Interleaved Zero-Knowledge
(A Preliminary Version)

Oded Goldreich* Shafi Goldwasser† Silvio Micali‡

July 8, 1999

Abstract

We introduce the notion of Interleaved Zero-Knowledge (iZK), a new security measure for cryptographic protocols which strengthens the classical notion of zero-knowledge, in a way suitable for multiple concurrent executions in an asynchronous environment like the internet. We prove that iZK protocols are *robust*: they are "parallelizable", and preserve security when run concurrently in a fully asynchronous network. Furthermore, this holds even if the prover's random-pads in all these concurrent invocations are identical. Thus, iZK protocols are ideal for smart-cards and other devices which cannot reliably toss coins on-line nor keep state between invocations.

Under general complexity asumptions (which hold in particular if the Discrete Logarithm Problem is hard), we construct iZK (computationally-sound) interactive proofs for all NP languages which run in constant-rounds. The protocols are in the *public key model:* the verifier is assumed to have a public key associated with it. This implies, concurrent constant-round zero-knowledge computationally-sound proofs for NP in the public key model, without resorting to any timing assumptions.

Analogously, we define Interleaved Witness-Indistinguishable (iWI), protocols which are witness indistiguishable even if the prover's random-pads in all concurrent executions are identical. Under general complexity assumptions we construct Interleaved Witness-Indistinguishable (iWI) interactive proofs for all NP languages which run in constant-rounds. These interactive proofs do not require any public keys, and make no assumptions about the prover computational ability.

We extend iZK (and iWI) interactive proofs to iZK (and iWI) proofs of identity: These are methods to prove identity that remain secure even if the prover can be forced to repeatedly run the identification protocol on the same coins. All previous ZK or WI proofs of identity were totally breakable in such case. In particular, this case arises whenever the prover is realized by means of a device which can be RESET to initial conditions, such as a "smart card". Here, our protocols call for the verifier of identity (but not the prover) to have an associated public key.

Keywords: Zero-Knowledge, Concurrent Zero-Knowledge, Smart Cards, ID Schemes, commitment schemes, DLP

*Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, ISRAEL, `oded@wisdom.weizmann.ac.il`

†Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA02139, `shafi@theory.lcs.mit.edu`

‡Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA02139, `silvio@theory.lcs.mit.edu`

[1]A subset of this work is included in patent application [21].

1 Introduction

The notion of zero-knowledge interactive proofs, was put forward and first instantiated by Goldwasser, Micali and Rackoff [22] in '85, before the onset of the internet. By now, zero-knowledge is the accepted way to define and prove security of various cryptographic tasks. Its generality was demonstrated by Goldreich, Micali and Wigderson [18], who showed that any NP-statement can be proven in zero-knowledge, provided commitment schemes exist (or, equivalently [26, 24], one-way functions exist). An important application of zero-knowledge proposed by Fiat and Shamir [11] was proving identity.

Alongside many applications, the notion of zero knowledge raises a few important questions.

Parallel composition. The first question is whether zero knowledge is preserved when (zero-knowledge) protocols are composed. For sequential composition of protocols, the question was answered in the affirmative (cf., [19]), provided zero knowledge is augmented in a natural manner (satisfied by all natural examples).[2] The situation with respect to *parallel composition,* however, turned out to be more complicated: There exist (unnatural) zero-knowledge protocols which yield knowledge if executed twice in parallel [16]. Furthermore, parallel executions of some natural zero-knowledge interactive proofs (like the abovementioned proof for NP by [18]) cannot be proven zero-knowledge using a black-box simulator (the only way known) unless $\mathcal{NP} \subseteq \mathcal{BPP}$ [16].

An alternative avenue toward the parallel composition problem was taken by the work on witness indistinguishability of [10]. In a sense their conclusion is that the zero-knowledge requirement is too strict and that one should suggest (as they do) weaker notions which are both preserved under parallel composition and suffice for (hopefully) many applications.

Concurrent execution. With the rise of the internet, the question 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 each prover–verifier pair should act independently. (It would be unrealistic to require all interactive parties to coordinate their actions such that zero-knowledge is preserved!) Note that maintaining zero-knowledge in the concurrent setting is even harder than preserving zero-knowledge under parallel composition, as all executions need not start at the same time nor remain totally synchronized.

A recent approach towards addressing the concurrent execution problem has been suggested by Dwork, Naor and Sahai [8], who assume that a very mild level of synchronization is guaranteed: the so-called *timing assumption.* Under this assumption, protocols may not start at the same time, but once an execution starts there are a-priori known bounds on the delays of messages with respect to some ideal global clock. Furthermore, it is assumed that 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 zero-knowledge (computationally-sound) proofs for NP were presented in [8]. In a later paper by Dwork and Sahai[9], it was shown how the use of timing can be pushed up to a pre-processing protocol, to be executed before the concurrent executions of protocols (which do not use timing), and still get constant-round zero-knowledge computationally-sound proofs for NP.

[2] The augmentation is indeed necessary; see [16].

More recent work by Ransom and Kilian [27] seems to indicate that it may be possible to get rid of the timing assumption, alas their protocols are either not constant-round or only simulatable in quasi-polynomial time.

Resettability. It is known that zero-knowledge proof in which the prover is deterministic exist only for BPP languages (cf., [19]). Furthermore, (standard) Zero-Knowledge protocols do not necessarily remain zero-knowledge if the prover uses the same sequence of coins in many interactions (even if the interactions are with a honest verifier). In fact, a wide class of ZK protocols (e.g., Zero-Knowledge proof for 3-Coloring [18] or proofs of identification a la Fiat-Shamir [11]), are trivially breakable in such setting. Furthermore, by definition in any proof of knowledge protocol, if the verifier can reset the prover to the same initial conditions (i.e., to use the same coins) for a polynomial number of executions, then the verifier can easily extract the very same secret key which the prover is claiming knowledge of. (For the original protocol of [11] it suffices to repeat the protocol twice with the same prover coins to be able to extract the prover's secret).

This 'feature' restricts the physical implementations of the prover to those of essentially fully-reliable hardware, where it is impossible for the verifier to "reset" the prover. For instance, assume that the prover is a smart card which has neither a built-in power supply nor a (tamper-proof) non-volatile writable memory. Then, one could (e.g., by disconnecting and reconnecting the power supply) restore a fixed internal configuration of the card, thus essentially cloning it any time one pleases. (In the unlikely case that the card generates a random-pad by sampling the environment, one can possibly determine the random-pad.) Thus, being zero-knowledge in the classical sense does not provide security against such an attack on the card. An interesting question is whether it is possible to have zero-knowledge protocols even when the prover is realized by a resettable device.

