Resettable Zero-Knowledge*

(Extended Abstract)

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

Abstract

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

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

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

Keywords: Zero-Knowledge, Concurrent Zero-Knowledge, Public-Key Cryptography, Witness-Indistinguishable Proofs, Smart Cards, Identification Schemes

1 Introduction

The notion of a zero-knowledge interactive proof was put forward and first exemplified by Goldwasser, Micali and Rackoff [22]. The generality of this notion was demonstrated by Goldreich, Micali and Wigderson [19], who showed that any NP-statement can be proven in zero-knowledge, provided that commitment schemes exist.[1] Subsequently, related notions have been proposed; in particular, zero-knowledge arguments [2], witness indistinguishability [11], and zero knowledge proofs of knowledge [22, 27, 10, 1]. By now, zero-knowledge is the accepted way to define and prove security of various cryptographic tasks; in particular, as proposed by Fiat and Shamir [12], it provides the basis for many proofs of identity.

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

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

For zero-knowledge proofs of knowledge (and thus for all proofs of identity à la Fiat-Shamir [12]), *by definition*, the answer is NO: if the verifier can force the prover to use the same coins for a polynomial number of executions, then even the honest verifier can easily extract the very same secret which the prover is claiming knowledge of.[3]

For zero-knowledge proofs (of language membership), the answer also appeared to be negative: all known examples of zero-knowledge proofs (including the 3-Coloring protocol of [19]) are trivially breakable if the prover is "reset" (to his initial state) and forced to use the same coins in future

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

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

‡Dept. of Computer Science, Weizmann Institute of Science, Rehovot, ISRAEL; oded@wisdom.weizmann.ac.il. Supported by MINERVA Foundation, Germany.

§Laboratory for Computer Science, MIT, Cambridge, MA02139; shafi@theory.lcs.mit.edu

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

STOC 2000 Portland Oregon USA

[1] Or, equivalently [25, 23], that one-way functions exist.

[2] Zero-knowledge proofs in which the prover is deterministic exist only for $\mathcal{BPP}$ languages (cf., [20]).

[3] For instance, in [12] it suffices to repeat the protocol twice with the same prover-coins to be able to extract the prover's secret.

interactions, even if these interactions are with the honest verifier.

Example. For instance, to prove that $z = x^2 \bmod n$ is quadratic residue mod n, in [22] the following basic protocol is repeated: the prover randomly chooses $r \in Z_n^*$ and sends $r^2 \bmod n$ to the verifier; the verifier sends a random bit b to the prover; and the prover sends back r if $b = 0$, and $xr \bmod n$ if $b = 1$. Assume now that the prover is forced to execute twice with the same coins r the basic protocol. Then, by sending $b = 0$ in the first execution and $b = 1$ in the second execution, the verifier learns both r and xr and thus trivially extract x, a square root of z mod n.

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

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

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

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

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

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

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

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

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

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

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

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

MAIN RESULTS. We prove the following theorems:

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

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

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

By the public-key model, we mean that a verifier has a public key that has been registered —i.e., fixed— prior to his interaction with the prover. We stress that we only assume that public-keys can be registered in the sense that it has been posted. Registration does not have to include any interaction with a trusted system manager that may verify properties of the registered public-key. We also stress that the prover does not need a public key.[6] (As we shall point out later on, this quite standard model of fixing a key before interaction starts can be further relaxed.) For a more detailed discussion of this model see section 5.

[4] In a preliminary version of this work [15], the same notion was called *rewind zero-knowledge* and *interleaved zero-knowledge*.

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

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

1.1 Resettable vs. Concurrent ZK

A WEAKER NOTION. In the past few years, considerable attention has been devoted to *concurrent* zero-knowledge (cZK) protocols. In essence, these are ZK proofs that withstand malicious verifiers who can interact several times with the prover, in an "interleaved way," about the same theorem. In each interaction, however, the prover will use a "fresh" random tape. (This model was first considered in [9].)

