Resettable Zero-Knowledge*

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

October 24, 1999

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 him 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 a 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 [28].

[†]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 **3**

2 Overview **6**
2.1 The notion of resettable zero-knowledge . . . 6
2.2 NP has constant-round resettable-WI . . . 8
2.3 NP has resettable-ZK proofs . . . 9
2.4 NP has resettable-ZK arguments . . . 9
2.5 The Public-Key model . . . 10

3 Preliminaries **12**
3.1 Standard Conventions . . . 12
3.2 The models considered . . . 13

I The Vanilla Model **14**

4 Definition **14**

5 Resettable Witness-Indistinguishable proofs for NP **17**

6 NP has Resettable ZK proof systems **21**
6.1 The protocol . . . 21
6.2 The simulation . . . 23
6.3 An alternative proof . . . 30

7 NP has Resettable ZK Arguments **34**
7.1 The Strong DLP Assumption for Safe Primes . . . 34
7.2 Initial Remarks About Our Protocol . . . 35
7.3 DLP-Based Commitment Schemes . . . 36
7.4 An rZK Argument for 3-Colorability . . . 37

II The Public-Key Model **41**

8 Discussion and Definition **41**

9 Constant-round RZK for NP in the public-key model **42**
9.1 RZK for NP in the preprocessing model . . . 42
9.2 Back to the bare public-key model . . . 45
9.3 Almost constant-round RZK under weaker assumptions . . . 50

10 Alternative Constant Round RZK Protocol for NP in the Public Key Model **51**
10.1 Two Types of Commitments . . . 51
10.2 Discrete-Log Implementations of Type-1 and Type-2 Commitment . . . 54
10.3 An RZK Protocol For 3-Coloring Using Public Keys . . . 55

Bibliography **57**

III Appendices **60**

Appendix A: Commitment Schemes **60**
A.1 The Strong DLP Intractability Assumption 60
A.2 Standard Commitment Schemes 60
A.3 Perfect Commitment Schemes 61

Appendix B: Blum's Proof of Knowledge **62**
B.1 Proofs of Knowledge 63
B.1.1 Preliminaries 63
B.1.2 Knowledge verifiers 63
B.2 Blum's Protocol 64

1 Introduction

The notion of a zero-knowledge interactive proof was put forward and first exemplified by Goldwasser, Micali and Rackoff [29]. The generality of this notion was demonstrated by Goldreich, Micali and Wigderson [25], 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 [8], witness indistinguishability [16], and zero-knowledge proofs of knowledge [29, 38, 15]. 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 [17], 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 [17]), *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 [25]) 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 [29] the following basic protocol is repeated: the prover randomly chooses $r \in Z_n^*$ and sends $r^2 \bmod n$ to the verifier; the verifier sends a random bit b to the prover; and the prover sends back r if $b = 0$, and $xr \bmod n$ if $b = 1$. Assume now that the prover is forced to execute twice with the same coins r the basic protocol. Then, by sending $b = 0$ in the first execution and $b = 1$ in the second execution, the verifier learns both r and xr and thus trivially extract x, a square root of z mod n.

A New Notion. In this paper we extend the classical notion of zero-knowledge by introducing the notion of *Resettable Zero-Knowledge* (rZK for short).[4] In essence, a rZK proof is a zero-knowledge proof in which a verifier learns nothing (except for the verity of a given statement) even if he can interact with the prover polynomially many times, 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 [36, 32], that one-way functions exist.

[2] Zero-knowledge proofs in which the prover is deterministic exist only for BPP languages (cf., [26]).

[3] For instance, in [17] 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 [21], 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.

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 commitment schemes. In some cases, any commitment scheme with perfect privacy would do. In other cases, we need a more sophisticated primitive, which we call *Verifiable Commitment* (see Section 10 for a definition). Verifiable commitment can be implemented under traditional complexity assumptions, such as the hardness of the Discrete Log Problem (DLP), or on 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 $2^{k^{\epsilon}}$, 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 $2^{n^{\epsilon}}$, 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 which 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 [37] 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 pre-processing, 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 rZKprotocols yields a constant-round and simple solution to the important (and extensively investigated) concurrent ZK problem.

Consequences for proofs of identity. Fiat and Shamir in [17] 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 [17] 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 [17]). 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 here 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 actually used. Furthermore, assuming the existence of pseudorandom functions, protocols satisfying the simpler definition can be transformed into ones satisfying the full-fledge definition.

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 $P_{x,y,\omega}$. 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 the current interaction are independent of prior interaction (since $P_{x,y,\omega}$ mimics the "single interaction

[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.

strategy" P), the actions of the adversary may depend on prior interactions. In particular, the adversary may repeat the same messages sent in a prior 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 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., [27, 39])*:*

`Distribution 1` *is defined by the following random process which 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 $i \geq 0$ and $j = 1, ..., t$, the $it + j^{\text{th}}$ message sent by V^* is treated as the j^{th} message in the i^{th} interaction of P', and accordingly the response is $P'(\mathtt{msg}_{it+1}, ..., \mathtt{msg}_{it+j})$, 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 [25], 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 (i.e., actually linear in the number of edges), 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. 1) is to allow the adversary to 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. Indeed, as stated above, using pseudorandom function, one may transform protocols satisfying the single-incarnation definition (of above) to protocols satisfying the general definition in which polynomially-many independent incarnations are allowed.

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 [16] 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 *provably* 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. 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., [26]). 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 [20]. 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 which the verifier may want to alter later (by rewinding) in order to extract knowledge. Thus, the zero-knowledge proof system for 3-Colorability of [25] 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 [22]: 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 decommmit) or refuse to continue (i.e., send an invalid decommitment message). This small 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 [37]. 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 [37] with our resettable witness indistinguishable proof system. Whereas any WI proof (cf. [16]) is also concurrent-WI (cf. [14]), 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). We present two proofs for the claim that the resulting proof systems is indeed resettable zero-knowledge. The first proof adapts the simulation argument of [37], extending it from their concurrent model to our stronger resettable model. The second (i.e., alternative) proof 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 it is also resettable zero-knowledge.

Again, we warn that the explanation provided above ignores several important issues. For further details see Section 6.

2.4 NP has resettable-ZK arguments

Computationally-sound proofs (a.k.a arguments) [8] are a weaker notion that interactive proofs [29]: it is infeasible rather than impossible to fool the verifier to accept wrong statements with non-negligible probability. Still, we present an alternative construction achieving resettable zero-knowledge arguments for any language in NP (indeed a weaker result than the one reviewed in previous subsection). The reason this alternative construction is interesting is that we don't use a reduction to some NP-complete language (as in the proof system above). Loosely speaking, given a suitable resettable witness indistinguishable argument for a language L (in NP), we show how to transform it to a resettable zero-knowledge argument for L (without using a reduction of L to some NP-complete language, as done in the above proof system).[8]

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}, 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 Section 7.

2.5 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 registered. One crucial aspect of our work consists of using the public-key model for tasks totally unrelated to privacy and authenticity.[9]

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

[8] In the proof system above, which follows the strategy of [37], a reduction to some NP-complete language is employed in order to obtain an instance on which the resettable witness indistinguishable proof system is executed. Specifically, a statement of the form "$x \in L$" is reduced to a statement of the form "either $x \in L$ or χ", where χ is a statement which depends on a preliminary part of the execution. For further details see Section 6.

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

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.*

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

We mention that the above constant-round resettable zero-knowledge arguments also employ the idea of "telescopic" usage of intractability assumptions discussed in the previous subsection. For further details see Sections 8 through 10: Specifically, Sections 9 and 10 provide two alternative presentations of essentially the same protocol. (The simulator provided in Section 9 can be easily adapted to simulate the protocol as presented in Section 10.)

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 public-key model postulates that the public-keys depositted 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 to this result is that it assumes 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 make the assumption that a prover will not interact with a verifier unless the verifier's public-key was registered a sufficiently long time before, 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 (which all newly registered public-keys must undergo before being used) and a small delay (which 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 $r : \mathsf{N} \to \mathsf{N}$, an $r(\cdot)$-round resettable zero-knowledge argument for $\mathcal{NP}$ in the public-key model. Again, such protocols are concurrent zero-knowledge (as a special case). (For further details see Section 9.3.)

3 Preliminaries

3.1 Standard Conventions

Throughout this paper we consider interactive proof systems [29] 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*).

Likewise, when we talk of computationally-sound proof systems (a.k.a arguments) [8] 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 $1/p(|x|)$.

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.

We adopt the basic paradigm of the definition of zero-knowledge [29]: 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. (Recall that it is not known whether constant-round zero-knowledge proofs for $\mathcal{NP}$ exists, if one insists on strictly polynomial-time simulators (rather than expected polynomial-time ones); See [22, 19]. Recall that Part II focuses on constant-round resettable zero-knowledge systems.)

We also refer (or, actually, extend) the definition of witness indistinguishable proof systems (cf., [16]). 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.

3.2 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 2.5.

Part I
The Vanilla Model

4 Definition

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, 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 independent 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 equivalent variants of the model. 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 any 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 equivalence of the two variants is shown below.)

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 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. (In fact, an adversary of the resettable model may easily obtain the NP-witness used in the concurrent zero-knowledge protocols of [37].) 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.[10] That is, we allow to interact polynomially-many times with each of polynomially-many

[10] We comment that assuming that one-way function exists, we may transform any polynomial-time prover that is resettable zero-knowledge with respect to a single common input and a single random-pad into one that is resettable zero-knowledge with respect to polynomially-many common inputs and random-pads (as defined below). The key idea, used more extensively in Section 5, is to apply a pseudorandom function to the identifier of the prover's copy in order to derive "computationally independent" random-pads.

P_{x_i,y_i,ω_j}'s where the ω_j's are random and independent of one another.

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-pad 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.[11] That is, A' checks that $\alpha_i = A(\beta_1, ..., \beta_{i-1})$, but it does not (and in general cannot) check that $\beta_i = B(\alpha_1, ..., \alpha_{i-1})$ as 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 2 (resettable security – 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, 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 $i, j \in \{1, ..., t\}$.*
2. *Machine V^* is allowed to initiate polynomially-many interactions with the $P^{(i,j)}$'s.*

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

- *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 response of $P^{(i,j)}$ to such message.*
- *In the* sequential (or rewindable) version *V^* is required to complete its current interaction with the current copy of $P^{(i,j)}$ before starting an interaction with any $P^{(i',j')}$, regardless if $(i,j) = (i',j')$ or not. Thus, the activity of V^* proceeds in rounds. In each round it selects one of the $P^{(i,j)}$'s and conducts a complete interaction with it.*

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

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

In case there exists a universal probabilistic polynomial-time machine, M, so that M^* can be implemented by letting M have oracle-access to V^*, we say that P is resettable zero-knowledge via a black-box simulation.[12]

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

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 $P^{(i,j)}$).[13] Thus, *every resettable zero-knowledge protocol is concurrent zero-knowledge.*

Recall that, as stated above (and shown below), all known zero-knowledge protocols are NOT resettable zero-knowledge. Furthermore, they are even NOT resettable witness indistinguishable. For example, ability to "reset" the original zero-knowledge proof for 3-Colorability [25], 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 $\mathcal{NP}$ exists, under standard intractability assumptions.

Equivalence of the two variants

As stated above, the restricted non-interleaved model is actually as powerful as the general (interleaved) model. That is, any prover strategy which is resettable zero-knowledge in the non-interleaved model is also resettable zero-knowledge in general (in the interleaved model). In fact, a stronger claim holds:

Theorem 3 *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 $\langle P(\overline{y}), V^*\rangle(\overline{x})$.*

[12] Recall that the existence of black-box simulators implies auxiliary-input zero-knowledge (cf. [26, 23]).

[13] Indeed, the possibility to run various $P^{(i,j)}$'s (i.e., same j and varying j'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.

So, in particular, a simulator guaranteed for W^* will do also for V^*.

Proof Sketch: Using V^* as a black-box and interacting with instances of P in a non-interleaved manner, W^* emulates for V^* interleaved interactions with the same 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.

For simplicity, 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 $\beta_{t+1} \equiv (i, \beta_{j+1}^{(i)})$ (i.e., the $j+1$st message of session with ID i). 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 $\beta_k^{(i)}$ and obtains the response of $P_{x,y,\omega}$. It forward to V^* (only) the last response of $P_{x,y,\omega}$ (i.e., the response of $P_{x,y,\omega}$ to $\beta_k^{(i)}$), and aborts the current (non-interleaved) session.

The argument extends easily to the general case (in which V^* 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). ■

5 Resettable Witness-Indistinguishable proofs for NP

As a first indication towards the feasibility of the resettable model and as a tool towards the construction of resettable zero-knowledge proof systems, we show that under standard intractability assumptions, any NP-statement has a resettable witness indistinguishable proof system. We stress that whereas any witness indistinguishable proof (cf. [16]) is also concurrent-WI (cf. [14]), all previously known witness indistinguishable proof are not resettable witness indistinguishable. We actually prove the following:

Theorem 4 *If* two-round *perfectly-hiding commitment schemes exists then every language in $\mathcal{NP}$ has a* constant-round *resettable witness indistinguishable interactive proof system.*

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 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.