1.1 A New Notion: Interleaved Zero Knowledge

This work puts forward the notion of Interleaved Zero-Knowledge (iZK), a new security measure for cryptographic protocols which strengthens the classical notion of zero knowledge [22], to fit todays "internet age".

Informally, an interactive proof or a computationally-sound proof is iZK, if a verifier learns nothing (except for the verity of a given statement) even when he can make the prover interact with him polynomially-many times, *while the prover uses the same sequence of random moves in all these interactions.* In other words, a polynomial-time verifier learns nothing extra 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. (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.

The above definition may seem unnatural; however, it is a powerful abstraction which yields results for a variety of realistic settings. This makes our positive results only stronger. In particular, our notion enjoys the following properties:

- iZK protocols are closed under parallel composition;
- iZK is preserved even when the protocol is executed polynomially-many times in a totally asynchronous network (and furthermore the number of executions may be a-priori unknown);

- iZK allows the prover's random-pad to be randomly selected and fixed, once and for all: The resulting deterministic prover can prove in zero-knowledge polynomial many claims without keeping record of any kind, and again the number of interaction need not be a-priori known.

The importance of the first two properties has been recognized in the litureature and requires no further emphasis. We now discuss the importance of the third property, and specifically its relevance to some realistic settings. Randomly selecting and fixing the prover's random coins is advantageous in settings where the proving device can be physically tampered with. In particular, when one can effect on-line random choices made by the device and reset all its registers to their initial state. A iZK prover (with few random bits that can be wired in) maintains zero-knowledge also under such an attack, since it uses no on-line random choices and resetting it is guaranteed (by definition) to cause no harm.

In general, iZK enables enlarging the number of physical ways in which ZK proofs may be implemented while guaranteeing that security is preserved. Previous notions of zero-knowledge (and identity proving schemes in particular) worked only for a *restricted class of physical implementations of the prover*; typically, implementations over fully-reliable hardware. Note that for example a smart-card (playing the role of a prover of identity say in an interactive proof) cannot be considered such a device as we discussed in the previous subsection. In contrast, if a smart card implements an iZK proof and we hard-wire a (short) random-pad into its program then the card's security is maintained even under such an attack.

The Public Key Model. We will consider protocols which achieve iZK in the public-key model. Namely, we assume that participants are part of a network, where parties can register a public key which can be read by all other participants. 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 registrated public-key (e.g., that it valid or even that the user registrating it knows a corresponding secret key).

For sake of simplicity, we assume throughout the paper that registration occurs before any interaction between the users takes place. More flexible models allowing registration at all times, seem to require a mild notion of timing (or synchronization), and are deferred to a future version of this work.

We note that in the iZK computationally-sound proofs for NP which we construct only one of the participants of the two-party protocol – the verifier – is required to have an associated public-key. The fact that only the verifier is required to have a public-key is crucial for using iZK protocols in the context of identity proving protocols, where the prover does not necessarily have a registered public key but the verifier which is typically a server governing use of a resource (e.g., a computer or a data base) does have a pubic key.

1.2 Interleaved Witness-Indistinguishable

The notion of witness indistinguishability was introduced by [10] 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, witness indistinguishable protocols are closed under parallel composition and concurrent execution.

We define (analogously to iZK) Interleaved Witness-Indistinguishable protocols (iWI). Informally, a witness indistinguisable proof is iWI, if it remains witness indistinguishable even if the verifier can make the prover interact with him polynomially-many times, *while the prover uses the* *same sequence of random moves in all these interactions.* In other words, a polynomial-time verifier can still not distinguish between two different witnesses for an NP statement that the prover is proving 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.

We note that existing WI protocols (e.g., [18]) are not Interleaved Witness-Indistinguishable protocols. Furthermore, even the honest verifier can easily extract the entire witness (not to mention distinguish between witnesses), when the protocol is executed polynomial many times with a prover using the same coins.

Interestingly, as we shall see in an upcoming subsection, we can achieve iWI interactive proofs (rather than computationally-sound ones) without requiring neither prover or verifier to have a public key. This will be under essentially the same complexity assumptions as required for the iZK proofs in the public-key model.

1.3 Public Key Model iZK

The advantages of the new notion are clear, but do iZK protocols exist for languages outside BPP? Our main result is answering this question affirmatively, under reasonable complexity assumptions, in the public key model. We prove

Main Theorem 1 (informal statement): *Any language in NP has a constant-round iZK computationally-sound interactive proof in the public key model, under the assumption that trapdoor strong claw-free pairs of permutations exist.*

By strong claw-free pairs of permutations we mean that the clawfree property should hold also with respect to subexponential-size circuits (i.e., circuits of size 2^{n^ϵ}, where n is the input length and $\epsilon > 0$ is fixed), rather than only with respect to polynomial-size circuits. By trapdoor we mean that these pairs can be generated along with auxiliary information which allows to form (random) claws.

The existence of trapdoor strong claw-free pairs of permutations is, in particular, implied by the computational intractability of the Discrete-Log Problem (DLP). More generally, our result holds if constant-round commitment schemes with certain features exist.

Recall that zero-knowledge protocols (for NP) are extremely powerful tools with numerous applications in cryptography. This holds both if these protocols are absolutely sound or just computationally-sound (a.k.a arguments). The above says that this powerful tool is available, at moderate price (i.e., constant-round), in the fully concurrent setting of asynchronous networks (e.g., the internet) assuming verifiers have established public keys using the public-keys infrastructure. Moreover, this tool is available even when the prover is realized by a device which can potentially be reset to its initial state (coin tosses included).

Techniques. We believe some of our technique will prove useful in other applications and in particular ones related to security in the internet. These techniques include:

1. Replacing random choices by application of a pseudorandom function (cf., [14]) on the history of the interaction. This technique is at the heart of the "cloning-robustness" property of our zero-knowledge protocols. It enables to move the randomness of a party from the on-line interaction to a prior off-line stage.

2. Proving that such "cloning-robustness" (for sequential executions) guarantees security in interleaved executions, and thus security in an asynchronous concurrent setting.

3. Using public-keys as a tool to achieving zero-knowledge protocols. It is customary to use public-keys for encryption and digital signatures. However, here the fact that the verifier uses a registered public key in concurrent executions of the protocol enables achieving ZK in a concurrent and resettable setting. What is important is that the number of total public-keys ever utilized by verifiers (including multiple verifiers in multiple executions) is bounded by a fixed polynomial in the security parameter.

4. "Telescopic" usage of intractability assumptions. The idea is to use two "secure" schemes, one with security parameter K and one with a smaller security parameter k. Suppose that, for some $\epsilon > 0$, the security of the first scheme (with security parameter K) is maintained aganist 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-mallability" 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.

