Resettable Zero-Knowledge[*]

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

January 11, 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.

Under general complexity assumptions, which hold for example if the Discrete Logarithm Problem is hard, we construct

- (non-constant round) Resettable Zero-Knowledge proof-systems for NP
- constant-round Resettable Witness-Indistinguishable proof-systems for NP
- constant-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, we show that 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. 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 [27].

[†]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`

[§]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 [28]. The generality of this notion was demonstrated by Goldreich, Micali and Wigderson [24], 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 [7], witness indistinguishability [15], and zero-knowledge proofs of knowledge [28, 35, 14]. 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 [16], 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 if 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 [16]), *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 [24]) 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 [28] 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 $z \bmod n$.

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, 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 [33, 30], that one-way functions exist.

[2] Zero-knowledge proofs in which the prover is deterministic exist only for $\mathcal{BPP}$ languages (cf., [25]).

[3] For instance, in [16] 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 [20], the same notion was referred to by the names *rewind zero-knowledge* and *interleaved zero-knowledge*.

Resettability 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-IndistinguishableProofs* (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 executions.

The Importance of the New Notion. Resettable zero knowledge sheds new light on what is it that make secure protocol possible. In particular, constructing such protocols, makes 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 enlarges 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 enemy could reset the device implementing the prover to its initial conditions (which include his random tape). Unfortunately, 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 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:

[5] "Strong" refers to those in which the claw-free property should hold also with respect to subexponential-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 1: *Under the weak DLP assumption, there is a (non-constant round) rZK proof for NP.*

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 literal sense of the word. Registration does not have to include interaction with a trusted system manager that may verify properties of the registered public-key (e.g., that it valid or even that the user registering it knows a corresponding secret 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.)

Consequences for concurrent zero knowledge. With the rise of the internet, the importance of *concurrent execution* of zero-knowledge protocols emerged. In a concurrent setting, many executions of protocols can be running at the same time, involving many verifiers which may be talking with the same (or many) provers simultaneously. This presents the new risk of an overall adversary who controls the verifiers, interleaving the executions and choosing verifiers queries based on other partial executions. This risk is made even more challenging by the fact that it is unrealistic for the honest provers to coordinate their action so that zero-knowledge is preserved in this setting. Thus, we must assume that in each prover-verifier pair the prover acts independently.

A recent approach for solving the concurrent execution problem has been suggested by Dwork, Naor and Sahai [12], 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 [12]. In a later paper, Dwork and Sahai [11] 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 work by Ransom and Kilian [34] does not use the timing assumption, alas their protocols are either not constant-round or only simulatable in quasi-polynomial time. We stress that none of these concurrent ZK protocols is rZK.

Because secure concurrent executability is critical for protocols to be played over the internet, and because the number of rounds is an important resource for internet protocols, establishing whether constant-round concurrent ZK protocols exist is a critical problem. Theorem 3 provides an answer to this question by means of the following

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

The importance of this corollary stems from the fact that the public-key model is quite standard whenever cryptography is used, specifically it underlies any public-key encryption or digital

[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.

signature scheme. Note, that this model may indeed be both simpler, and more realistic than the timing assumption of [12, 11]. Even if one thinks of the public-key model as a mild form of preprocessing, Corollary 5 directly improves on Dwork and Sahai's protocol based on pre-processing with the timing assumption. In fact, *we would just rely on the existence of a pre-processing step, while they do rely on the existence of a pre-processing step in which the timing assumption holds.* Thus, the theory of rZK protocols yields a constant-round solution to the important (and extensively investigated) concurrent ZK problem.

Consequences for proofs of identity. Fiat and Shamir in [16] 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 knowledge of some secret (e.g. in the original [16] it was 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 [16]). 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.

We will elaborate further about the notion of Resettable Proofs of Identity and specific implementations of it in a separate paper.

2 Overview

Due to length of this write-up we provide an overview of our work. Details are found in subsequent sections.

2.1 The notion of resettable zero-knowledge

For sake of simplicity, we present here a simple definition of resettable zero-knowledge. This definition captures the most important aspects of the more general definition that is actually used.

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 sequential interactions with the residual deterministic prover strategy $P_{x,y,\omega}$ determined by uniformly selecting and fixing P's coins, ω. That is, ω is uniformly selected and fixed once and for all, and the adversary may sequentially invoke and interact with $P_{x,y,\omega}$ polynomially-many times. In each such invocation, $P_{x,y,\omega}$ behaves as P would have behaved on common input x, auxiliary-input y, and random-tape ω. Thus, the adversary and $P_{x,y,\omega}$ engage in polynomially-many interactions; but whereas $P_{x,y,\omega}$'s actions in each interaction are independent of prior interactions (since $P_{x,y,\omega}$ mimics the "single interaction strategy" P), the actions of the adversary in the current interaction may depend on prior interactions. In particular, the adversary may repeat the same messages sent in a prior

[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.

interaction, 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.

Definition 2.1 (resettable security – simple case – vanilla model): *A prover strategy P is said to be* resettable zero-knowledge (on L) *if for every probabilistic polynomial-time adversary V^* as below there exists a probabilistic polynomial-time simulator M^* so that the following distribution ensembles, indexed by a common input $x \in L$ and a prover auxiliary input y, are computationally indistinguishable* (cf., [26, 36]):

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

1. *Randomly select and fix a random-tape, ω, for P, resulting in a deterministic strategy $P' = P_{x,y,\omega}$ defined by $P_{x,y,\omega}(\text{history}) = P(x, y, \omega; \text{history})$.*
2. *Machine V^* is allowed to initiate polynomially-many* sequential *interactions with P'. The actions of V^* in the i^{th} interaction with P' may depend on previous interactions, but the i^{th} interaction takes place only after the $i - 1^{\text{st}}$ interaction was completed.*

   More formally, V^* sends whatever message its pleases, yet this message is answered as indicated above. That is, suppose P' expects to get t messages per interaction. Then, for every $i \geq 1$ and $j = 1, ..., t$, the $(i-1)t + j^{\text{th}}$ message sent by V^* is treated as the j^{th} message in the i^{th} interaction of P', and accordingly the response is $P'(\mathtt{msg}_{(i-1)t+1}, ..., \mathtt{msg}_{(i-1)t+j})$, where $\mathtt{msg}_k$ is the k^{th} message sent by V^*.
3. *Once V^* decides it is done interacting with P', it* (i.e., V^*) *produces an output based on its view of these interactions* (*which, as usual, includes the internal coin-tosses of* V^*).

`Distribution 2:` *The output of $M^*(x)$.*

We note that all known zero-knowledge protocols are NOT resettable zero-knowledge. (Furthermore, they are even NOT resettable witness indistinguishable.) For example, ability to "rewind" the original zero-knowledge proof for 3-Colorability [24], allows the adversary to fully recover the 3-coloring of the input graph used by the prover: The adversary merely invokes the proof system many times, and asks the prover to reveal a uniformly selected edge in each invocation. Since the prover's randomness is fixed in all these invocations, it will commit to the same coloring of the graph, and reveal the values (w.r.t this fixed coloring) of two adjacent vertices in each invocation. Thus, after polynomially-many invocations,[8] the adversary will obtain the values of all vertices w.r.t one fixed coloring. (Recall that in the standard zero-knowledge model the adversary will merely obtain in each invocation two different values w.r.t an independently chosen random coloring.)