Proof Sketch of Theorem 4

Traditional zero-knowledge interactive proofs rely on the randomized nature of the prover strategy. In a sense, this is essential (cf., [26]). In our context, the prover's randomization occurs only once and is fixed for all subsequent interactions. So the main 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 [20].[14] However, just "using a pseudorandom function" does not suffice. The function has to be

[14] Recall, that by combining [32] and [20] one may construct pseudorandom functions using any one-way function. Furthermore, relying on the intractability of the DLP, a much more efficient construction is available by combining [6]

applied to "crucial steps" of the verifier; that is, exactly the steps which the verifier may want to alter later (by rewinding) in order to extract knowledge. Thus, the zero-knowledge proof system for 3-Colorability of [25] 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 [22]: 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. Thus, on an abstract level, the proof system 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 $\phi : [n] \mapsto \{1, 2, 3\}$ of G.

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

The rest is an adaptation of the [22] proof system, where the only modification is at Step (P1).

(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, 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) As in [25, 22], the prover commits to t random relabeling of colors. The commitment is done using an ordinary commitment scheme, providing computational-secrecy and perfect-binding. The key point is that the prover's random choices (both for the relabeling and randomization needed for the commitment scheme) are replaced by the value of the function f applied to the message sent by the verifier in Step (V1).

Actually, we may apply f to the pair $((G, \phi), \texttt{msg})$, where $\texttt{msg}$ denotes the message sent by the verifier in Step (V1). That is, let $(\pi_1, r_1), ..., (\pi_t, r_t) = f(G, \phi, \texttt{msg})$, and use $\pi_i : \{1, 2, 3\} \stackrel{1-1}{\mapsto} \{1, 2, 3\}$ as the i^{th} randomization of ϕ (i.e., $\phi_i(v) = \pi_i(\phi(v))$), and $r_i = (r_{i,1}, ..., r_{i,n})$ as randomness to be used when committing to the values of ϕ_i on $[n]$. That is, for $i = 1, ..., t$ and $j = 1, ..., n$, the prover commits to $\phi_i(j)$ using randomness $r_{i,j}$.

(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". That is, suppose that the i^{th} edge revealed in Step (V2) is (u, v), then the prover reveals $\phi_i(u)$ and $\phi_i(v)$.

(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.

---

and [20].

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).[15] Thus, we need to integrate such (two-round) commitment schemes in the above protocol. Recall that the existence of such a scheme implies the existence of one-way functions [33], which suffices for constructing pseudorandom generators [32], pseudorandom functions [20], and (two-round) perfectly-binding commitment schemes [36]. (Our description can be easily modified to utilize the latter, rather than a one-round perfectly-binding scheme which may be constructed assuming that one-way permutations exist.)

Clearly, the above protocol constitutes an interactive proof system for 3-colorability (since as far as cheating provers are concerned there is no difference between the above protocol and the one in [22]). Thus the task is to show that the protocol is resettable witness indistinguishable.

Comment: It appears as if the above protocol is resettable zero-knowledge; however, we were not able to prove this. The subtle problem is that the verifier may fail to de-commit in Step (V2). Specifically, it may fail to decommit in one session and decommit properly in a later session in which it has sent the same message in Step (V1). Doing so will harm the straightforward simulation attempt, which proceeds session-by-session so that in each session one first tries to obtain the verifier's commitment values via a dummy (P1)-message so that one can later simulate Step (P1) (and the subsequent steps of the same session) properly. The problem is that we cannot answer the same Step (V1) message, sent in two sessions, in two different ways (in the two sessions), since the prover would answer identically in the real execution.[16]

Showing that the protocol in resettable witness indistinguishable: Fortunately, the subtle problem mentioned above has much milder effect on the proof that the above protocol is witness-indistinguishable. As a mental experiment, we consider an ideal prover that uses a truly random function rather than a pseudorandom one.[17] The key observation is that whenever a different Step (V1) message is sent, the corresponding Step (P1) is an independently selected random commitment to an independently selected random relabeling of the specific coloring ϕ. Our goal is to show that the dependence of the interaction on the specific witness coloring ϕ is computationally unnoticeable. That is, we show that multiple sessions (with an adversary V^*) in which one possible witness coloring ϕ is used are computationally indistinguishable from such sessions in which

[15] 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. [18]).)

[16] Specifically, suppose that the simulator always tries first to send a dummy message in Step (P1), and consider two consequetive sessions with a cheating verifier. In the first session, the verifiers commits to some edge sequence in Step (V1) but refuses to decommit in Step (V2). The simulator will thus produce a truncated session (which, by itself, is fine). Now suppose the verifier repeats the same Step (V1) message in the second session, but does decommit properly in Step (V2). The simulator would like now to send a corresponding commitment to a pseudo-coloring, but the problem is that this message is different from the dummy commitment sent in Step (P1) of the first session. Note that the real prover will always send the same (P1)-message in response to the same (V1)-message, and so if the simulator behaves differently this is easily detectable.

[17] As usual, once the claim is established for such a prover, we replace back the random function by a pseudorandom one (and so derive the stated result).

another witness coloring ϕ' is used. For simplicity (and, w.l.o.g., in view of Theorem 3), we may assume that the adversary verifier is a non-interleaving one (i.e., it completes or aborts a session before initiating a new one). The proof uses a hybrid argument, where the i^{th} hybrid is defined as follows:

- For every j, if in the j^{th} session the message sent in Step (V1) is identical to one sent in session j' (for some $j' < j$) then the prover repeats the corresponding Step (P1) message.
- Otherwise (i.e., the message sent in Step (V1) of session j did not appear in any prior session), if $j \leq i$ then the prover uses ϕ when executing Step (P1) else (i.e., $j > i$) it uses ϕ'.
- Loosely speaking, in all cases, the execution of Step (P1) determines also the prover's actions in Step (P2).

  This would have been accurate if the verifier's commitment scheme had been perfectly-binding, but it is only computationally-binding. Still we ignore here and below the possibility that the verifier may decommit to the same Step (V1) message (appearing in two different sessions) in two different way. We claim that such an event occurs with negligible probability, or else a contradiction to the computationally-binding feature of the verifier's commitment scheme follows.

Thus, ability to distinguish the ensembles in question (which correspond to the extreme hybrids) implies ability to distinguish neighboring hybrids (i.e., hybrids i and $i+1$). Note that hybrids i and $i+1$ may differ only when session $i+1$ has a Step (V1) message that did not appear in any of the first i sessions. The key observation is that in such a case the commitment generated in response is independent of the prior iterations. The actual argument utilizes several standard tricks: First we use an averaging argument to fix the transcript of the first i sessions. Since our distinguisher is non-uniform we may omit these first i sessions from our discussion and consider an distinguisher which receives as input only transcripts starting at session $i+1$ (which are taken either from hybrid i or $i+1$). We next transform this distinguisher into one that only receives the transcript of session $i+1$ (by emulating the following sessions in a straightforward manner using ϕ and ϕ' which as usual can also be incorporated in the distinguisher). Note that if a latter session repeats the same Step (V1) message then we just copy the prover's response from Step (P1) of the prior session. Thus we obtain a polynomial-size circuit that distinguishes between a single execution of the protocol in which ϕ is used and single execution in which ϕ' is used, which contradicts the fact that such a single execution corresponds to the standard zero-knowledge/witness-indistinguishability model (in which the protocol is equivalent to the one of [22] which is zero-knowledge and thus witness-indistinguishable).

An alternative approach for constructing resettable witness indistinguishable proofs

An alternative (but somewhat related) approach for constructing resettable witness indistinguishable proofs is to start with a non-interactive zero-knowledge proof system (cf., [5, 13]). The idea is to employ "coin tossing into the well" (cf., [3]), but with a small twist: First, the verifier commits to a sequence of random bits using a perfect (two-round) commitment scheme. Next, the prover sends a corresponding sequence of bits which are determined by applying a pseudorandom function to the verifier's message. Then, the verifier de-commits and a reference-string for the non-interactive zero-knowledge proof is defined (as usual in "coin tossing into the well"), and finally the prover sends such a (non-interactive) proof (relative to that reference-string). Further details are omitted from the current version.

6 NP has Resettable ZK proof systems

In this section 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 [37]. We modify this proof system using the techniques presented in Section 5, and replace the concurrent witness indistinguishable (concurrent-WI) proof system employed by [37] with our resettable witness indistinguishable proof system. We stress that whereas any witness indistinguishable proof (cf. [16]) is also concurrent-WI (cf. [14]), all previously known witness indistinguishable proof are not resettable witness indistinguishable. Thus, the resettable witness indistinguishable proof system of Section 5 plays a major role in obtaining the following result.

Theorem 5 *If two-round perfectly-hiding commitment scheme exist then any language in NP has a resettable zero-knowledge proof system. Furthermore, this is obtained via black-box simulation.*

Recall that the hypothesis holds if claw-free permutations exist, which in turn holds if DLP is intractable.

We start by reviewing the Ransom and Kilian protocol [37]. 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 the verifier commits to k random bit sequences, $r_1, ..., r_k \in \{0,1\}^k$, and next k iterations proceed so that in each iteration the prover commits to a random bit sequence, s_i, and the verifier decommits to the corresponding r_i. The result of the i^{th} coin-toss is defined as $r_i \oplus s_i$ and is known only to the prover. In *the second stage*, the prover provides a witness indistinguishable (WI) proof (cf. [16]) 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., $r_i = s_i$ 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 $r_i = s_i$ 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 r_i as witness. (By the WI property, such a run will be indistinguishable from a run in which an NP-witness for the common input being in the language is used.)

To transform the above protocol into one that is resettable zero-knowledge, we replace the prover's random choices in the first phase by choices determined by the application of a pseudo-random function to the verifier's initial commitment. In addition, we replace the WI proof system used by [37] by our resettable witness indistinguishable proof system.

6.1 The protocol (sketch)

The implementation of the protocol uses two complementary types of commitment schemes: The prover's commitments are via a perfectly-binding commitment scheme (which is only computationally-hiding), whereas the verifier's commitments are via a perfectly-hiding commitment scheme (which is only computationally-binding). For simplicity of presentation, we will use a one-round scheme based on any one-way permutations[18] for the first type, and a two-round scheme based on claw-free pairs[19] for the second type.

Common Input: x supposedly in the language $L \in \mathcal{NP}$, and a security parameter k.[20]

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

Prover's Randomness is used to define a pseudorandom function $f : \{0,1\}^{\leq \mathrm{poly}(k)} \to \{0,1\}^{\mathrm{poly}(k)}$.

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

1. The verifier commits to k uniformly selected k-bit strings. This is done as follows. First the prover uses f to determine its first message in the two-round perfectly-hiding commitment scheme.[21] (Towards this end, the prover applies f to an arbitrary fixed string different from all strings to which f is applied in the sequel.) In response, the verifier uniformly selects $r_1, ..., r_k \in \{0,1\}^k$, and sends the prover its commitment to each of the r_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 $r_1, ..., r_k$.
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 s_i, and next the verifier decommits to β_i by providing r_i along with the randomness used in forming β_i from r_i. The prover's choice (i.e., s_i) as well as the randomization used in its commitment are determined by applying f to the transcript so far.[22] We stress that s_i is uniquely determined by the string, denoted α_i, sent by the prover.

Stage 2: The prover provides a resettable witness indistinguishable proof that *either* $x \in L$ *or* $r_i = s_i$*, for some* i. The NP-witness used by the prover is w, and the witness indistinguishable proof is the one presented in Section 5. Specifically, we reduce the NP-statement *either* $x \in L$ *or there exists an* i *and an* s *so that* α_i *is a valid commitment to* s *and* $r_i = s$ to Graph 3-Colorability. (The graph is formed depending on x, the sequence of α_i's and the sequence of r_i's; whereas w is efficiently transformed into a 3-coloring of this graph.)

We stress that whenever a party fails to provide a message as instructed the other party halts (detecting an obvious cheating attempt).

It is quite obvious that the above protocol constitutes a proof system for L. In particular, the soundness property follows from the perfect-hiding property of the verifier's commitment in Stage 1, the perfect-binding property of the prover's commitment, and the soundness of the proof system used in Stage 2. The difficult part is to show that the above protocol is indeed resettable

[18] Specifically, given a one-way permutation f with a hard-core b (e.g., see [24]), one commits to bit σ by selecting uniformly a string x, and sending the value $f(x), b(x) \oplus \sigma$. Decommitment is done by providing (σ and) x.

[19] Specifically, given a family of claw-free pairs, $\{(f_a^0, f_a^1) : a \in I \subseteq \{0,1\}^*\}$ (e.g., see [19]), the sender commits to bit σ as follows. The receiver first selects at random an index $a \in I$ and sends it to the sender, which uniformly selects x in the domain of f_a^σ, and sends the value $f_a^\sigma(x)$. Decommitment is done by providing (σ and) x.

[20] 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, the security parameters used in the pseudorandom function and in the commitment schemes, etc.

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

[22] Alternatively, it suffices to apply f to the pair $(\overline{\beta}, i)$. The alternative adopted in the main text merely simplifies the simulation a little.

zero-knowledge. We present two alternative proofs for this claim. The first alternative is outline below and executed in subsection 6.2, whereas the second is presented in subsection 6.3. The first alternative consists of extending the simulation strategy of [37] from the concurrent setting to the more demanding resettable setting. The basic strategy is based on constructing transcripts of Stage 1 in which $r_i = s_i$ for some i, and running the witness indistinguishable proof system (as if we were the prover) with witness s_i. We wish to stress two points which are crucial to the fact that our protocol is indeed resettable zero-knowledge:

1. The way in which [37] construct transcripts (of Stage 1) in which $r_i = s_i$ for some i, can be extended from the concurrent model to the (stronger) resettable model. The extension involves changing the location to which the simulator is rewinded so to meet the constraints of the resettable model.

2. The protocol used in Stage 2 is witness indistinguishable *in the resettable model*. We refer the reader to Section 5 for discussion of this feature.

