Min-Round Resettable Zero-Knowledge In The Public-Key Model

Silvio Micali and Leonid Reyzin

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
`reyzin@theory.lcs.mit.edu`
`http://theory.lcs.mit.edu/~reyzin`

Abstract. In STOC 2000, Canetti, Goldreich, Goldwasser, and Micali put forward the strongest notion of zero-knowledge to date, *resettable zero-knowledge* (RZK) and implemented it in constant rounds in a new model, where the verifier simply has a public key registered before any interaction with the prover.

To achieve ultimate round efficiency, we advocate a slightly stronger model. Informally, we show that, as long as the honest verifier does not use a given public key more than a fixed-polynomial number of times, there exist 3-round (which we prove optimal) RZK protocols for all of NP.

1 Introduction

The Notion of Resettable Zero-Knowledge. A zero-knowledge (ZK) proof [GMR89], is a proof that conveys nothing but the verity of a given statement. As put forward by Canetti, Goldreich, Goldwasser, and Micali [CGGM00], *resettable zero-knowledge* (RZK) is the strongest form of zero-knowledge known to date. In essence, an RZK proof is a proof that remains ZK even if a polynomial-time verifier can force the prover to execute the proof multiple times with the same coin tosses. More specifically,

- *The verifier can reset the prover.* In each execution, the verifier can choose whether the prover should execute the protocol with a new random tape or with one of the tapes previously used.
- *The verifier can arbitrarily interleave executions.* The verifier can always start (in particular, in a recursive way) new executions in the middle of old ones, and resume the old ones whenever it wants.
- *The prover is oblivious.* As far as it is concerned, the prover is always executing a single instance of the protocol.

Resettable ZK is a strengthening of Dwork, Naor and Sahai's [DNS98] notion of concurrent ZK (CZK). In essence, in a CZK protocol, a malicious verifier acts as in an RZK protocol, except that it lacks the power of resetting the prover's random tapes.

Constructing RZK protocols. Perhaps surprisingly, it is possible to implement such a strong notion: RZK proofs for NP-complete languages are constructed in [CGGM00] under standard complexity assumptions. Their construction is concretely obtained by properly modifying the CZK protocol of Richardson and Kilian [RK99]. Because this underlying CZK protocol is not constant-round, neither is the resulting RZK protocol. (The minimum known number of rounds for implementing the protocol of [RK99] is polylogarithmic in the security parameter, as shown by Kilian and Petrank [KP00].)

Unfortunately, it may not be possible to obtain a constant-round RZK protocol: at least in the black-box model, Canetti, Kilian, Petrank and Rosen [CKPR01] recently proved that no constant-round protocol exists even for CZK. However, [CGGM00] also put forward an appealingly simple model, which we call the *bare public-key* (BPK) model, and provide a 5-round[1] RZK argument for any NP language in this model.

Let us now quickly recall what their model is.

The Bare Public-Key Model. An interactive proof system in the BPK model simply assumes that the verifier $\mathcal{V}$ has a public key, PK, that is registered before any interaction with the prover begins. No special protocol needs to be run to publish PK, and no authority needs to check any property of PK. It suffices for PK to be a string known to the prover, and chosen by the verifier prior to any interaction with the prover.

The BPK model is very simple. In fact, it is a *weaker* version of the frequently used public-key infrastructure (PKI) model, which underlies any public-key cryptosystem or digital signature scheme. In the PKI case, a secure association between a key and its owner is crucial, while in the BPK case no such association is required. The single security requirement of the BPK model is that a bounded number of keys (chosen beforehand) are "attributable" to a given user.[2]

We have recently pointed out in [MR01] that the BPK model has four distinct notions of soundness, depending on the power enjoyed by a malicious prover $\mathcal{P}^*$: informally,

1. *one-time soundness,* arising when $\mathcal{P}^*$ is allowed a single interaction with $\mathcal{V}$ per theorem statement;
2. *sequential soundness*, arising when $\mathcal{P}^*$ is allowed multiple but sequential interactions with $\mathcal{V}$;

[1] Their paper actually presents two related constructions: (1) a 4-round protocol with an additional 3-round preprocessing stage with a trusted third party, and (2) an 8-round protocol without such preprocessing. Their constructions can be easily modified to yield the 5-round protocol attributed above.

[2] Indeed, having a prover $\mathcal{P}$ work with an incorrect public key for a verifier $\mathcal{V}$ does not affect soundness nor resettable zero-knowledgeness; at most, it may affect completeness. (Working with an incorrect key may only occur when an active adversary is present— in which case, strictly speaking, completeness does not even apply: this fragile property only holds when all are honest.)

3. *concurrent soundness,* arising when $\mathcal{P}^*$ is allowed multiple interleaved interactions with the same $\mathcal{V}$; and
4. *resettable soundness,* arising when $\mathcal{P}^*$ is allowed to reset $\mathcal{V}$ with the same random tape *and* interact with it concurrently.

As we have already said, the BPK model permits constant-round RZK protocols for all of NP. Indeed, the CGGM protocol is 5-round and sequentially sound, and we have recently constructed a 4-round one that also is sequentially sound [MR01]. To achieve ultimate round efficiency, we advocate strengthening the BPK model a bit.

The Upperbounded Public-Key Model. How many public keys can a verifier establish before it interacts with the prover? Clearly, no more than a polynomial (in the security parameter) number of keys. Though innocent-looking, this bound is a source of great power for the BPK model: it allows for the existence of constant-round black-box RZK, which is impossible in the standard model.

How many times can a verifier use a given public key? Of course, at most a polynomial (in the security parameter) number of times. Perhaps surprisingly, we show that if such an innocent-looking polynomial upperbound U is made explicit *a priori*, then we can further increase the round efficiency of RZK protocols.

In our *upperbounded public-key* (UPK) model, the honest verifier is allowed to fix a polynomial upperbound, U, on the number of times a public key will be used; keep track, via a counter, of how many times the key has been used; and refuse to participate once the counter has reached the upperbound.

Let us now make the following remarks about the UPK model:

– In the RZK setting, the “strong party” is the verifier (who controls quite dramatically the prover’s executions). Such a strong party, therefore, should have no problems in keeping a counter in order to save precious rounds of interaction.
– The UPK model does not assume that the prover knows the current value of the verifier’s counter. (Guaranteeing the accuracy of such knowledge would de facto require public keys that “change over time.”)
– While our RZK protocol satisfies interesting efficiency constraints with respect to U, we believe that these should be considered properties of our specific protocol rather than requirements of the UPK model.
  (For instance, our public key length is independent of U, while the secret key length and each execution of the protocol depend on U only logarithmically. Only the verifier’s key-generation phase depends linearly on U —a dependency that hopefully will be improved by subsequent protocols.)
– The UPK model is somewhat similar to the one originally envisaged in [GMR88] for secure digital signatures, where the signer posts an explicit upperbound on the number of signatures he will produce relative to a given public key, and keeps track of the number of signatures produced so far. (The purpose and use of our upperbound, however, are totally different.)
– While sufficient for constant-round implementation of the stronger RZK notion, the UPK model is perhaps simpler than those of [DS98] (which uses

"timing assumptions") and that of [Dam00] (which uses trusted third parties to choose some system parameters) for efficient implementation of CZK.

