 of the proposition follows. ■

6.4 Almost constant-round rZK under weaker assumptions

Using a perfectly-hiding commitment scheme which enjoys the trapdoor feature *but not necessarily the strong computational-binding feature*, one may obtain resettable zero-knowledge computationally-sound proof system for $\mathcal{NP}$ in the public-key model. These protocols, however, have an unbounded number of rounds. The idea is to use sequential repetitions of the basic protocols (both for Steps (2)–(6) of the main protocol as well as for the POK sub-protocol) rather that parallel repetitions. That is, both Steps (2)–(6) of the main protocol and the POK sub-protocol consists of parallel executions of a basic protocol, and what we suggest here is to use sequential repetitions instead. The number of (sequential) repetitions can be decreased by using Blum's protocol (rather than the one of [28]) also as a basis for the main proof system (i.e., in Steps (2)–(6)). To minimize round complexity, one may use a parallel-sequential hybrid in which one performs $s(n)$ sequential repetitions of a protocol composed of parallel execution of $p(n) = O(\log n)$ copies of the basic protocol (of Blum). This yields a $O(s(n))$-round resettable zero-knowledge computationally-sound proof system for $\mathcal{NP}$ in the public-key model, for any unbounded function $s : \mathsf{N} \to \mathsf{N}$. In particular, we obtain

Theorem 20 *Let $r : \mathsf{N} \to \mathsf{N}$ be any unbounded function which is computable in polynomial-time, and suppose that for every polynomial p and all sufficiently large n's, any circuit of size $p(n)$ solves DLP correctly only on a negligible fraction of the inputs of length n. Then every language in $\mathcal{NP}$ has a* $r(\cdot)$-round *resettable zero-knowledge computationally-sound proof system in the public-key model.*

Alternatively, we note that by using the perfectly-hiding commitment scheme PC1 also in role of the ("weaker") scheme PC2, we obtain resettable zero-knowledge property also against sub-exponential adversaries. Specifically, even adversaries of running-time bounded by $2^{k^\epsilon} = 2^{K^{\epsilon^2}}$ gain nothing from the interaction, where K (the primary security parameter), $k = K^\epsilon$ (the secondary security parameter) and ϵ (the exponent in the strong computational-binding feature) are as above.

6.5 An alternative presentation of resettable zero-knowledge systems

An alternative presentation of the above protocol may proceed as follows: Rather than relying on general proofs of knowledge we introduce an additional requirement from the PC1 commitment scheme. The new feature referred to as *One-Or-All* asserts that obtaining two different decommitments to the same commitment allows to (feasibly) decommit any way one wants. In our application, the verifier is supposed to know the trapdoor to an instance of the PC1 scheme, allowing it to decommit any way it wants. Thus, if the verifier demonstrates ability to decommit at will then this effectively yields a proof of knowledge of the trapdoor. Put in other words, if the simulator may obtain from the verifier (by rewinding, which is not possible for the actual prover) two different decommitments to the same commitment then it can later decommit at will. Of course, the verifier's demonstration of ability to decommit at will should be performed in a "zero-knowledge" manner. The natural protocol is to have the verifier commit to a k-bit string, and later decommit any way as required by the prover. The natural way to (weakly) simulate this is to select at random a single k-bit string, commit to it and hope that the prover will require to decommit to this value.

References

[1] M. Bellare and O. Goldreich, Proofs of Computational Ability. Crypto '92, August 1992. Full version available on the *Theory of Cryptography Library*, `http://philby.ucsd.edu/old.html`, Record Arc-03.

[2] M. Bellare, R. Impagliazzo and M. Naor. Does Parallel Repetition Lower the Error in Computationally Sound Protocols? In *38th FOCS*, pages 374–383, 1997.

[3] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali and P. Rogaway. Everything Provable is Probable in Zero-Knowledge. In *CRYPTO88*, Springer-Verlag LNCS403, pages 37–56, 1990

[4] M. Blum. Coin Flipping by Phone. *IEEE Spring COMPCOM*, pages 133–137, February 1982. See also *SIGACT News*, Vol. 15, No. 1, 1983.

[5] M. Blum, A. De Santis, S. Micali, and G. Persiano. Non-Interactive Zero-Knowledge Proof Systems. *SIAM J. Computing*, Vol. 20, No. 6, pages 1084–1118, 1991. (Considered the journal version of [6].)