In both cases, the key to robustness in the resettable model is that the verifier's actions are almost determined by its first message in the current stage, and so applying a pseudorandom function to this message provides the prover with sufficiently good randomization. By saying that the verifier's actions are almost determined by its first message in this stage we mean that this message effectively commits the verifier to all other actions in the stage. Essentially, its only choice is whether to continue in its "predetermined" actions or abort the protocol.[23] Whereas this restricted choice (of whether to proceed properly or abort) suffices to prevent us from proving that the (resettable witness indistinguishable) protocol used in Stage 2 is itself resettable zero-knowledge, we'll show that the combined protocol (of both stages) is in fact resettable zero-knowledge.

6.2 The simulation game (sketch)

We first define a *simulation game* that captures the simulation strategy. Our aim in this game is to capture the way the simulator "abuses" Stage 1 in order to produce transcripts of both stages (and specifically of Stage 2). As usual, the simulator has black-box access to an adversary strategy for a cheating verifier. Here this strategy operates in the resettable model which means that when supplied with a history transcript h the verifier may send a message corresponding to some *prefix* of h (i.e., such a message may be the next message in any of the sessions described in the prefix or an invocation of a new session).[24] Now, suppose that we replace Stage 2 by the verifier asking the prover if it has an NP-witness to the statement made in this stage, and by the prover answering honestly (i.e., always "yes" in the actual execution). (We stress that this replacement for Stage 2 takes place only if the verifier has properly decommitted in all k iterations of Stage 1.) Furthermore, let us postulate that the simulator is required to answer honestly too. It follows that in order to produce good simulations the simulator must force $r_i = s_i$ for some i before completing (Stage 1 of) a session in which the verifier never fails to properly decommit. Alternatively, we postulate that

[23] This text is slightly imprecise since computational-binding does not disallow a different decommitment to the same value (i.e., providing (v, σ_1) and (v, σ_2) so that applying the commitment scheme to value v while using coins σ_i yields the same committal, regardless of σ_i. But we may assume, w.l.o.g., that this does not occur (since it only increases the prover's freedom): Suppose that the verifier decommits to some β_i in two different ways both corresponding to the same r_i. Then, this does not effect our attempt to force $s_i = r_i$; it only allows makes the prover's actions in next iterations more independent (as these may depend on the entire transcript).

[24] Note that in the standard zero-knowledge model, the verifier must send the next message in the single session described in h, whereas in the concurrent model the verifier must send the next message in any session described in h or invoke a new session.

the simulator may make query h to the verifier strategy only if for any (Stage 1) session that is properly completed in h it is the case that $r_i = s_i$ for some i. Recall, that without loss of generality, we may assume that before outputting the full transcript h, the simulator queries the verifier on h (and the latter responses with `halt`).

Recall that the simulator makes queries which are partial transcripts of a possible execution, where each execution is a sequence of sessions. To simplify the exposition, the reader may assume that these sessions are not interleaved but rather that a new session (possibly with a copy of the same P_{x_i,y_i,ω_j}) is initiated only after the previous session was terminated (i.e., completed or aborted). We stress that the same exposition applies also to an adversary that interleaves sessions (only the formalism is a bit more complex as we need to make conventions regarding the correspondence of messages in an execution to various sessions). Furthermore, recall that restricting the adversary to the non-interleaving case does not limit its power (since such an adversary may emulate an interleaving one; see Theorem 3).

Definition 6 (the simulation game): *We call a session* (of Stage 1) *in an execution prefix* properly completed *if (during this execution prefix) the verifier has decommitted properly in all its k iterations. A black-box simulator* (for Stage 1) *is called* honest *if for every query h made by the simulator and for every properly completed session in h, it is the case that for some i the i^{th} prover's commitment in this session is to a value equal to the i^{th} value revealed* (and properly decommitted) *by the verifier in this session.*

Recall that the queries are made to adversary strategies which may respond by a next message in an existing session or by an initiation of a new session. We stress that new sessions may be with a copy P_{x_i,y_i,ω_j} which was already used in a previous session (described in the prefix). Thus, it is instructive to prepend each such initiation with the corresponding pair (i,j). Finally, since the adversary's strategy to which we have black-box access captures the operation of a poly(k)-time adversary, we know that for some polynomial p it is the case that for any query made by the simulator either the verifier strategy aborts or the total number of sessions appearing in the query is at most $p(k)$.

Theorem 7 *Suppose that the commitment schemes used in Stage 1 are as postulated. Specifically, suppose that the computational security conditions hold with respect to* poly(k)*-size circuits. Then, there exists an* honest *(black-box) simulator operating in* poly(k)*-time that, for every* poly(k)*-time verifier V^* in the ressetable model, has output that is computationally indistinguishable (again w.r.t parameter k) from the transcript of executions of Stage 1 by V^*.*

The proof is constructive. It will be clear from the construction that the simulator runs in strict (not merely expected!) poly(k)-time. As in the proof of Theorem 4, we will consider an imaginary prover that utilizes a truly random function rather than a pseudorandom one. We will make extensive use of the hypothesis that the verifier's commitment is computational-binding, and ignore the rare cases (which may occur only with negligible probability) in which during the simulation game the verifier properly decommits to the same commitment so to support two different values. Thus, at any point where the verifier sends a decommit message, it is either a proper decommitment to a unique value (associated with the corresponding commitment message) or a special message indicating refusal to decommit. Let $K \stackrel{\text{def}}{=} \text{poly}(k)$ (i.e., $p(k)$ above) be a bound on the number of sessions initiated by the verifier strategy (say in the actual execution). We assume K is known (but this assumption may be easily removed by using the standard doubling trick; see [37]).

Motivation towards the simulator: Standard simulator (and the current one is no exception) proceed by rewinding the verifier so to force situations that occur rarely in actual executions (i.e., in our case $r_i = s_i$ for some i). (In a sense, impossibility results as [23] indicate that such "rewinding" in unavoidable.) A key observation, originating in [12], is that whenever many sessions are to be simulated rewinding may result in loss of work done and cause the simulator to do the same amount of work again. Specifically, all simulation-work done for sessions starting after the point to which we rewind may be loss. The key observation is that only work invested for such "fresh" sessions is lost. Loosely speaking, considering a specific session, since there are in total at most K sessions, there must be an iteration (i.e., an $i \in \{1, ..., k\}$) so that at most K/k sessions start in the interval corresponding to possible rewindings of the i^{th} iteration (of this specific session). So if we try to rewind on the correct i, we will invest (and so waste) only work proportional to K/k sessions. So the idea is to abort the rewinding attempt on the i^{th} iteration if more than K/k session are initiated in the corresponding interval. The same reasoning applies also recursively (i.e., to the rewinding in these K/k sessions). So at the ℓ^{th} level of the recursion we may need to deal with $K_\ell = K/k^{\ell-1}$ sessions, and each may cause us work proportional to the simulation of $K_\ell/k = K/k^\ell$ sessions. Denoting by $W(m)$ the amount of work invested for m sessions, we obtain the recursion $W(m) = O(m \cdot W(m/k))$, which solves to $W(K) = K^{\Theta(\log_k K)}$. Since $K = \text{poly}(k)$ we get $W(K) = K^{O(1)}$.[25] Clearly, the above is but a rough sketch of the argument and numerous details should be dealt with (as done below).

The main procedure: This procedure uses a subroutine, denoted `NextProverMsg`, that given a partial transcript of an execution ending in a verifier message, returns the next prover message (in the session indicated by the verifier). Initiating h to equal the sequence of common inputs $\overline{x}$ and $S \leftarrow \emptyset$, the main procedure repeats the following steps until V^* terminates, at which point it outputs h.

1. Extend the transcript by one verifier message: $h \leftarrow (h, V^*(h))$;
2. Extend the transcript by one prover message: $(S, \texttt{msg}) \leftarrow \texttt{NextProverMsg}(K, S, h)$ if $\texttt{msg} = \texttt{fail}$ then we abort without output else we let $h \leftarrow (h, \texttt{msg})$.

The rest of the description is recursive: `NextProverMsg` calls a subroutine `Solve` which in turn calls `NextProverMsg` itself. However the first parameter in these calls decreases and this provides a bound on the running time. The second parameter passed to `NextProverMsg` is a (monotonically growing) set of "solved" sessions (i.e., sessions for which we've forced $r_i = s_i$ for some i) along with adequate witnesses (i.e., the r_i's). We assume all procedures know K (as above). Recall that by our convention (see Section 4), each message (of both prover and verifier) includes as prefix the entire current transcript of the current session.

Procedure `NextProverMsg`: On input (m, S', h') the procedure extracts the last message in h', which is a verifier message denoted `msg`, and acts as follows.

1. *Initiate session message*: If `msg` is a verifier's message asking to initiate a new session then we act as follows:

[25] It is indeed remarkable that the running-time of the simulator is polynomial, whenever the running-time of the adversary is polynomial, but the former polynomial is not a power of the latter one. Specifically, if the running-time of the adversary is $T_A(k) = k^a$ then the simulator runs in (polynomial) time $t_S(k) = k^{O(a^2)}$ (rather than $t_S(k) = k^{O(a)}$, or even $t_S(k) = k^{a+O(1)}$, as is typically the case). This (bad) feature is inherited from [37].

- If this is the first session to be initiated (in h') then we emulate the prover's answer by selecting a random first receiver-message for the commitment scheme.
- Otherwise (i.e., other sessions already exist), we merely answer with the same prover's message provided in the previous initializations.

2. *Repeated message*: If `msg` appears several times in h' (this cannot occur in the concurrent zero-knowledge model) then we act here to match the action in the previous occurrence:

   - Let i denote the currently handled session (to which `msg` belongs), I denote the previous sessions in which `msg` appears, and a be the answer given to `msg` in these previous sessions.
   - If `msg` is the last verifier message in the session then return (S', a).
   - Otherwise, suppose that a is the prover's j^{th} commitment in session i (i.e., `msg` is either the verifier's commitment in session i or its $j-1^{\text{st}}$ decommitment, where $j \leq k$). If there exists $(i', j, r) \in S'$ so that $i' \in I$ then return $(S' \cup \{(i, j, r)\}, a)$ else return (S', a).

3. *Invalid message*: If `msg` is an invalid message (i.e., an illegal decommit) then we return $(S', (\mathtt{msg}, \mathtt{halt}))$ (i.e., the prover halts the current session).

4. *Message for a solved session*: If `msg` relates to a session in S' then we proceed in the straightforward manner. That is,

   - if `msg` the last verifier (valid) decommitment (viewed as the question whether the simulator knows an NP-witness) then we return $(S', (\mathtt{msg}, \mathtt{yes}))$ (which is correct);
   - if, for $i < k$, `msg` is the i^{th} verifier decommitment (and it is proper and only appears once in h') then we return $(S', (\mathtt{msg}, \alpha))$, where α is a (random) commitment to a random k-bit string. Same if `msg` is the initial commitment of the verifier (and it only appears once in h').

5. *Last message for an unsolved session*: If `msg` is a valid decommit to the last iteration in a session not in S' then we've failed and return $(S', \mathtt{fail})$. (This is because at this point we are asked if we have an NP-witness and being honest we cannot answer "yes".)

6. *Non-last message for an unsolved session*: If `msg` is a valid decommit (which appears once in h') to a non-last iteration in a session, denoted i, not in S' then we try to "solve" the session as follows. Basically, we'll try to obtain the *next* verifier decommitment for the current session (that's why we insist that the current decommitment is not the last one). Denote by $j \leq k$ the index of the *next* decommitment. If we obtain the latter decommitment then we set the prover's next message to be a commitment to the corresponding value (and so force $s_j = r_j$ which satisfies session i).

   Specifically, we repeat the following steps for $O(K^2)$ times:

   - Let $S' \leftarrow \mathtt{Solve}(\lceil 2m/k \rceil, S', h', i, j)$.
     (The values of i and j are implicit in h', but we provide them explicitly for clarity of exposition.)
   - If for some r the revised S' contains (i, j, r) then return $(S', (\mathtt{msg}, \alpha))$, where α is a (random) commitment to the value r.

In case none of these attempts succeeded, return $(S', (\mathtt{msg}, \alpha))$, where α is a (random) commitment to a random k-bit string. (In all these case the string α returned by the procedure is the prover's commitment in the next iteration of session i.)

Same if `msg` is the initial commitment of the verifier (in a session i not appearing in S' and `msg` only appears once in h').

We stress that Item 6 guarantees that the standard "rewinding" technique can be applied here. This is because we rewind to the first place which determines the randomization in the next prover's step. In the description of `Solve` we ignore the case of `msg` equal the initiation message of a session, which is handled as in Item 1 above.

Procedure `Solve`: By the construction of `NextProverMsg`, when `Solve` is invoked with input (m, S', h', i, j), session i is not solved yet (i.e., it does not appear in S'), and it is the procedure's aim to redeem this state of affairs (specifically by obtaining the verifier's j^{th} decommitment for session i). We stress that, unlike in [37], this decommitment is not necessarily obtained from session i: It may be obtained from a session i' in which the initial verifier's commitment message is identical to the first message in session i. We call such a session a conjugate of session i.

A key point regarding `Solve` is that throughout its attempt to solve the current session, the procedure never wastes too much time. This is guaranteed by maintaining a growing set, denoted I, of the sessions it has initiated during the current execution of `Solve`, and aborting when $|I|$ exceeds m.

On input (m, S', h', i, j), the procedure denotes by E the set of sessions existing in h', and initializes $I \leftarrow \emptyset$. It then executes iteratively the following steps:

1. Get the verifier's respond to the current history; i.e., $\mathtt{msg} \leftarrow V^*(h')$ and $h' \leftarrow (h', \mathtt{msg})$.