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

SOME PRIOR cZK PROTOCOLS. Concurrent ZK protocols have been suggested by Dwork, Naor and Sahai [8], assuming that a certain level of synchronization is guaranteed: the so-called *timing assumption*. Under this assumption, (1) there are a-priori known bounds on the delays of messages with respect to some ideal global clock, and (2) each party uses a local clock whose rate is within a constant factor of the rate of the ideal clock. Under the timing assumption (and some standard intractability assumption), constant-round, ZK arguments for NP were presented in [8]. In a later paper, Dwork and Sahai [7] show how the push up the use of the timing assumption to a pre-processing protocol, to be executed before the concurrent executions of protocols. More recent work by Richardson and Kilian [26] does not use the timing assumption, however their protocols are not constant-round. We stress that none of these concurrent ZK protocols is rZK.

1.1.1 rZK vs. cZK in the standard model

In the standard (non public-key) model, we construct our resettable ZK protocols based on concurrent ZK ones, and in particular the cZK protocol of Richardson and Kilian [26]. (Therefore, in this model, our constructions do not result in better cZK protocols.)

CONSTRUCTIONS OF rZK PROOF SYSTEMS. We actually present two constructions of rZK protocols for NP: one "by reduction" and a "direct" one. Our first construction consists of two steps. In a first step, we provide a transformation mapping any cZK protocol satisfying a special condition (the *admissible* cZK *protocols*) into rZK protocols. In the second step, we show how to transform the cZK protocol of [26] into an admissible one.

Our direct construction also consists of two steps. In the first step, we provide a constant-round resettable witness-indistinguishable (rWI) protocol for NP (a step of independent interest). In the second step, we properly combine our rWI protocol with the cZK protocol of [26] so as to obtain an rZK protocol for NP. The combined protocol inherits the round complexity of the [26] protocol, and thus is not constant-round.

LOWER BOUNDS FOR rZK PROOF SYSTEMS. Demonstrating limitations on the ability to construct cZK protocols, Kilian et.al. [24] show that four-round cZK protocols whose security is proved via black-box simulation exist only for languages in BPP. Rosen (priv. comm.) has recently extended this result to seven-round protocols. Since any rZK protocol is also cZK, these lower bounds apply to rZK protocols as well.

1.1.2 rZK vs. cZK in the public-key model

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

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

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

Moreover, the above result is widely implementable, because the public-key model is ubiquitous whenever cryptography is used (specifically, it underlies any public-key encryption or digital signature scheme). As the public-key model is both simpler and more realistic than the timing assumptions of [8, 7], we believe that the constant-round cZK protocol of Corollary 4 is preferable to the constant-round one of [8, 7]. Indeed, even if one thinks of the public-key model as a form of preprocessing, Corollary 4 provides an alternative to Dwork and Sahai's protocol which is based on pre-processing with the timing assumption. For further comparison see Section 5.

Another constant-round cZK argument for NP (but not an rZK one!) has been independently provided by Damgård [4, 5], but his protocol relies on a stronger public-key model: one in which a trusted center generates the (secret key, public key) pairs (i.e the soundness of the protocols depends on the trusted center keeping the secret key confidential). Alternatively, this trusted center can be replaced by a pre-processing interactive protocol between users (setting up their public-keys) and certification authorities.

1.2 Implications of rZK for Proofs of Identity

Fiat and Shamir in [12] introduced a paradigm for ID schemes based on the notion of Zero Knowledge Proof of Knowledge. In essence, a prover identifies himself by convincing the verifier of knowing a given secret (e.g., in [12], of knowing a square root of a given square mod n). All subsequent ID schemes follow this paradigm, and are traditionally implemented by the prover being a smart card (as suggested in [12]). However, Zero Knowledge Proof of Knowledge are impossible in a resettable setting (i.e., they exist only in a trivial sense[7]), and thus *all* Fiat-Shamir like ID schemes fail to be secure whenever the prover is resettable.

Instead, an alternative paradigm emerges for constructing ID schemes so that the resulting schemes are secure when the identification is done by a device which can be reset to its initial state such as a smart card. The new paradigm consists of viewing the *ability to convince the verifier that a fixed input is in a "hard" NP-language* as a proof of identity,

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

and employing an rZK proof to do so. Further elaboration on the notion and the construction of Resettable Proofs of Identity will appear in a separate paper.

Organization. For lack of space, this abstract contains only a rough sketch of our results. Details appear in our technical report [3]. Section 2 defines the notions of rZK and rWI. Section 3 provides a general method for transforming a certain class of proof systems designed for the concurrent setting into resettable ones. Section 4 presents a direct construction of a rZK proof system. Sections 2 through 4 concentrate on the standard model. Section 5 overviews our results in the public key model.