3-Round RZK in the UPK Model. Because the powerful RZK notion seems to require substantial interaction, it is important to establish how many rounds a reasonable model can save. As we have already said, the BPK model can reduce the number of rounds to four [MR01]. We show that the UPK model can do even better, reducing the number of rounds to the minimum and, at the same time, increasing soundness:

Main Theorem: *In the UPK model there exist 3-round concurrently sound RZK arguments for any language in NP, assuming collision-resistant hashing and the subexponential hardness of discrete logarithm and integer factorization.*[3]

Round-Optimality of the UPK Model. Our result is optimal (in either the UPK or the BPK model), at least for black-box RZK. This fact is evident from the following argument. Assume that a 2-round RZK (or even just ZK!) protocol (P, V) existed, in the BPK or the UPK model, for a language $L \notin \text{BPP}$. Then one could construct from it a 3-round ZK protocol (P', V') by adding an initial round in which the verifier sends its public key PK to the prover.[4] Protocol (P', V') would thus contradict the result of [GK96], which states that no 3-round, black-box ZK proofs or arguments exist for non-trivial languages.

Necessity of the UPK Model. In the cited [MR01], we also show that it is impossible in the BPK model to achieve 3-round ZK with concurrent soundness. Thus, to achieve 3-round RZK, one needs either to come up with a protocol that is sequentially (but not concurrently) sound, or to enhance the model in some reasonable fashion. The former approach seems quite elusive, and whether such a protocol exists remains an open problem. Our solution is an example of the latter approach.

2 Resettable Zero-Knowledge in the UPK Model

In this section, we define RZK in the UPK model. Let us refer the reader to the original exposition of [CGGM00] for motivation and intuition of RZK, which we do not provide here due to space constraints. Here we focus on:

[3] We can replace the integer factorization assumption with the more general assumption that subexponentially secure dense public-key cryptosystems [DDP00] and subexponentially secure certified trapdoor permutations [FLS99] exist. Or we can replace both the DL and the factorization assumptions with the assumption that decision Diffie-Hellman is subexponentially hard.

[4] Note that the so constructed (P', V') will not be RZK (else, being 4-round, it would contradict the recent lowerbound of [CKPR01]—and indeed even the older lowerbound of [KPR98]). However, it will still be ZK. To see this, observe that the old black-box simulator, designed to handle very powerful resetting malicious verifier (who can choose from among multiple public keys in the public file) can be also used with the weaker standard verifier (who simply uses only a single public key transmitted in the first message).

- RZK *arguments* (rather than proofs). That is, we assume that the prover is polynomial-time and we let soundness hold in a computational (rather than probabilistic) sense. Our protocol in Section 4 and the public key protocol of [CGGM00] are RZK arguments.
- *Black-box* zero-knowledgeness. That is, we demand that there exist a single simulator that works for all malicious verifiers $\mathcal{V}^*$ (given oracle access to $\mathcal{V}^*$). This is a stronger notion, and is indeed the one we satisfy in Section 4.

The Players

Let

- A *public file* F be a polynomial-size collection of records (id, PK_{id}), where id is a string identifying a verifier, and PK_{id} is its (alleged) public key.
- A *prover* $\mathcal{P}$ *(for a language* L*)* be an interactive deterministic polynomial-time TM that is given as inputs (1) a security parameter 1^n, (2) a n-bit string $x \in L$, (3) an auxiliary input y, (4) a public file F, (5) a verifier identity id, and (6) a random tape ω.
  For simplicity of exposition, one can view $\mathcal{P}$ as a *non-interactive* TM that is given, as an additional input, the entire history of the messages already exchanged in the interaction, and outputs the next message. Fixing all inputs, this view allows one to think of $\mathcal{P}(1^n, x, y, F, id, \omega)$ as a simple deterministic oracle, which is helpful in defining the notion of RZK below.
- A *U-bounded (honest) verifier* $\mathcal{V}$, for a positive polynomial U, be an interactive polynomial-time TM that, on first input a security parameter 1^n, works in $U(n) + 1$ stages, with the ability of keeping state information. In the first *key generation* stage, on input a security parameter 1^n, $\mathcal{V}$ outputs a public key PK and remembers the corresponding secret key SK. In subsequent $U(n)$ *verification* stages, on input an n-bit string x, $\mathcal{V}$ performs an interactive protocol with a prover.
- An *(s, t)-resetting verifier* $\mathcal{V}^*$, for any two positive polynomials t and s, be a TM that runs in two stages so that, on first input 1^n,
  1. In stage 1, $\mathcal{V}^*$ receives $s(n)$ values $x_1, \ldots, x_{s(n)} \in L$ of length n each, and outputs an arbitrary public file F and a list of $s(n)$ identities $id_1, \ldots, id_{s(n)}$.
  2. In stage 2, $\mathcal{V}^*$ starts in the final configuration of stage 1, is given oracle access to $s(n)^3$ provers, and then outputs whatever it desires (in particular, it can output its "view" of the interactions, which includes its random string).
  3. The total number of steps of $\mathcal{V}^*$ in both stages is at most $t(n)$.
- A *black-box simulator* M be a polynomial-time machine that is given oracle access to $\mathcal{V}^*$. By this we mean that it can run $\mathcal{V}^*$ multiple times, each time picking $\mathcal{V}^*$'s inputs, random tape and (because $\mathcal{V}^*$ makes oracle queries itself) the answers to all of $\mathcal{V}^*$'s queries. M is also given $s(n)$ values $x_1, \ldots, x_{s(n)} \in L$ as input.

The Definitions

To define RZK in the UPK model, we must define (1) completeness, (2) soundness and (3) resettable zero-knowledgeness proper. For lack of space, we omit a formal discussion of completeness in the UPK model. (This property is the usual one for interactive proofs, except that it has to hold only for the first $U(n)$ interactions, and to assume that $\mathcal{P}$ gets the correct public key for $\mathcal{V}$.) For the same reason, we omit a formal discussion of concurrent soundness, the type of soundness actually enjoyed by our protocol and informally specified in our introduction. (The reader is referred to [MR01] for formal details.) The third notion is the same as in [CGGM00]. Nonetheless, we find it useful to recall it below.