2. Act as follows according to `msg`:

   (a) If `msg` is a valid decommit to the j^{th} iteration in a session that is conjugate to session i then we return $S' \cup \{(i, j, r)\}$, where r is the corresponding value revealed in the decommitment.

   (This completes a successful execution of `Solve`.)

   (b) If `msg` is a valid decommit to iteration j' in a session i' that is conjugate to sessions $i_1, ..., i_t \in E \setminus S'$ then we return $S' \cup \{(i_1, j', r), ..., (i_t, j', r)\}$, where r is the corresponding value.[26]

   (Although we did not solve session i, we did make progress since $E \setminus S'$ became smaller.)[27]

   (c) If `msg` is an invalid decommit to any iteration in any session i' then set $h' \leftarrow (h', (\mathtt{msg}, \mathtt{halt}))$ (i.e., the prover halts session i').[28]

   (Here, as in cases 2(d)ii and 2e below, we just continue to extend the current execution.)

[26] Note that i' may but need not be one of the i_j's. We comment that in the concurrent setting, treated in [37], this case may occur only if $i' \in E \setminus S'$ (since there each session is only conjugated to itself).

[27] In case $j' < k$ we may indeed continue our solving attempts (rather than returning), but in case $j' = k$ we must return as in Step 5 of `NextProverMsg`. Since the analysis does not seem to benefit from this distinction, we prefer, for sake of simplicity, we the current presentation.

[28] Alternatively, we may invoke $\mathtt{NextProverMsg}(m, S', h')$, which will do exactly the same. We remark that in [37] if `msg` is an invalid decommit to the j^{th} iteration in session i then we may abort (since we have no chance to succeed in the current invocation of `Solve`). This is however NOT the case in the resettable model, and it is crucial to proceed as in case 2c. It will become clear from the analysis that no damage is caused by continuing the current solving trial (and this holds, of course, also for the concurrent model treated in [37].

(d) If `msg` is the first message (i.e., a verifier's commitment) in a new session, $i' \notin E \cup I$, then

  i. If $|I| = m$ then we return S'.
  (This failure to solve session i is due to too many new session encountered in the current execution of `Solve`.)

  ii. Otherwise (i.e., $|I| < m$), we set $I \leftarrow I \cup \{i'\}$, invoke $(S', \texttt{msg}) \leftarrow \texttt{NextProverMsg}(m, S', h')$, and set $h' \leftarrow (h', \texttt{msg})$.

(e) If `msg` is any message referring to a session in $I \cup S'$ then set $(S', \texttt{msg}) \leftarrow \texttt{NextProverMsg}(m, S', h')$ and $h' \leftarrow (h', \texttt{msg})$. In the unlikely case $\texttt{msg} = \texttt{fail}$ (i.e., failure of `NextProverMsg`), we return S'.

(f) If $\texttt{msg} = \texttt{halt}$ (i.e., verifier terminates) then we return S'.
(This failure to solve session i is due to not encountering a proper decommit in the j^{th} iteration prior to the termination of all sessions.)

(The reader may verify that these cases cover all possibilities; e.g., valid decommits are handled in Items 2a, 2b and 2e. Case 2e is handled as the other cases, but this is done by the recursive call.)

The reader may easily verify the following:

Fact 6.1 *Let $T_N(m)$ (resp., $T_S(m)$) denote the running time of* `NextProverMsg` *(resp.,* `Solve`*) when invoked with first parameter m. Then,*

1. *the running time of the main procedure is $\text{poly}(k) \cdot T_N(\text{poly}(k))$;*

2. *$T_N(m) < \text{poly}(k) \cdot T_S(3m/k)$; and*

3. *$T_S(m) = \text{poly}(k) \cdot T_N(m)$.*

Thus, $T_N(m) = \text{poly}(k)^{\log_{k/3} m}$, and the running time of the main procedure is $\text{poly}(k)$.

Ignoring the rare cases in which the verifier decommits to different values for the same commitment, it follows that the simulator works as claimed in Theorem 7, provided that the probability that `NextProverMsg` returns `fail` is negligible. That is,

Lemma 6.2 *Conditioned on the main procedure not aborting, its output is computationally indistinguishable from the output of V^* in executions of Stage 1 in the resettable model.*

The main procedure may abort only if `NextProverMsg` returns a fail symbol in one of its ($\text{poly}(k)$-many) invocations. As in [37], establishing a bound on the latter failure probability is the most difficult part in the analysis of the simulator. The key observation allowing to adapt the argument in [37] to our context is that the "rewinding intervals" remain disjoint. That is, in any execution and for every session i, we consider the set of all sessions, denoted $C(i)$, having the same first (verifier's commitment) message as in session i; that is, $C(i) \subseteq \{i\}$ is the set of all sessions conjugate to session i in this execution. Consider any session i occuring in an execution. For $j = 1, ..., k$, the j^{th} interval is between the first time a prover's (commitment) message is sent in the j^{th} iteration of a session in $C(i)$ and the first proper decommitment in the j^{th} iteration of a session in $C(i)$. The important facts regarding these intervals are:

1. Procedure `Solve` is only invoked with a message marking the beginning of such an interval, and it (as well as it recursive calls) only explores extensions which are contained in this interval.

2. The intervals corresponding to such a session i are disjoint. This is because, for $j > 1$, the prover sends j^{th} commitment to a session in $C(i)$ only after the verifier has provided a proper decommitment in the $j - 1^{\text{st}}$ iteration of this session.

We stress that the above facts hold trivially in the concurrent zero-knowledge context (considered in [37]). Using the above facts, we obtain:

Lemma 6.3 *The probability that* `NextProverMsg`$(m, \cdot, \cdot)$ *returns* `fail` *is negligible.*

Proof Sketch: We extend the argument of [37]: In their (concurrent) context, `NextProverMsg`$(m, \cdot, \cdot)$ returns `fail` only when encountering the last message of a session i for which the current transcript contains k proper decommitments to session i. (Otherwise, the simulator rightfully completes session i with a halting message on behalf of the prover.) In our (resettable) context, `NextProverMsg`$(m, \cdot, \cdot)$ returns `fail` only when encountering the last message of a session i so that for every $j \in \{1, ..., k\}$ the current transcript contains a proper decommitment to the j^{th} iteration of a session that is conjugate to i. (Otherwise, for some j all sessions conjugate to i (including i itself!) have the verifier send an invalid message in one of the first j iterations, and the simulator would have rightfully completes session i with a halting message on behalf of the prover.)

Thus, suppose that `NextProverMsg`$(m, \cdot, \cdot)$ returns `fail` when invoked to provide the last message of a session i, and let i_j be the first session conjugate to session i so that the current transcript contains a proper decommitment to the j^{th} iteration of it (i.e., of session i_j). Still, if `NextProverMsg`$(m, \cdot, \cdot)$ returns `fail` (when invoked to provide the last message of a session i) it means that for every j we failed to solve i_j in iteration j using a new-session bound of $2m/k$ (otherwise, for $i = i_j$ this means that Case 4 of `NextProverMsg` would have applied, and for $i > i_j$ Case 2 would have applied). So we are talking of $O(K^2)$ failures to solve each of these specific sessions for each of the relevant k iteration (i.e., $O(K^2)$ failures to solve i_j in iteration j). Loosely speaking, if at least $k/3$ of these iterations have low (say lower than 0.1) probability to yield a proper commitment then the above event occurs with probability smaller than $2^k \cdot 0.1^{k/3}$, which is negligible.[29] Otherwise, we consider $2k/3$ iteration which have proper decommitment with probability at least 0.1. For at least one of these iterations, the expected numbers of new sessions initiated during the relevant interval (defined above) is at most $\frac{m}{2k/3}$. Using Markov's Inequality it follows that with probability at least $1/4$ at most $2m/k$ new sessions are initiated in an attempt to solve for this iteration. Thus, it is highly unlikely that we fail in all our $O(K)$ attempts at this iteration: Case 2b of `Solve` (i.e., solving an old session) may occur in at most $m \leq K$ attempts, Case 2(d)i occurs with probability at most $3/4$ (see above), and Case 2f occurs with probability at most 0.9 (see above). ■

Conclusion: Combining Theorem 7 with the resettable witness indistinguishable proof systems of Section 5, we prove the main result of the current section. This can be seen by either employing a general composition theorem or by extending the arguments in the simulation game to our actual two-stage protocol, using the techniques of Section 5.

[29] Following [37], the fact that we are talking about a sequence of k decommitments and are content with solving one of them allows to simplify the otherwise complex analysis of such procedure. Compare the treatment of a single decommitment, which must be solved no matter what, in [22].

6.3 An alternative proof

We present an alternative proof which uses the [37] analysis as a black-box (rather than modifying it as above).

The modified protocol: We slightly modify the protocol presented in the previous subsection. Specifically, we move the verifier's second commitment, which takes place in the first two rounds of the resettable witness indistinguishable protocol taking place in 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 call this combined commitment strings, the verifier's commitment-message. We comment that there is a minor problem here, since the graph (to which the resettable witness indistinguishable proof (of Stage 2) is applied) is not determined yet. However, 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. [25, 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). 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 $(1 - n^{-2})^t = \exp(-n)$ (as before). From this point on, we referred to the modified protocol as (P, V).

Intuitively, the above modification, only restricts the power of the verifier (as it needs to commit earlier to its choices). Since the original protocol (of the previous subsection) is a specific (complex)[30] implementation of the [37] protocol, it follows that both the original protocol as well as its modification are concurrent zero-knowledge. It is not hard to see that the modified protocol remains an interactive proof (since the commitments moved to the front yield no information to the prover). The key observation regarding the new protocol is that (in a single execution of the protocol) all the verifier steps following its first message (i.e., its commitment-message) are essentially determined. (See similar claims made in Section 5 and 6.1.) Thus, if the verifier repeats its commitment-message in the current session then it is essentially bound to continue as in a previous session until a point of its choice where it may abort. The key issue is how to deal with such restricted verifier behavior; note this is not trivial (see discussion of a subtle problem in Section 5). The other case is that the verifier uses a new commitment-message in the current session. But in such a case, by virtue of the way in which the prover uses its pseudorandom function to determine its actions, the current session is essentially independent of the previous ones. Using these observations, we show that the fact that (P, V) is concurrent zero-knowledge implies that it is also resettable zero-knowledge. (We stress that this is a feature of the specific proof system (P, V), and it is not true that any concurrent zero-knowledge protocol is also resettable zero-knowledge: The original protocol of [37] is a good counter-example.)

Motivation: Loosely speaking, our proof transforms an arbitrary adversary, V^*, operating in the resettable model into an adversary W^* of the concurrent model so that the output distribution of W^* after interacting with concurrent sessions of P is computationally indistinguishable from the output distribution of W^* after interacting with resettable sessions of P. Assume, for simplicity, that V^* is restricted so that it interacts with a single "incarnation" of P (rather than polynomially many such copies), and that whenever it sends the same commitment-message in session ξ, in each next round of session ξ it either sends the same (corresponding) decommitment message or

[30] Recall that the complications were needed in order to achieve resettable zero-knowledge.

abort. Also ignore the first message sent by the prover (i.e., the receiver message in the two-round commitment scheme employed by the verifier). (Both assumptions will be removed in the actual argument given below.) Using V^* as a black-box, we construct an "equivalent" W^* that interacts with concurrent (independent) copies of P. Machine W^* serves the messages of V^* as follows:

- In case, V^* sends a new commitment-message, denoted com, machine W^* initiates a new session, denoted $\xi_{\texttt{com}}$, with P, and supplies V^* with P's response.
- In case, V^* repeats a message (either a commitment-message or a decommitment), machine W^* just retrieves the corresponding response of P and forwards it to V^*. (W^* does not send any message to any session of P.)
- In case, V^* sends a new *valid* decommitment message, associated with the session having commitment-message com, machine W^* forwards this valid decommitment to session $\xi_{\texttt{com}}$ of P.
- In case, V^* sends an *invalid* decommitment message, machine W^* just feeds V^* with the standard message indicating that P has aborted this session. (We stress that W^* does not send any message to any session of P; in fact, W^* may need to send P a valid decommitment corresponding to the current session at a later stage.)

Note that although V^* interaction with P is in the resettable model, the induced interaction of W^* with P is restricted to the concurrent model. Specifically, the sessions initiated by W^* use independent prover's coins (determined by applying a random function to different values), and in each session the interaction is indeed in order (i.e., the verifier's j^{th} message is sent after its $j-1^{\text{th}}$ message, since the prover proceeds to round j only after receiving a proper decommitment for round $j-1$).

The actual proof (sketch)

Actually, some minor modifications have to be applied in order to allow the actual proof go through.

The augmented-concurrent model: We prove this claim by considering an augmented-concurrent zero-knowledge model: In this model, the verifier first asks the prover to provide one string, and then concurrent executions of the protocol take place where in all of these executions (sessions) the first prover's message is taken to be the string provided above.

Let us first consider the security of the modified protocol, (P, V), in this augmented-concurrent zero-knowledge model. In this specific protocol, the prover's first message is merely the receiver's message in a two-round perfectly-hiding commitment scheme. Clearly, the computational-binding feature of such a scheme holds also when several executions with the same random receiver-message take place. Thus, the effect of augmentation on the specific protocol that we consider (i.e., on (P, V)) is immaterial. (To justify this claim, we observe that the only role of the prover's first message in the analysis of the protocol is to guarantee that the verifier cannot decommit to two different values.) It follows that the modified protocol (P, V) is zero-knowledge also in the augmented-concurrent model.