Resolving the zero-knowledge concurrency question in the public key model. As should be clear from the above discussion, our Main Theorem 1 is directly relevant to *the open problem of concurrent zero-knowledge*: We provide constant-round concurrent zero-knowledge protocols for NP, based on standard intractability assumptions in the public key model. As the existence of a public-key may be thought of as a mild form of preprocessing, this directly improves Dwork and Sahai's work that required a pre-processing protocol which uses the timing technique and thus makes timing assumptions to achieve concurrent computationally-sound proofs for NP. In our case, no timing assumptions are necessary, just the plain old public key model. Moreover, the notion of concurrent zero-knowledge achieved is a stronger notion than that in previous works as it allows concurrent executions with a prover who may use the same coins.

1.4 Constructing iWI Protocols for NP

Without resorting to the public key infrastructure, we show how to achieve Interleaved Witness-Indistinguishable interactive proofs for all NP statements. Note that here we can achieve interactive proofs rather than computationally sound proofs. Namely, no bound need be assumed about the computational power of the cheating prover. We prove

Main Theorem 2 (informal statement): *Under the assumption that claw-free pairs of permutations exist, every language in $\mathcal{NP}$ has a* constant-round *rewindable witness-indistinguishable interactive proof system.*

The assumption in Main Theorem 2 is seemingly weaker than in Main Theorem 1: Ordinary clawfree pairs will do, rather than strong clawfree pairs with trapdoor. Again, the assumption in the theorem can be alternatively stated in terms of the existence of constant-round perfect commitment schemes.

1.5 Other Results

iZK vs. proofs of knowledge. We show that proofs of knowledge cannot be iZK, except in a useless sense: That is, we show that if, on input x, one can provide an iZK 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. (This holds even for the weaker model presented in Section 3.)

iZK proof of identity. The fact that we cannot have (useful) iZK proofs of knowledge blocks the applicability – to the "prover cloning" model – of a known paradigm for constructing identification schemes. We refer to the Fiat-Shamir paradigm [11] by which any zero-knowledge proof of knowledge (for a "hard" NP-language) yields a secure identification scheme. Furthermore, the Fiat-Shamir identification scheme [11] (which is based on a zero-knowledge proof of knowledge of a square root of a given number modulo a given composite [22]), as well as any identification scheme based on the above paradigm, can be totally broken if one can force the identifying user to re-run the protocol on the same choice of coins several times. As discussed above, such an attack may be feasible whenever the identification is done by a device which uses hard-wired coins and may be reset to its initial state.

Instead, we propose an alternative paradigm for constructing identification schemes so that the resulting schemes are secure also when the identification is done by a device which uses hard-wired coins and may be reset to its initial state. Furthermore, the scheme remains secure even if the adversary can get hold of several copies of the same device (i.e., "clones") and interact with all copies while possibly interleaving the executions.

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. The legitimate user holds an NP-witness which allows it to successfully carry out such proofs, whereas it is infeasible for an adversary which is only given a YES-instance of the hard language to successfully carry out such a proof.[3] Using a iZK proof system, the identifying user can always convince the verifier while not yielding any information which may facilitate latter impersonating attempts (by the current verifier). This holds even if the verifier can attack the device used by the identifying user as described above.

2 Preliminaries

2.1 Standard Conventions

Throughout this paper we consider interactive proof systems [22] 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*).

[3] Note that an efficient strategy to convince the verifier on YES-instances yields an efficient decision procedure for the language (by emulating the interaction).

Likewise, when we talk of computationally-sound proof systems (a.k.a arguments) [7] we mean ones with perfect completeness in which it is infeasible to cheat with non-negligible probability. Specifically, for every polynomial p and all sufficiently large inputs x not in the language, every circuit of size $p(|x|)$ (representing a cheating prover strategy) may convince the verifier to accept only with probability less than $1/p(|x|)$.

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. Actually, we are most 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 [22]: 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 stress that throughout this paper, a *probabilistic polynomial-time machine* means a probabilistic machine running in expected polynomial-time (rather than strict 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 [15, 13].

We also refer (or, actually, extend) the definition of witness indistinguishable proof systems (cf., [10]). 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.

2.2 The models considered

In this paper we consider a variety of models. These models are defined by two parameters specifying the *initial set-up assumptions* and the generality of the *adversary attack*.

Initial set-up assumptions: The vanilla case 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. Whenever we make no mention of set-up assumptions/model, we mean the vanilla model.

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. This (and even more imposing variants) is 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 our work actually uses weaker assumptions. Specifically, in both the latter models, we only need that potential verifier will deposit public-keys and/or participate in a pre-computation. This is not required of users who are only going to play the role of provers.

The attacks: The most basic attack we consider allows the adversary to rewind any of the prior completed interactions with the prover to an arbitrary point and carry out a new interaction from that point. Once the adversary initiates a new interaction, it must complete it (or abandon it), and so interleaving of interactions is not allowed. The basic model is considered in Section 3, whereas a model allowing arbitrary interleaving of interactions is considered in Section 4. However, in both sections we consider only a single incarnation of the prover; all interactions with it are on a fixed input and fixed random-tape. In Section 5 we generalize the treatment to the case the adversary can interact (i.e., rewind and interleave) with polynomially many different incarnations of the prover.

The tasks: Our main focus is on zero-knowledge protocols. However, we also consider other variants such as witness-indistinguishable protocols and identification protocols.

3 Rewindable Zero-Knowledge

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 $P_{x,y,\omega}$ engange 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 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 (rewindable security – vanilla model): *A prover strategy P is said to be* rewindable 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., [20, 28])*:*

`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 $P'(\text{msg}_{it+1}, ..., \text{msg}_{it+j})$, where 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 $M^*(x)$.*

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^* (cf., [19]), *we say that P is* rewindable zero-knowledge via a black-box simulation.

A prover strategy P is said to be rewindable witness-indistinguishable (on L) *if every two distribution ensembles of Type 1, each indexed by a common input $x \in L$ and depending on a possibly different prover auxiliary input y, are computationally indistinguishable.*

We note that many known zero-knowledge protocols are NOT rewindable zero-knowledge. Furthermore, they are even NOT rewindable witness-indistinguishable. For example, ability to "rewind" the original zero-knowledge proof for 3-Colorability [18], allows the adversary to fully recover the 3-coloring of the input graph used by the prover. Still (as shown below), rewindable witness-indistinguishable interactive proofs for $\mathcal{NP}$ exists, provided that the Discrete Logarithm Problem (DLP) is hard modulo primes p of the form $2q+1$ where q is a prime. Actually, a more general result holds

Theorem 2 *If constant-round perfect commitment schemes exists then every language in $\mathcal{NP}$ has a* constant-round *rewindable witness-indistinguishable interactive proof system.*

Analogously to Definition 1, we may define rewindable 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.