[6] M. Blum, P. Feldman and S. Micali. Non-Interactive Zero-Knowledge and its Applications. In *20th STOC*, pp. 103–112, 1988.

[7] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM J. Computing*, Vol. 13, pages 850–864, 1984.

[8] J. Boyar, M. Krentel and S. Kurtz. A Discrete Logarithm Implementation of Perfect Zero-Knowledge Blobs. *Jour. of Cryptology*, Vol. 2, pp. 63–76, 1990.

[9] G. Brassard, D. Chaum and C. Crépeau. Minimum Disclosure Proofs of Knowledge. *JCSS*, Vol. 37, No. 2, pages 156–189, 1988.

[10] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable Zero-Knowledge. STOC 2000.

[11] I. Damgård. Concurrent Zero-Knowledge in Easy in Practice. Theory of Cryptography Library, 99-14, June 1999. `http://philby.ucsd.edu/cryptolib/1999.html`.

[12] I. Damgård. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. Eurocrypt 2000.

[13] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. In *23rd STOC*, pages 542–552, 1991.

[14] C. Dwork, and A. Sahai. Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints. In *Crypto98*, Springer LNCS 1462.

[15] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In *30th STOC*, pages 409–418, 1998.

[16] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Based on a Single Random String. In *31th FOCS*, pages 308–317, 1990. To appear in *SICOMP*.

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

[18] U. Feige, A. Fiat and A. Shamir. Zero-Knowledge Proofs of Identity. *Journal of Cryptology*, Vol. 1, 1988, pages 77–94.

[19] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd STOC*, pages 416–426, 1990.

[20] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solution to Identification and Signature Problems. In *CRYPTO86*, Springer-Verlag LNCS263, pages 186–189, 1987.

[21] O. Goldreich. A Uniform Complexity Treatment of Encryption and Zero-Knowledge. *Jour. of Cryptology*, Vol. 6, No. 1, pages 21–53, 1993.

[22] O. Goldreich. *Foundation of Cryptography – Fragments of a Book*. February 1995. Revised version, January 1998. Both versions are available from `http://theory.lcs.mit.edu/~oded/frag.html`.

[23] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *JACM*, Vol. 33, No. 4, pages 792–807, 1986.

[24] O. Goldreich, S. Goldwasser, and S. Micali. Interleaved Zero-Knowledge in the Public-Key Model. *ECCC*, TR99-024, 1999. Also available from the *Theory of Cryptography Library*.

[25] 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.

[26] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM J. Computing*, Vol. 25, No. 1, pages 169–192, 1996.

[27] O. Goldreich and L.A. Levin. Hard-core Predicates for any One-Way Function. In *21st STOC*, pages 25–32, 1989.

[28] 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, No. 1, pp. 691–729, 1991.

[29] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Jour. of Cryptology*, Vol. 7, No. 1, pages 1–32, 1994.

[30] S. Goldwasser and S. Micali. Probabilistic Encryption. *JCSS*, Vol. 28, No. 2, pages 270–299, 1984.

[31] S. Goldwasser and S. Micali. Patent applications on *Internet Zero-knowledge Protocols and Application* (3/3/99) and *Internet Zero-Knowledge and Low-Knowledge Proofs and Protocols* (6/11/99).

[32] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, Vol. 18, No. 1, pp. 186–208, 1989.

[33] S. Goldwasser, S. Micali, and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, April 1988, pages 281–308.

[34] S. Hada and T. Tanaka. On the Existence of 3-Round Zero-Knowledge Protocols. In *Crypto98*,

[35] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. Construction of Pseudorandom Generator from any One-Way Function. *SIAM Jour. on Computing*, Vol. 28 (4), pages 1364–1396, 1999.

[36] R. Impagliazzo and M. Luby. One-Way Functions are Essential for Complexity Based Cryptography. In *30th FOCS*, pages 230–235, 1989.

[37] J. Kilian and E. Petrank. An Efficient Non-Interactive Zero-Knowledge Proof System for NP with General Assumptions. *Jour. of Cryptology*, Vol. 11, pages 1–27, 1998.

[38] J. Kilian and E. Petrank. Concurrent Zero-Knowledge in Poly-logarithmic Rounds. Available at the IACR Eprint depository, http://ecrypt.iacr.org, 2000.