Definition 1. $(\mathcal{P}, \mathcal{V})$ *is* black-box resettable zero-knowledge for an NP-language *L if there exists a simulator M such that for every pair of positive polynomials (s, t), for every (s, t)-resetting verifier $\mathcal{V}^*$, for every $x_1, \ldots, x_{s(n)} \in L$ and their corresponding NP-witnesses $y_1, \ldots, y_{s(n)}$, the following probability distributions are indistinguishable (in time polynomial in n):*

1. *The output of $\mathcal{V}^*$ obtained after choosing $\omega_1, \ldots, \omega_{s(n)}$ uniformly at random, running the first stage of $\mathcal{V}^*$ to obtain F, and then letting $\mathcal{V}^*$ interact in its second stage with the following $s(n)^3$ instances of $\mathcal{P}$: $\mathcal{P}(x_i, y_i, F, id_k, \omega_j)$ for $1 \le i, j, k \le s(n)$.*
2. *The output of M with input $x_1, \ldots, x_{s(n)}$ interacting with $\mathcal{V}^*$.*

3 Tools

Let us quickly recall the notation, the definitions and the constructions that we utilize in our protocol.

3.1 Probabilistic Notation

(The following is taken verbatim from [BDMP91] and [GMR88].) 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. If S is a probability space, then "$x \overset{R}{\leftarrow} S$" denotes the algorithm which assigns to x an element randomly selected according to S. If F is a finite set, then the notation "$x \overset{R}{\leftarrow} F$" denotes the algorithm that chooses x uniformly from F.

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

3.2 Trapdoor Commitment Schemes

In this section we present trapdoor commitment schemes that are secure against subexponentially strong adversaries (satisfying an additional key-verification property).[5]

Informally, a trapdoor commitment scheme consists of a quintuple of algorithms. Algorithm TCGen generates a pair of matching public and secret keys. Algorithm TCCom 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. Algorithm TCVer takes the public key and c, v, d and checks whether c was indeed a commitment to v.

What makes the commitment *computationally binding* is that without knowledge of the secret key, it is computationally hard to come up with a single commitment c and two different decommitments d_1 and d_2 for two different values v_1 and v_2 such that TCVer would accept both c, v_1, d_1 and c, v_2, d_2. What makes it *perfectly secret* is that the value c yields no information about the value v. Moreover, this has to hold even if the public key is chosen adversarially. Thus, there has to be an algorithm TCKeyVer that takes a public key as input and verifies whether the resulting commitment scheme is indeed perfectly secret. (More generally, TCKeyVer can be an interactive protocol between the committer and the key generator, rather than an algorithm; however, for our application, the more restricted view suffices).

Perfect secrecy ensures that, information-theoretically, any commitment c can be decommitted arbitrarily: for any given commitment c to a value v_1, and any value v_2, there exists d_2 such that TCVer accepts c, v_2, d_2 and the public key (indeed, if for some v_2 such d_2 did not exist, then c would leak information about the actual committed value v_1). The trapdoor property makes this assurance computational: knowing the secret key enables one to decommit arbitrarily through the use of the TCFake algorithm.

Definition 2. *A* Trapdoor Commitment Scheme *(*TC*) is a quintuple of probabilistic polynomial-time algorithms* TCGen, TCCom, TCVer, TCKeyVer *and* TCFake*, such that*

1. Completeness. $\forall n$, $\forall v$,

$$\begin{aligned}\text{PROB}[(\mathit{TCPK}, \mathit{TCSK}) \overset{R}{\leftarrow} \text{TCGen}(1^n)\,;\ (c,d) \overset{R}{\leftarrow} \text{TCCom}(\mathit{TCPK}, v)\ :\\ \text{TCKeyVer}(\mathit{TCPK}, 1^n) = \text{TCVer}(\mathit{TCPK}, c, v, d) = \text{YES}] = 1\end{aligned}$$

2. Computational Soundness. $\exists\ \alpha > 0$ *such that for all sufficiently large* n *and for all* 2^{n^α}*-gate adversaries* ADV

$$\begin{aligned}\text{PROB}[\ &(\mathit{TCPK}, \mathit{TCSK}) \overset{R}{\leftarrow} \text{TCGen}(1^n)\,;\\ &(c, v_1, v_2, d_1, d_2) \overset{R}{\leftarrow} \text{ADV}(1^n, \mathit{TCPK})\ :\\ &\text{TCVer}(\mathit{TCPK}, c, v_1, d_1) = \text{YES and}\\ &\text{TCVer}(\mathit{TCPK}, c, v_2, d_2) = \text{YES and } v_1 \neq v_2] < 2^{-n^\alpha}\end{aligned}$$

We call α *the* soundness constant.

[5] We follow a similar discussion in [CGGM00] almost verbatim.

3. Perfect Secrecy. $\forall$ *TCPK such that* TCKeyVer(*TCPK*, 1^n) = YES *and* $\forall v_1, v_2$ *of equal length, the following two probability distributions are identical:*

$$[(c_1, d_1) \overset{R}{\leftarrow} \text{TCCom}(TCPK, v_1) : c_1] \quad \text{and}$$
$$[(c_2, d_2) \overset{R}{\leftarrow} \text{TCCom}(TCPK, v_2) : c_2]$$

4. Trapdoorness. $\forall$ $(TCPK, TCSK) \in \{\text{TCGen}(1^n)\}$, $\forall v_1, v_2$ *of equal length the following two probability distributions are identical:*

$$[\ (c, d_1) \overset{R}{\leftarrow} \text{TCCom}(TCPK, v_1)\ ;$$
$$d_2' \overset{R}{\leftarrow} \text{TCFake}(TCPK, TCSK, c, v_1, d_1, v_2)\ :\ (c, d_2')\ ] \quad \text{and}$$
$$[\ (c, d_2) \overset{R}{\leftarrow} \text{TCCom}(TCPK, v_2)\ :\ (c, d_2)\ ]$$

(In particular, the above states that faked commitments are correct: indeed, $d_2' \overset{R}{\leftarrow}$ TCFake($TCPK, TCSK, c, v_1, d_1, v_2$) *implies that* TCVer($TCPK, c, v_2, d_2'$) = YES*)*

In this paper, we will also require that the relation $(TCPK, TCSK)$ *be polynomial-time; this is easy to satisfy by simply including the random string used in key generation into the secret key.*

Such commitment schemes can be constructed, in particular, based on a subexponentially strong variant of the Discrete Logarithm assumption. We refer the reader to [BCC88] (where, in Section 6.1.2, it is called a DL-based "chameleon blob") for the construction.