2 The Notions of rWI and rZK

2.1 Preliminaries

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

Interactive proof and argument systems. 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. Likewise, when we talk of computationally-sound proof systems (a.k.a arguments) [2] we mean ones in which polytime provers are unable to cheat with non-negligible probability.

Zero-knowledge. 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 be simulatable by a probabilistic polynomial-time machine (which interacts with nobody), called the simulator. We mention that the simulators in Sections 3 and 4 run in *strict* polynomial-time, whereas those in Section 5 run in *expected* polynomial-time.

Witness indistinguishable proof systems [11]. Loosely speaking, these are proof systems in which the prover is a probabilistic polynomial-time machine with auxiliary input (typically, an NP-witness), having the property that interactions in which the prover uses different "legitimate" auxiliary-inputs are computationally indistinguishable from each other. Recall that any zero-knowledge proof system is also witness indistinguishable, and there are witness indistinguishable proof systems that are not zero-knowledge.

2.2 Definition of rWI and rZK

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

Convention: Without loss of generality, we assume that each message of the verifier contains the entire communication history in the session that the message relates to. Furthermore, we assume that the prover is *memoryless:* it responds to each message based solely on the received message and on its input and random input. Whereas traditionally this convention could add power to a cheating verifier, in the context of resettable zero-knowledge this convention does not add any power to a cheating verifier (such freedom is allowed anyhow in the resettable model).

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

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

1. *Randomly select and fix $t = \mathrm{poly}(n)$ random-tapes, $\omega_1, ..., \omega_t$, for P, resulting in deterministic strategies $P^{(i,j)} = P_{x_i,y_i,\omega_j}$ defined by $P_{x_i,y_i,\omega_j}(\alpha) = P(x_i, y_i, \omega_j, \alpha)$, for $i,j \in \{1,...,t\}$. Each $P^{(i,j)}$ is called an* incarnation *of P.*
2. *Machine V^* is allowed to arbitrarily interact with all the incarnations of P. That is, V^* sends arbitrary messages to each of the $P^{(i,j)}$, and obtains the responses of $P^{(i,j)}$ to such messages. Once V^* decides it is done interacting with the $P^{(i,j)}$'s, it* (i.e., V^*) *produces an output based on its view of these interactions. Let us denote this output by $\langle P(\overline{y}), V^*\rangle(\overline{x})$.*

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

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

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

Comments on the Definition:

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

[8] Recall that the existence of black-box simulators implies auxiliary-input zero-knowledge (cf. [20, 17]).

different incarnation of the prover, so that the prover uses different coin tosses in every session. Thus, *every resettable zero-knowledge protocol is concurrent zero-knowledge.*

Two more variations of definition 1 are natural to consider (for details see [3]). The first one requires V^* to complete one session before starting another (either with the same incarnation of P or with a different one.) This variation results in a definition that is equivalent to the one here. The second variation allows V^* to have multiple concurrent sessions but only with a single incarnation of P. This variation results in a strictly weaker definition than the one here; this a main reason why the formulation of Definition 1 allows many incarnations.

3 Constructing rWI and rZK Protocols Via Reductions

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

Finally we demonstrate how to transform known constructions of concurrent WI and ZK proof systems for $\mathcal{NP}$ (specifically, the constructions of Goldreich and Kahan [16] and Richardson and Kilian [26]) into admissible ones that are WI and ZK in the hybrid model, obtaining:

Theorem 2 *Suppose that there exists a* two-round *perfectly-hiding commitment scheme. Then the following holds:*

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

The Hypothesis Perfectly-hiding computationally binding commitment schemes between a committer and a receiver consist of a commitment protocol, followed by a decommitment protocol where the receiver outputs a decommitment value. Roughly speaking the requirements are that the receiver's view in the commitment protocol is statistically independent of the committed value, and that it is computationally infeasible for the committer to run the commitment protocol and later be able to run the decommitment protocol in two ways, so that the receiver outputs two different valid decommitment values. Here we restrict ourselves to *two round* schemes, where the commitment protocol consists of a single message sent by the receiver, followed by a single message sent by the committer. The decommitment protocol consists of a single message sent by the committer. See details in [3].