[39] J. Kilian, E. Petrank, and C. Rackoff. Lower Bounds for Zero-Knowledge on the Internet. In *39th FOCS*, pages 484–492, 1998.

[40] M. Naor. Bit Commitment using Pseudorandom Generators. *Jour. of Cryptology*, Vol. 4, pages 151–158, 1991.

[41] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *EuroCrypt99*, Springer LNCS 1592, pages 415–413.

[42] A. Rosen. A note on the round complexity of concurrent zero-knowledge protocols. In *Crypto'00*, 2000.

[43] M. Tompa and H. Woll. Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information. In *28th FOCS*, pages 472–482, 1987.

[44] A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd FOCS*, pages 80–91, 1982.

A Constructing rZK and rWI proof-systems in the single-incarnation model

This appendix shows that Construction 7 is sufficient for transforming any WI (ZK) proof-system in the concurrent model into an rWI (rZK) proof-system. The proof of this case is simpler than that of the multiple-incarnation case (i.e., Theorem 8) and can serve as a warm-up for the proof there. For the purpose of this warm-up we further simplify the analysis by assuming (unrealistically!)[37] that the first message of prover P is a fixed string, independent of r_1. This simplifying assumption allows us to focus here on the main ideas underlying the analysis of the protocol transformation. Specifically, we show:

Proposition 21 (for warm-up purposes only): *Suppose that (P, V) is admissible and that the first message of P is a fixed string. Let $\mathbf{P}$ be the prover strategy obtained from P by applying Construction 7, assuming that pseudorandom functions exist. Then for every probabilistic polynomial-time V^* in the single-incarnation resettable model, there exists a probabilistic polynomial-time W^* in the concurrent model so that $\langle P(\overline{y}), W^* \rangle(\overline{x})$ is computationally indistinguishable from $\langle \mathbf{P}(\overline{y}), V^* \rangle(\overline{x})$.*

It follows that if P is concurrent zero-knowledge (resp., witness-indistinguishable) then $\mathbf{P}$ is resettable zero-knowledge (resp., witness-indistinguishable) in the single-incarnation model.

Proof Sketch: By Theorem 2, it suffices to consider the sequential variant of the resettable model. Thus, V^* proceeds in rounds, where in each round it initiates a new session with the single incarnation of $\mathbf{P}$, and carries it out till completion. Our analysis will refer to a mental experiment in which $\mathbf{P}$ utilizes a truly random function rather than a pseudorandom one. As usual, the corresponding views of the verifier V^* in the two cases (i.e., random versus pseudorandom function)

[37] Unfortunately, this assumption does not hold in the protocols to which we want to apply the transformation. In fact, in some sense, this assumption cannot possibly hold when one considers (as we do) adversaries V^* that may be non-uniform polynomial-size circuits.

are computationally indistinguishable. From this point on, we identify the random-tape of P with a truly random function.

Working in the concurrent model, W^* handles the messages of V^* as follows:

1. *V^* initiates a new session*: In this case W^* initiates a new session with the prover P, obtains its first message, denoted $\mathtt{msg}$, and forwards $\mathtt{msg}$ to V^*.

   (Here we capitalize on the fact that, by our hypothesis, independent sessions of P yield the very same first prover-message. This is important because V^* always initiates the same incarnation of $\mathbf{P}$, and hence expects to always obtain the same first prover-message.)

2. *V^* sends a new first-message*: That is, we refer to the case where the current message sent by V^* is the first verifier-message in the current session (carried out by V^* with $\mathbf{P}$), and assume that this message is different from all first-verifier-messages sent in prior sessions. Let $\mathtt{msg}'$ denote the message sent by V^*. Then W^* sends $\mathtt{msg}'$ to one of the sessions (which it carries out with P) that still awaits a first-verifier-message,[38] obtains the prover's response, and forwards it to V^*. Finally, W^* designates this session (with P) as the active session of $\mathtt{msg}'$, and stores the prover's response.

   (All subsequent sessions of V^* in which the first-verifier-message equals $\mathtt{msg}'$ will be "served" by the single session of W^* designated as the active session of $\mathtt{msg}'$. Non-active sessions will not be used (i.e., W^* does not send any message in them).)

3. *V^* repeats a first-message*: That is, we refer to the case where the current message sent by V^* is the first verifier-message in the current session, and assume that this message equals a first-verifier-message, $\mathtt{msg}'$, sent in a prior session. In this case, W^* retrieves from its storage P's answer in the active session of $\mathtt{msg}'$, and forwards it to V^*.