In the public-key model, we may obtain computationally-sound rewindable 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 3 *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 *rewindable zero-knowledge computationally-sound proof system in the public-key model. Furthermore, the prescribed prover is rewindable zero-knowledge via a black-box simulation.*

3.1 Proof Sketch of Theorem 2

Traditional zero-knowledge interactive proofs rely on the randomized nature of the prover strategy. In a sense, this is essential (cf., [19]). 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 [14].[4] 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 [18] 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 [15]: 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 [15] 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 perfect 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.

(P1) As in [18, 15], the prover commits to t random relabeling of colors. The commitment is done using an ordinary commitment scheme, providing computational-secrecy and absolute/perfect binding. The key point is that the prover's random choices (both for the relabelling 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, to allow smooth extension to the general model discussed in Section 5, we apply f to the pair $((G, \phi), \text{msg})$, where msg denotes the message sent by the verifier in Step (V1). That is, let $(\pi_1, r_1), ..., (\pi_t, r_t) = f(G, \phi, \text{msg})$, 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 $\phi_i(v)$.

[4]Recall, that by combining [24] and [14] 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 [5] and [14].

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

There is one problem, however, with the above presentation. In Step (V1) we have assumed the existence of a 1-round (i.e., uni-directional communication) perfect commitment scheme. However, any commitment scheme with perfect secrecy which is computational-binding require 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).[5] Thus, we need to integrate such (two-round) commitment schemes in the above protocol.

It appears as if the above protocol is rewindable 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 interaction and decommit properly in a later interaction in which it has sent the same message in Step (V1). Doing so will harm the straightforward simulation attempt, which proceeds interaction-by-interaction so that in each interaction 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 interaction) properly. The problem is that we cannot answer the same Step (V1) message, sent in two interactions, in two different ways (in the two interactions), since the prover would answer identically in the real execution.[6] (We comment that if the verifier always decommits in Step (V2) then we can simulate polynomially many interactions with the above prover. That is, the above protocol is rewindable zero-knowledge with respect to verifiers which always decommit properly in Step (V2). For details see Appendix B.)

Fortunately, the subtle problem mentioned above has much milder effect on the proof that the above protocol is witness-indistinguishable. Specifically, as a mental experiment, we first consider an ideal prover that uses a truely random function rather than a pseudorandom one. 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 relabelling 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 interactions (with an adversary V^*) in which one possible witness coloring ϕ is used are computationally indistinguishable from such interactions in which another witness coloring ϕ' is used. This is proved using a hybrid argument, where the i^{th} hybrid is defined as follows: In each of the i first iterations, the prover uses ϕ when executing Step (P1), and this determines also its action in Step (P2). For $j > i$, if in the j^{th} iteration the message sent in Step (V1) is identical to one sent in iteration j' (for some $j' < j$) then the prover repeats the corresponding Step (P1) message. (In particular, this happens in case $j' \leq i$,

[5] Here, as in all work on zero-knowledge, with the exception of a fully-uniform treatment (cf. [12]), the computational condition refers to non-uniform adversaries. The reason being that the standard zero-knowledge condition is itself somewhat non-uniform (as it refers to any input). Inspecting Definition 14 (in Appendix A), it is evident that no 1-round scheme may satisfy it.

[6] Specifically, suppose that the simulator always tries first to send a dummy messsage in Step (P1), and consider two consequetive interactions with a cheating verifier. In the first interaction, the verifiers commits to some edge sequence in Step (V1) but refuses to decommit in Step (V2). The simulator will thus produce a truncated interaction (which, by itself, is fine). Now suppose the verifier repeats the same Step (V1) message in the second interaction, 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 interaction. 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. This problem is further discussed and resolved in the next subsection.

which means that in this case the Step (P1) message will also be a commitment to ϕ.) Otherwise, Step (P1) message is computed as a commitment to ϕ'. Using another standard trick (cf., [18, pp. 719–721]), we prove Theorem 2. We stress that the argument uses in an essential way the key observation above: If the i^{th} iteration utilizes a new Step (V1) message then the commitment generated in response is independent of the prior iterations.

3.2 Proof Sketch of Theorem 3

We first present a rewindable 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 [13, 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 clawfree property holds w.r.t subexponential circuits.[7]

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 proof system of the previous subsection, 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 indentity; we merely rely on the fact that somebody knows the trapdoor to the index i if indeed it is in the public file. Since we view the adversary as controlling the entire "world outside the prover" it really does not matter who knows the trapdoor.]

(2) This step is analogous to Step (V1) in the protocol of 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

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

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 paramter k, is insufficient to violate the strong computational-binding property of PC1 when run on security paramter 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 relabelling 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., [22, 18]), 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 interation 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 noticeablly different from the output in real executions. Using the computational-binding property of the scheme PC2 (as done in Appendix B), we argue that this event may only occur with negligible probability. This establishes the rewindable 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 paramter 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 "ambigious 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-ambigious. This establishes the computationally-soundness of the above protocol.

Back to the bare public-key model (i.e., without preprocessing): 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 $(i, \mathrm{trap}(i))$).

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

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 $\mathrm{trap}(i)$.) 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 retreive $\mathrm{trap}(i)$.

The proof of knowledge (POK) of Hamiltonicity is based on Blum's proof system for this language, which is reproduced in Appendix C. 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).[8] 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[9] (i.e., error of 2^{-k}), and will be simulatable in time $\mathrm{poly}(K) \cdot 2^k$. 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 $\mathrm{poly}(K)$-time).

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

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

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

[8] This property holds also for other protocols for NP, but not for the 3-Colorability protocol of [18]. Any protocol having the property will do.

[9] 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 C.

(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.)

Again, 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., [22, 18]), 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 rewindable zero-knowledge property of the entire protocol. As a first step towards this goal, we establish that the above subprotocol is indeed a POK with knowledge-error 2^{-k} (see Def. 17 in Appendix C). In other words, we analyze a single execution of the subprotocol, and thus we may assume that Step (**pok2**) is replaced by sending a truely random string c. This assumption is not valid when the subprotocol 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 challanges* we can easily reconstruct the Hamiltonian cycle (and from it the trapdoor).[10] To extract the Hamiltonian cycle we just invoke the knowledge-prover many times, each time it answers with the same Step (**pok1**) message but then we challenge it with a new randomly chosen c (i.e., chosen independently of all prior attempts). If we ever obtain its answer to two successful challenges then we are done. Denoting by p the probability that a uniformly chosen challenge is succesful, we conclude that if $p > 2^{-k}$ then given oracle access to the knowledge-prover (played by the adversary) we can (with overwhelmingly high probability) find the trapdoor in time $\text{poly}(k)/(p - 2^{-k})$. By a trivial modification, we obtain a knowledge extractor which for any $p > 0$ with overwhelming

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

probability runs for time $\mathrm{poly}(k)/p$, and in case $p > 2^{-k}$ also retreives the trapdoor.[11]

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 rewindable 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 annoncing 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 indistingushiable 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 succeeeds. 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.[12] This need to "rewind" interactions which were already completed may lead to exponential blow-ups as discussed by Dwork, Naor and Sahai [8]. What saves us here is that the number of times we possibly need to "rewind" is a-priori bounded by the total number of indices in the public file. (This is the key and only place where we use the assumption underlying the public-key model.)