3.3 Hash-Based Commitment Schemes

We also have a need of non-trapdoor, non-interactive, computationally-binding commitment schemes (which, unlike trapdoor commitments, need not be secure against subexponentially strong adversaries). Because of the absence of the trapdoor requirement, these simpler commitment schemes can be implemented more efficiently if one replaces perfect secrecy by the essentially equally powerful property of *statistical secrecy* (i.e., even with infinite time one can get only a statistically negligible advantage in distinguishing the commitments of any two different values). In particular [DPP97,HM96] show how to commit to any value by just one evaluation of a collision-free hash function $H : \{0,1\}^* \rightarrow \{0,1\}^k$. To differentiate trapdoor commitments from these simpler ones, we shall call them *hash-based commitments.*

Though the trapdoor property does not hold, we still insist that, given any commitment and any value, it is possible in time 2^k to decommit to that value.

Definition 3. *A* Hash-Based Commitment Scheme *(*HC*) is a pair of probabilistic polynomial-time algorithms* HCCom, HCVer*, along with the algorithm* HCFake *that runs in time* 2^kpoly *when its first input is k and* poly *is some polynomial in the size of its input, such that*

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

$$\text{PROB}[(c,d) \overset{R}{\leftarrow} \text{HCCom}(1^k, v) : \text{HCVer}(1^k, c, v, d) = \text{YES}] = 1$$

2. Computational Soundness. *For all probabilistic polynomial-time machines* ADV*, and all sufficiently large k,*

$$\text{PROB}[(c, v_1, v_2, d_1, d_2) \overset{R}{\leftarrow} \text{ADV}(1^k) : \\ v_1 \neq v_2 \text{ and } \text{HCVer}(1^k, c, v_1, d_1) = \text{YES} = \text{HCVer}(1^k, c, v_2, d_2)]$$

is negligible in k.

3. Statistical Secrecy. *$\forall v_1, v_2$ of equal length, the statistical difference between the following two probability distribution is negligible in k:*

$$[(c_1, d_1) \overset{R}{\leftarrow} \text{HCCom}(1^k, v_1) : c_1] \textit{ and } [(c_2, d_2) \overset{R}{\leftarrow} \text{HCCom}(1^k, v_2) : c_2]$$

4. Breakability. *$\forall v_1, v_2$ of equal length, the statistical difference between the following two probability distribution is negligible in k:*
$[(c, d_1) \overset{R}{\leftarrow} \text{HCCom}(1^k, v_1)\ ;\ d_2' \overset{R}{\leftarrow} \text{HCFake}(1^k, c, v_1, d_1, v_2) : (c, d_2')]$ *and*
$[(c, d_2) \overset{R}{\leftarrow} \text{HCCom}(1^k, v_2) : (c, d_2)]$

We refer the reader to [DPP97,HM96] for the constructions of such schemes, which are based on the assumption that collisions-resistant hash functions exist.

3.4 Non-Interactive Zero-Knowledge Proofs of Knowledge

Non-interactive zero-knowledge (NIZK) proofs for any language $L \in \text{NP}$ were put forward and exemplified in [BFM88,BDMP91]. Ordinary ZK proofs rely on interaction. NIZK proofs replace interaction with a random *shared string*, σ, that enters the view of the verifier that a simulator must reproduce. Whenever the security parameter is 1^n, σ's length is NIσLen(n), where NIσLen is a fixed, positive polynomial.

Let us quickly recall their definition, modified for polynomial-time provers and security against subexponentially strong adversaries.

Definition 4. *Let non-interactive prover* NIP *and non-interactive verifier* NIV *be two probabilistic polynomial-time algorithms, and let* NIσLen *be a positive polynomial. We say that* (NIP, NIV) *is a NIZK argument system for an NP-language L if*

1. Completeness. *$\forall\ x \in L$ of length n, σ of length* NIσLen(n)*, and NP-witness y for x,*

$$\text{PROB}[\Pi \overset{R}{\leftarrow} \text{NIP}(\sigma, x, y) : \text{NIV}(\sigma, x, \Pi) = \text{YES}] = 1.$$

2. Soundness. *$\forall\ x \notin L$ of length n,*

$$\text{PROB}[\sigma \overset{R}{\leftarrow} \{0,1\}^{\text{NI}\sigma\text{Len}(n)} : \exists\ \Pi \textit{ s. t. } \text{NIV}(\sigma, x, \Pi) = \text{YES}]$$

is negligible in n.

3. Zero-Knowledgeness. *$\exists\ \alpha > 0$ and a probabilistic polynomial-time simulator* NIS *such that, $\forall$ sufficiently large n, $\forall$ x of length n and NP-witness y for x, the following two distributions are indistinguishable by any 2^{n^α}-gate adversary:*

$$[(\sigma', \Pi') \overset{R}{\leftarrow} \mathrm{NIS}(x) : (\sigma', \Pi')] \quad \text{and}$$
$$[\sigma \overset{R}{\leftarrow} \{0,1\}^{\mathrm{NI}\sigma\mathrm{Len}(n)} \ ; \ \Pi \overset{R}{\leftarrow} \mathrm{NIP}(\sigma, x, y) : (\sigma, \Pi)]$$

We call α the zero-knowledgeness constant.

In [DP92], De Santis and Persiano propose to add a *proof of knowledge* property to NIZK. Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a polynomial-time relation (i.e., given a pair of strings (x, y), it is possible to check in time polynomial in $|x|$ whether $(x, y) \in R$). L be the NP language corresponding to R ($L = \{x : \exists\ y \text{ s.t. } (x, y) \in R\}$). Let $(\mathrm{NIP}, \mathrm{NIV})$ be a NIZK proof system for L. An *extractor* is a probabilistic polynomial-time TM that runs in two stages: in stage one, on input 1^n, it outputs a string σ of length $\mathrm{NI}\sigma\mathrm{Len}(n)$ (and saves any information it wants to use in stage two); in stage two, on input x of length n and a proof Π for x relative to shared string σ, it tries to find a witness y for x.

Definition 5. *An NIZK argument* $(\mathrm{NIP}, \mathrm{NIV})$ *is a NIZKPK if there exists an extractor* $\mathrm{NIExt} = (\mathrm{NIExt}_1, \mathrm{NIExt}_2)$ *such that, for all probabilistic polynomial-time malicious provers* NIP^**, for all constants $a > 0$, for all sufficiently large n and for all x,*

$$\mathrm{PROB}\ [(\sigma, \mathit{state}) = \mathrm{NIExt}_1(1^n) \ ; \ \Pi = \mathrm{NIP}^*(\sigma, x) \ ;$$
$$y = \mathrm{NIExt}_2(\mathit{state}, x, \Pi) : (x, y) \in R] \geq \quad p_{n,x}(1 - n^{-a}),$$

where $p_{n,x} = \mathrm{PROB}[\sigma \overset{R}{\leftarrow} \{0,1\}^n; \Pi = \mathrm{NIP}^*(\sigma, x) : \mathrm{NIV}(\sigma, x, \Pi) = 1]$.