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

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

Although the above intuitive formulation suffices for our main results (i.e., deriving the conclusion of Theorem 2 under the standard DLP assumption), we wish to relax it for greater generality. We syntactically partition each subsequent message of the verifier into two parts: a *main part* and an *authenticator*. In the special case considered above (of the first verifier-message being a commitment), the main part (of a subsequent verifier-message) is the revealed value and the authenticator is the extra decommitment information that establishes the validity of this value. The relaxation is that the main part (in this case the revealed value) must be determined by the first verifier message (i.e., the commitment), but the authenticator (i.e., the decommitment information) may vary.

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

Definition 3 (admissible proof-systems): *A proof-system (P, V) is called* admissible *if the following requirements hold:*

1. *The prover P consists of two modules, P_1, P_2. Similarly, the random input w is partitioned into two disjoint parts, $w^{(1)}, w^{(2)}$, where $w^{(i)}$ is given to P_i. The prover initialization message is sent by P_1.*
2. *Each verifier message (other than the first one) is first received by P_1 and is interpreted as consisting of two parts, called* main *and* authenticator. *P_1 decides[9] whether to accept the message or to abort. If P_1 accepts, it forwards the* main part *of the message to P_2, who generates the next prover message.*
3. *Let V^* be an arbitrary* (deterministic) *polynomial-size circuit representing a possible strategy for the verifier in the interactive proof (P, V). Then, except with negligible probability, V^* is unable to generate two different messages for some round ℓ that specify the same session determining message in their corresponding prefixes, and such that P_1 accepts both.*

Construction 4 *Given an admissible proof system (P, V), where $P = (P_1, P_2)$, and a collection $\{f\}$ of pseudorandom functions (see [14]), we define a new proof system $(\mathbf{P}, \mathbf{V})$ as follows.*

[9] The above phrase postulates a deterministic decision, which suffices for our applications.

The new verifier *is identical to* V.
The new prover: *The new prover's randomness is viewed as a pair* $(w^{(1)}, f)$, *where* $w^{(1)} \in \{0,1\}^{\text{poly}(n)}$ *is of length adequate for the random-tape of* P_1, *and* $f : \{0,1\}^{\leq \text{poly}(n)} \to \{0,1\}^{\text{poly}(n)}$ *is a description of a function taken from an ensemble of pseudorandom functions. For convenience we describe the new prover,* $\mathbf{P}$, *as a pair* $\mathbf{P} = \mathbf{P}_1, \mathbf{P}_2$. $\mathbf{P}_1$ *is identical to* P_1 *with random-tape* $w^{(1)}$; $\mathbf{P}_2$ *emulates the actions of* P_2 *with random tape that is determined by applying* f *to the determining message and the input. That is, upon receiving the determining message, denoted* $\mathtt{msg}'$, $\mathbf{P}_2$ *sets* $w^{(2)} = f(x, \mathtt{msg}')$ *and runs* P_2 *with random input* $w^{(2)}$. *From this step on,* $\mathbf{P}_2$ *emulates the actions of* P *using* $(w^{(1)}, w^{(2)})$ *as* P*'s random-tape.*

Intuitively, Construction 4 "takes care" of the fact that in the resettable model the random-tape of $\mathbf{P}_2$ is fixed in all the sessions of an incarnation of $\mathbf{P}$. However, the construction does not modify $\mathbf{P}_1$, and in particular does not solve potential problems that may occur when $\mathbf{P}_1$ uses the same $w^{(1)}$ in different sessions. In order to formalize the security guarantee provided by Construction 4, we formulate a hybrid type of cheating verifier that works against admissible proof systems. (In such proof systems, an incarnation of the prover is identified via three indices: $P^{(i,j,k)} = P_{x_i, y_i, w_{j,k}}$, where $w_{j,k} = w_j^{(1)}, w_k^{(2)}$. That is, i specifies the input, j specifies the random input to P_1 and k specifies the random input to P_2.) This hybrid verifier is identical to the cheating verifier in Definition 1, with the exception that is not allowed to have more than a single session with incarnations of P that have the same index k. (In other words, in each session the prover gets a new value for $w^{(2)}$.) The hybrid verifier may be regarded as closer in spirit to the verifier in the concurrent setting (than to the verifier in the resettable setting), since the first message usually only contains initialization information for the session (and in particular is independent of the input.)