In Section 4, the above definition is generalized by allowing the adversary to interleave the various executions (rather than execute them sequentially one after the other). Interestingly, this does not change the power of the model: every protocol that is resettable zero-knowledge in the non-interleaved model is also resettable zero-knowledge in the interleaved model. This equivalence is important since it allows us to analyze protocols in the simpler non-interleaved model and infer their security in the general (interleaved) model for free. (We use this fact to simplify the exposition of the analysis of our various protocols.) Another extension (to Def. 2.1) is to allow the adversary to

[8] Actually, a linear (in the number of vertices) number of invocations suffices.

interact (many times) with several random independent incarnations of P (rather than with a single one). That is, rather than interacting many times with one $P_{x,y,\omega}$, where ω is randomly selected, the adversary many interact many times with each P_{x_i,y_i,ω_j}, where the ω_j's are independently and randomly selected. Intuitively, this should not add power to the model either.

Note that the general definition (i.e., the one allowing polynomially-many independent incarnations of the prover) implies concurrent zero-knowledge. In fact, concurrent zero-knowledge is (syntactically) a very restricted case of resettable zero-knowledge (in which one may interact only once with each of these polynomially-many incarnations).

For further details see Section 4.

2.2 NP has constant-round resettable-WI

The notion of Witness Indistinguishability (WI) was introduced by [15] as a relaxation of the zero-knowledge requirement which could be still suitable in many applications and may be achieved with greater ease and efficiency. For example, *all* witness indistinguishable protocols are closed under parallel composition and concurrent execution.

Resettable-WI (resettable witness indistinguishable) relates to resettable zero-knowledge as standard WI relates to ZK. Informally, in a resettable witness indistinguishable protocol a polynomial-time verifier can still not distinguish between two different witnesses for an NP statement used by the prover, even if it can "clone" the prover (each time with the same initial configuration, random tape included) as many times as it pleases, and then interact with these clones in any order and manner it wants. More formally, instead of requiring that Distribution 1 (in Def. 2.1 above) be simulatable by a probabilistic polynomial-time machine, we require that instances of Distribution 1 – induced by the prover using different NP-witnesses – be computationally-indistinguishable.

We stress that all existing WI protocols are not rWI protocols. (Even the honest verifier can easily extract the entire witness —let alone distinguish between witnesses— when the protocol is executed polynomially many times with a prover using the same coins.) In contrast, as stated in Theorem 2, we can achieve constant-round rWI interactive proofs.

To build resettable witness indistinguishable proof-systems for NP, we start with a ZK proof-system for NP. Traditionally, the latter proof-systems rely on the randomized nature of the prover strategy (in a sense, this is essential —cf., [25]). In our context, the prover's randomization occurs only once and is fixed for all subsequent interactions. So the idea is to utilize the initial randomization (done in the very first invocation of the prover) in order to randomize all subsequent invocations. The natural way of achieving this goal is to use a pseudorandom function, as defined and constructed in [19]. However, just "using a pseudorandom function" does not suffice. The function has to be applied to "crucial steps" of the verifier; that is, exactly the steps that the verifier may want to alter later (by rewinding) in order to extract knowledge. Thus, the zero-knowledge proof system for 3-Colorability of [24] is not an adequate starting-point (since there the prover's randomization takes place before a crucial step by the verifier). Instead, we start with the zero-knowledge proof system of Goldreich and Kahan [21]: In that proof system, the verifier first commits to a sequence of edge-queries, then the prover commits to random colorings, and then the verifier reveals its queries and the prover reveals the adequate colors. Starting with this proof system, we replace the prover's random choices (in its commitment) by the evaluation of a pseudorandom function (selected initially by the prover) on the verifier commitment. The resulting proof system can be shown to be resettable witness indistinguishable.

An indication of the non-triviality of the result is given by the fact that we don't know whether the resulting protocol is resettable zero-knowledge. The key observation regarding the *specific* protocol sketched above is that, in each single execution of it, all the verifier steps following its first message (i.e., its commitment message) are "essentially determined." The only choice left to the verifier is whether to reveal the correct value (i.e., properly decommit) or refuse to continue (i.e., send an invalid decommitment message). This limited level of freedom allows to prove that the protocol is resettable witness indistinguishable (however, it prevents us from proving that the protocol is resettable zero-knowledge): intuitively, if the verifier's subsequent steps are determined (except for the abort possibility) then its only real freedom is in selecting its first message. Now, if it selects the same first message as in a prior interaction, it will only get the same interaction transcript again (which being easily simulatable by mere copying is quite useless). If, on the other hand, the verifier selects as first message a string different from the one used as first message in all prior interactions then the prover's actions in the current interaction will be independent of its actions in prior interactions (since the prover's actions are determined by applying a pseudorandom function to the verifier's first message). So in this case the verifier obtains no more than in standard sequential composition of zero-knowledge protocols (which are well-known to remain zero-knowledge).

We warn that the explanation provided above ignores several important issues. For further details see Section 5.

2.3 NP has resettable-ZK proofs

We show how to construct resettable zero-knowledge proof systems for any language in NP. Our starting point is a concurrent zero-knowledge proof system of Ransom and Kilian [34]. We modify this proof system using the techniques discussed above (i.e., determining the prover's actions by applying a pseudorandom function to suitable transcripts of the interaction so far), and replace the concurrent witness indistinguishable (concurrent-WI) proof system employed by [34] with our resettable witness indistinguishable proof system. Whereas any WI proof (cf. [15]) is also concurrent-WI (cf. [13]), let us stress again that all previously known WI proofs are not resettable witness indistinguishable. Thus, our resettable witness indistinguishable proof system plays a major role in showing that NP has resettable zero-knowledge proofs.

It is easy to show that the protocol resulting from the above sketched transformation remains a proof system for the same language. The tricky part is to show that it is indeed resettable zero-knowledge (and not merely zero-knowledge in the standard sense, which is obvious). Our original proof, which can be found in [8], adapts the simulation argument of [34], extending it from their concurrent model to our stronger resettable model. The proof presented in this version[9] refers to a slight modification of the above protocol. Very loosely speaking, it consists of showing that, for any protocol in which the verifier's actions are essentially determined by its first message (as in the case of the modified protocol), if the protocol is concurrent zero-knowledge then (by using pseudorandom functions as above) it can be modified to become resettable zero-knowledge.

Again, we warn that the explanation provided above ignores several important issues. For further details see Section 5.

2.4 The Public-Key model

So far in this overview (and the corresponding Part I of this work), no set-up assumptions have been made. This is indeed the "simplest" model used for two-party and multi-party computation. Another model, used routinely in the different context of providing privacy and/or authenticity of messages, is the *public-key* model, which instead relies on a set-up stage in which public-keys are

[9] A variant of this alternative proof can also be found in our original technical report [8].

registered. One crucial aspect of our work consists of using the public-key model for tasks totally unrelated to privacy and authenticity.[10]

In the mildest form of the latter model, users are assumed to have deposited a public-key in a public file that is accessible by all users at all times. 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 only assumption about this file is that it is guaranteed that entries in it were deposited before any interaction among the users takes place. No further assumption about this file is made. In particular, an adversary may deposit in it arbitrarily many public-keys, including public key are are "non-sensical" or "bad" (e.g., for which no corresponding secret key exist or are known).