A last modification model: Actually, we need to modify the augmented-concurrent model a bit as follows. Rather than selecting one string (at the onset of the interaction), the prover selects polynomially-many such strings and puts them in a list. When later the verifier initiates a new (concurrent) session it (i.e., the verifier) specifies which of these strings should be used as first message in this session. Clearly, (P, V) is zero-knowledge also in this modified augmented-concurrent model. ¿From this point on, we referred to the modified augmented-concurrent model as the augmented-concurrent model.

Finally, we claim that for every probabilistic polynomial-time adversary, V^*, interacting with P in the resettable zero-knowledge model there exists a probabilistic polynomial-time adversary, W^*, interacting with P in the augmented-concurrent zero-knowledge model so that the output distribution of W^* in actual executions (in the augmented-concurrent model) is computationally indistinguishable from the output distribution of V^* in executions (of the resettable model).[31] Thus, the simulator provided for W^* (by the above claim) is also adequate for V^*. Following is an outline of our construction of W^*, which uses V^* as a black-box. We stress that W^* may concurrently execute several sessions with prover P, but in these sessions – with the exception of the first message that is determined as postulated in the augmented model – the prover's actions are independent of its actions in other sessions. In contrast, V^* (being in the resettable model) may wish to conduct several sessions with a fixed random incarnation of the prover, denoted $P^{(i,j)} = P_{x_i,y_i,\omega_j}$ (see terminology in Section 4). For simplicity, we may again assume that V^* is in the non-interleaving model (see Theorem 3).

The construction of W^*: Working in the augmented-concurrent model, W^* first asks the prover to supply polynomially-many strings (to be used as the prover's first message). These strings are put in a list in which entries correspond to pairs of integers. Next, adversary W^* handles the messages of V^* as follows:

1. V^* *initiates a new session*: Suppose that V^* initiates a new session with one of the incarnations of P. Specifically, suppose it is the ξ^{th} time that V^* initiates a session with $P^{(i,j)}$. Then, adversary W^* initiates a new session with the actual prover, and feeds V^* with the prover's response. Specifically, the adversary asks the prover to use the string (i, j) as its first message in this session, which is denoted (i, j, ξ).

   (It is crucial that W^* is in the augmented-concurrent model, rather than being in the standard concurrent model: otherwise, W^* could not have initiated two sessions with the actual prover while insisting that the actual prover uses the same first message in both of them.)

2. V^* *sends a new commitment-message*: Suppose that V^* sends as its commitment-message, in session ξ with $P^{(i,j)}$, a string different from all commitment-messages is has sent in prior sessions with $P^{(i,j)}$. Let $\mathtt{com}$ denote this commitment-message. Then W^* sends $\mathtt{com}$ to session (i, j, ξ) of the actual prover, and feeds V^* with the prover's response. It records ξ as the active session of $P^{(i,j)}$ for commitment $\mathtt{com}$.

   (In subsequent messages directed by V^* to $P^{(i,j)}$, for every commitment-message $\mathtt{com}$, adversary W^* will communicate only with session $(i, j, \xi_{\mathtt{com}})$, where $\xi_{\mathtt{com}}$ is the active session of $P^{(i,j)}$ for $\mathtt{com}$.)

   (Note that in case V^* sends different commitment-messages (to sessions of the same $P^{(i,j)}$), W^* uses as reply messages obtained from two different $(i, j, \cdot)$-sessions. Case 1 ensures that these sessions share the first prover's message, but are otherwise independent (from the prover's point of view).)

[31] As in other cases in this paper, the difference may be due to a case in which V^* decommits differently to the same message, in two sessions it conducts with the same "incarnation" of the prover (i.e., induced by the same prover's random-pad).

3. V^* *sends a new valid decommitment*: Suppose that V^* makes for the first time a valid decommitment to a specific commitment component in a session with a specific $P^{(i,j)}$ in which a specific commitment-message is sent. Let $\mathtt{dec}$ denote this decommitment, ξ denote the current session of $P^{(i,j)}$, $\mathtt{com}$ denote the commitment-message in this session, and $\xi_{\mathtt{com}}$ be the active session of $P^{(i,j)}$ for commitment $\mathtt{com}$. Then W^* sends $\mathtt{dec}$ to session $(i, j, \xi_{\mathtt{com}})$ of the actual prover, and feeds V^* with the prover's response.

   (We wish to clarify the conditions made in this case: Recall that the commitment-message $\mathtt{com}$ has $k+1$ components; one per each of the k iterations of Stage 1, and a single one in Stage 2. Let ℓ be the index of the current decommitment; that is, $\mathtt{dec}$ is the ℓ^{th} decommitment taking place in session ξ of $P^{(i,j)}$. The current case that $\mathtt{dec}$ be a valid decommitment. Furthermore, it requires that in any prior session of $P^{(i,j)}$, in which the commitment-message is $\mathtt{com}$, the ℓ^{th} decommitment either did not take place or was invalid.)

4. V^* *sends an invalid decommitment*: Suppose that V^* makes an invalid decommitment to a specific commitment component in a session with a specific $P^{(i,j)}$. (We don't care whether or not this is the first time such an invalid commitment is sent for this specific commitment-message.) In this case, all that W^* does is feed V^* with a message indicating that the prover has halted this specific session.

   (We stress that W^* does not communicate this halting message to (session (i, j, ξ) of) the actual prover, where ξ is the active session of $P^{(i,j)}$ for the corresponding commitment. This point is crucial, since when responding to future messages of V^*, machine W^* may need to send a valid decommitment to the same commitment in session (i, j, ξ), so to obtain the prover's response (which it may not be able to generate). We are fortunate not to need any help in generating the prover's response to an invalid decommit. Note that W^* never sends an invalid message to the actual prover; this should not bother the reader – our aim is to simulate the view of V^*, not the joint view of copies of the provers interacting with V^*.)

5. V^* *repeats a message*: Suppose that V^* repeats in session ξ with $P^{(i,j)}$ a message that has appeared in a prior session with $P^{(i,j)}$. (Such a message may either be the verifier's commitment-message or one of its decommitment messages.) Let $\mathtt{msg}$ be the current message, $\mathtt{com}$ be the corresponding ($(k+1)$-component) commitment-message of this session, and $\xi_{\mathtt{com}}$ be the active session of $P^{(i,j)}$ for commitment $\mathtt{com}$. Then W^* just copies the response to $\mathtt{msg}$ obtained from the prover in session $(i, j, \xi_{\mathtt{com}})$, and feeds it to V^*.

   Our description assumes that whenever V^* provides a valid decommitment to the same commitment, it always provides exactly the same text; that is, decommit to the same value while providing the same proof (witness) to vouch for the validity of the value. Recall, that we may ignore the rare cases (occurring with negligible probability) in which the verifier decommits properly to two different values. So the issue is what to do when the verifier decommits to the same value using two different witnesses. A possible solution is to treat the latter case as we treat the case of identical messages (i.e., pretend that the decommit message is identically the same). This requires a minor modification of the protocol, following the alternative suggested in Footnote 22; that is, don't include the witness in the session history to which the pseudorandom function is applied. (Note that, intuitively, V^* has nothing to gain from using a different witness for proper decommitment to the same value; it is the value itself that matters. In fact, given the above modification, the prover ignore the witness once the correctness of the decommitted value has been established.)

(It is not important that W^* does not communicate with the actual prover in case a massage is repeated, provided it does so in a way which is not conflicting with its other actions (e.g., W^* should not communicate on session $(i, j, \xi_{\texttt{com}})$, since the actual prover of this session has already responded to the current message msg). What is important is that the message fed to V^* must be exactly the one given in prior sessions with $P^{(i,j)}$ in which V^* sent the same msg (in the same round). For simplicity, we just specify that W^* does not communicate with the actual prover in the current case.)

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

7 NP has Resettable ZK Arguments

In this section we show how to achieve rZK arguments for NP. Our protocol uses no timing assumption, preprocessing, or public-files, but runs in $O(n)$ number of rounds, where n is the security parameter. Mutatis mutandis, the situation is analogous to the original ZK proof of quadratic residuosity, which, in order to be zero-knowledge, must be executed in $O(n)$ *sequential* rounds for having an error probability of 2^{-n}. (Only years later was a constant-round ZK protocol for quadratic residuosity found. We do not know whether the same may happen here.)

As we have said already, our protocol can be implemented with any verifiable commitment scheme, though for simplicity in this extended abstract we prefer to rely on a rather more concrete complexity assumption.

7.1 The Strong DLP Assumption for Safe Primes

Let p be a prime, g a generator for Z_p^* (the multiplicative group modulo p), and y an element in Z_p^*. Then the Discrete Logarithm Problem (DLP) consists of finding, on inputs p, g amd y, an element $x \in [1, p-1]$ such that $g^x \equiv y \pmod{p}$. In this extended abstract we assume that (1) the circuit complexity of this task is sub-exponential also in the special case where p is a *safe* prime (i.e., of the form $p = 2q + 1$ where also q is a prime), and that (2) safe primes are easily samplable. More precisely

Strong DLP Assumption For Safe Primes: The following two properties hold:

1. *Samplability.* There exists a probabilistic polynomial-time algorithm that, for all sufficiently large n, on input 1^n outputs a random safe prime of length n.

2. *Hardness.* $\exists \epsilon > 0$ such that, for every sufficiently large n, for every circuit C of size at most 2^{n^ϵ}, for a random, safe prime p of length n, for a random generator g of Z_p^*, and a random $x \in Z_p^*$:
$$\Pr[C(p, g, g^x \bmod p) = x] \ < \ 2^{-n^\epsilon}.$$

The above DLP assumption is quite reasonable, and enables us to implement our needed verifiable commitment in a very easy manner. Notice that we could also implement our verifiable commitment based on a weaker assumption, but at the price of additional complexities. In particular, we could rely on the same DLP assumption as above, but stated for general primes rather than for safe ones.[32] Alternatively, we could implement verifiable commitment based on the assumption that the circuit complexity of factoring is sub-exponential.

[32] We use safe primes because it is easy to check whether g is a generator mod p if the prime factorization of $p-1$ is an available input. Note that, in the case of a safe prime p, such a factorization is indeed available: one has to divide

7.2 Initial Remarks About Our Protocol

To show that all NP languages have rZK arguments, we show one such argument for 3-colorability.

Security parameters. Our protocol uses instances of the DLP relative to safe primes of two different lengths: K and k. (In the general version, this corresponds to two commitment schemes with different security parameters.) K and k are not chosen independently. Rather, the protocol requires that K be polynomially bigger than k: more precisely, $K = k^{\frac{1}{2\epsilon}}$ where ϵ is the hardness constant of our DLP assumption. Thus, solving a random instance of the strong DLP will be much harder modulo a K-bit safe prime than modulo a k-bit safe prime. In fact, even if one were given the ability of performing a number of computational steps equal to those necessary for solving the DLP modulo a k-bit safe prime by exhaustive search (rather than by the best known discrete-log algorithm), then his chance of solving the DLP modulo a K-bit safe prime would be totally negligible.

Pseudo-random functions. In our protocol the prover random tape consists of a secret input s, which he uses as the seed of a pseudo-random function à la [GGM], f_s. The length of seed s may be chosen quite independently of K and k: it is only for simplicity that below we choose it to be K-bit long. The prover is de-facto deterministic: at each step of the protocol, all of his "random" choices are made by applying f_s to the history of the communication so far.

Rounds. The number of rounds of our protocol is not constant, but is a quite independent security parameter. For simplicity, below we let this other parameter also be equal to K, and implement our protocol 3K+4 rounds.

Two logical parts. Our protocol consists of two logical parts.

In the first part (steps 1-2) the verifier uses a K-bit instance of the DLP to establish a trapdoor commitment scheme. The prover will use this commitment scheme later on in the protocol to encode the 3-coloring of the input graph, $\mathcal{G}$. The commitment scheme is *perfectly private* for the verifier (i.e., the verifier will have absolutely no information about the colors the prover commits to) and *computationally binding* for the prover (i.e., in our case, the prover cannot "change" the committed colors unless he solves a K-bit instance of the DLP). The commitment scheme has an additional property: trapdoorness. Namely, the verifier embeds in the instance of the DLP (used for implementing a commitment scheme) some auxiliary information (the trapdoor) which allows him to decommit in more than 1 way. This extra feature does not allow him to cheat: recall that this DLP instance is used to commit by the prover, who knows no trapdoor information. In step 2, the verifier actually proves in ZK to the prover that he knows such trapdoor. This (ordinary) ZK proof of knowledge is necessary for making the whole protocol rZK.

In the second part (steps 3-8), the prover convinces the verifier that there exists a proper 3-coloring for the input graph. At a high level, this is done following a traditional approach consisting of four logical stages executed independently and in parallel: (1) the verifier commits in advance to an edge, e of $\mathcal{G}$; (2) the prover commits to a (randomized) 3-coloring of $\mathcal{G}$; (3) the verifier decommits e; and (4) the prover decommits the coloring of the e's end points. The commitment scheme used by the verifier for e has properties that are symmetric to those of the commitment scheme used

$p-1$ by 2 and check that the result is also prime. We could also rely on the assumption that the DLP remains hard even if the factorization of $p-1$ is available. This assumption too is widely believed; moreover, it has proven that it is possible to generate random n-bit integers in factored form [Bach]. One could therefore generate such random integers and then add one until a prime is found, without relying on any type of samplability assumption.

by the prover for the coloring. That is, it is perfectly private for the prover and computationally binding for the verifier. This second commitment scheme is actually chosen by the prover using a k-bit instance of the DLP. (It will not matter whether this second scheme has an embedded trapdoor or not.)