   We stress that W^* does not communicate with any session of P in this case. (Note that if W^* were to send the same message $\mathtt{msg}'$ to two sessions of P then the responses could have differed, whereas V^* expects to see exactly the same answer in sessions in which it sends the same $\mathtt{msg}'$.)

4. *V^* sends a valid non-first-message*: That is, we refer to the case where V^* sends a non-first-message in the current session and this message is valid; that is, P accepts it as valid as per Definition 5. (In this case, the message is essentially determined by the first-verifier-message in that session.)

   We stress that it is universally verifiable whether the current message of V^* is valid or not (i.e., this depends only on the current and first verifier-messages, and on all prover-messages in the current session).

   We distinguish two cases, depending on whether this is the first time that a valid verifier-message of the current round was sent in a session of V^* in which the first verifier-message equals $\mathtt{msg}'$, where $\mathtt{msg}'$ is the first verifier-message sent by V^* in the current session. Let $\xi > 1$ denote the index of the current message sent by V^*.

   (1) *The current session is the first session in which the first verifier-message equals $\mathtt{msg}'$ and the ξ^{th} verifier-message is valid*: In this case W^* forwards the current message to the active session of $\mathtt{msg}'$ (with P), obtains P's response, stores it, and forwards it to V^*.

[38] Such a session exists since W^* initiates a new session per each new session initiated by V^*, whereas W^* sends at most one first verifier-message per each such message sent by V^*.

(2) *The current session is* NOT *the first session in which the first verifier-message equals* $\mathtt{msg}'$ *and the* ξ^{th} *verifier-message is valid*: In this case W^* does not communicate with any session of P. Instead, it merely retrieve the corresponding prover response from its storage, and forwards it to V^*. Note that the corresponding answer is stored in the history of the active session of $\mathtt{msg}'$.

(Note that by Definition 5, it is infeasible for V^* to send, in two sessions starting with any fixed verifier-message, valid messages for the same round that differ in their main part. Thus, the responses of $\mathbf{P}$ to valid ξ^{th} messages, in sessions starting with any fixed verifier-message, are identical. It follows that V^* will be content with the identical responses supplied to it by W^*.)

5. V^* *sends an invalid non-first-message*: That is, we refer to the case where V^* sends a non-first-message in the current session and this message is invalid. In this case, W^* just forwards P's standard `abort` message to V^*.

   We stress that W^* does NOT forward the invalid message of V^* to any session of P, most importantly not to an active session. This allows W^* to handle a corresponding valid message that may be sent by V^* in a future session.

6. V^* *terminates*: When V^* sends a termination message, which includes its output, W^* just outputs this message and halts.

We stress that W^* is defined to operate in the concurrent model. That is, in every session it invokes with P, the action of the latter are independent of other sessions. In contrast, V^* that operates in the (stronger) resettable model interacts with a single incarnation of $\mathbf{P}$, and so the actions of $\mathbf{P}$ in various sessions are potentially related. Nevertheless, we claim that the output of W^* is computationally indistinguishable from the output of V^*. The key observations justifying this claim refer to the actions of $\mathbf{P}$ in the various sessions invoked by V^*:

- In sessions having different first-verifier-messages, the actions of $\mathbf{P}$ are independent. This is because $\mathbf{P}$ determines its actions by applying a random function on the first-verifier-message, and in this case the results are independent random-tapes.

- In sessions having the same first-verifier-message, the actions of $\mathbf{P}$ are practically determined by that first message. This is because in this case $\mathbf{P}$ determines the same random-tape, and the only freedom of V^* is essentially to choose at each message whether to send a predetermined (by the first-verifier-message) value or to abort. Thus, the transcripts of all these sessions correspond to various augmented prefixes of one predetermined transcript, where each prefix is either the complete transcript or a strict prefix of it augmented by an `abort` message.