Resolving the problem – a sketch: Let us reproduce and further abstract the problem we need to analyze. 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.

[11] 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 $(p - 2^{-k})^{-1} \leq 2/p$.

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

(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, $f', f'', \ldots$, and the rule is that whenever we have two different successes (i.e., with two different challenges c) for the same pair (i, C) we can complete the extra steps referred to in Step (c). [This corresponds to extracting the trapdoor of i, which allows the simulation of the rest of the steps in the current interaction of the main protocol.]

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

A more precise description is provided in Figures 1 and 2, and the actual analysis presented below refers to this formal description. The main procedure (of Figure 1) attempts $|I|+1$ times to generate a full transcript, while constructing the random function, f, on the fly. Typically, each attempt which fails to generate a full transcript provides "progress" in the form of a new element being covered. The non-typicaly case, which (as we will show) occurs with negligible probability, is that neither happens. Another thing to be proven is that the expected number of times that the main procedure repeats (M8)–(M10) is inversely proportion to the probability that for uniformly chosen $r \in \{0,1\}^k$ it holds that $V^*(H', (i, C), r)$ succeeds, where H' and (i, C) are as defined in (M6). The extension of transcripts, either initial ones as in (M4) or partial ones as in (M9), is performed (in a straightforward manner) by the `Extend` procedure depicted in Figure 2. In particular, once `Extend` "gets into trouble" (reaches a success w.r.t (i', C') where i' is uncovered) it returns control to the main procedure. In case `Extend` is invoked in (M4), we next try to get i covered. In case `Extend` is invoked in (M9), if $(i', C') = (i, C)$ then we obtain a different success to the one obtained already, and consequently i gets covered.

Proposition 4 (Analysis of the main procedure): *We consider a single execution of the outer loop in the main procedure.*

1. *The total expected time spent in such an execution is* $\text{poly}(k)$*.*

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

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

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

Figure 1: The main simulation procedure

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

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

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

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

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

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

$\texttt{Extend}^f_{V^*}(U, H)$
(E1) Repeat till $V^*(H)$ halts
(E2) $(i', C') \leftarrow V^*(H)$ (assuming $V^*(H)$ does not halt).
(E3) If f is not defined on (i', C') then select r' uniformly in $\{0,1\}^k$ and let $f(i', C') \leftarrow r'$.
(E4) If $V^*(H, (i', C'), f(i', C'))$ fails then $H \leftarrow (H, (i', C'), f(i', C'), \perp)$ and goto (E1).
[Comment: Otherwise $V^*(H, (i', C'), f(i', C'))$ succeeds.]
(E5) If i' is covered (i.e., $i' \in I \setminus U$) then complete H as in Step (c) and goto (E1).
[Comment: Otherwise i' is not covered, and we return a partial transcript.]
(E6) $\texttt{return}(H, (i', C'), f'(i', C'))$.
[Comment: If $(i', C') = (i, C)$ we retrun a transcript contating a success w.r.t (i, C).]
(E7) End [of repeat loop]
[Comment: Reaching this point means completion of simulation.]
(E8) $\texttt{return}(H)$.

Figure 2: The Extend procedure

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

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

and Part 1 of the proposition follows.

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

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

and Part 2 of the proposition follows. ■

3.3 Comments

Using a perfect commitment scheme which enjoys the trapdoor feature (but not necessarily the strong computational-binding feature), one may obtain rewindable 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 subprotocol) rather that parallel repetitions. That is, both Steps (2)–(6) of the main protocol and the POK subprotocol 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 [18]) 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 rewindable 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 5 *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* rewindable 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 rewindable 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.

We mention that the idea of applying a pseudorandom function to the verifier's message may be used to derive alternative schemes secure in the rewindable sense. For example, starting with a non-interactive zero-knowledge proof system (cf., [4])[13], we may obtain an alternative rewindable witness-indistinguishable proof system for NP (establishing Theorem 2) as follows. The idea is to employ "coin tossing into the well" (cf., [2]), but with a small twist: We let the verifier commit 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).

3.4 An alternative presentation of rewindable zero-knowledge systems

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

[13]The basic version (of non-interactive zero-knowledge) allowing for a proof of a single assertion relative to one reference-string suffices for our application.

4 Interleaved Zero-Knowledge

We now show that the restriction on the interaction of V^* with copies of the prover made in Definition 1 is inessential. That is, allowing the adversary to interleave these interactions (rather than conduct them in a sequential manner) does not add to its power. This result is very important since it places rewindable (or interleaved) zero-knowledge quite close to concurrent zero-knowledge. The extra step is taken in the next section, but first we formally define the interleaving model and prove the above stated result.

In the actual definition we introduce a small technicality, which we motivate now. In the model of Definition 1, the adversary does not need to specify which interaction with the prover it wishes to continue next, since at any point in time only one interaction is active. In the model below, several interactions may be active (i.e., not completed) concurrently, and hence such an indication is necessary. The simplest way of addressing this technicality is to modify the original interactive proof so that each party prepends its 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.[14] 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 input to B). However, in the context of interleaved zero-knowledge (or even in rewindable zero-knowledge) this 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 interleaved (resp., rewindable) zero-knowledge model. In the following definition, we assume that P is a machine resulting from the modification above. We start again with the vanilla version.

Definition 6 (interleaved zero-knowledge – vanilla model): *A prover strategy P is said to be* interleaved 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 common input $x \in L$ and prover's auxiliary input y, are computationally indistinguishable:*

[14] 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.

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

1. *As in Definition 1: Randomly select and fix a random-tape, ω, for P, resulting in a deterministic strategy $P' = P_{x,y,\omega}$ defined by $P_{x,y,\omega}(\alpha) = P(x, y, \omega; \alpha)$.*
2. *Machine V^* is allowed to initiate polynomially-many* interleaved *interactions with P'. Formally speaking, we may allow V^* to send arbitrary messages to P' and obtain the response of P' to any such message.*[15]
3. *As in Definition 1: Once V^* decides it is done interacting with P', it* (i.e., V^*) *produces an output based on its view of these interactions.*

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

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 interleaved zero-knowledge via a black-box simulation.

Note that once one adopts the above convention, the definition of the interleaving model is simpler than the definition of the rewindable model (i.e., Definition 1). Analogously, we may define interleaved zero-knowledge in the public-key model as well interleaved witness-indistinguishable in both models.