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. This fact plays a key role in our main result for this model:

> Under the strong DLP assumption, we show *how to construct* constant-round *resettable zero-knowledge arguments for $\mathcal{NP}$ in the public-key model.*

Recall that *arguments* (a.k.a computationally-sound proofs) [7] are a weaker notion than interactive proofs [28]: 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:

> constant-round *concurrent zero-knowledge arguments for $\mathcal{NP}$ in the public-key model*, under the strong DLP assumption.

We stress that unlike [12], the above stated result does not use any timing assumption.

Our construction uses a technique which may be of independent interest. We use two secure schemes, one with security parameter K and the other with a smaller security parameter k. Suppose that, for some $\epsilon > 0$, the security of the first scheme (with security parameter K) is maintained against adversaries running in time 2^{K^ϵ},[11] 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. [10]) 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 $U(n) = 2^n$.)

For further details see Sections 6 and 7.

Related work. Using weaker assumptions but a stronger public-key model, Damgard has independently shown that NP has constant-round concurrent zero-knowledge arguments [9]. His

[10] A similar use was independently suggested by Damgard [9] (see discussion below).

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

public-key model postulates that the public-keys deposited in the public-file are legal, and furthermore that the user (or somebody else) knows the corresponding private-key. We stress that our public-key model is much milder (see above).

More on the model. A possible critique the public-key model (even in our mild form) is that this model postulates that registration takes place before any interaction between users may take place. One may claim that in *some* settings this is not desirable, as one may want to allow users to join-in (i.e., register) also during the active life-time of the system. It is indeed desirable to allow parties to register at all times. Note, however, that such a flexible model requires some restriction (as otherwise it coincides with the "vanilla" model– that is, the model in which no set-up stage or special stage or model is used). We thus suggest two intermediate models in which we can obtain our result.

1. One possibility is to postulate that 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" ensures that whatever sessions were in progress before registration have terminated by now. Namely, parties 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 [12], 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 [12] 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.

2. A different possibility is to require newly registered public-keys to be used only after authorization by a trusted "switchboard", which may interact 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.

Let us repeat here that registration is only required of verifiers. Again, this 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).

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 $r : \mathsf{N} \rightarrow \mathsf{N}$, 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). (For further details see Section 7.3.)

3 Preliminaries

Interactive proof systems. Throughout this paper we consider interactive proof systems [28] 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) [7] 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 $1/p(|x|)$.

Round-complexity. For simplicity, we consider only interactive proof systems in which the total number of message-exchanges (a.k.a. *rounds*) is a pre-determined (polynomial-time computable) function of the common input. We are specially interested in interactive proof systems in which this number is a constant; these are called *constant-round* interactive proof systems.

Zero-knowledge – strict versus expected polynomial-time simulators. We adopt the basic paradigm of the definition of zero-knowledge [28]: The output of every probabilistic polynomial-time adversary which interacts with the designated prover on a common input in the language, ought to simulatable by a probabilistic polynomial-time machine (which interacts with nobody). The latter machine is called a simulator. We mention that the simulators in Part I of the paper work is *strict* polynomial-time, whereas those in Part II work in *expected* polynomial-time. (As Part II 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 [21, 18].)

Witness indistinguishable proof systems. We also refer (or, actually, extend) the definition of witness indistinguishable proof systems (cf., [15]). 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.

The models considered: In this paper we consider two main models, depending on the *initial set-up assumptions*. The vanilla case, considered in Part I, is when no set-up assumptions are made. This is indeed the "simplest" model typically employed in theoretical works regarding secure two-party and multi-party computation. In Part II we consider the *public-key model* as described in subsection 2.4.

Part I
The Vanilla Model

4 Definitional Issues

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 residual 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 $P_{x,y,\omega}$ many times, each such interaction is called a session. In each such session, $P_{x,y,\omega}$ behaves as P would have behaved on common input x, auxiliary-input y, and random-tape ω. Thus, the adversary and $P_{x,y,\omega}$ engage in polynomially-many sessions; but whereas $P_{x,y,\omega}$'s actions in the current session are oblivious of other sessions (since $P_{x,y,\omega}$ mimics the "single session strategy" P), 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.

4.1 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.[12] 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.

Definition 4.1 (resettable security – vanilla model): *A prover strategy P is said to be* resettable zero-knowledge (rZK) *on L if for every probabilistic polynomial-time adversary V^* as below there exists a probabilistic polynomial-time simulator M^* so that the following two distribution ensembles, where each distribution is 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 $\overline{y} = y_1, ..., y_{\text{poly}(n)}$, are computational indistinguishable:*

`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-tape, $\omega_1, ..., \omega_t$, 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 initiate polynomially-many interactions 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*

[12] Actually, this part of the history may be omitted from these messages, since it can be re-computed by the receiver itself. Furthermore, it is actually not needed at all. We choose the current convention for greater explicitness.

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.[13]

A prover strategy P is said to be resettable witness indistinguishable (rWI) *on L if every two distribution ensembles of Type 1, where each distribution is indexed by a sequence of common inputs $\overline{x} = x_1, ..., x_{\text{poly}(n)} \in L \cap \{0,1\}^n$, depending on two different sequence of prover's auxiliary-inputs, $\text{aux}^{(1)}(\overline{x}) = y_1^{(1)}, ..., y_{\text{poly}(n)}^{(1)}$ and $\text{aux}^{(2)}(\overline{x}) = y_1^{(2)}, ..., y_{\text{poly}(n)}^{(2)}$, are computationally indistinguishable. That is, we require that $\{\langle P(\text{aux}^{(1)}(\overline{x})), V^* \rangle(\overline{x})\}_{\overline{x}}$ and $\{\langle P(\text{aux}^{(2)}(\overline{x})), V^* \rangle(\overline{x})\}_{\overline{x}}$ are computationally indistinguishable.*

Several previously investigated aspects of zero-knowledge can be casted as special cases of the above general model. For example, *sequential composition* of zero-knowledge protocols coincides with a special case of the *non-interleaved model*, where one is allowed to run each $P^{(j,j)}$ once (and may not run any other $P^{(i,j)}$). More importantly, *concurrent zero-knowledge* coincides with a special case of the *interleaving model* where one is allowed to run each $P^{(j,j)}$ once (and may not run any other $P^{(i,j)}$).[14] Thus, *every resettable zero-knowledge protocol is concurrent zero-knowledge.*

Recall that, as stated above, all known zero-knowledge protocols are NOT resettable zero-knowledge. Furthermore, they are even NOT resettable witness indistinguishable; not even in the sequential, single-incarnation version. For example, ability to "reset" the original zero-knowledge proof for 3-Colorability [24], allows the adversary to fully recover the 3-coloring of the input graph used by the prover. Still (as shown below), resettable zero-knowledge interactive proofs for $\mathcal{NP}$ do exist, under standard intractability assumptions.

4.2 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 4.1.
2. *Single versus multiple incarnations*: Definition 4.1 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:

[13] Recall that the existence of black-box simulators implies auxiliary-input zero-knowledge (cf. [25, 22]).

[14] Indeed, the possibility to run various $P^{(i,j)}$'s (i.e., same j and varying i's) was never considered before. This refers to running the prover on the same random-tape but on different input, and is a natural extension of our notion of resettable zero-knowledge.