The corresponding transcripts (of imaginary sessions with $\mathbf{P}$) are generated by W^* by merely copying from real sessions it conducts with P. Each set of $\mathbf{P}$-sessions sharing the same first-verifier-message, is generated from a single (distinct) session with P (called the active session of that message). The way in which W^* handles invalid messages of V^* guarantees that it never aborts an active session, and so such a session can always be extended (up-to completion) to allow the generation of all $\mathbf{P}$-sessions sharing that first-verifier-message. We stress again that W^* does not need to (and in fact does not) abort a session in order to produce $\mathbf{P}$'s abort message; it merely determines whether $\mathbf{P}$ aborts and, if so, generates the standard `abort` message by itself. ∎

Comment: We emphasize the concurrent nature of the adversary W^* constructed in the proof above. If V^* first abort a session with first-verifier-message $\mathtt{msg}'$, and later sends a corresponding valid message in a later session with the same first-verifier-message, then W^* answers V^* by obtaining a response from the active session of $\mathtt{msg}'$. However, the latter session was initiated at the time when $\mathtt{msg}'$ was first sent, and other sessions could have been initiated between the two times in which V^* sent $\mathtt{msg}'$ as a first message. Thus, W^* conducts concurrent sessions with P, although V^* only interacts sequentially with $\mathbf{P}$.

B Commitment Schemes

We formally define the various types of commitment schemes used in the main text. We start with the more standard notion of a commitment scheme in which secrecy is preserved only w.r.t computationally bounded adversaries, and later pass to the dual notion of a perfectly-hiding commitment scheme (in which secrecy is preserved in an information theoretic sense). Recall that the binding property in standard schemes is absolute (i.e., information theoretical), whereas in perfectly-hiding commitment schemes it holds only w.r.t computationally bounded adversaries. But before defining any of these, let use define a sufficient condition for the existence of all these schemes – a strong DLP assumption.

B.1 The Strong DLP Intractability Assumption

The Discrete Logarithm Problem (DLP) is defined as follows. On input p, g, y, where p is a prime, g is a primitive element in the multiplicative group modulo p, and $y \in \mathsf{Z}_p^*$, one has to find x such that $g^x \equiv y \pmod p$. We assume that this task is intractable also in the special case where $p = 2q + 1$ and q is a prime too. Such p's are often called *safe primes*, and the above assumption is quite standard. It follows that the same would hold when g is of order q and so is y. Finally, we assume that intractability refers to sub-exponential size circuits rather merely to super-polynomial ones. Thus we assume the following:

> **The Strong DLP Assumption**: *For some $\epsilon > 0$, for every sufficiently large n, and every circuit C of size at most 2^{n^ϵ}*
>
> $$\Pr[C(p, g, g^x \bmod p) = x] \;<\; 2^{-n^\epsilon}$$
>
> *where the probability is taken uniformly over all n-bit long safe primes p, elements g of order $q \stackrel{\text{def}}{=} (p-1)/2$, and $x \in \mathsf{Z}_q^*$.*

We comment that, although stronger than the standard assumption, the above Strong DLP Assumption seems very reasonable.

B.2 Standard Commitment Schemes

By a standard commitment scheme we refer to one providing computational-secrecy and absolute (or perfect) binding. For simplicity, we consider here only one-round commitment schemes.

Definition 22 (standard commitment scheme): *A* standard commitment scheme *is a probabilistic polynomial-time algorithm, denoted C satisfying:*

(Computational) Secrecy: *For every v, u of equal* $\mathrm{poly}(n)$*-length, the random variables $C(1^n, v)$ and $C(1^n, u)$ are computationally indistinguishable by circuits. That is, for every two polynomials p, q, all sufficiently large n's and all $v, u \in \{0,1\}^{p(n)}$ and every distinguishing circuit D of size $q(n)$,*

$$|\Pr[D(C(1^n, v)) = 1] \; - \; \Pr[D(C(1^n, u)) = 1]| \; < \; \frac{1}{q(n)}$$

(Perfect) Binding: *For every v, u of equal* $\mathrm{poly}(n)$*-length, the random variables $C(1^n, v)$ and $C(1^n, u)$ have disjoint support. That is, for every v, u and α, if* $\Pr[C(1^n, v) = \alpha]$ *and* $\Pr[C(1^n, u) = \alpha]$ *are both positive then $u = v$.*

The way such a commitment scheme is used should be clear: To commit to a string v, under security parameter n, the sender invokes $C(1^n, v)$ and sends the result as its commitment. The randomness used by C during this computation, is to be recorded and can latter be used as a decommitment.