Definition 5 (hZK and hWI): *A hybrid cheating verifier* V^* *works against admissible proof systems as described above. That is,* V^* *proceeds as in Distribution 1 of Definition 1 with the exception that no two sessions started by* V^* *may interact with incarnations* $P^{(i,j,k)}$ *and* $P^{(i',j',k')}$ *such that* $k = k'$. *An admissible proof system is* hZK *(resp.,* hWI*) if it satisfies Definition 1 with respect to hybrid cheating verifiers.*

Theorem 6 *Suppose that* (P, V) *is admissible, and let* $\mathbf{P}$ *be the prover strategy obtained from* P *by applying Construction 4, assuming that pseudorandom functions exist. Then for every probabilistic polynomial-time cheating verifier* V^* *(as in Definition 1 there exists a probabilistic polynomial-time hybrid cheating verifier* W^* *so that* $\langle P(\overline{y}), W^*\rangle(\overline{x})$ *is computationally indistinguishable from* $\langle \mathbf{P}(\overline{y}), V^*\rangle(\overline{x})$.

Corollary 7 *If a proof system* (P, V) *is* hZK *(resp.,* hWI*) then* $(\mathbf{P}, \mathbf{V})$ *is* rZK *(resp.,* rWI*).*

The proof of Theorem 6 appears in our technical report [3]. We provide a brief outline (that omits numerous important details): Adversary W^* serves as a "mediator" between adversary V^* and the prover P. That is, W^* runs V^*; whenever V^* starts a new session whose determining message is different from all previous ones, W^* merely relays the messages of this session between V^* and P. When V^* "replays" an existing session s (i.e., V^* starts a new session whose determining message is identical to that of an existing session s) W^* responds to V^* using the answers of P in session s, *without interacting with* P. Finally W^* outputs whatever V^* outputs.
The proofs of the following propositions can be found in [3].

Proposition 8 *Suppose that there exists a* two-round *perfectly-hiding commitment scheme. Then the (5 round) zero-knowledge proof system of [16] for any language in* $\mathcal{NP}$ *can be modified to be both admissible and* hWI.

Comments:
1. In fact, the "modification" to the [16] proof system does not modify any protocol messages; it only specifies the separation of the prover P into P_1 and P_2 as in Definition 3. Essentially, P_1 will play the role of the receiver in the (unconditionally binding) commitment scheme in the [16] protocol; the other functions of P are played by P_2. See details in [3].
2. The resulting proof system is probably *not* rZK; in fact, it is probably not even cZK. This follows from recent work of A. Rosen (priv. comm.). Specifically, extending [17, 24], Rosen shows that no language outside $\mathcal{BPP}$ can have a 7-round proof system that is concurrent zero-knowledge via black-box simulation.

Proposition 9 *Suppose that there exists a* two-round *perfectly-hiding commitment scheme. Then the proof system of [26] can be modified so that the resulting proof system (for any language in* $\mathcal{NP}$*) is both admissible and* hZK.

4 A Direct Construction of an rZK Protocol For NP

We present an rZK proof system for NP. Our proof-system is obtained by properly modifying the cZK proof system of [26]. The modification proceeds along the lines of the general construction presented in the previous section; nonetheless, some additional modifications, specific to the construction of [26], are necessary. Let us therefore briefly recall their proof system and then describe our modifications of it.
The Richardson-Kilian Protocol. In essence, given a common input x (allegedly a member of an NP-complete language L), their proof system consists of two stages.

The *first stage* is independent of x. At its start, the verifier commits to k random bit sequences, $r_1, ..., r_k \in \{0,1\}^n$, where n is the security parameter and k is a parameter of the proof. We fix k to be polynomial in the security parameter.[10] This initial commitment is then followed by k *iterations*. In iteration i, the prover commits to a random bit sequence, s_i, and the verifier decommits to the corresponding r_i, thereby pinning down the i^{th} *coin-toss* $r_i \oplus s_i$, the bit-by-bit exclusive or of s_i and r_i. Note that $r_i \oplus s_i$ string is known only to the prover.