Theorem 4.2 *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.[15] 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, $\beta_{i_{j+1}} \stackrel{\text{def}}{=} \beta_{t+1}$ is the $j+1^{\text{st}}$ 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).[16]

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 4.2, 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 4.2 is false:

Proposition 4.3 *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.*

[15] 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.

[16] 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 [22], 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 4.1). 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.

5 How to construct resettable protocols

In this section we show how to construct resettable zero-knowledge proofs and constant-round resettable witness indistinguishable proofs for $\mathcal{NP}$.

Theorem 5.1 *Suppose that there exists a* two-round *perfectly-hiding commitment scheme.*[17] *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 a black-box simulator.*

Recall that the hypothesis holds if families of claw-free permutations exists, which in turn holds if the Discrete Logarithm Problem (DLP) is hard modulo primes p of the form $2q+1$ where q is a

[17] Recall that, by definition, the receiver's decision at the reveal phase depends only on the messages sent (i.e., the two messages exchanged between the two parties during the commit phase, and the single message sent by the sender in the reveal phase. The latter provides, without loss of generality, the sender's coins, and verification of the revealed message is done by running the (predetermined) sender's program. Thus, the receiver's decision in the reveal phase is deterministic.

prime. We note that the theorem holds also under the assumption that there exist constant-round (rather than two-round) perfectly-hiding commitment schemes that is computationally-binding also in the resettable model (i.e., when the receiver may be reset). Note that any two-round perfectly-hiding commitment scheme is computationally-binding in the resettable model.

We note that all known zero-knowledge protocols are not even resettable witness indistinguishable, and so the mere existence of a resettable witness indistinguishable protocol for languages outside $\mathcal{BPP}$ is interesting.

Theorem 5.1 is proven by presenting a general transformation that applies to a subclass of protocols that are zero-knowledge (resp., witness-indistinguishable) in the concurrent model. When applied to the concurrent zero-knowledge proof system of Ransom and Kilian [34], the transformation yields an resettable zero-knowledge proof system (and so establishes Part 2 of Theorem 5.1). Part 1 of Theorem 5.1 is established by applying the transformation to the constant-round zero-knowledge proof system of Goldreich and Kahan [21]. (We use the fact that zero-knowledge proofs are witness-indistinguishable, and that the latter feature is preserved under concurrent composition (cf. Feige [13]).)

We start by presenting a class of protocols to which our transformation applies. Next, we present and analyze our transformation. Finally, we show that the two protocols mentioned above (i.e., of [34] and of [21]) indeed belong to the class.

5.1 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., decommited) 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 and actually causes the prover to abort) or properly decommit to the predetermined value.

Although the above formulation suffices for our main results (i.e., deriving the conclusion of Theorem 5.1 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.

Before presenting the actual definition, we stress that the first verifier message is not necessarily the first message sent in the protocol. It may be preceded by a (single) message sent by the prover.

However, we require that the prover's randomization for this first message (in case it exists, which we assume w.l.o.g.), is decoupled from its randomization in later steps. Also, without loss of generality, we assume that the first prover message contains the common input.

Definition 5.2 (admissible protocols – a sketch): *A protocol* (P,V) *is called* admissible *if the following requirements hold:*

1. *The strategy of* P *utilizes two distinct random strings; the first is used to determine the very first prover-message* (denoted `msg` below)*, and the second is used to determine all later prover messages as described below. Both parts are of length polynomial in the length of the common input.*

2. *Each verifier message consists of two parts, called* main *and* authenticator*. It is required that the prover response to such a message is determined as follows: First the prover decides, based on its own messages*[18]*, the main part of the very first verifier-message and* (both parts of) *the current verifier-message, whether to abort or not.*[19] *In case the prover does not abort, it determines its respond message as a function of* the second part *of its random-tape and the* main part *of all verifier-messages.*[20]

3. *Let* V^* *be an arbitrary* (deterministic) *polynomial-size circuit representing a possible strategy for the verifier in the interactive proof* (P,V)*. Let* `msg` *denote the first prover message, generated at random according to* P*'s strategy. We consider a probabilistic polynomial-time oracle machine that is given oracle access to* $V^*_{\mathtt{msg}}(\cdot) \stackrel{\rm def}{=} V^*(\mathtt{msg},\cdot)$*. That is, the oracle machine may query* V^* *on every history of prover messages that starts with* `msg`*, which guarantees that all responses correspond to an interaction in which the first verifier message equals* $V^*(\mathtt{msg})$*. Then it is required that the probability, taken over the choices of* `msg` *and the internal coin tosses of the oracle machine, that the oracle machine obtains for some verifier-message two different main parts that are both properly authenticated is negligible.*

5.2 The Transformation – A Warm-up

Given a proof system (P,V) that satisfies Definition 5.2 and is "secure in the concurrent model", we construct a new strategy $\mathbf{P}$ for the resettable model as follows:

- The initial random-tape of $\mathbf{P}$ is viewed as a pair (r_1, f_2), where r_1 is of length adequate for the first part of P's random-tape and f_2 is a description of a pseudorandom function.
- Using r_1, prover $\mathbf{P}$ determines the first message just as P does, and sends it to the verifier.

[18] The above phrase postulates a deterministic decision, which suffices for our applications. We may allow the decision to be probabilistic; that is, depend also on designated portions of the second part of the prover's random-tape. 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 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.

[19] The definition can be further extended by allowing the prover 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.

[20] Note that the case in which the current verifier-message has an empty authenticator is a special case covered by the above.

- Upon receiving the first verifier-message, denoted $\mathtt{msg}'$, prover $\mathbf{P}$ first determines a string r_2 of length adequate for the second part of P's random-tape; that is, $r_2 \leftarrow f_2(\mathtt{msg}')$.
- All subsequent messages of $\mathbf{P}$ are determines as P would have determined them using r_2 as the (second part of the) random-tape.

The above description suffices for constructing a correspondingly secure prover for the *single-incarnation* resettable model, and it will be slightly augmented (in the next subsection) in order to cover the *multiple-incarnation* model. As a warm-up, we further simplify the analysis by assuming (unrealistically!)[21] that the first message of prover P is a fixed string, independent of r_1. This simplifying assumption will be removed in the next subsection, but it allows us to focus here on the main ideas underlying the analysis of the protocol transformation. Specifically, we prove the following:

Proposition 5.3 (for warm-up purposes only): *Suppose that (P, V) satisfies Definition 5.2, and that the first message of P is a fixed string. Let $\mathbf{P}$ be the prover strategy obtained from P by applying transformation above, assuming that pseudorandom functions exist. Then for every probabilistic polynomial-time V^* for 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. (But, unfortunately, we don't know whether there exists a concurrent zero-knowledge prover P as in the hypothesis of Proposition 5.3.)

Proof Sketch: By Theorem 4.2, 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) 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

[21] 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.

carries out with P) that still awaits a first-verifier-message,[22] 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 5.2. (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^*.

   (a) *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^*.

   (b) *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 5.2, 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 indentical responses supplied to it by W^*.)

[22] 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^*.

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 $\mathbf{P}$.