A commitment scheme as above can be constructed based on any one-way permutation: Loosely speaking, given a permutation $f : D \to D$ with a hard-core predicate b (cf., [27]), one commits to a bit σ by uniformly selecting $x \in D$, and sending $(f(x), b(x) \oplus \sigma)$ as a commitment.

A strong version of the standard commitment scheme requires computational-secrecy to hold also with respect to sub-exponential-size circuits (i.e., replace the polynomial q above by a function f of the form $f(n) = 2^{n^\epsilon}$, for some fixed $\epsilon > 0$). This is analogous to the strong computational-binding feature discussed below. The Strong DLP Assumption implies the existence of such strong computational-secrecy commitment schemes.

B.3 Perfectly-hiding Commitment Schemes

We start by defining two-round perfectly-hiding commitment schemes. In such schemes the party's strategies may be represented by two algorithms, denoted (S, R), for *sender* and *receiver*. The sender has a secret input $v \in \{0,1\}^*$ and both parties share a security parameter n. Thus, the first message sent (by an honest receiver) is $R(1^n)$, and the response by a sender wishing to commit to a value v (of length bounded by a polynomial in n) is $S(1^n, v, \mathtt{msg})$, where $\mathtt{msg}$ is the message received in the first round. To "de-commit" to a value v, the sender may provide the coin tosses used by S when committing to this value, and the receiver may easily verify the correctness of the de-committed value.

Definition 23 (perfectly-hiding two-round commitment scheme): *A* perfectly-hiding two-round commitment scheme *is a pair of probabilistic polynomial-time algorithms, denoted (S, R) satisfying:*

(Perfect) Secrecy: *For every mapping R^** (representing a computationally-unbounded cheating receiver)*, and for every v, u of equal* $\mathrm{poly}(n)$*-length, the random variables $S(1^n, v, R^*(1^n))$ and $S(1^n, u, R^*(1^n))$ are statistically close. That is, for every two polynomials p, q, all sufficiently large n's and all $v, u \in \{0,1\}^{p(n)}$*

$$\sum_{\alpha} |\Pr[S(1^n, v, R^*(1^n)) = \alpha] - \Pr[S(1^n, u, R^*(1^n)) = \alpha]| \; < \; \frac{1}{q(n)}$$

(Computational) Binding: *Loosely speaking, it should be infeasible for the sender, given the message sent by the honest receiver, to answer in a way allowing it to later de-commit in two different ways.*

In order to formulate the above, we rewrite the honest sender move, $S(1^n, v, \mathtt{msg})$*, as consisting of uniformly selecting* $s \in \{0,1\}^{\mathrm{poly}(n,|v|)}$*, and computing a polynomial-time function* $S'(1^n, v, s, \mathtt{msg})$*, where* $\mathtt{msg}$ *is the receiver's message. A cheating sender tries, given a receiver message* $\mathtt{msg}$*, to find two pairs* (v, s) *and* (v', s') *so that* $v \neq v'$ *and yet* $S'(1^n, v, s, \mathtt{msg}) = S'(1^n, v', s', \mathtt{msg})$*. This should be infeasible; that is, we require that for every polynomial-size circuit* S^* (representing a cheating sender invoked as part of a larger protocol)*, for every polynomial* p*, all sufficiently large* n*'s*

$$\Pr[\, V_n \neq V'_n \ \& \ S'(1^n, V_n, S_n, R(1^n)) = S'(1^n, V'_n, S'_n, R(1^n)) \,] \ < \ \frac{1}{q(n)}$$

where $(V_n, S_n, V'_n, S'_n) = S^*(1^n, R(1^n))$.

A perfectly-hiding two-round commitment scheme can be constructed using any claw-free collection (cf., [25]). In particular, it can be constructed based on the standard assumption regarding the intractability of DLP (as the latter yields a claw-free collection). Combing the two constructions, we get the following perfectly-hiding two-round commitment scheme: On input a security parameter n, the receiver selects uniformly an n-bit prime p so that $q \stackrel{\mathrm{def}}{=} (p-1)/2$ is prime, a element g of order q in Z_p^*, and z in the multiplicative subgroup of Z_p^* formed by g, and sends the triple (p, g, z) over. To commit to a bit σ, the sender first checks that (p, g, z) is of the right form (otherwise it halts announcing that the receiver is cheating[39]), uniformly selects $s \in \mathsf{Z}_q$, and sends $g^s z^\sigma \bmod p$ as its commitment.