By the above convention, we may assume without loss of generality that V^* only sends messages which result by augmenting some message (i.e., a sequence of strings) received before by a string of its choice. That is, V^* is allowed to send the message $(e_1, ..., e_i, e_{i+1})$ only if it has received the message $(e_1, ..., e_i)$ before. This claim is justified by the following modification. Assume, without loss of generality, that the verifier V^* moves in odd rounds, and that it currently wishes to send the message $\text{msg} \stackrel{\text{def}}{=} (e_1, ..., e_{2i}, e_{2i+1})$. Then, the modified verifier, denoted V^{**}, proceeds in iterations as follows: For $j = 1, ..., i$, in the j^{th} iteration V^{**} sends the message $(e_1, ..., e_{2j-1})$, and continues to the next iteration only if the response equals $(e_1, ..., e_{2j})$. In case all i iteration were completed successfully, V^{**} sends the message $\text{msg} = (e_1, ..., e_{2i}, e_{2i+1})$; otherwise, $(e_1, ..., e_{2i})$ is not a valid history w.r.t the prover, and V^{**} refrains from sending msg and behaves as V^* does after sending msg and receiving a prover's response equal to a special symbol (indicating detection of cheating by the verifier).

The main result of this section is that security (e.g., zero-knowledge) in the rewindable model implies security in the seemingly stronger interleaving model. For simplicity we state and prove this result for zero-knowledge in the vanilla model. It is straightfoward to modify the proof to the public-key model as well as other variants.

Theorem 7 *Suppose that P is rewindable zero-knowledge. Then P is interleaved zero-knowledge. Furthermore, simulation via black-box simulators is preserved.*

Proof Sketch: This argument too is by a transformation of one verifier strategy into another so that a more general adversay is transformed into a milder one, which nevertheless produces exactly the same output. Here we transform an adversary of the interleaving model into a more (syntaxically) restricted adversary of the rewindable model. Thus, simulability of the latter adversary implies simulability of the former, which means that the syntaxical restriction is actually immaterial.

[15]That is, V^* is given full oracle access to P'.

Let V^* be an adversary for the interleaving model, and suppose that it invokes P' (to obtain a single message) at most t times. We construct an adversary, W^*, for the sequential (rewindable) model which has output distribution identical to V^*. The adversary W^* will use V^* as a black-box. Thus, combining the black-box simulator provided for the sequential model with the operation of W^* (and providing it with black-box access to V^*), we obtain a black-box simulator for the interleaving model (which works given black-box access to V^*).

Machine W^* will run t copies of P', sequentially. (Thus, if P expect to get r messages per interaction, then W^* will send a total of $r \cdot t$ messages, where t was defined as the number of messages sent by V^*.) Firstly, W^* selects and fixes a random-pad for V^* and so make the interaction of V^* with copies of P' totally determined (as both V^* and P' are now deterministic). We next employ exactly the same transformation used above (when arguing that w.l.o.g each message of V^* is obtained by appending a string to a message it has received below), stressing the relevant fact that it yields an adversary in the rewindable model. Specifically, we assume without loss of generality, that whenever V^* sends a message to a copy of P', the response arrives immediately. Thus, all that we need to do is embed each message sent by V^* in a new interaction of W^* with a fresh copy of P'. Machine W^* sequentially reconstructs the messages exchanged between V^* and the (interleaved) copies of P' as follows.

Assuming, w.l.o.g., that the verifier takes odd moves in interaction with P'. For $k = 1, ..., t$, suppose that the k^{th} message sent by V^* is $\text{msg} \stackrel{\text{def}}{=} (e_1, ..., e_{2i}, e_{2i+1})$. Then, W^* initiates a new interaction (i.e., the k^{th} one) with P' and proceeds in it as follows: For $j = 1, ..., i$, in the j^{th} step, W^* sends the message $(e_1, ..., e_{2j-1})$ to P', and continues to the next step only if the response of P' equals $(e_1, ..., e_{2j})$. (The condition is violated only when V^* does not satisfy the second convention. By the above transformation we may assume that V^* always satisfies the latter.) In case all i steps were completed successfully, W^* sends the message $\text{msg} = (e_1, ..., e_{2i}, e_{2i+1})$, and obtains the response of P'. (Again, we may ignore the case in which these steps were not completed successfully, which occurs when $(e_1, ..., e_{2i})$ is not a valid history w.r.t the prover. In such a case W^* refrains from sending msg and behaves as V^* does after sending msg and receiving a "cheating detected" response from P'.) Machine W^* records this response of P' to the message msg. (We stress that this response of P' equals its k^{th} response in the interaction with V^*.) Finally, W^* aborts the current (i.e., k^{th}) interaction (or, actually, to fit the exact definition of the sequential model, runs this copy to termination arbitrarily). (We stress that the current copy of P' may detect that W^* is "cheating" in the subsequent steps in the current interaction, but this information does not pass to and effect future copies of P'.)

To summarize, W^* is able to conduct sequential interactions with P' as allowed in the rewindable model, and still obtain a transcript of the corresponding execution of V^* in the interleaved model. Thus, upon termination, W^* may output exactly the same value as output by V^*. ■

Comment: We note that various properties of V^* are not necessarily inherited by W^*. For example, even in case V^* is honest (i.e., merely runs a single copy of P' while behaving as the prescribed verifier), the resulting W^* is not. In general, as noted at the end of the above proof, the interaction of W^* with each copy may be terminated in a "bad" way. Specifically, applying the transformation to the construction of the previous section, we obtain an adversary W^* which does not always decommit properly (even in case the underlying V^* always decommits properly).

Comment: The ideas underlying the above proof can be employed also when the above convention is not adopted. In such a case we need an alternative convention for regulating simultaneously interactions with several copies of P'. Suppose that we adopt a convention by which each message of V^* is prepanded with an index of a copy of P', and that responses are accordingly. Suppose that the $k^{\rm th}$ message that V^* wishes to send has the form $(i, {\rm msg})$; that is, it is a message to the $i^{\rm th}$ copy of P'. Then W^* invokes a new (i.e., $k^{\rm th}$) copy of P', and interacts with it as follows. Firtst W^* sends to the $k^{\rm th}$ copy each of the previous messages sent by V^* to the $i^{\rm th}$ copy (i.e., all messages of the form $(i, \cdot)$), next W^* sends to the $k^{\rm th}$ copy the current message msg, and records the answer. Finally, W^* dismisses the $k^{\rm th}$ copy of P' (or, actually, run this copy to termination arbitrarily). Note that, again, each message of the interleaved adversary is embeded in a new interaction with a fresh copy of P' so that there is no interleaving among these copies (and so the requirements of the rewindable model are satisfied).