7.3 DLP-Based Commitment Schemes

Relying on the intuitive notion of what a commitment scheme is, let us informally describe some commitment schemes under the DLP assumption.
A FOLKLORE BIT-COMMITMENT SCHEME. Assume that p is a prime, g a generator for Z_p^*, and y an element of Z_p^*. Then the following protocol $COMM_{p,g,y}$ is a well-known way for a party P to commit to a bit b.

Scheme $COMM_{p,g,y}$

1. (Committing Instructions for P) To commit to a bit b, randomly select r in Z_p^* and output the value $C = COMM_{p,g,y}(b) = y^b g^r \bmod p$.
2. (Decommitting Instructions for P) Output $DECOMM_{p,g,y}(C) = r$.

It is immediately seen that the value C is uniformly distributed in Z_p^* both when $b = 0$ and when $b = 1$. Thus $COMM_{p,g,y}$ enjoys perfect secrecy. Moreover $COMM_{p,g,y}$ also enjoys computational soundness in the sense that, to be able to decommit C both as 0 and as 1, P must find the discrete log of y in base g (mod p).

A FOLKLORE STRING COMMITMENT. Scheme $COMM_{p,g,y}$ is immediately extended to a string commitment scheme (with perfect secrecy and computational binding under the DLP) as follows: to commit to a binary value $v = v_1, \ldots, v_n$, commit to each v_i individually and independently. That is,

$$C = COMM_{p,g,y}(v) = COMM_{p,g,y}(v_1), \ldots, COMM_{p,g,y}(v_n) = C_1, \ldots, C_n$$

and

$$DECOMM_{p,g,y}(C) = DECOMM_{p,g,y}(C_1), \ldots, DECOMM_{p,g,y}(C_n).$$

A SECOND BIT- AND STRING-COMMITMENT SCHEME. Assume that p is a prime, that g is a generator for Z_p^*, and that $y_1, \ldots, y_n$ are n distinct elements of Z_p^*. Then, the following protocol is a bit commitment with perfect secrecy and computational binding under the DLP.

Scheme $COMM_{p,g,y_1,\ldots,y_n}$

1. (Committing Instructions) To commit to a bit b, randomly select bits $b_1, \ldots, b_n$ at random so that $\sum_{i=1}^n b_i = b$ and output
$$C = COMMp, g, y_1, \ldots, y_n(b) = COMM_{p,g,y_1}(b_1), ..., COMM_{p,g,y_n}(b_n) = C_1, \ldots, C_n.$$
2. (Decommitting Instructions for P) To decommit $C = C_1, \ldots, C_n$, output
$$DECOMM_{p,g,y_1,\ldots,y_n}(C) = DECOMM_{p,g,y_1}(C_1), \ldots, DECOMM_{p,g,y_n}(C_n).$$

It is immediately seen that $COMM_{p,g,y_1,\ldots,y_n}$ is a commitment scheme with perfect secrecy and computational binding if the DLP is hard. More precisely: P can decommit C in more than one way, if and only if P finds the discrete log in base g of y_i for some $i = 1, \ldots, n$.

Note: With an eye to the final paper, the above DLP-based schemes also are trapdoor, verifiable commitment schemes if they are implemented with safe primes, and enriched with the steps of verifying that p is a safe prime and that g is a generator mod p.

7.4 An rZK Argument for 3-Colorability

Protocol (P, V)

- **Security Parameter(s):** k and K (where $K = k^{\frac{1}{2\epsilon}}$, and ϵ is the hardness constant of our strong DLP assumption).
- **Common input to P and V:** A 3-colorable graph $\mathcal{G}$ with vertex set VERS and edge set EDGES (where VERS has cardinality n and EDGES has cardinality m).
- **P's secret seed:** a random string $s \in_R \{0,1\}^K$ (specifying a GGM pseudo-random function f_s).
- **Secret input to P :** a 3-coloring of $\mathcal{G}$, $COL : VERS \rightarrow \{1,2,3\}$ (where $COL(v)$ is the color of vertex v).

1. (Instructions for V)

   Randomly select: (a) a safe prime p of length K, (b) a random generator g for Z_p^*, and (c) elements $x_1, ..., x_K$ in Z_p^*. Compute $X_i = g^{x_i} \bmod p$ for $i = 1, \ldots, K$, and send p, g, $X_i, \ldots, X_K$ to P.

   (Instructions for P)

   Check that p is a safe prime of length K, that g is a generator for Z_p^*, and that $X_1, \ldots, X_K \in Z_p^*$. If not, halt.

2. For $i = 1...K$,

   (a) (instructions for V)
   Pick r_i in Z_p^* at random and send $R_i = g^{r_i} \bmod p$ to P.

   (b) (Instructions for P)
   Check that $R_i \in Z_p^*$; if not, halt. Compute b_i as the last bit of $f_s(pgX_1...X_KR_1...R_i)$, and send b_i to V.

   (c) (Instructions for V)
   If $b_i = 0$, then send r_i to P. Else, send $t_i = r_i + x_i \bmod p - 1$ to P.

   (d) (Instructions for P)
   If $b_i = 0$, then check that $g^{t_i} = R_i \bmod p$. If not, halt. If $b_i = 1$, then check that $g^{t_i} = R_iX_i \bmod p$. If not, halt.

3. (Instructions for P)

   Select: (a) a safe prime q of length k, (b) a random generator h for Z_q^*, and (c) an element y in Z_q^*. Compute $Y = g^y \bmod q$, and send q, h, and Y to P.

   (This selection is done "at random", but using as coin tosses the outcome of f_s applied to all previous messages sent so far by the verifier.)

4. (Instructions for V)

   Check that q is a safe prime of length k, that h is a generator for Z_q^*, and that $Y \in Z_q^*$. If not, halt.

   For $j = 1 \ldots, n^3$, randomly select edge $e_j = (u_j, v_j)$ in $\mathcal{G}$, and send to P the commitment values $E_j = COMM_{q,h,Y}(e_j)$.

5. (Instructions for P)

   For $j = 1, \ldots, n^3$, choose π_j, a random permutation of $\{1, 2, 3\}$, and, for all $u \in VERS$, send $C_{j,u} = COMM_{p,g,X_1,\ldots,X_k}(\pi_j(COL(u)))$ to V.

6. (Instructions for V)

   For $j = 1, \ldots, n^3$, decommit $e_j = (u_j, v_j)$ by sending $DECOMM_{q,h,Y}(E_j)$.

7. (Instructions for P)

   For $j = 1 \ldots, n^3$, if any of the decommitments are invalid, then reject. Else, for $j = 1 \ldots, n^3$ decommit the colors, $\pi_j(COL(u_j)$ and $\pi_j(COL(v_j)$, of the end points of e_j by sending $DECOMM_{p,g,X_1,\ldots,X_k}(C_{j,u_j})$ and $DECOMM_{p,g,X_1,\ldots,X_k}(C_{j,v_j})$ to V.

8. (Instructions for V)

   If for some j an edge with two end points of the same color is discovered, or a wrong decommitment is received, then reject. Else, accept.

Theorem: (P, V) *is a an resettable zero-knowledge argument for 3-colorability.*

Proof Sketch

COMPLETENESS. If the prover P knows a 3-coloring of the input graph, then it is immediate to show that, as long as he will follow the protocol, he will convince the honest verifier V with probability 1.

SOUNDNESS.[33] We have to show that, if the input graph $\mathcal{G}$ is not 3-colorable, then a polynomial-time cheating prover P' has but a negligible chance of leading honest V to acceptance. The essence of this proof is best sketched in the extreme case in which we assume (for contradiction purposes) that such polynomial-time P' has probability 1 of convincing V. From this hypothesis we shall derive that there exists a 2^k-size circuit C that solves K-size instances of the DLP for safe primes with probability roughly $1/2K$. This contradicts our DLP assumption given the relative sizes of security parameters k and K.

We construct C by combing the polynomial-time P' with the following $O(2^k)$-time machine M. On input p, g and X, a random K-bit instance of the DLP for safe primes, M randomly selects

[33] The argument is analogous to the one presented in Section 9.1.

$i \in \{1, \ldots, K\}$, and sets $X_i = X$. Then, for $j \in \{1, \ldots, K\} - \{i\}$, it randomly selects $x_j \in Z_p^*$ and sets $X_j = g^{x_j} \bmod p$. Finally, it sends p, g, and $X_1, \ldots, X_K$ to P'. Note therefore that, by construction, M has sent P' a message that is distributed exactly as the step-1 message from V to the prover in protocol (P, V).

Notice now that M has a probability 1/2 of being able to answer the "questions" of P' in all iterations of Step 2, and with exactly the same distribution as in a random execution of (P', V). This is so because M has probability 1/2 of "passing" the question about X_i, and probability 1 of passing the questions about all other X_j (for which it knows the discrete log in base g by construction). Note also that step 2 is a perfect zero-knowledge proof of knowledge (though with probability of cheating $\frac{1}{2}$) of the discrete logarithms of $X_1, \ldots, X_K$ where P' serves as the step-2-verifier and V (M inside this argument) serves as the step-2-prover. Thus, P' cannot use it in any manner to gain knowledge of the secrets x_j nor of the discrete log of X_i. [34]

Notice also that in proving that step 2 is a perfect zero knowledge proof of knowledge (where P' is the step-2-verifier and V the step-2-prover) we do not have to worry about resettability of the step-2-verifier, as in an rZK protocol the prover P' is not resettable, rather V is.

Let us now describe how M executes step 4. Upon receiving from P' a k-size instance of the DLP for safe primes, q, h and Y, in step 3, M conducts an exhaustive search and finds the discrete log, y, of Y in base h (mod q). This will take 2^k computational steps. At this point, it commits to every edge e_j by committing to the right number of 0s. That is, by sending $COMM_{q,h,Y}(0 \cdots 0)$ to P'. Notice that each such message is equally distributed to the corresponding one of honest V because the commitment scheme has perfect privacy. Therefore, by hypothesis, P' still has probability 1 of convincingly executing the remaining steps when he sends M his commitments $C_{j,u}$ in step 5. Call S the internal state of P' at this point. Now M behaves as follows:

First, it executes step 6 by decommitting e_j to be always (i.e., for $i = 1, \ldots, n^3$) the first edge of $\mathcal{G}$. (Notice that it can do this because $COMM_{q,h,Y}$ is a trap-door commitment scheme, and M has, by exhaustive search found the trapdoor.) Because such behavior were possible (though unlikely) for the honest V, and because P' has a probability 1 of satisfying V, P' will respond by decommitting the colors of the end point of the first edge for each of the n^3 committed, colored copies of $\mathcal{G}$. Then, M resets P' in state S.

Second, M executes step 6 again, this time by decommitting e_j to be always the second edge of $\mathcal{G}$. P' decommits accordingly, and then M resets P' in state S.

And so on, for m times, that is, once for each edge of $\mathcal{G}$.

Focus now on the first committed colored copy of $\mathcal{G}$, and see what happens on an edge-by-edge basis. Because P' has probability 1 of fooling V, then the decommitted colors of the end points of edge 1 will be locally correct (i.e., different from each other); the same holds for the colors of edge 2, edge 3, ..., edge m. However, because we are also assuming that $\mathcal{G}$ is NOT 3-colorable, it must be that these decommitted colors are NOT globally consistent. Hence, there must exist some some j between 1 and m, such that, letting (u_j, v_j) be the jth edge of $\mathcal{G}$, either (a) the color of u_j has been decommitted by P' in two different ways, or (b) the color of v_j has been

[34] Formally, to argue that step 2 is a perfect zero-knowledge proof of knowledge, let us simulate P' view during the step 2 (note that here we need not worry about resettability as we arguing that the verifier is giving no knowledge to the prover rather than the other way around). The simulator SIM has access to the prover P' (which is a polynomial time algorithm) and has as inputs: all the inputs of the protocol, and what was sent in step 1 of the protocol to the prover by the verifier, i.e $p, g, X_i = g^{x_i}$. Initialize $SIM - OUTPUT = \epsilon$. In round i, (1) SIM picks $c_i \in \{0, 1\}$ at random. If $c_i = 0$, then SIM chooses at random R_i and sends g^{R_i} mod p to P; otherwise SIM chooses at random R_i and sends $g^{R_i - x_i}$ mod p to P. (2) if P answer with $b_i = c_i$, then P send V R_i, and SIM sets $SIM - OUTPUT = SIM - OUTPUT \| (g^{R_i} modp, b_i, R_i)$ (where $\|$ denotes concatenation), otherwise, SIM goes back to step 1 of round i, and rewinds the P' to the previous round.

decommitted by P' in two different ways. Either way, because these commitments are made using our $COMM_{p,g,X_1,...,X_K}$ scheme, and because of the property of our scheme, this entails that from such two different decommitments for the same value the discrete log of either X_1, or X_2, or,..., X_K has been found. Because i was randomly chosen in $\{1,...,K\}$ (and the proper probability distribution totally respected), with probability $1/K$, it will be the discrete log of X_i to have been found, that is, the discrete log base g of our initial random element X mod p. Thus, keeping track also of the probability 1/2 of passing step 2, this leaves us that an algorithm consisting of running P' and M together, and thus taking at most (roughly) 2^k steps, has chance $1/2K$ of solving a random K-size instance of the DLP for safe primes. The contradiction establishes soundness for the case in which P' can cheat with probability 1. It is not hard to see how M should be modified so as to take in consideration small probability of cheating.