Additional features: The additional requirements assumed of the perfectly-hiding commitment schemes in Section 6.3 can be easily formulated. The strong computational binding feature is formulated by extending the Computational Binding Property (of Def. 23) to hold for sub-exponential circuits S^*. Again, the Strong DLP Assumption yields such a stronger binding feature. The trapdoor feature requires the existence of a probabilistic polynomial-time algorithm $\overline{R}$ that outputs pairs of strings so that the first string is distributed as in R (above), whereas the second string allows arbitrary decommitting. That is, there exists a polynomial-time algorithm A so that for every $(\mathtt{msg}, \mathrm{aux})$ in the range of $\overline{R}(1^n)$, every $v, u \in \{0,1\}^{\mathrm{poly}(n)}$, and every $s \in \{0,1\}^{\mathrm{poly}(n,|v|)}$, satisfies

$$S'(1^n, v, s, \mathtt{msg}) \ = \ S'(1^n, u, A(\mathrm{aux}, (v, s), u), \mathtt{msg})$$

That is, $a = A(\mathrm{aux}, (v, s), u)$ is a valid decommit of the value u to the sender's commitment to the value v (i.e., the message $S'(1^n, v, s, \mathtt{msg})$). Thus, one may generate random commitments c (by uniformly selecting s and computing $S'(1^n, 0^{\mathrm{poly}(n)}, s, \mathtt{msg})$) so that later, with knowledge of aux, one can decommit to any value u of its choice (by computing $a = A(\mathrm{aux}, (0^{\mathrm{poly}(n)}, s), u)$). The DLP construction (of above) can be easily modified to satisfy the trapdoor feature: Actually, the known implementation for the random selection of z (in the subgroup generated by g) is to select r uniformly in Z_q^* and set $z = g^r \bmod p$. But in this case r is the trapdoor we need, since $g^s z^v \equiv g^{s+(v-u)r} z^u \pmod{p}$, and so we may decommit to u by presenting $s + (v-u)r \bmod q$.

C Blum's Proof of Knowledge

For sake of self-containment, we first recall the definition of a proof of knowledge. The following text is reproduced from [22].

[39] Actually, to fit the definition, the sender should commit via a special symbol which allows arbitrary decommit. Surely, such a commitment-decommit pair will be rejected by the honest receiver, which never cheats.

C.1 Proofs of Knowledge

Preliminaries

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. Then $R(x) \stackrel{\text{def}}{=} \{s : (x,s) \in R\}$ and $L_R \stackrel{\text{def}}{=} \{x : \exists s \text{ s.t. } (x,s) \in R\}$. If $(x,s) \in R$ then we call s a *solution* for x. We say that R is *polynomially bounded* if there exists a polynomial p such that $|s| \leq p(|x|)$ for all $(x,s) \in R$. We say that R is an *NP relation* if R is polynomially bounded and, in addition, there exists a polynomial-time algorithm for deciding membership in R (i.e., $L_R \in \mathcal{NP}$). In the sequel, we confine ourselves to polynomially bounded relations.

We wish to be able to consider in a uniform manner all potential (knowledge) provers, without making distinction based on their running-time, internal structure, etc. Yet, we observe that these interactive machine can be given an auxiliary-input which enables them to "know" and to prove more. Likewise, they may be lucky to select a random-input which enables more than another. Hence, statements concerning the knowledge of the prover refer not only to the prover's program but also to the specific auxiliary and random inputs it has. Hence, we fix an interactive machine and all inputs (i.e., the common-input, the auxiliary-input, and the random-input) to this machine, and consider both the corresponding accepting probability (of the verifier) and the usage of this (prover+inputs) template as an oracle to a "knowledge extractor". This motivates the following definition.

Definition 24 (message specification function)*: Denote by $P_{x,y,r}(\overline{m})$ the message sent by machine P on common-input x, auxiliary-input y, and random input r, after receiving messages $\overline{m}$. The function $P_{x,y,r}$ is called the* message specification function *of machine P with common-input x, auxiliary-input y, and random input r.*

An oracle machine with access to the function $P_{x,y,r}$ will represent the knowledge of machine P on common-input x, auxiliary-input y, and random input r. This oracle machine, called the *knowledge extractor*, will try to find a solution to x (i.e., an $s \in R(x)$). (As postulated below, the running time of the extractor is inversely related to the corresponding accepting probability (of the verifier).)