5.3 The Actual Transformation

The above description suffices for the *single-incarnation* resettable model. To deal with the *multiple-incarnation* model, we slightly augment the above construction: instead of setting $r_2 = f_2(\mathtt{msg}')$, where $\mathtt{msg}'$ is the first verifier-message, we set $r_2 = f_2(x, r_1, \mathtt{msg}')$, where x is the common-input. We also assume, without loss of generality, that $|r_1| > |x|$ (so that the probability that the same r_1 is selected at random is negligible). For sake of clarity we reproduce and expand the description of the transformation:

Construction 5.4 *Given an admissible protocol (P, V), and a collection of pseudorandom functions, we define a new protocol $(\mathbf{P}, \mathbf{V})$ as follows.*

The inputs: *the common input and auxiliary inputs to the two parties are as in (P, V). Let x denote the common input, and n denote the security parameter, which for simplicity may equal the length of x.*

The new prover's randomness *is viewed as a pair (r_1, f_2), where $r_1 \in \{0,1\}^{\mathrm{poly}(n)}$ is of length adequate for the first part of the random-tape of P, and $f_2 : \{0,1\}^{\leq \mathrm{poly}(n)} \to \{0,1\}^{\mathrm{poly}(n)}$ is a description of a function taken from an ensemble of pseudorandom functions.*

The new verifier *is identical to V.*

The new prover: *The new prover emulates the actions of P, when the latter uses random-tape (r_1, r_2), where r_2 is determined by applying f_2 to the verifier's first message. Specifically:*

1. *The first message sent by $\mathbf{P}$ equals the message that P would have sent on random-tape having r_1 as its first part. That is, $\mathbf{P}$ sends $\mathtt{msg} = P(x, r_1)$ to the verifier.*
2. *Upon receiving the verifier's first message, denoted $\mathtt{msg}'$, the prover $\mathbf{P}$ sets $r_2 = f_2(x, r_1, \mathtt{msg}')$. From this step on, $\mathbf{P}$ emulates the actions of P using (r_1, r_2) as P's random-tape. Recall that, by Definition 5.2, these actions are actually independent of r_1.*

To overcome the gap left by Proposition 5.3, we need to consider admissible protocols (P, V) in which the first prover-message is not fixed, but is rather a function of r_1. The problem with such protocols (P, V) is that an adversary may initiate two sessions with (the same incarnation of) $\mathbf{P}$, and use different first verifier-messages (i.e., $\mathtt{msg}'$) in them, whereas the first prover-message $\mathtt{msg}$ determined (before) by $\mathbf{P}$ will be the same. Thus, applying the proof of Proposition 5.3, the resulting W^* needs to interact with two sessions of P having the same first part of random-tape (i.e., r_1), something that is not allowed in the concurrent model (in which W^* is supposed to operate). Such interactions are explicitly allowed in the hybrid model presented below, which allows interactions with sessions of P sharing the same r_1 but still postulates that the second part of the random-tape (i.e., r_2) is independently chosen per each session (as in the concurrent model). Thus, the hybrid model lies between the resettable model and the concurrent model. We feel that the hybrid model is closer in spirit to the concurrent model (than to the resettable model). Specifically, in natural protocols (such as the ones to which we apply the transformation of Construction 5.4), the effect of re-using the same r_1 is much lesser than the effect of re-using the same r_2. The reason is that in such protocols the randomness of r_1 is used merely to establish that the verifier's messages are essentially determined by the first verifier-message.

Definition 5.5 (hybrid model): *We consider an adversary as in the resettable model, interacting with incarnations of P, where incarnation $P^{(i,j,k)} = P_{x_i, y_i, \omega_{j,k}}$ is determined by the common-input x_i, auxiliary-input y_i, and* (two-part) *random-tap $\omega_{j,k} = (r_1^{(j)}, f_2^{(k)})$, where the $r_1^{(j)}$'s and the $f_2^{(k)}$'s are independently and uniformly selected in the relevant space and fixed. Here the adversary is* *allowed to initiate at most one session with each incarnation $P^{(i,j,k)}$. Furthermore, for every k, the adversary is allow to hold at most one session with one of the incarnations in $\{P^{(i,j,k)} : i,j\}$. That is, for every k, the number of sessions held with incarnations of the form $P^{(\cdot,\cdot,k)}$ is at most 1.*

Thus, in the hybrid model, for every i,j, the adversary may hold polynomially-many sessions with incarnations in $\{P^{(i,j,k)} : k\}$, but in each of these sessions a different k is used. In contrast, in the (standard) resettable model, when applied to provers as above, the adversary may hold polynomially-many sessions with each $P^{(i,j,j)}$. On the other hand, in the concurrent session, for every j, the number of sessions held with incarnations of the form $P^{(\cdot,j,j)}$ is at most 1.

In the next theorem we relate the security of $\mathbf{P}$ in the resettable model to the security of P in the hybrid model. (Security means either zero-knowledge or witness-indistinguishability properties.)

Theorem 5.6 *Suppose that (P,V) satisfies Definition 5.2, and let $\mathbf{P}$ be the prover strategy obtained from P by applying Construction 5.4, assuming that pseudorandom functions exist. Then for every probabilistic polynomial-time V^* for the resettable model, there exists a probabilistic polynomial-time W^* in the hybrid 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 zero-knowledge (resp., witness-indistinguishable) in the hybrid model then $\mathbf{P}$ is resettable zero-knowledge (resp., witness-indistinguishable).

We comment that in the hybrid model there is no difference between the actions of P and of $\mathbf{P}$. For sake of clarity, we preferred to stated and prove Theorem 5.6 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.

Proof Sketch: By Theorem 4.2, it suffices to consider the sequential variant of the resettable model. Recall that we deal with the multiple incarnation model, and so V^* may interact (sequentially) with polynomially-many incarnations of $\mathbf{P}$, invoking each such incarnation polynomially many times. As in Definition 4.1, these incarnations are denoted $\mathbf{P}^{(i,j)} = \mathbf{P}_{x_i,y_i,\omega_j}$, where x_i is a common input, y_i a corresponding auxiliary input, and the ω_j's are uniformly and independently selected random pads. We will construct a hybrid-model adversary, W^*, that interacts with incarnations of P, denoted $P^{(i,j,k)}$'s (as in Def. 5.5). To satisfy Definition 5.5, 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 $P^{(i',j',k)}$. The proof (slightly) extends the ideas presented in the proof of Proposition 5.3.

Again, 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 are computationally indistinguishable. From this point on, we identify the random-tape of P with a pair (r_1, f_2), where f_2 is a truly random function.

The construction of W^*. (The following construction slightly extends the construction presented in the proof of Proposition 5.3.) 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 the prover P, obtains its first message, and forwards `msg` to V^*. Specifically, W^* initiates a session with $P^{(i,j,k)}$, where k is a new index not used so far. That is, W^* maintains a counter for k that is incremented each time the current case is encountered.

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 first message obtained from $P^{(i,j,k)}$ is identical to the first message obtained previously from $P^{(i,j,k')}$.

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

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

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 first verifier-message in the current session, and assume that this message equals a first-verifier-message, $\texttt{msg}'$, 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 $(i, j, \texttt{msg}')$, 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 $\texttt{msg}'$ 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 $\texttt{msg}'$.)

4. *V^* sends a valid non-first-message to* $\mathbf{P}^{(i,j)}$: That is, we refer to the case where V^* sends a non-first-message in the current session with $\mathbf{P}^{(i,j)}$, and assume that this message is valid; that is, P accepts it as valid as per Definition 5.2. (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^* with $\mathbf{P}^{(i,j)}$ in which the first verifier-message equals $\texttt{msg}'$, where $\texttt{msg}'$ 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^*.