The authors of [DP92] show that NIZKPKs exist for all polynomial-time relations under the RSA assumption. Furthermore, the results of [DDP00] (combined with those of [FLS99]) show the same under more general assumptions: that dense public-key cryptosystems and certified trapdoor permutations exist. They also present constructions secure under the specific assumptions of factoring Blum integers or decision Diffie-Hellman. Because we need NIZKPKs to be secure against subexponentially strong adversaries, we need subexponentially strong versions of these assumptions. We refer the reader to these papers for details.

3.5 Additional Basic Tools

We also use two basic and commonly used tools, whose definitions are recalled in the appendix. The first is a Merkle tree [Mer89], which can be constructed based on a collision-resistant hash function. The second is a subexponentially-strong pseudorandom function [GGM86], i.e., one that is secure against adversaries of size 2^{n^α} (such α is called the *pseudorandomness constant*). It can be constructed based on subexponentially strong one-way functions [HILL99].

4 Our Construction

Why the Obvious Solution Does Not Work. Before we begin, let us demonstrate that our goal cannot be more easily achieved by the following simpler construction.

Let $cmax = U(n)$ be the upperbound on the number of uses of the verifier's public key (i.e., the max value for the verifier's counter). Take a four-round ZK protocol, and have the verifier post $cmax$ independently generated first-round messages in its public key. Then execution number c simply uses first-round message number c appearing in the public key, and performs the remaining three rounds of the protocol as before.

The above construction does not work, because the prover does not know the real value c of the verifier's counter. This enables a malicious verifier to choose the value of c after it sees the prover's first message. Thus, if such a verifier resets the prover while varying c, it will typically gain knowledge. (Typically, in a 4-round ZK protocol, the verifier commits to a question without revealing it, the prover sends a first message, the verifier asks the question, and the prover answers it. However, if the prover were to answer two different questions relative to the same first message, then zero-knowledgeness disappears. Now, in the above construction, varying c enables the verifier to ask different questions.)

High-Level Description. As in the CGGM protocol, we use the NP-complete language of graph 3-colorability and the parallel repetition of the protocol of [GMW91] as our starting point. Thus, in the first round, $\mathcal{P}$ commits to a number of random recolorings of a graph G, in the second round $\mathcal{V}$ requests to reveal the colors of one edge for each committed recoloring, and in the third round $\mathcal{P}$ opens the relevant commitments.

To allow the RZK simulator to work, our protocol uses trapdoor commitment schemes as in many prior ZK protocols (e.g., the RZK one of [CGGM00], the CZK one of [DNS98], and the ZK one of [FS89]). That is, $\mathcal{V}$'s public key contains a key for a trapdoor commitment scheme, and $\mathcal{P}$'s first-round commitments with respect to that public key. If the simulator knows the trapdoor, then it can open the commitments any way it needs in the the third round.

To ensure that the simulator knows the trapdoor, the CGGM protocol uses a three-round proof-of-knowledge subprotocol, with $\mathcal{V}$ proving to $\mathcal{P}$ knowledge of the trapdoor. This requires $\mathcal{V}$ to send two messages to $\mathcal{P}$. Because we have a *total* of only three rounds, we cannot use such a subprotocol—in three rounds $\mathcal{V}$ only sends one message to $\mathcal{P}$. We therefore use *non-interactive* ZK proofs of knowledge. This, of course, requires $\mathcal{P}$ and $\mathcal{V}$ to agree on a shared random string σ.

It is because of the string σ that we cannot use the BPK model directly, and have to strengthen it with a counter. Let $cmax = U(n)$ be the bound on the number of times public key is used. During key generation, $\mathcal{V}$ generates $cmax$ random strings $\sigma_1, \ldots, \sigma_{cmax}$, and commits to each one of them using hash-based commitments (to make the public key length independent of $cmax$, the resulting commitments are then put into a Merkle tree). In its first message,

$\mathcal{P}$ sends a fresh random string $\sigma_{\mathcal{P}}$, and in its message $\mathcal{V}$ decommits σ_c, where c is the current counter value and provides the NIZKPK proof with respect to $\sigma = \sigma_{\mathcal{P}} \oplus \sigma_c$.

The RZK simulator, after seeing the value of σ_c can rewind the verifier and choose $\sigma_{\mathcal{P}}$ so that $\sigma = \sigma_{\mathcal{P}} \oplus \sigma_c$ allows it to extract the trapdoor from the NIZKPK proof. Of course, there is nothing to prevent a malicious verifier $\mathcal{V}^*$ from choosing a value of c after seeing $\sigma_{\mathcal{P}}$; but because the number of choices for $\mathcal{V}^*$ is only polynomial, the simulator has an inverse polynomial probability of guessing c correctly.

One question still remains unresolved: how to ensure that a malicious verifier $\mathcal{V}^*$ does not ask $\mathcal{P}$ multiple different queries for the same recoloring of the graph? If $\mathcal{V}^*$ resets $\mathcal{P}$, then it will get the same committed recolorings in the first round; if it can then ask a different set of queries, then it gain a lot of information about the coloring of the graph (eventually even recovering the entire coloring). To prevent this, the CGGM protocol makes the verifier commit to its queries before it receives any information from $\mathcal{P}$. Our protocol, however, cannot afford to do that, because we only have three rounds. Instead, during key generation the verifier commits (using hash-based commitments) to a seed *PRFKey* for a pseudorandom function PRF, and adds the commitment to the public key. The verifier's queries are then computed using $\mathrm{PRF}(\mathit{PRFKey}, \cdot)$ applied to the relevant information received from $\mathcal{P}$ in the first round and the counter value c. To prove to $\mathcal{P}$ that they are indeed computed correctly, the verifier has to include in its NIZKPK proofs of knowledge of *PRFKey* that leads to such queries and knowledge of the decommitment to *PRFKey*.

A Few More Technical Details. In our protocol, just like in the CGGM protocol all probabilistic choices of the prover are generated as a pseudorandom function of the input. (This is indeed the first step towards resettability, as it reduces the advantages of resetting the prover with the same random tape.) Because the prover makes no probabilistic choices in its second step, we do not need to include the verifier's message in the input to the pseudorandom function.

To ensure soundness and avoid problems with malleability of $\mathcal{V}$'s commitments, we use complexity leveraging in a way similar to the CGGM protocol. That is, and we shall use two polynomially-related security parameters: n for all the components except the hash-based commitment scheme HC, and $k = n^{\epsilon}$ for HC.