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

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

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

Our Modifications The above proof system is cZK but not rZK. To obtain our resettable proof system $(\mathcal{P}, \mathcal{V})$, we introduce the following three modifications in their proof system.

1. The prover's random tape will only be used to provide the randomness for the initializing message for the verifier's commitment, plus a short random string s. The prover shall use s to implement a random function $f(\cdot) = f_s(\cdot)$ à la Goldreich, Goldwasser and Micali [14] (having sufficiently long inputs and outputs), and then use f, as follows, to generate all the random bits needed in subsequent rounds. Let msg denote the verifier's commitment message (i.e., the one committing the verifier to the strings $r_1, \ldots, r_k$); then the random tape that the prover of [26] will use throughout the rest of the execution (i.e, for all k iterations of Stage 1 and for the entire Stage 2), is set to $f(x, msg)$.

2. We replace their WI proof system with our rWI of Section 3. This guarantees the (local) resettability of Stage 2.

3. We require the verifier to send its first message in the rWI proof system of Stage 2 together with his initial commitment message of phase 1.

The statement S to be proven in Stage 2 is "$x \in L \vee \exists i$ s.t. $r_i = s_i$. This "NP statement" is tranformed in a standard fashion in an instance of 3 colorability, that is, in a graph G, and this G will be the common input of our rWI proof system. Therefore, because (the structure and) the size of S is known in advance (i.e., it is independent of the particular execution of Stage 1 and thus of the particular values of the strings s_i and r_i), so is the number of nodes in G. Consequently, because the verifier of our rWI proof system should commit to choosing random edges in G, such commitment can be made right away (i.e., together with the verifier's commitment at the start of Stage 1), based solely of the number of nodes of G. The verifier simply commits to a random pair of nodes, and when such a pair (u, v) will not correspond to an actual edge of G, it will simply be ignored.

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

The implementation of the protocol uses two complementary types of commitment schemes: The prover's commitments are via a perfectly-binding commitment scheme (which is only computationally-hiding), whereas the verifier's commitments are via a perfectly-hiding commitment scheme (which is only computationally-binding). For simplicity of presentation, we will use a one-round scheme based on any one-way permutations[11] for the first type, and a two-round scheme based on claw-free pairs[12] for the second type. The protocol is visually summarized in Figure 1.

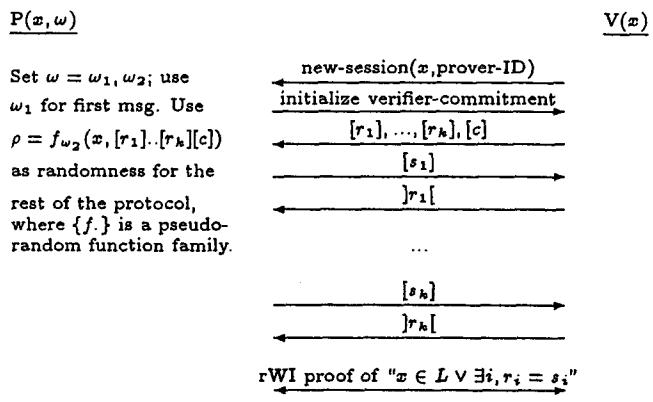

Figure 1: A sketch of the resettable zero-knowledge protocol for language L. Here $[a]$ and $]a[$ denote commitment and decommitment to a, respectively, and c is the verifier's challenge in the rWI protocol.

Completeness and soundness of $(\mathcal{P}, \mathcal{V})$ are straightforward. The difficulty consists of proving that $(\mathcal{P}, \mathcal{V})$ is resettable zero-knowledge (as per Definition 1). To simplify that proof, we make the following assumptions on the model, but stress that none of them restricts the generality of the result.

4.1 Without Loss of Generality

Recall that the verifier V^* is taken to be deterministic. We also assume that V^* does not generate invalid messages (e.g., invalid decommitments). In addition, we make the following simplifying assumptions. (Yet other simplifying assumptions are made when presenting the actual proof, in [3].)