(Comment: M's resetting of P' in the above argument should not be confused with the prover-resettability of rZK protocols. In fact, the latter resettability applies to provers whose coin tosses are not known to an adversary verifier, and thus the prover is a kind of unknown (at least for his coin tosses) device that can nonetheless be reset. In the present context, instead, resettability is quite straightforward. Namely, if algorithm P' exists, then M can run with random coins that M chooses and controls. Therefore M knows the entire internal configuration of P' at each moment in time, and thus it can easily reset P' any time it likes.)

Resettable Zero-Knowledge feature. What remains to be shown, having established that the protocol is complete and sound is that it is Resettable Zero-Knowledge. The simulator can be constructed by a combination of the ideas presented in Sections 9.1 and 6.2. Details are omitted.

Part II
The Public-Key Model

8 Discussion and Definition

The vanilla model, considered in Sections 4–6, 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).[35] 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 precomputation. This is not required of users who are only going to play the role of provers.

Definition (sketch): Analogously to Definition 2, 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

[35] 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.)

9 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 8 *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.*

9.1 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 [19, 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.[36]

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 $\phi : [n] \mapsto \{1, 2, 3\}$ of G.

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

The protocol itself is an adaptation of the resettable witness indistinguishable proof system of Section 5 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.]

[36] 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., [29, 25]), the current simulator does not "rewind" the verifier. Instead, it simulates an execution of the protocol by emulating the actions of the prover in Steps (1)–(4) using some dummy sequence, rather than a sequence of colorings, in Step (3). However, when getting to Step (5), and in case the verifier has decommitted properly, the simulator uses trap(i)

in order to decommit to the corresponding edge queries in a random legal-looking way (i.e., it decommits to a uniformly and independently chosen pair of distinct colors, for each such edge). This uses the trapdoor feature of PC1 and the hypothesis that the verifier (and so the simulator) knows this trapdoor. The above description corresponds to simulation of the first 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.

9.2 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).[37] 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[38] (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

[37] This property holds also for other protocols for NP, but not for the 3-Colorability protocol of [25]. Any protocol having the property will do.

[38] 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., [29, 25]), 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. 13 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).[39] To extract the Hamiltonian cycle we just invoke the knowledge-prover many times, each time it answers with the

[39]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 $\mathrm{poly}(k)/(p - 2^{-k})$. By a trivial modification, we obtain a knowledge extractor which for any $p > 0$ with overwhelming probability runs for time $\mathrm{poly}(k)/p$, and in case $p > 2^{-k}$ also retrieves the trapdoor.[40]

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 $\mathrm{poly}(k)/p$ 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 $(1-p)\cdot \mathrm{poly}(k) + p \cdot (\mathrm{poly}(k)/p) = \mathrm{poly}(k)$. 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 $\mathrm{poly}(k)$.

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.[41] 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

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

[41] 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 6) 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).

9.3 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 [25]) 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 $s : \mathsf{N} \mapsto \mathsf{N}$. In particular, we obtain

Theorem 9 *Let $r : \mathsf{N} \mapsto \mathsf{N}$ 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.

10 Alternative Constant Round RZK Protocol for NP in the Public Key Model

In this section we give an alternative version of the Resettable Zero Knowledge (RZK) Proof for NP. This presentation does not use Blum's (or any other version) of the general proof that NP statement has Zero-Knowledge proofs. Rather, we define two types of commitment schemes Type-1 and Type-2 and show how to use them directly to give RZK protocol for NP statements in the public key model where the verifier has a public key assigned to it. These commitment schemes exist under the strong DLP assumption. We note that this version of the protocol is very similar to the perfect rZK arguments for NP protocol, except that the latter does not run in constant number of rounds. Having introduced the public-key model enables achieving rZK arguments for NP in constant number of rounds. Whereas in Section 7 we describe the protocol in terms of the discrete problem, here we will define the commitments schemes abstractly.

Preliminaries

Probability spaces.[42] If $A(\cdot)$ is an algorithm, then for any input x, the notation "$A(x)$" refers to the probability space that assigns to the string σ the probability that A, on input x, outputs σ. The set of strings having a positive probability in $A(x)$ will be denoted by "$\{A(x)\}$".

If S is a probability space, then "$x \stackrel{R}{\leftarrow} S$" denotes the algorithm which assigns to x an element randomly selected according to S, and "$x_1, \ldots, x_n \stackrel{R}{\leftarrow} S$" denotes the algorithm that respectively assigns to, $x_1, \ldots, x_n$, n elements randomly and independently selected according to S. If F is a finite set, then the notation "$x \stackrel{R}{\leftarrow} F$" denotes the algorithm that chooses x uniformly from F.

If p is a predicate, the notation $PROB[x \stackrel{R}{\leftarrow} S; y \stackrel{R}{\leftarrow} T; \cdots : p(x, y, \cdots)]$ denotes the probability that $p(x, y, \cdots)$ will be true after the ordered execution of the algorithms $x \stackrel{R}{\leftarrow} S$; $y \stackrel{R}{\leftarrow} T; \cdots$. The notation $[x \stackrel{R}{\leftarrow} S; y \stackrel{R}{\leftarrow} T; \cdots : (x, y, \cdots)]$ denotes the probability space over $\{(x, y, \cdots)\}$ generated by the ordered execution of the algorithms $x \stackrel{R}{\leftarrow} S$, $y \stackrel{R}{\leftarrow} T, \cdots$.

10.1 Two Types of Commitments

In this section we introduce two types of commitment schemes which will be useful for our result.

[42] Verbatim from [4] and [30].

Type-1 Commitments (Verifiable Commitments)

Informal Description. A type-1 commitment (or alternatively called a "verifiable commitment") consists of a quintuple of algorithms. Algorithm GEN1 generates a pair of matching public and secret keys. Algorithm COM1 takes two inputs, a value v to be committed to and a public key, and outputs a pair, (c, d), of commitment and decommitment values. Without knowledge of the secret key, it is computationally hard —given c, v and d— to decommit to any value other than v (*computational soundness*). On the other hand, having seen c yields no information about the value v (*perfect secrecy*).

The knowledge of the secret key enables decommitting the same value c in arbitrary ways (*trapdoorness*). This arbitrary decommitment ability is achieved by by running the FAKE1 algorithm.

Finally, succeeding in decommitting any single value into more than one way is essentially equivalent to knowing the secret key (*one-or-all*). This property is achieved by algorithm FAKE$'$.

Put together, the properties of type-1 commitment yield (using standard terminology) a perfect-secrecy computationally- binding commitment scheme for which there exists auxiliary information (the secret key) whose knowledge enables decommitment in more than one way. Moreover, it is possible to give a *secure* "proof-of-knowledge" of the secret key. This commitment scheme will be used in the rZK protocol for graph 3-colorability in the following way: the verifier will publish the public key of the commitment scheme ahead of the protocol and keep to himself the secret key. At the onset of the rZK protocol itself, the verifier will essentially proves to the prover that he knows the matching secret key. This proof will be secure to the extent that the prover cannot learn any knowledge which will allow him to cheat. Next, the prover will use commitment scheme specified by the verifiers public key to encode the coloring of the input graph.

The Formal Notion.

Definition 1: A *Type-1 Commitment Scheme* is a tuple of probabilistic polynomial-time algorithms $GEN1(\cdot)$, $COM1(\cdot,\cdot)$, $VER1(\cdot,\cdot)$, $KEYVER1$, $FAKE1(\cdot,\cdot)$, and $FAKE1'$ such that

1. *Completeness.* $\forall k$, $\forall v$,

   $$PROB[(PK, SK) \stackrel{R}{\leftarrow} GEN1(1^k); (c,d) \stackrel{R}{\leftarrow} COM1(PK, v) : KEYVER1(PK, 1^k) = VER1(1^k, PK, c, v, d) = YES] = 1$$

2. *Computational Soundness.* $\exists \alpha > 0$ such that $\forall$ sufficiently large k and $\forall$ 2^{k^α}-gate adversary ADV

   $PROB[(PK, SK) \stackrel{R}{\leftarrow} GEN1(1^k)\ ;\ (c, v_1, v_2, d_1, d_2) \stackrel{R}{\leftarrow} ADV(1^k, PK)$:

   $v_1 \neq v_2$ and $VER1(1^k, PK, c, v_1, d_1) = YES = VER1(1^k, PK, c, v_2, d_2)] < 2^{-k^\alpha}$

   (We call α the soundness constant.)

3. *Trapdoorness.* $\forall$ $(PK, SK) \in \{GEN1(1^k)\}$, $\forall v_1, v_2$ such that $v_1 \neq v_2$ the following two probability distributions are identical:

   $[(c, d_1) \stackrel{R}{\leftarrow} COM1(PK, v_1)\ ;\ d_2' \stackrel{R}{\leftarrow} FAKE1(PK, SK, c, v_1, d_1, v_2) : (c, d_2')]$

   and

   $[(c, d_2) \stackrel{R}{\leftarrow} COM1(PK, v_2) : (c, d_2)]$

   (*Comment:* $d_2' \stackrel{R}{\leftarrow} FAKE1(PK, SK, c, v_1, d_1, v_2)$ implies $VER1(1^k, PK, c, v_2, d_2') = YES$)

4. *Perfect Secrecy.* $\forall$ PK such that $KEYVER1(PK, 1^k) = 1$ and $\forall v_1, v_2$:

   $[(c_1, d_1) \stackrel{R}{\leftarrow} COM1(PK, v_1) : c_1] = [(c_2, d_2) \stackrel{R}{\leftarrow} COM1(PK, v_2) : c_2]$

5. *One-Or-All.* $\forall$ $(PK, SK) \in \{GEN1(1^k)\}$, and $\forall c, v_1, v_2, d_1, d_2, C, V_1, D_1, V$ such that $v_1 \neq v_2$, $VER1(1^k, PK, c, v_1, d_1) = YES = VER1(1^k, PK, c, v_2, d_2)$, $(C, D_1) \in \{COM1(V_1, PK)$, and $V_1 \neq V_2$,:

   $PROB[D_2 \stackrel{R}{\leftarrow} FAKE1'(PK, c, v_1, v_2, d_1, d_2, C, V_1, D_1, V_2) : VER1(1^k, PK, C, V_2, D_2) = YES] = 1$

Type-2 Commitment

Informal Description. In type-1 commitment schemes, one commits to a value by means of a public key, and can de-commit at will if he knows the matching secret key.

In a type-2 commitment scheme, there is a single key used to commit to values, but this key can be easily inspected (by algorithm KEYVER2) to determine that a corresponding trap-door information exists (and thus can be used by algorithm FAKE2 to decommit at will). Because such trapdoor information exists, it can be found by an exhaustive search. It is not required, however, that there is a easy way to generate type-1 commitment keys and their trapdoor information *together*.

Type-1 and type-2 requirement appear to be incomparable.

The use we make of type-2 commitment in the rZK protocol for graph 3-colorability is for the verifier to commit to his questions about colors of end points of edges in the graph before he sees an encoding of the graph.

The Formal Notion.

Definition 2: A *type-2 commitment scheme* is a quintuple of probabilistic polynomial-time algorithms $GEN2(\cdot)$, $COM2(\cdot, \cdot)$, $VER2(\cdot, \cdot, \cdot, \cdot)$, $FAKE2(\cdot, \cdot)$ and $KEYVER2(\cdot)$,

1. *Completeness.* $\forall k$, $\forall v$,

   $$PROB[key \stackrel{R}{\leftarrow} GEN2(1^k)\ ;\ (c, d) \stackrel{R}{\leftarrow} COM2(key, v) : VER2(key, c, v, d) = YES] = 1$$

2. *Computational Soundness.* $\exists \alpha, > 0$ such that $\forall$ sufficiently large k and $\forall$ 2^{k^α}-gate adversary ADV

   $PROB[key \stackrel{R}{\leftarrow} GEN2(1^k)\ ;\ (c, v_1, v_2, d_1, d_2) \stackrel{R}{\leftarrow} ADV(key) :$

   $v_1 \neq v_2$ and $VER2(key, c, v_1, d_1) = YES = VER2(key, c, v_2, d_2)] < 2^{k^\alpha}$

   (α is referred to as the soundness constant.)

3. *Verifiable Trapdoorness.* $\forall$ key such that $KEYVER2(key, 1^k) = YES$ $\exists$ $trap \in \{0,1\}^k$ such that, $\forall v_1, v_2$ such that $v_1 \neq v_2$:

   $PROB[c \stackrel{R}{\leftarrow} COM2(key, v_1)\ ;\ d \stackrel{R}{\leftarrow} FAKE2(key, trap, c, v_1, d_1, v_2) : VER2(key, c, v_2, d) = YES] = 1$

4. *Verifiable Perfect Secrecy.* $\forall key$ such that $KEYVER2(key)$=YES and $\forall v_1, v_2$

   $[(c_1, d_1) \stackrel{R}{\leftarrow} COM2(key, v_1) : c_1] = [(c_2, d_2) \stackrel{R}{\leftarrow} COM(key, v_2) : c_2]$

Remarks on Type-1 and Type-2 Commitments

The above commitment schemes can be implemented under a variety of assumptions. For example, the assumption that family of claw-free trapdoor permutation pairs defined by [GoMiRi] suffices for Type-1 commitment. Moreover, this same assumption suffices for implementing type-2 commitment if KEYVER2 is relaxed to be an interactive procedure (or if it has access to a random string as required for non-interactive zero-knowledge proofs).