Knowledge verifiers

Now that all the machinery is ready, we present the definition of a system for proofs of knowledge. At first reading, the reader may set the function κ to be identically zero.

Definition 25 (System of proofs of knowledge): *Let R be a binary relation, and $\kappa : \mathsf{N} \to [0,1]$. We say that an interactive machine V is a* knowledge verifier for the relation R *with* knowledge error κ *if the following two conditions hold.*

- Non-triviality: *There exists an interactive machine P so that for every $(x,y) \in R$ all possible interactions of V with P on common-input x and auxiliary-input y are accepting.*
- Validity (with error κ): *There exists a probabilistic oracle machine K such that for every interactive machine P, every $x \in L_R$ and every $y, r \in \{0,1\}^*$, on input x and access to $P_{x,y,r}$ machine K finds a solution $s \in R(x)$ within expected time inversely proportional to $p - \kappa(|x|)$, where p is the probability that V accepts x when interacting with $P_{x,y,r}$. More precisely:*

  Denote by $p(x,y,r)$ the probability that the interactive machine V accepts, on input x, when interacting with the prover specified by $P_{x,y,r}$. Then if $p(x,y,r) > \kappa(|x|)$ then, on input x and *access to oracle $P_{x,y,r}$, machine K outputs a solution $s \in R(x)$ within an expected number of steps bounded above by*

$$\frac{\text{poly}(|x|)}{p(x,y,r) - \kappa(|x|)}$$

The oracle machine K is called a universal knowledge extractor.

When $\kappa(\cdot)$ is identically zero, we just say that V is a knowledge verifier for the relation R. *An interactive pair (P, V) so that V is a knowledge verifier for a relation R and P is a machine satisfying the non-triviality condition (with respect to V and R) is called a* system for proofs of knowledge *for the relation R.*

C.2 Blum's Protocol

In the main text, we consider k parallel repetitions of the following basic proof system for the *Hamiltonian Cycle* (HC) problem which is NP-complete (and thus get proof systems for any language in $\mathcal{NP}$). We consider directed graphs (and the existence of directed Hamiltonian cycles).

Construction 26 (Basic proof system for HC):

- Common Input: *a directed graph $G = (V, E)$ with $n \stackrel{\text{def}}{=} |V|$.*
- Auxiliary Input to Prover: *a directed Hamiltonian Cycle, $C \subset E$, in G.*
- Prover's first step (P1): *The prover selects a random permutation, π, of the vertices V, and commits to the entries of the adjacency matrix of the resulting permuted graph. That is, it sends an n-by-n matrix of commitments so that the $(\pi(i), \pi(j))^{\text{th}}$ entry is a commitment to 1 if $(i, j) \in E$, and is a commitment to 0 otherwise.*
- Verifier's first step (V1): *The verifier uniformly selects $\sigma \in \{0, 1\}$ and sends it to the prover.*
- Prover's second step (P2): *If $\sigma = 0$ then the prover sends π to the verifier along with the revealing (i.e., preimages) of all commitments. Otherwise, the prover reveals to the verifier only the commitments to entries $(\pi(i), \pi(j))$ with $(i, j) \in C$. In both cases the prover also supplies the corresponding decommitments.*
- Verifier's second step (V2): *If $\sigma = 0$ then the verifier checks that the revealed graph is indeed isomorphic, via π, to G. Otherwise, the verifier just checks that all revealed values are 1 and that the corresponding entries form a simple n-cycle. In both cases the verifier checks that the decommitments are proper (i.e., that they fits the corresponding commitments). The verifier accepts if and only if the corresponding condition holds.*

We stress that the above protocol uses a standard commitment scheme.

Proposition 27 *The protocol which results by k parallel repetitions of Construction 26 is a proof of knowledge of Hamiltonicity with knowledge error 2^{-k}. Furthermore if, for every positive polynomial p, the commitment scheme used in Step (P1) maintain secrecy with respect to circuits of size $p(n) \cdot 2^{3k}$ and distinguishing gap of $2^{-3k}/p(n)$ then, for every positive polynomial q, the interaction can be simulated in time $\text{poly}(n) \cdot 2^k$ so that no circuit of size $q(n) \cdot 2^{2k}$ can distinguish the simulation from the real interaction with gap of $2^{-2k}/q(n)$ or more.*