This will ensure that any algorithm that is endowed with a subroutine for breaking HC commitments, but is polynomial-time otherwise, is still unable (simply by virtue of its running time) of breaking any other of our components. This property will be used in our proof of soundness.

We actually choose the constant ϵ in a particular way. Namely, we shall use a trapdoor commitment scheme TC with soundness constant α_1, an NIZKPK system (NIP, NIV) (for a relation to be specified later) with zero-knowledgeness constant α_2, and a pseudorandom function PRF with pseudorandomness constant α_3, and set $\epsilon < \min(\alpha_1, \alpha_2, \alpha_3)$.

The Full Description. The complete details of $\mathcal{P}$ and $\mathcal{V}$ are given below.

Key Generation Algorithm for $\mathcal{V}$

System Parameter:
A polynomial U
Security Parameter:
1^n
Procedure:

1. Let $cmax = U(n)$.
2. Generate random strings $\sigma_1, \dots, \sigma_{cmax}$ of length NIσLen(n) each. (Note: to save secret key length, the strings σ_c can be generated using a pseudorandom function of c, whose short seed can be made part of the secret key).
3. Let $k = n^\epsilon$.
4. Commit to each σ_c using $(\sigma Com_c, \sigma Decom_c) \overset{R}{\leftarrow}$ HCCom$(1^k, \sigma_c)$.
5. Combine the values σCom_c into a single Merkle tree with root R. (Note: If the values σ_c's are generated via a PRF to save on secret key length then also the values σCom_c, the resulting Merkle tree, etc. can be computed efficiently in space logarithmic in $cmax$.)
6. Generate a random string $PRFKey$ of length n.
7. Commit to the $PRFKey$ using $(PRFKeyCom, PRFKeyDecom) \overset{R}{\leftarrow}$ HCCom$(1^n, PRFKey)$.
8. Generate keys for the trapdoor commitment scheme: $(TCPK, TCSK) \overset{R}{\leftarrow}$ TCGen(1^n).

Output:
$PK = (R, PRFKeyCom, TCPK)$
$SK = (\{(\sigma_c, \sigma Decom_c)\}_{c=1}^{cmax}, (PRFKey, PRFKeyDecom), TCSK)$.

Protocol $(\mathcal{P}, \mathcal{V})$

Public File:
A collection F of records (id, PK_{id}), where PK_{id} is allegedly the output of the Key Generation Algorithm above
Common Inputs:
A graph $G = (V, E)$, and a security parameter 1^n

$\mathcal{P}$ **Private Input:**
A valid coloring of G, $col : V \to \{0, 1, 2\}$; $\mathcal{V}$'s id and the file F; a random string ω
$\mathcal{V}$ **Private Input:**
A secret key SK, a counter value c, and a bound $cmax$.
$\mathcal{P}$ **Step One :**

1. Using the random string ω as a seed for PRF, generate a sufficiently long "random" string from the input to be used in the remaining computation.

2. Find PK_{id} in F; let $PK_{id} = (R, \mathit{PRFKeyCom}, \mathit{TCPK})$ (if more than one PK_{id} exist in F, use the alphabetically first one).
3. Verify TCPK by invoking $\mathrm{TCKeyVer}(1^n, \mathit{TCPK})$.
4. Let $\sigma_{\mathcal{P}}$ be a random string of length $\mathrm{NI}\sigma\mathrm{Len}(n)$.
5. Commit to random recolorings of the G as follows.
   Let $\pi_1, \ldots, \pi_n$ be random permutations on $\{0, 1, 2\}$.
   For all i $(1 \le i \le n)$ and $v \in V$, commit to $\pi_i(\mathit{col}(v))$ by computing $(\mathit{cCom}_{i,v}, \mathit{cDecom}_{i,v}) \overset{R}{\leftarrow} \mathrm{TCCom}(\mathit{TCPK}, \pi_i(\mathit{col}(v)))$.
6. If all the verifications hold, send $\sigma_{\mathcal{P}}$ and $\{\mathit{cCom}_{i,v}\}_{1 \le i \le n, v \in V}$ to $\mathcal{V}$.

$\mathcal{V}$ **Step One:**

1. Increment c and check that it is no greater than cmax.
2. For each j $(1 \le j \le n)$, compute a challenge edge $e_j \in E$ by applying PRF to the counter value c, j and the commitments received from $\mathcal{P}$:
   $e_j = \mathrm{PRF}(\mathit{PRFKey}, c \circ j \circ \{\mathit{cCom}_{i,v}\}_{1 \le i \le n, v \in V})$
3. Let $\sigma = \sigma_{\mathcal{P}} \oplus \sigma_c$. Compute a NIZKPK proof $\mathit{\Pi}$ using NIP on σ and the following statement:
   "$\exists$ key K for PRF that generated the challenge edges $\{e_j\}_{1 \le j \le n}$;
   $\exists$ decommitment D s. t. $\mathrm{HCVer}(1^n, \mathit{PRFKeyCom}, K, D) = \mathrm{YES}$;
   $\exists$ secret key S corresponding to the public key TCPK."
   (Note: this can computed efficiently because $\mathcal{V}$ knows witnesses PRFKey for K, $\mathit{PRFKeyDecom}$ for D, and TCSK for S).
4. Send c, σ_c, $\sigma\mathit{Com}_c$ together with its authenticating path in the Merkle tree, $\sigma\mathit{Decom}_c$, $\mathit{\Pi}$ and $\{e_j\}_{1 \le j \le n}$ to $\mathcal{P}$.

$\mathcal{P}$ **Step Two:**

1. Verify the authenticating path of $\sigma\mathit{Com}_c$ in the Merkle tree
2. Verify that $\mathrm{HCVer}(1^k, \sigma_c, \sigma\mathit{Com}_c, \sigma\mathit{Decom}_c) = \mathrm{YES}$.
3. Let $\sigma = \sigma_P \oplus \sigma_c$. Verify $\mathit{\Pi}$ using NIV.
4. If all the verifications hold, for each $e_j = (v_j^0, v_j^1)$ and $b \in \{0, 1\}$, send $c_j^b = \pi_j(\mathit{col}(v_j^b))$ and $\mathit{cDecom}_{j,v_j^b}$ to $\mathcal{V}$.

$\mathcal{V}$ **Step Two:**

1. Verify that, for all j $(1 \le j \le n)$, and for all $b \in \{0, 1\}$ $\mathrm{TCVer}(\mathit{TCPK}, \mathit{cCom}_{j,v_j^b}, c_j^b, \mathit{cDecom}_{j,v_j^b}) = \mathrm{YES}$.
2. Verify that for all j $(1 \le j \le n)$, $c_j^0 \ne c_j^1$.
3. If all the verifications hold, accept. Else reject.