Alternatively, based on the assumption that the discrete logarithm problem is hard, both Type-1 and Type-2 commitment can be achieved as we show below. Even though the two commitment schemes implementations follow from the same complexity assumption, our rewindable zero-knowledge protocol uses commitments in two fundamentally different ways. Thus, having two different types of commitments enhances the understanding of the protocol, and may possibly lead to minimizing the complexity assumptions necessary in future implementations.

Finally, as shown within our RZK protocol (i.e., in its first 4 steps), the producer of a public-secret key pair for a type-1 commitment scheme, can prove in constant round that he knows the secret key corresponding to the public key without enabling the verifier of this proof of knowledge to "decommit at will".

10.2 Discrete-Log Implementations of Type-1 and Type-2 Commitment

Definition: We define the language DLP$'$ to consist of the quadruples $(p,g,x,\overline{p-1})$, where p is a prime, g a generator of Z_p^*, x an element of Z_P^*, and $\overline{p-1}$ is an encoding of the prime factorization of $p-1$. We denote by DLP'_k the set of quadruples in DLP' whose prime has length k: $DLP'_k \stackrel{\text{def}}{=} \{(p,g,x,\overline{p-1}) \in DLP' : |p| = k\}$.
Lemma 1: Under the strong DLP assumption, there exist a type-1 commitment scheme.

Proof: Define algorithms $GEN1$, $COM1$, $VER1$, $FAKE1$, and $FAKE1'$ as follows:

$GEN1$ is a probabilistic, polynomial-time algorithm that, on input 1^k, randomly selects a k-bit prime p, a generator g for Z_p^*, and $x \in [1, p-1]$ and outputs $PK = (p,g,y,\overline{p-1})$ and $SK = x$.

(Note: GEN1 makes use of the fact that one can generate k-bit composite numbers in factored form as shown by Bach.)

$COM1$ is a probabilistic polynomial-time algorithm that, on inputs $(p,g,y,\overline{p-1}) \in DLP'_k$ and a bit b, randomly selects $d \in \{1,...,p-1\}$, computes $c = g^d y^b \bmod p$, and outputs (c,d).

(Note: Longer binary strings are committed in a "bit-by-bit fashion")

$VER1$ takes as input $(p,g,y,\overline{p-1})$ and c,v,d. If $(p,g,y,\overline{p-1}) \in DLP'_k$ and $c = g^d y^v \bmod p$ it outputs YES, else it outputs NO.

$KEYVER1$ is a probabilistic polynomial time algorithm that takes as input $(p,g,y,\overline{p-1})$ and outputs YES if p is prime, g is generator for Z_p^*, and $y \in Z_p^*$, and NO otherwise.

$FAKE1$ takes as input $(p,g,y,\overline{p-1}) \in DLP'_k$ and (x,c,v_1,d_1,v_2) where $g^x = y \bmod p$, $v_1 \neq v_2 \bmod p-1$, and $c = g^{d_1} y^{v_1} \bmod p$, and outputs $d_2 = d_1 + x(v_1 - v_2) \bmod p-1$.

$FAKE1'$ takes as input $PK \in DLP'_k$ and $c, v_1, v_2, d_1, d_2, C, V_1, D_1, V$ such that $v_1 \neq v_2 \bmod p-1$, $V_1 \neq V \bmod p-1$, and $VER1(PK,c,v_1,d_1) = YES = VER1(PK,c,v_2,d_2)$. It computes $x = (d_1 - d_2)(v_1 - v_2)^{-1} \bmod p-1$ and outputs $D = D_1 + x(V_1 - V) \bmod p-1$. ■

Lemma 2: Under the strong DLP assumption, there is a type-2 commitment scheme.

Proof: Define algorithms $GEN2$, $COM2$, $VER2$, $FAKE2$, and $KEYVER2$ as follows:

$GEN2$ is a probabilistic, polynomial-time algorithm that, on input 1^k, randomly selects a k-bit prime q together with $\overline{q-1}$, a generator h for Z_q^*, and $z \in Z_q^*$ and outputs $PK = (q, h, z, \overline{q-1})$.

$COM2$ is a probabilistic polynomial-time algorithm that, on input $(q, h, z, \overline{q-1}) \in DLP'_k$ and a bit b, randomly selects $d \in \{1, ..., q-1\}$, computes $c = h^d z^b \bmod q$, and outputs (c, d).

(Note: Longer binary strings are committed in a "bit-by-bit fashion")

$VER2$ takes as input $(q, h, z, \overline{q-1}) \in DLP'_k$ and c, v, d. If $c = h^d h^v \bmod q$ it outputs YES, else it outputs NO.

$KEYVER2$ takes as input $(q, h, z, \overline{q-1}) \in DLP'_k$ and 1^k. If q is prime, h is a generator for Z_q^* and $z \in Z_q^*$ it outputs YES, else it outputs NO.

$FAKE2$ takes as input $key = (q, h, z, \overline{q-1})$ such that $KEYVER(key, 1^k) = YES$, $trap \in \{0,1\}^k$ and c, v_2, v_1, d_1 such that $v_1 \neq v_2 \bmod p-1$ and $VER2((q, h, z), c, v_1, d_1) = YES$. If $h^{trap} = z \bmod q$, then output $d_2 = d_1 + trap(v_1 - v_2) \bmod q-1$.

Note that for every $key = (q, h, z)$ where $KEYVER(key, 1^k) = YES$, there exists $trap$ such that $h^{trap} = z \bmod q$ (as required above). ■

10.3 An RZK Protocol For 3-Coloring Using Public Keys

Initial Remarks

The protocol utilizes a type-1 commitment scheme, $(GEN1, COM1, VER1, KEYVER1, FAKE1, FAKE1')$, with soundness constant α_1. Before the protocol starts, the verifier runs GEN1 with security parameter K to obtain a public key, PK, and its matching secret key, SK. This public key will be a common input of prover and verifier. The second common input will be G, a graph that the prover claims to be 3-colorable. The private input of the verifier consist of SK, while the private input to the prover consist of a seed s for a pseudo-random function à la [GGM], f_s.

The protocol also uses a type-2 commitment scheme, $(GEN2, COM2, VER2, FAKE2, KEYVER2)$, with soundness constant α_2. The prover generates a (single) key for this commitment scheme by using GEN2, with security parameter k, during run time.

Note that the security parameters K and k are not chosen equal, nor independently. Rather, the protocol requires that K be suitably bigger than k: more precisely, $K = k^{\frac{1}{2\alpha_1}}$. [43] The length of the seed s, may however, be chosen quite independently of K and k: it is only for simplicity that we chose it to be K-bit long.

At a very high level, the protocol consists of two phases. First, the verifier convinces the prover that he "knows" SK(steps 1-4). Second, the prover convinces the verifier that the input graph $\mathcal{G}$ is 3-colorable (steps 5-10). The prover is de-facto deterministic: at each step of the protocol, all his "random" choices are made by applying f_s to the history of the communication so far.

The Protocol

[43] As a result, "cheating" will be hard in both schemes, but it will be much harder for the type-1 scheme than for the type-2 scheme. In particular, as it will become clear later on, finding two different decommitments for the same type-2 commitment cannot significantly help in finding SK, the type-1 secret key.

Protocol $\mathcal{RZK}$

- **Security Parameter(s):** k and K where $K = k^{\frac{1}{2\alpha_1}}$.
- **Verifier's public and secret key:** $(PK, SK) \stackrel{R}{\leftarrow} GEN1(1^K)$.
- **Prover's secret seed:** $s \in_R \{0,1\}^K$
- **Common input to protocol:** A 3-colorable graph $\mathcal{G}$ with vertex set VERTICES and edge set EDGES, where VERTICES has cardinality n and EDGES has cardinality m.
- **Secret input to prover :** a 3-coloring of $\mathcal{G}$, $COL : VERS \rightarrow \{1,2,3\}$, where $COL(v)$ is the color of vertex v.

Comment: The following 10 steps are executable in 7 rounds of communication.

1. (Instructions for V)

   For $i = 1, \ldots, k$, let $(X_i, R_i) \stackrel{R}{\leftarrow} COM1(PK, 0)$, send X_i to P.
2. (Instructions for P) If $KEYVER1(Pk, 1^k) = NO$, then halt. Else, Compute $a_1, \ldots, a_k = f_s(X_1| \ldots |X_k)$ where f_s is a GGM random function with seed s and send them to V.
3. (Instructions for V)

   For $i = 1, \ldots, k$, compute $R'_i \stackrel{R}{\leftarrow} FAKE1(PK, SK, X_i, 0, R_i, 1)$.

   If $a_i = 0$, then set $D_i = R_i$, else set $D_i = R'_I$. Send D_i to P.
4. (Instructions for P)

   For $i = 1 \ldots, k$, if $VER1(PK, X_i, a_i, D_i) = NO$, then reject.
5. (Instructions for P)

   Select $key \stackrel{R}{\leftarrow} GEN2(1^k)$ and send it to V.
6. (Instructions for V) If $KEYVER2(key, 1^k) = NO$, then reject. Else, For $j = 1 \ldots, n^3$, randomly select edge $e_j = (u_j, v_j)$ in $\mathcal{G}$, compute $(ce_j, de_j) \stackrel{R}{\leftarrow} COM2(key, e_j)$, and send P the commitment values ce_j.
7. (Instructions for P) For $j = 1, \ldots, n^3$, choose π_j, a random permutation of $\{1,2,3\}$, and:

   for all $u \in VERTICES$ do: $(cu_j, du_j) \stackrel{R}{\leftarrow} COM1(PK, \pi_j(COL(u)))$ and send cu_j to V.
8. (Instructions for V) For $j = 1, \ldots, n^3$, decommit $e_j = (u_j, v_j)$ by sending e_j and de_j to P.
9. (Instructions for P) For $j = 1 \ldots, n^3$, if $VER2(key, ce_j, e_j, de_j) = NO$, then reject. Else, decommit the colors of the endpoints of e_j by sending $\pi_j(COL(u_j))$, du_j, $\pi_j(COL(v_j))$ and dv_j to P.
10. (Instructions for V)
    (a) For $j = 1, \ldots, n^3$, if $VER1(PK, cv_j, \pi_j(COL(v_j)), dv_j) = NO$ or $VER1(PK, cu_j, \pi_j(COL(u_j)), du_j) = NO$, then reject.
    (b) For $j = 1, \ldots, n^3$, if $\pi_j(COL(u_j)) = \pi_j(COL(v_j))$ (where $e_j = (u_j, v_j)$), then reject.
    (c) Else, accept.

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

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

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

[7] 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.

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

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

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

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

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

[17] 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.

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

[19] 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`.

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

[21] 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*.

[22] 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.

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

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

[25] 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.

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

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

[28] 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).

[29] 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.

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

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

[32] 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.

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

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

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

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

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

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

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

Part III
Appendices

These appendices are reproduced from an 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 $q \stackrel{\rm def}{=} (p-1)/2$, 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 10 (standard commitment scheme): *A* standard commitment scheme *is a probabilistic polynomial-time algorithm, denoted C satisfying:*

(Computational) Secrecy: *For every v, u of equal* $\text{poly}(n)$*-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* $\text{poly}(n)$*-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 $f : D \mapsto D$ with a hard-core predicate b (cf., [24]), 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 11 (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* $\text{poly}(n)$*-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* $s \in \{0,1\}^{\text{poly}(n,|v|)}$, *and computing a polynomial-time function* $S'(1^n, v, s, \texttt{msg})$, *where* $\texttt{msg}$ *is the receiver's message. A cheating sender tries, given a receiver message* $\texttt{msg}$, *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., [22]). 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 $q \stackrel{\text{def}}{=} (p-1)/2$ 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[44]), 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 9 can be easily formulated. The strong computational binding feature is formulated by extending the Computational Binding Property (of Def. 11) 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 $(\texttt{msg}, \text{aux})$ in the range of $\overline{R}(1^n)$, every $v, u \in \{0,1\}^{\text{poly}(n)}$, and every $s \in \{0,1\}^{\text{poly}(n,|v|)}$, satisfies

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

That is, $a = A(\text{aux}, (v, s), u)$ 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 $S'(1^n, 0^{\text{poly}(n)}, s, \texttt{msg})$) so that later, with knowledge of aux, one can decommit to any value u of its choice (by computing $a = A(\text{aux}, (0^{\text{poly}(n)}, s), u)$). 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 [19].

[44] 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

B.1.1 Preliminaries

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. Then $R(x) \stackrel{\rm def}{=} \{s : (x,s) \in R\}$ and $L_R \stackrel{\rm def}{=} \{x : \exists s \text{ s.t. } (x,s) \in R\}$. 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 12 (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).)

B.1.2 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 13 (System of proofs of knowledge): *Let R be a binary relation, and $\kappa : \mathsf{N} \to [0,1]$. We say that an interactive machine V is a* knowledge verifier for the relation R *with* knowledge error κ *if the following two conditions hold.*

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

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

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

The oracle machine K is called a universal knowledge extractor.

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

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 14 (Basic proof system for HC):

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

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

Proposition 15 *The protocol which results by k parallel repetitions of Construction 14 is a proof of knowledge of Hamiltonicity with knowledge error 2^{-k}. Furthermore if, for every positive polynomial p, the commitment scheme used in Step (P1) maintain secrecy with respect to circuits of size $p(n) \cdot 2^{3k}$ and distinguishing gap of $2^{-3k}/p(n)$ then, for every positive polynomial q, the interaction can be simulated in time $\text{poly}(n) \cdot 2^k$ so that no circuit of size $q(n) \cdot 2^{2k}$ can distinguish the simulation from the real interaction with gap of $2^{-2k}/q(n)$ or more.*