No duplication of sessions. We assume that the adversary V^* does not start two sessions with the same verifier commitment message. Informally, no generality is lost by this assumption since, as long as the verifier does not open a commitment in more than one way, the verifier's messages in a session are uniquely determined by its first message. Thus two sessions that have the same verifier commitment message will have transcripts which may differ only in that one transcript is a prefix of the other. Consequently, the verifier "gains nothing" by opening two sessions with the same verifier commitment message. (Specifically, for any verifier V that may open many sessions with the same verifier commitment message there exists an equivalent verifier W that does not open two sessions with the same verifier commitment message. Full formalization of this argument would proceed along the lines of the proof of Theorem 6.)

A fixed bound on the number of sessions. Our simulator works assuming that a malicious (resetting) verifier

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

[12] Specifically, given a family of claw-free pairs, $\{(f_a^0, f_a^1) : a \in I \subseteq \{0,1\}^*\}$ (e.g., see [13]), the sender commits to bit σ as follows. The receiver first selects at random an index $a \in I$ and sends it to the sender, which uniformly selects x in the domain of f_a^σ, and sends the value $f_a^\sigma(x)$. Decommitment is done by providing (σ and) x.

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

4.2 The High-Level Strategy of the Simulator

In this extended abstract we only present the high-level strategy of the simulator (full details appear in [3]). At this high level, this strategy is similar to that of the simulator of the cZK protocol of [26]; but significant differences exist in the actual simulation. (In fact, our proof in [3] also provides a detailed alternative to the proof of the underlying cZK protocol.)

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

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

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

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

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

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

Specifically, $\mathcal{S}$ uses three main procedures. Procedure `Simulate` tries to generate a simulated run of V^*. Each time a new prover's message is needed, `Simulate` calls procedure `NextProverMsg`, which returns the next prover's message to be used in the simulated run. When `NextProverMsg` encounters an unsolved session in an iteration $j < k$, it calls procedure `Solve`. Procedure `Solve` tries to solve session s at iteration $j+1$ by means of a *bounded look ahead*. That is, before committing to prover's string s_{j+1}, `Simulate` tries to find the value r_{j+1} that V^* might decommit to in iteration $j+1$ of s, so as to choose $s_{j+1} = r_{j+1}$. Such look-ahead is no different than the main simulation: it is done via a recursive call to `Simulate`. At each call, however, a properly initialized counter ℓ is decreased by 1, and when $\ell = 0$ no more recursive calls are made, and the simulator always commits to random prover values s'_i for all iterations i of other sessions s' (unless it happens to know the right r'_i), hoping to solve s at iteration $j+1$. If it fails, it abandons all hope to solve s in iteration $j+1$, hoping instead to solve it at a future iteration. (In this case the simulator sticks with a commitment, α, to a randomly selected value s_i, and proceeds with the rest of the simulation). In [3], we show that the running time of the simulator is $n^{O(\log_k K)}$, and that, as long as $k = \Omega(\log_k K)$, the above strategy solves all sessions with high probability.

5 rZK in the Public-Key Model

Thus far in the paper no set-up assumptions have been made in the model. This is indeed the "simplest" model used for two-party and multi-party computation. Another model, used routinely in the context of providing privacy and/or authenticity of messages (i.e public-key encryption and digital signatures), is the *public-key* model, which relies on a set-up stage in which public-keys are registered. In the work presented in this section, the public-key model is used for tasks totally unrelated to privacy and authenticity, but rather for proving security of protocols whose participants hold public-keys.[13]

5.1 The Public Key Model

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

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

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

[13] A similar use was independently suggested by Damgård [4, 5] (see discussion below in related work).

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

Making such a distinction is quite reasonable in practice (e.g., say that a user in nowadays internet may start using its key a couple of days after registration, whereas each internet session is assumed to be completable within a couple of hours). Notice that, unlike usage of timing in [8], our usage of timing here *does* NOT *affect typical interactions*, which can be and actually are completed much faster than the conservative upper bound (of message delay) being used. In contrast, in [8] each user delays each critical message by an amount of time that upper bounds normal transmission delay. This means that all communication is delayed by this upper bound. Thus, in their case, this *always* causes *significant* delays: in fact the upper bound should be conservative enough so to guarantee that communication by honest users are rarely rejected.

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

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

5.2 Main Results for the Public Key Model

Theorem 10 *Under the strong DLP assumption, there exist* constant-round *resettable zero-knowledge arguments for $\mathcal{NP}$ in the public-key model.*

Recall that *arguments* (a.k.a computationally-sound proofs) [2] are a weaker notion than interactive proofs [22]: it is infeasible rather than impossible to fool the verifier to accept wrong statements with non-negligible probability.