Theorem 1. *$(\mathcal{P}, \mathcal{V})$ is a 3-round RZK protocol in the UPK model.*

As usual, completeness is easily verified. We address soundness in Section 4.1 and resettable zero-knowledgeness in Section 4.2.

4.1 Computational Soundness

Suppose G is a graph that is not 3-colorable, and $\mathcal{P}^*$ is a circuit of size $t < 2^k$ that can make $\mathcal{V}$ accept $(G, 1^n)$ with probability $p > 1/2^k$. Then, we shall construct a small circuit A that receives $TCPK$ as input, and, using $\mathcal{P}^*$, will output two trapdoor decommitments for the same TC commitment. The size of A will be $\text{poly}(n) \cdot t \cdot 2^k/\text{poly}(p)$. Thus, A will violate the soundness of TC, because its size is less (for a sufficiently large n) than $2^{n^{\alpha_1}}$ allowed by the soundness property of TC (recall in fact that $k = n^\epsilon$ and $\epsilon < \alpha_1$).

A is constructed as follows. It receives as input a public key $TCPK$ for TC generated by TCGen(1^n). A then generates PK as if it were the public key of the specified honest verifier $\mathcal{V}$, using the $\mathcal{V}$'s key generation procedure with the exception of step 7, for which it simply uses $TCPK$. Note that A knows all the components of corresponding secret key of $\mathcal{V}$, with the exception of $TCSK$. A selects an identity id and creates a file F to contain the single record (id, PK) (or embeds it into a larger such file containing other identities and public keys, but honestly generated).

A will now run $\mathcal{P}^*$ multiple times with inputs F and id (G and 1^n are already known to $\mathcal{P}^*$), each time with the same random tape. Thus, each time, $\mathcal{P}^*$ will send the same set of strings $\sigma_\mathcal{P}$ and $\{cCom_{i,v}\}_{1\le i\le n, v\in V}$. Our goal, each time, is to allow A to respond with a different random set of challenges $\{e'_j\}_{1\le j\le n}$. Then, after an expected number of tries that is inversely polynomial in p, there will exist a recoloring i and a node v such that $cCom_{i,v}$ has been opened by $\mathcal{P}^*$ in two different ways. That is, there will be a "break" of the commitment scheme TC.

Therefore, all there remains to be shown is *how* A can ask a different random set of challenges, despite the fact that it has committed to $\mathcal{V}$'s $PRFKey$ in PK. Recall that honest $\mathcal{V}$ executes the protocol at most $cmax$ time, and that the current value of $\mathcal{V}$'s counter will be known to P^*. If P^* has such an overall success probability p of proving G 3-colorable, then there exists a value of $\mathcal{V}$'s counter for which the success probability of $\mathcal{P}^*$ is at least p. Let c be such a value. Because of A's non-uniformity, we assume A "knows" c.

To issue a set of (different) random challenges in response to the same first message of $\mathcal{P}^*$, A uses the NIZKPK simulator NIS as follows. First, A selects a set of random challenges $\{e'_j\}_{1\le j\le n}$. Second, it invokes NIS to obtain a "good looking proof" σ' and Π' for the following statement Σ:

$$\begin{aligned}\Sigma = \text{“}&\exists \text{ key } K \text{ for PRF that generated the challenge edges } \{e'_j\}_{1\le j\le n};\\ &\exists \text{ decommitment } D \text{ s. t. HCVer}(1^n, PRFKeyCom, K, D) = \text{YES};\\ &\exists \text{ secret key } S \text{ corresponding to the public key } TCPK.\text{”}\end{aligned}$$

(Note that Σ is potentially false, because it may be the case that no such K exists at all; we address this below.) Third, A sets $\tau = \sigma' \oplus \sigma_\mathcal{P}$. Fourth, A comes up with a decommitment $\tau Decom$ that decommits σCom_c (the commitment to the c-th shared string computed during key generation) to τ rather than the originally committed σ_c. This can be done by implementing HCFake by means of a (sub)circuit of size $\text{poly}(k)2^k$. Fifth, A sends $\tau, \sigma Com_c$ together with its authenticating path in the Merkle tree (A knows that path from key generation), $\tau Decom$, Π' and $\{e'_j\}_{1\le j\le n}$ to $\mathcal{P}^*$.

Thus, all that's left to show is that $\mathcal{P}^*$ will behave the same way as it would for the true verifier $\mathcal{V}$, even though it received random, rather than pseudorandom, challenges, together with a faked decommitment and a simulated proof of a potentially false statement Σ. This is done by a properly constructed hybrid argument that relies on the zero-knowledgness of (NIP, NIV), the pseudorandomness of PRF and the statistical secrecy and breakability of HC.

First, note that random $\{e'_j\}_{1\leq j\leq n}$ cannot be distinguished from pseudorandomly generated $\{e'_j\}_{1\leq j\leq n}$ (wihout knowledge of *PRFKey*): otherwise, we'd violate the pseudorandomness of PRF. Moreover, this holds even in the presence of *PRFKeyCom*, because *PRFKeyCom* is statistically secret, and thus reveals a negligible amount of information about *PRFKey*. It follows that the tuple $(PRFKeyCom, \{e'_j\}_{1\leq j\leq n}, \sigma', \Pi')$ cannot be distinguished from the tuple $(PRFKeyCom, \{e_j\}_{1\leq j\leq n}, \sigma'', \Pi'')$, where the challenge edges $\{e_j\}_{1\leq j\leq n}$ are produced by the true PRF with the true committed-to *PRFKey*, and σ'', Π'' are produced by NIS. This, in turn, by zero-knowledgeness is indistinguishable from $(PRFKeyCom, \{e_j\}_{1\leq j\leq n}, \sigma, \Pi)$, with the pseudorandomly generated $\{e_j\}_{1\leq j\leq n}$, a truly random σ and Π honestly generated by NIP. By a hybrid argument, therefore, the tuple $(PRFKey, \{e_j\}_{1\leq j\leq n}, \sigma, \Pi)$ is indistinguishable from the tuple $(PRFKey, \{e'_j\}_{1\leq j\leq n}, \tau, \Pi')$. Of course, if we replace σ by the pair $(\sigma_{\mathcal{P}}, \tau = \sigma \oplus \sigma_{\mathcal{P}})$ and σ' by the pair $(\sigma_{\mathcal{P}}, \sigma_c = \sigma \oplus \sigma_{\mathcal{P}})$, the statement still holds. Moreover, it holds in the presence of σCom_c, because the commitment to σ_c is statistically secret (and thus is almost equally as likely to be a commitment to τ). The authenticating path of σCom_c in the Merkle tree is just a (randomized) function of σCom_c and root R of the tree, and thus does not affect indistinguishability. Finally, note that this indistinguishability holds with respect to any distinguishing circuit of size $2^k\text{poly}(n)$, because the zero-knowledgeness and pseudorandomness constants α_2 and α_3 are greater than ϵ. Therefore, indisntinguishability holds even in the presence of the decommitment $\tau Decom$ or $\sigma Decom_c$, because this decommitment can be computed by such a circuit from σCom_c using HCFake.