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

      (Note that by Definition 5.2, 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 fixed verifier-message, are identical. It follows that V^* will be content with the indentical responses supplied to it by W^*.)

5. V^* *sends an invalid non-first-message to* $\mathbf{P}^{(i,j)}$: That is, we refer to the case where V^* sends a non-first-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 first-verifier-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 first-verifier-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 first-verifier-message, the actions of $\mathbf{P}$ are essentially determined by the first verifier-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 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 $\mathbf{P}^{(i,j)}$-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 $\mathbf{P}^{(i,j)}$-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). ■

5.4 Deriving a constant-round rWI proof for NP

Part 1 of Theorem 5.1 is proved by applying Construction 5.4 to an admissible (as per Definition 5.2) proof system for $\mathcal{NP}$ that is constant-round and witness-indistinguishable in the hybrid model (of Definition 5.5). Thus, we need to assert the existence of such a protocol.

Proposition 5.7 *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 witness-indistinguishable in the hybrid model.*

Combining Theorem 5.6 and Proposition 5.7, Part 1 of Theorem 5.1 follows.

Proof Sketch: It suffices to present a proof system for some NP-complete problem (and we refer to 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 5.2) or are not witness-indistinguishable in the hybrid model.[23] Fortunately, as we show below, the (5-round) zero-knowledge proof system of Goldreich and Kahan [21] is both admissible and witness-indistinguishable in the hybrid model. On an abstract level, the proof system of [21] 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,[24] 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".

(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.

[23] For example, in the zero-knowledge proof system of Goldreich, Micali and Wigderson [24], 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 protocol 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. [33], the protocol is not admissible (since the verifier has total freedom in selecting the edges).

[24] See discussion following this abstract presentation.

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).[25] Thus, we need to integrate such (two-round) commitment schemes in the above protocol. 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 protocol is indeed admissible, Furthermore, as shown in [21], the protocol is indeed a 5-round zero-knowledge proof system for Graph 3-Colorability. Thus, it follows that the protocol is witness-indistinguishable in the concurrent model (cf. [13]). 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 feature of the verifier's commitment, and this feature continues to hold also when the sender commits to several values using the same receiver message.[26] Thus, the above protocol is witness-indistinguishable in the hybrid model, and the proposition follows. ■

Comment: We mention that the above protocol is probably NOT resettable zero-knowledge; in fact, it is probably NOT even *concurrent* zero-knowledge. This follows from recent work of A. Rosen (priv. comm.): extending [22, 32], Rosen shows that no language outside $\mathcal{BPP}$ can have a 8-round protocol that is concurrent zero-knowledge via black-box simulation.

5.5 Deriving a resettable zero-knowledge proof for NP

Part 2 of Theorem 5.1 is proved by applying Construction 5.4 to an admissible (as per Definition 5.2) proof system for $\mathcal{NP}$ that is zero-knowledge in the hybrid model (of Definition 5.5). Thus, we need to assert the existence of such a protocol.

Proposition 5.8 *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.*

Combining Theorem 5.6 and Proposition 5.8, Part 2 of Theorem 5.1 follows.

Proof Sketch: We show that the concurrent zero-knowledge proof system of Ransom and Kilian [34] can be slightly modified so that the resulting protocol satisfies all requirements.

We start by reviewing the Ransom and Kilian protocol [34]. In essence, the protocol consists of two stages. In *the first stage*, which is independent of the actual common input, k instances of *coin tossing into the well* [3] are executed in a specific manner to be described, where k is the security parameter (or a parameter that is polynomially related to the security parameter). Specifically, first

[25] 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 decommits 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. [17]).)

[26] See an analogous discussion in the proof of Proposition 5.8.

the verifier commits to k random bit sequences, $v_1, ..., v_k \in \{0,1\}^k$, and next k iterations proceed so that in each iteration the prover commits to a random bit sequence, p_i, and the verifier decommits to the corresponding v_i. The result of the i^{th} coin-toss is defined as $v_i \oplus p_i$ and is known only to the prover. In *the second stage*, the prover provides a witness indistinguishable (WI) proof that either the common input is in the language or one of the outcomes of the k coin-tosses is the all-zero string (i.e., $v_i = p_i$ for some i). Intuitively, since the latter case is unlikely to happen in an actual execution of the protocol, the protocol constitutes a proof system for the language. However, the latter case is the key to the simulation of the protocol in the concurrent zero-knowledge model: Whenever the simulator may cause $v_i = p_i$ to happen for some i, it can simulate the rest of the protocol (and specifically Stage 2) by merely running the WI proof system with v_i (and the prover's coins) as a witness. (By the WI property, such a run will be indistinguishable from a run in which an NP-witness for the membership of the common input (in the language) is used.)

Ransom and Kilian do not say which WI protocol should be used; in fact, for their purposes any such protocol will do [34]. However, as a starting point to our modification, we use a *specific* WI proof system: the zero-knowledge proof system of [21], which is reviewed in the proof of Proposition 5.7. Thus, at an abstract level, the protocol is as follows.

Common Input: x supposedly in the language $L \in \mathcal{NP}$, and a security parameter k.[27]

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

Stage 1: This stage has little effect in real interactions between the prover and the verifier, yet it provides a "trapdoor" for the simulation.

1. The verifier commits to k uniformly selected k-bit strings, using a two-round perfectly-hiding commitment scheme. That is, the prover, which acts a receiver, sends the first message, to which the verifier responds by uniformly selecting $v_1, ..., v_k \in \{0,1\}^k$, and sending to the prover its commitment to each of the v_i's. 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 $v_1, ..., v_k$.
2. For $i = 1, ..., k$, the following two-round interaction goes on. First the prover commits (in a perfectly-bidding way) to a random k-bit string, denoted p_i, and next the verifier decommits to β_i by providing v_i along with the randomness used in forming β_i from v_i. We stress that p_i is uniquely determined by the string, denoted α_i, sent by the prover.

Stage 2: The prover provides a proof that *either $x \in L$ or $v_i = p_i$, for some i* (i.e., either $x \in L$ or *for some i there exists a decommitment string validating that the string α_i is a commitment to the value v_i*). The NP-witness used by the prover is w, and the proof system is the one reviewed above (after reducing the above NP-statement to a statement about 3-Colorability of the resulting graph). Recall that this proof system starts with a two-round perfectly-hiding commitment of the verifier (to a sequence of uniformly selected graph-edges), followed by a perfectly-binding commitment of the prover (to a random relabeling of colors), then the verifier reveals all values (i.e., edges), and finally the prover reveals the indicated values (colors of indicated endpoints).

As usual, in both stages, whenever a party fails to provide a message as instructed the other party halts (detecting an obvious cheating attempt).

[27] For simplicity we equate all "security governing" parameters such as the number of iterations in Stage 1, the length of strings committed to in Stage 1, and the security parameters used in the various commitment schemes, etc.

Recall that the above protocol is concurrent zero-knowledge (cf. [34]). However, it is not admissible. Below we modify the protocol so as to make it admissible and zero-knowledge also in the hybrid model.