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

Corollary 11 *Under the strong DLP assumption, there exist* constant-round *concurrent zero-knowledge arguments for $\mathcal{NP}$ in the public-key model.*

We stress that unlike [8], the above stated result does not use any timing assumption.

5.3 Techniques

Several techniques used by our construction are worth singling out. First, in all messages of the prover which require randomization, the prover will use, instead of fair coins, the result of applying a pseudo random function [14] to the prover's input and the sequence of messages exchanged in the interaction thus far. (In fact, it suffices to apply the pseudo random function only to the input plus some critical parts of the communication.) This ensures that on the same prefix of an interaction, the verifier will always get the same response from the prover. Thus, the verifier will not be able to collect different responses of the prover to the same questions – a capability which can lead to breaking the security of the protocol, and is an obvious attack strategy for a verifier who can run several executions of the protocol each time resetting the prover to the same initial state and same random tape.

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

Third, our construction uses actually two secure commitment schemes which interact in a novel way, One commitment scheme is with security parameter K and the other with a smaller security parameter k. We assume that , for some $\epsilon > 0$, the security of the first commitment scheme (with security parameter K) is maintained against adversaries running in time 2^{K^ϵ},[14] and that instances of the second scheme (with security parameter k) can be broken in time 2^k. Then setting $k = K^\epsilon/2$ guarantees both security of the second scheme as well as "non-malleability" (cf. [6]) of the first scheme in presence of the second one. The reason for the latter fact is that breaking the second scheme can be incorporated into an adversary attacking the first scheme without significantly effecting its running-time: Such an adversary is allowed running-time 2^{K^ϵ} which dominates the time $2^k = 2^{K^\epsilon/2}$ required for breaking the second scheme. This "telescopic" usage of intractability assumptions can be generalized to a case in which we have a lower and upper bound on the complexity of some problem; specifically, we

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

need a lower bound $L(n)$ on the average-case of solving n-bit long instances, and an upper-bound $U(n) \gg L(n)$ on the corresponding worst-case complexity. Suppose that we can choose polynomially-related security parameters k and K so that $L(k)$ is infeasible and $U(k) \ll L(K)$ (i.e., $L(k)$ is infeasible and $U(k) \ll L(\mathrm{poly}(k))$). Then the above reasoning still holds. (Above we used $L(n) = 2^{n^\epsilon}$ and $U(n) = 2^n$.) For further details see [3].

5.4 Outline of the Protocol

The detailed protocol can be found in our technical report [3]. Due to lack of space we will here only outline it.

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

1. prover looks up public-key i of the verifier (no interaction required) which specifies a commitment scheme $Comm_i$ (which is perfectly private and computationally binding and has security parameter K)
2. verifier runs a sub-protocol in which he convinces the prover that he knows the matching secret key to i
3. verifier commits to a sequence of edges $e_1, ...e_n$ of the graph G using 2nd commitment scheme $Comm'$ (which is also perfectly private and computationally binding but its security parameter is k)..
4. prover commits independently to n copies $EG_1, ..., EG_n$ of the graph G, each copy is colored with a permutation of the 3-coloring which the prover knows, using the commitment scheme $Comm_i$ specified by i.
5. verifier decommits edges e_1,..., e_n.
6. prover decommits for each EG_i the colors of the end points of the edge e_i
7. the verifier accepts if indeed all edges e_i were colored properly and rejects otherwise.

Almost constant-round rZK under weaker assumptions. We mention that using the weak DLP assumption (rather than the strong one), we obtain for every unbounded function $r : \mathsf{N} \to \mathsf{N}$, an $r(\cdot)$-round resettable zero-knowledge argument for $\mathcal{NP}$ in the public-key model. Again, such protocols are concurrent zero-knowledge (as a special case). (For further details see [3].)

References

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

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

[3] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable Zero-Knowledge. *ECCC*, TR99-042, 1999. Also available from the *Theory of Cryptography Library*.

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

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

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

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

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

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

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

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

[12] 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.

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

[16] 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.

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

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

[19] 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.

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

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

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

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

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

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

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