5 Generalization to multiple inputs

In order to relate interleaved zero-knowledge to concurrent zero-knowledge we need to somewhat extend the former model. Specifically, we need to allow the adversary to invoke polynomially-many random indepedently selected incarnations of P, rather than only one. Similarly, we need to allow the adversary to invoke incarnations of P on several inputs, rather than on a single one. We stress that in all cases, the adversary may invoke each incarnation polynomially-many times, the issue is whether there are many incarnations of P or only one. Intuitively, having many (independent!) incarnations should not add power to the adversary since communicating with the same incarnation on the very same common input seems most advantageous for the adversary (and most dangerous for the prover). Currently, we do not know if this intuition is correct (in general). Fortunately, it can be easily verified that the results of the previous two sections extend to the more general setting. We start by extending the models of the two previous sections. Again, we provide definitions and state most results only for zero-knowledge in the vanilla model, and trust the reader to infer other varaints (such as interleaved zero-knowledge in the public-key model as well interleaved witness-indistinguishable in both models).

Definition 8 (generalization to multiple inputs – vanilla model): *A prover strategy P is said to be* interleaved zero-knowledge (resp., rewindable zero-knowledge) *on L* (in the general model) *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 sequence of common inputs $x_1, ..., x_{{\rm poly}(n)} \in L \cap \{0,1\}^n$ and a corresponding sequence of prover's auxiliary-inputs $y_1, ..., y_{{\rm poly}(n)}$, are computational indistinguishable:*

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

1. *Randomly select and fix $t = {\rm poly}(n)$ random-tape, $\omega_1, ..., \omega_t$, for P, resulting in deterministic strategies $P^{(i,j)} = P_{x_i, y_i, \omega_j}$ defined by $P_{x_i, y_i, omega_j}(\alpha) = P(x_i, y_i, \omega_j, \alpha)$.*
2. *Machine V^* is allowed to initiate polynomially-many interactions with the $P^{(i,j)}$'s.*
   - *In 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.*

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

*We let i*ZK *denote an interactive proof which is interleaved zero-knowledge in the general model. Zero-knowledge via black-box simulation is defined as in the previous cases.*

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 *sequential 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 $P^{(i,j)}$).[16] Thus, we immediately have

Theorem 9 *Suppose that P is interleaved zero-knowledge in the general model* (i.e., iZK). *Then P is concurrent zero-knowledge. Furthermore, simulation via black-box simulators is preserved.*

By a straightforward adaptation of the proof of Theorem 7, it follows that rewindable zero-knowledge in the general model implies interleaved zero-knowledge in the general model.[17] That is,

Theorem 10 *Suppose that P is rewindable zero-knowledge in the general model. Then P is interleaved zero-knowledge in the general model* (i.e., iZK). *Furthermore, simulation via black-box simulators is preserved.*

Finally, as stated above, the (constant-round) protocols presented in Section 3 are in fact secure in a setting allowing multiple inputs. Specifically, both Theorems 2 and 3 hold with respect to multiple inputs. Combining the above, we obtain the main result of this paper:

Theorem 11 *Suppose DLP is hard for subexponential circuits. Then every language in $\mathcal{NP}$ has a* constant-round *interleaved zero-knowledge computationally-sound proof system in the public-key model, even when multiple inputs are allowed* (i.e., iZK). *Furthermore, the prescribed prover is interleaved zero-knowledge via a black-box simulation.*

Thus, as a special case we get results for concurrent execution. For example,

Corollary 12 *Suppose DLP is hard for subexponential circuits. Then every language in $\mathcal{NP}$ has a* constant-round *concurrent zero-knowledge computationally-sound proof system in the public-key model.*

The above actually holds under more general conditions (see discussion and proof of Theorem 3). Likewise, note that under more general assumptions, every language in $\mathcal{NP}$ has a five-round interleaved witness-indistinguishable interactive proof system (in the vanilla) model, again even when multiple inputs are allowed (i.e., iWI).

---

[16]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 extention of our notion of rewindable zero-knowledge.

[17]Note that even in case the adversary V^* of the general interleaving model only runs each $P^{(i,j)}$ once, the adversary W^* derived for the general sequential (rewindable) model may run the same $P^{(i,j)}$ several times.

6 Alternative Rewindable Zero Knowledge Protocol

In this section we give an alternative version of the Rewindable 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. As before these commitment schemes exist if for example the DLP is hard.

6.1 Preliminaries

Probability spaces.[18] 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 $x \stackrel{R}{\leftarrow} S$, $y \stackrel{R}{\leftarrow} T, \cdots$.

6.2 Two Types of Commitments

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

6.2.1 Type-1 Commitments

Informal Description. A type-1 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 decommiting 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 auxilary 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 iZK 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 iZK protocol itself, the verifier will essentially proves to the prover that he knows

[18] Verbatim from [3] and [23].

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 $V_1 \neq V_2$,:

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

6.2.2 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 iZKprotocol 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 $\forall v_1, v_2$

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

6.2.3 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 noninteractive 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 commitmentments 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".

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

Definition: We define the language DLP$'$ to consist of the quadruplets $(p, g, x, \overline{p-1})$, 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 quadruplets in DLP' whose prime has length k: $DLP'_k \stackrel{\text{def}}{=} \{(p, g, x, \overline{p-1}) \in DLP' : |p| = k\}$.

The DLP$'$ Assumption: $\exists c, d > 0\ \forall k > d\ \forall\ 2^k$-gate circuits C:

$$PROB[(p, g, y, \overline{p-1}) \xleftarrow{R} DLP'_{k^c}\ ;\ C(p, g, y, \overline{p-1}) = x : g^x \bmod p) = y] < 2^{-k}$$

Remark: We include the factorization of $p-1$ in the Discrete-Logarithm Problem to enable one to check easily that g is a generator for Z_p^*. We may avoid this "difficulty" in varius ways. For instance, by (1) redefining DLP' to consist of triplets (p, g, x), where p is a prime of the form $2q+1$, and q is itself prime, and (2) assuming that one may easily randomly select a member of the corresponding sublanguage DLP'_k on input 1^k. The Discrete-Logarithm Problem appears to remain hard to solve for primes of this special form.

Lemma 1: Under the 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 algoritm 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 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). ■

6.4 An iZK Protocol For 3-Coloring Using Public Keys

6.4.1 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, $K = k^{\frac{1}{2\alpha_1}}$. [19] 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 verifer 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.

6.4.2 The Protocol

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.

[19] 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.

- **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.

Acknowledgements

We are indebted to Ran Canetti for discussion on the notion of rewindability and its possibility in earlier stages of this work. We are indebted to Amit Sahai for his invaluable help in finding a flaw in an earlier version of this paper.