The modified protocol: We slightly modify the above protocol by moving the verifier's second commitment, which takes place in the first two rounds of Stage 2, to Step 1 of Stage 1. That is, both the verifier commitment to k uniformly selected k-bit strings as well as its commitment to random edges in the reduced graph are made up-front, as the very first thing in the entire protocol. We comment that there is a minor problem here, since the graph (to which the proof system (of Stage 2) is applied) is not determined yet (i.e., in the very beginning of Stage 1). However, (an upper boun on) the size of this graph is known. Thus, we let the verifier commit to random pairs of vertices, and use a standard convention by which the prover interprets each non-edge as some fixed edge (cf. [24, p. 714]). The only (other) thing requiring change is to increase the number of parallel repetitions; that is, set $t = n^3$ (rather than $t = n \cdot |E|$), where n (resp., $|E|$) is the number of vertices (resp., edges) in the graph. This is done in order to guarantee that the probability that a bad edge is hit in t tries, where in each trial we select a random pair of vertices, is at least $(1 - n^{-2})^t = \exp(-n)$.[28]

Firstly, we observe that the modified protocol is indeed admissible. Next, we analyze the effect of our modification. Intuitively, the above modification only restricts the power of the verifier (as it needs to commit earlier to its choices). Thus, we should verify (which is easy to do) that the above protocol maintains its soundness (i.e., remains a proof system for Graph 3-Colorability). On the other hand, one can show that the protocol remains zero-knowledge in the concurrent model (since the verifier power has only decreased). Furthermore, we need and will show that the modified protocol is zero-knowledge in the hybrid model.

Let P_{org} denote the original prover and P_{mod} the modified one. We consider the following auxiliary model as applied to BOTH P_{org} and P_{mod}: First, for each of the two instances of the two-round perfectly-hiding commitment scheme (in which the verifier sends a commitment to the prover), a polynomial number of independent receiver-messages are generated and given to the adversary verifier. (We refer to the first message in the two-round commitment scheme, which is a message from the receiver to the sender.) Next, the adversary is allowed to initiate polynomially-many independent sessions with the prover. In each session, the verifier indicates which pair of receiver-messages (out of the above list) it wishes to use, and the prover executes its program skipping the generation of these messages, which are instead taken as the messages selected by the verifier (from the list). We stress that in each session, the prover uses an independently chosen random-tape (which corresponds to the second part of the random-tape of prover P_{mod}). We observe that:

1. The proof that P_{org} is zero-knowledge in the concurrent model extends to showing that P_{org} is zero-knowledge in the auxiliary model.

   This is the case since the only difference between the two models is in whether the same receiver message is used for several sender's commitments or not. The computational-binding property, which is stated for a single use of a receiver's message, holds also for multiple uses. Furthermore, the randomness of the receiver's message is only used to argue that the verifier's commitment is binding, and the simulator and its analysis use this fact as a "black box". Thus, the simulator and its analysis extend to show that P_{org} is zero-knowledge in the auxiliary model.

[28] Before, the modification we had $(1 - |E|^{-1})^t = \exp(-n)$.

2. Intuitively, the transformation of $P_{\rm org}$ to $P_{\rm mod}$ only restricts the power of the verifier, and so the fact that $P_{\rm org}$ is zero-knowledge in the auxiliary model implies that so is $P_{\rm mod}$. This intuition can be proven valid (in the auxiliary model).

   Specifically, for every adversary V^* interacting with $P_{\rm mod}$ (in the auxiliary model), we can construct an adversary W^* that interacts with $P_{\rm org}$ (in the auxiliary model) so that W^* has output identical to the output of V^*: We merely let W^* emulate the actions of V^* in the natural manner. Specifically, the first message that W^* sends in each session is determined by the first message sent by V^* in the corresponding session, where the latter message is a pair of verifier's commitments and W^* sends only the first one (as its first message in the current session) and saves the second message for future use. When W^* encounters the point (in the beginning of Stage 2) where it needs to send a second commitment, it merely sends the message stored at the very beginning of the emulation of the current session.

3. When applied to prover $P_{\rm mod}$, the hybrid model is equivalent to the auxiliary model: Specifically, if $P_{\rm mod}$ is zero-knowledge in the auxiliary model then it is zero-knowledge in the hybrid model.

   Furthermore, for every adversary V^* interacting with $P_{\rm mod}$ in the hybrid model, we can construct an adversary W^* that interacts with $P_{\rm mod}$ in the auxiliary model so that W^* has output identical to the output of V^*: Given an upper bound B on the running time of V^*, machine W^* first asks to be given B receiver-messages (as it is entitled in the auxiliary model). Next, W^* emulates the actions of V^* in the natural manner. Specifically, when V^* invokes incarnation ${P_{\rm mod}}^{(i,j,k)}$ (as in the hybrid model), W^* invokes a new session with $P_{\rm mod}$ asking it to use the $j^{\rm th}$ receiver-message (in the list provided above), and uses the latter session in order to emulate ${P_{\rm mod}}^{(i,j,k)}$.

Combining the above three facts, the proposition follows. ■

Part II

The Public-Key Model

6 Discussion and Definition

The vanilla model, considered in Sections 4–5, is when no set-up assumptions are made. This is indeed the "cleanest" model typically employed in theoretical works regarding secure two-party and multi-party computation.

By the public-key model we mean a model in which all users are assumed to have deposited a public-key in a file that is accessible by all users at all times. The only assumption about this file is that it guarantees that entries in it were deposited before any interaction among the users takes place. No further assumption is made about this file, and so in particular an adversary may deposit many (possibly invalid) public-keys in it (and, in particular, without even knowing corresponding secret keys or whether such exist). Access to the file may be implementable by either several identical servers or by providing users with certificates for their deposited public-keys.

A more realistic public-key model allows parties to register at all times. Note however that such a flexible model requires some restriction (as otherwise it coincides with the vanilla model). One possibility is to make some mild timing assumption such as that all parties can distinguish between some predetermined large delay (which all newly registered public-keys must undergo before being used) and a small delay (which upper bounds the communication delays in actual interaction).[29] A different possibility is to require newly registered public-keys to be used only after authorization by a trusted "switchboard", and occasionally updating (i.e., replacing) the entire system. The second alternative seems better suited to the smart-card application discussed in the introduction. For sake of simplicity, we assume throughout the rest of this section that registration occurs before any interaction between the users takes place. The treatment of more flexible models is deferred to a future version of this work. We comment that variants of the public-key model are a standard model in many applied works.

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 which issues them 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.

We stress that we actually use weaker assumptions. Specifically, in both the latter models, we only need that potential verifier will deposit public-keys and/or participate in a pre-computation. This is not required of users who are only going to play the role of provers.

Definition (sketch): Analogously to Definition 4.1, we may define resettable zero-knowledge in the public-key model: The only modification is that the prover and verifier (as well as the simulator) have access to a public-file which was generated by the adversary V^* before all interactions began. Thus, the public-file may be viewed as part of the common input as far as the zero-knowledge (i.e., RZK) condition holds. (In the soundness (in fact computational-soundness) condition one needs to

[29] As explained in Section 2 such an assumption does not effect typical interactions; whereas the timing assumption in [12] amounts to slowing down all interactions to meet some a-priori upper bound (which must be quite conservative to prevent abort of honest interactions).

consider what happens when the public-file is randomly generated (by a honest verifier), and the actual input is fixed possibly afterwards.)