References

[1] 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

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

[3] 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 [4].)

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

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

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

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

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

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

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

[11] 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.

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

[13] 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`.

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

[15] 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.

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

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

[18] 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.

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

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

[21] 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).

[22] 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.

[23] 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.

[24] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. Construction of Pseudorandom Generator from any One-Way Function. To appear in *SIAM Jour. on Computing*. Preliminary versions by Impagliazzo et. al. in *21st STOC* (1989) and Håstad in *22nd STOC* (1990).

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

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

[27] R. Ransom and J. Kilian. Non-Synchronized Composition of Zero-Knowledge Proofs. Manuscript, 1998.

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

Appendix A: Commitment Schemes

We formally define the various types of commitment schemes used by our main protocol. 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 $q \stackrel{\text{def}}{=} (p-1)/2$, and $x \in \mathsf{Z}_q^*$.*

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

A.2 Standard Commitment Schemes

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

Definition 13 (standard commitment scheme): *A* standard commitment scheme *is a probabilistic polynomial-time algorithm, denoted C satisfying:*

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

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

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

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

A commitment scheme as above can be constructed based on any one-way permutation: Loosely speaking, given a permutation $f : D \mapsto D$ with a hard-core predicate b (cf., [17]), 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 $S(1^n, v, \mathrm{msg})$, 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 14 (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* $\mathrm{poly}(n)$*-length, the random variables* $S(1^n, v, R^*(1^n))$ *and* $S(1^n, u, R^*(1^n))$ *are statistically close. That is, for every two polynomials* p, q*, all sufficiently large* n*'s and all* $v, u \in \{0,1\}^{p(n)}$

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

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

In order to formulate the above, we rewrite the honest sender move, $S(1^n, v, \mathrm{msg})$*, as consisting of uniformly selecting* $s \in \{0,1\}^{\mathrm{poly}(n,|v|)}$*, and computing a polynomial-time function* $S'(1^n, v, s, \mathrm{msg})$*, where* msg *is the receiver's message. A cheating sender tries, given a receiver message* msg*, to find two pairs* (v, s) *and* (v', s') *so that* $v \neq v'$ *and yet* $S'(1^n, v, s, \mathrm{msg}) = S'(1^n, v', s', \mathrm{msg})$*. 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., [15]). 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[20]), 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 subsection 3.2 can be easily formulated. The strong computational binding feature is formulated by extending the Computational Binding Property (of Def. 14) 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 (msg, 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

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

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

Appendix B: the subtle problem or rewindable zero-knowledge wrt limited verifiers

Clearly, the protocol described in Subsection 3.1 is constant-round (specifically 5-round). It is also easy to see that it constitutes an interactive proof for Graph 3-Colorability: Perfect completeness is obvious (by just following the specified strategies). Soundness follows by noting that the protocol differs from the one in [15] only in Step (P1), but this is irrelevant since the analysis of soundness is with respect to an arbitrary cheating prover. Thus, soundness follows from [15]. We therefore focus on the zero-knowledge aspect of the protocol:

Proposition 15 *The protocol described in Subsection 3.1 is rewindable zero-knowledge* with respect to verifiers which decommit properly.

Proof Sketch: Here we use some extra properties of the simulator presented in [15] for the related protocol. We first consider, as a mental experiment, a prover which uses a truly random function rather than a pseudorandom one. Let V^* a probabilistic polynomial-time adversary which

[20] 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.

interacts with this prover as in (`Distribution 1` of) Definition 1. We first claim that in case V^* sends the same first message (i.e., the Step (V1) commitment) in two invocations of the prover then, except with negligible probability, it cannot reveal two different edge-sequences in the corresponding Step (V2). This follows from the computational-binding property of the commitment scheme used for the verifier's commitments, and from the fact that the simulator presented in [15] starts by fixing the coins of V^* (once and for all). That is, for the i^{th} iteration, the claim follows by the computational-binding property and the fact that a simulation of the previous $i-1$ interactions can be incorporated into a cheating sender (played by the verifier).

Thus, ignoring these negligible probability events, whenever the verifier repeats a Step (V1) message it has to reveal the same values in the decommitment Step (V2). The reason being that failure to decommit is disallowed by the hypothesis, whereas decommitment to a different value may only occur with negligible probability (by the above). But whenever the Step (V1) commitment message as well as the decommitment Step (V2) are the same as in a previous interaction, so are the values revealed in Step (P2). Thus, in such a case, we can simulate the current interaction by copying values from the previous interaction. It follows that, without loss of generality (and ignoring negligible probability events), we may assume that V^* never repeats the same Step (V1) message in two different interactions.

But in such a case, the interaction amounts to polynomially-many sequential composition of the [15] protocol (since an application of a totally random function $f:\{0,1\}^m \mapsto \{0,1\}^m$ to q different values is equivalent to uniform and independent selection of q elements in $\{0,1\}^m$). Since the [15] protocol has a black-box zero-knowledge simulator it follows that the above ideal prover (which uses a truly random function) is rewindable zero-knowledge via a black-box simulation.

Turning back to the actual prover (which uses a pseudorandom function rather than a totally random one), we claim that the simulator provided for the ideal prover also simulates the actual prover. Otherwise, one can combine the prover and adversary interactive programs into a (non-uniform) algorithm which distinguishes pseudorandom functions from truly random ones, in contradiction to the definition of pseudorandom functions. Here we use the fact that the prover's strategy can be implemented in polynomial-time given the input graph and a 3-coloring of it. The latter are incorporated into the distinguishing algorithm (and constitute – as usual in the area – its only non-uniform aspect).[21] ■

Appendix C: Blum's Proof of Knowledge

For sake of self-containment, we first recall the definition of a proof of knowledge. The following text is reproduced from [13].

C.1 Proofs of Knowledge

C.1.1 Preliminaries

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation. Then $R(x) \stackrel{\text{def}}{=} \{s : (x,s) \in R\}$ and $L_R \stackrel{\text{def}}{=} \{x : \exists s \text{ s.t. } (x,s) \in R\}$. If $(x,s) \in R$ then we call s a *solution* for x. We say that R is *polynomially bounded* if there exists a polynomial p such that $|s| \le p(|x|)$ 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.

[21] A fully-uniform treatment is applicable here too; see [12].

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 16 (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).)

C.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 17 (System of proofs of knowledge): *Let R be a binary relation, and $\kappa : \mathsf{N} \to [0,1]$. We say that an interactive machine V is a* knowledge verifier for the relation *R with* knowledge error *κ if the following two conditions hold.*

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

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

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

  The oracle machine K is called a universal knowledge extractor.

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

C.2 Blum's Protocol

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

Construction 18 (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 decomitments.*
- 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 19 *The protocol which results by* k *parallel repeations of Construction 18 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.*