7 Constant-round RZK for NP in the public-key model

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 7.1 *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.*

7.1 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 has some extra features informally stated below. For a precise definition see Appendix A.

1. A two-round perfect 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 subexponential circuits. That is, there exists a constant $\epsilon > 0$ so that for sufficiently large security parameter K no sender strategy which is implementable by a circuit of size 2^{K^ϵ} can decommit in two different ways with probability greater than 2^{-K^ϵ}.

2. A constant-round perfect commitment scheme, denoted PC2. (This scheme corresponds to the one used in the actual implementation of Step (V1) above.) 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 [18, Sec. 4.8.2]. We comment that a PC1 scheme can be constructed under the assumption the DLP is hard for subexponential 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 subexponential circuits.[30]

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.

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 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 $f : \{0,1\}^{\text{poly}(n)} \to \{0,1\}^{\text{poly}(n)}$.

The protocol itself is an adaptation of the resettable witness indistinguishable proof system of Section 5.4 with Step (V1) being replaced (or rather implemented) by current Steps (1) and (3). Another important change is the replacement of former Step (P1) 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 which 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.]

[30] In fact, it suffices to have collision-intractable family of hashing function, provided it carries trapdoors and is strong wrt subexponential circuits.

(2) This step is analogous to Step (V1) in the protocol of the previous subsection: The verifier commits to a sequence of $t \stackrel{\text{def}}{=} n \cdot |E|$ uniformly and independently chosen edges. The commitment is done using the constant-round perfect 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 are 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 the previous subsection: 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, 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 $(jn + v)^{\text{th}}$ 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 $(G, \phi, \mathit{history})$, 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., [28, 24]), 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 decommited 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 interaction with the prover. Subsequent interactions are simulated in the same way assuming that the execution of Steps (1)–(2) of the current interaction is different than in all previous interactions. Otherwise, we simulate Steps (3) and (5) by copying the values used in the previous interaction. 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.

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 which 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 which 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.

7.2 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 subexponential 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 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 $\text{trap}(i)$.

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).[31] 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[32] (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

[31] This property holds also for other protocols for NP, but not for the 3-Colorability protocol of [24]. Any protocol having the property will do.

[32] 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.

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., [28, 24]), which merely selects a random string $c \in \{0,1\}^k$ and "simulates" Step (**pok1**) so 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 $\text{poly}(k) \cdot 2^k$ on input the index i.

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. B.6 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 the trapdoor).[33] To extract the Hamiltonian cycle we just invoke the knowledge-prover many times, each time it answers with the

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

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 $\text{poly}(k)/(p - 2^{-k})$. By a trivial modification, we obtain a knowledge extractor which for any $p > 0$ with overwhelming probability runs for time $\text{poly}(k)/p$, and in case $p > 2^{-k}$ also retrieves the trapdoor.[34]

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 invocations of the entire protocol). Assuming that this is indeed the case avoids the subtle problem discussed in the previous subsection. Still 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 of the main protocol. The simulator will proceed by simulating one interaction after the other, where a single interaction 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 interaction 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 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 interactions with the prover. Furthermore, suppose that in the first interaction it fails to convince KV played by the prover, but in the second 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 interaction, since the challenge is determined already by the first interaction. Thus, the simulator cannot complete the simulation of the second interaction, unless it "rewinds" upto the first interaction in which the same Step (**pok1**) message is used.[35] This need to "rewind" interactions which were already completed may lead to exponential blow-ups as discussed by Dwork, Naor and Sahai [12]. What saves us here is that the number of times we possibly need to "rewind" is a-priori bounded by

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

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

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.)

Resolving the problem – a sketch: Let us reproduce and further abstract the problem we need to analyze. For sake of simplicity, we will consider only non-interleaving adversaries (yet this assumption can be removed as in Section 5) We are dealing 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 Step (**pok1**) message.]

(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.

[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, $f', f'', ...$, 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 occuring, 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 upto $|I| + 1$ 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' (wrt to the function f') just as we do in the main simulation (where we currently try to extend H wrt 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).

7.3 Almost constant-round RZK under weaker assumptions

Using a perfect 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 [24]) 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 $s : \mathsf{N} \rightarrow \mathsf{N}$. In particular, we obtain

Theorem 7.2 *Let $r : \mathsf{N} \rightarrow \mathsf{N}$ 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 perfect commitment scheme PC1 also in role of the ("weaker") scheme PC2, we obtain resettable zero-knowledge property also against subexponential 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.

7.4 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.

Details can be found in our original technical report [8].

References

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

[2] 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

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

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

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

[6] 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.

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

[8] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable Zero-Knowledge. *ECCC*, TR99-024, 1999. Also available from the *Theory of Cryptography Library.*

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

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

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

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

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

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

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

[16] 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.

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

[18] 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`.

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

[20] 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.*

[21] 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.

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

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

[24] 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.

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

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

[27] 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).

[28] 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.

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

[30] 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.

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

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

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

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

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

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

Part III
Appendices

These appendices are reproduced from old versions; some inconsistencies with and repetitions of the main text may occur.

Appendix A: 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 perfect 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 perfect 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.

A.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 suct 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 $q \overset{\text{def}}{=} (p-1)/2$, and $x \in \mathsf{Z}_q^*$.*

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

A.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 A.3 (standard commitment scheme): *A* standard commitment scheme *is a probabilistic polynomial-time algorithm, denoted C satisfying:*

(Computational) Secrecy: *For every v, u of equal* 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* 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., [23]), 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 subexponential-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.

A.3 Perfect Commitment Schemes

We start by defining two-round perfect 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 $S(1^n, v, \texttt{msg})$, where `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 A.4 (perfect two-round commitment scheme): *A* perfect 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* 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, $S(1^n, v, \texttt{msg})$, as consisting of uniformly selecting $s \in \{0,1\}^{\mathrm{poly}(n,|v|)}$, and computing a polynomial-time function $S'(1^n, v, s, \texttt{msg})$, where msg *is the receiver's message. A cheating sender tries, given a receiver message* msg*, to find two pairs (v, s) and (v', s') so that $v \neq v'$ and yet $S'(1^n, v, s, \texttt{msg}) = S'(1^n, v', s', \texttt{msg})$. 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 perfect two-round commitment scheme can be constructed using any claw-free collection (cf., [21]). 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 perfect 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[36]), 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 perfect commitment schemes in Section 7 can be easily formulated. The strong computational binding feature is formulated by extending the Computational Binding Property (of Def. A.4) to hold for subexponential 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 decommiting. That is, there exists a polynomial-time algorithm A so that for every $(\texttt{msg}, \mathrm{aux})$ 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

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

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 $S'(1^n, v, s, \texttt{msg})$). Thus, one may generate random commitments c (by uniformly selecting s and computing $S'(1^n, 0^{\mathrm{poly}(n)}, s, \texttt{msg})$) 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 $s + (v-u)r \bmod q$.

Appendix B: 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 [18].

[36]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.

B.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 B.5 (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 B.6 (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*

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

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.*

B.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 B.7 (Basic proof system for HC):

- Common Input: *a directed graph $G = (V, E)$ with $n \overset{\mathrm{def}}{=} |V|$.*
- 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 $(\pi(i), \pi(j))^{\mathrm{th}}$ 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 B.8 *The protocol which results by k parallel repetitions of Construction B.7 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* $\mathrm{poly}(n) \cdot 2^k$ *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.*