4.2 Resettable Zero-Knowledgeness (Sketch)

Let $\mathcal{V}^*$ be an (s,t)-resetting verifier. We will show how to construct the simulator M as required by Definition 1. Due to lack of space in this extended abstract, below we present only the essential points of our construction.

As alredy proven in [CGGM00], resettability is such a strong capability of a malicious verifier, that it has nothing else to gain by interleaving its executions with the honest prover. Thus, we can assume that our $\mathcal{V}^*$ executes with $\mathcal{P}$ only sequentially.

Recall that $\mathcal{V}^*$ runs in two stages. Then M operates as follows. First, M runs the first stage of $\mathcal{V}^*$ to obtain a public file F. Then, for every record (id, PK_{id}) in F, M remembers some information (whose meaning will be explained later on):

1. M remembers whether PK_{id} is "broken" or not
2. If PK_{id} is broken, M also remembers the value of $TCSK$
3. If PK_{id} is not broken, M also remembers a list of tuples $(c, \sigma_c, \sigma Com_c)$

Initially, every PK_{id} in F is marked as not broken, and the list of pairs for each record is empty.

Whenever $\mathcal{V}^*$ starts a new session for an *id* that is not broken and whose list of pairs is empty, M computes the "first prover message" as follows: it commits to arbitrary color values for graph G, and then selects $\sigma_{\mathcal{P}}$ at random. (Of course, if $\mathcal{V}^*$ dictates that M's random tape and inputs be equal to those in a prior interaction, M has no choice but to use the same first message as in that interaction.) When $\mathcal{V}^*$ responds with the verifier message, M takes $(c, \sigma_c, \sigma Com_c)$ from this message and adds it to the list of tuples maintained for PK_{id}. M then rewinds $\mathcal{V}^*$ to the beginning of $\mathcal{V}^*$'s second stage.

Whenever $\mathcal{V}^*$ starts a new session for an *id* that is not broken but whose list of pairs is non-empty, M randomly chooses a tuple $(c', \sigma_{c'}, \sigma Com_{c'})$ from the list of tuples for PK_{id}. M then uses the extractor of the non-interactive ZK proof of knowledge, NIExt_1, to obtain a shared string σ, and sets $\sigma_{\mathcal{P}} = \sigma \oplus \sigma_{c'}$. M then commits to arbitrary color values for graph G and sends the commitment and $\sigma_{\mathcal{P}}$ as the "first prover message" to $\mathcal{V}^*$. When $\mathcal{V}^*$ responds with the verifier message, M compares the counter value c included in this response to the value c' from the pair chosen above.

1. If $c = c'$, then it must be the case that $\sigma_c = \sigma_{c'}$. (Otherwise, if the commitment $\sigma Com_{c'}$ previously stored by M is equal to the commitment σCom_c included in $\mathcal{V}^*$'s response, σ_c and $\sigma_{c'}$ have been easily found, so as to violate the soundness of HC; and if the $\sigma Com_c \neq \sigma Com'_c$, then a collision has been easily found in the Merkle tree). Thus, the string Π, also included in the response of V^*, is an NIZK proof of knowledge with respect to the string σ output by NIExt_1. Therefore, M can use NIExt_2 to extract a witness $TCSK$ for the secret key of the commitment scheme. In this case, PK_{id} is marked as broken and M remembers $TCSK$.
2. If $c \neq c'$, then M has learned a potentially new tuple $(c, \sigma_c, \sigma Com_c)$, which it remembers as its list of pairs for PK_{id}.

M then rewinds $\mathcal{V}^*$ to the beginning of $\mathcal{V}^*$'s second stage.

Whenever $\mathcal{V}^*$ starts a new session for an *id* that is broken, M can always simulate $\mathcal{P}$'s behavior because M knows the trapdoor to the commitment scheme. Thus, it can commit to arbitrary color values in its first message, and then decommit in its second message so that they look like a valid response to $\mathcal{V}^*$'s challenge edges.

The expected running time of M is polynomial, because the expected number of rewinds before M breaks a given PK_{id} is polynomial in *cmax* and inverse polynomial in the frequency with which $\mathcal{V}^*$ uses *id*.

It remains to show that $\mathcal{V}^*$ cannot ask for two different sets of challenge edges for the same first message of M (if it could, then, unless M knows the correct 3-coloring of the graph, it maybe unable to faithfully simulate the decommitments). However, if $\mathcal{V}^*$ has a non-negligible probability of doing so, then one can build a machine ADV to violate the soundness of HC in polynomial time with non-negligible probability, as follows.

ADV guesses, at random, for what instance of $\mathcal{P}$ the machine $\mathcal{V}^*$ will first give two different sets of challenges on the same first message. A also guesses, at random, the counter values c_1 and c_2 that $\mathcal{V}^*$ will use in these two cases. A then attempts to find out σ_{c_1} and σ_{c_2} by using the same technique as M. A then runs the second stage of $\mathcal{V}^*$ two more times: once to extract a witness K for *PRFKey* and its decommitment D in the first case, and the other to extract a witness K' for *PRFKey* and its decommitment D' in the second case (this witness extraction is done the same way as M). $K \neq K'$ and D and D' are valid decommitments, which violates soundness of HC.

Acknowledgements

We would like to thank Amit Sahai, Anna Lysyanskaya and the anonymous referees for helpful comments. The second author was supported, in part, by a National Science Foundation Graduate Research Fellowship and a grant from the NTT corporation.

References

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.

[BDMP91] M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, December 1991.

[BFM88] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 103–112, 1988.

[Bra89] G. Brassard, editor. *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[CGGM00] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable zero-knowledge. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, 2000. Updated version available at the Cryptology ePrint Archive, record 1999/022, `http://eprint.iacr.org/`.

[CKPR01] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-box concurrent zero-knowledge requires $\tilde{\Omega}(\log n)$ rounds. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, 6–8 July 2001.

[Dam00] I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, ed., *Advances in Cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, Springer-Verlag, 2000.

[DDP00] A. De Santis, G. Di Crescenzo, and G. Persiano. Necessary and sufficient assumptions for non-interactive zero-knowledge proofs of knowledge for all np relations. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *Automata Languages and Programming: 27th International Colloquim (ICALP 2000)*, volume 1853 of *Lecture Notes in Computer Science*, pages 451–462. Springer-Verlag, July 9–15 2000.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent zero knowledge. In *30th Annual ACM Symposium on Theory of Computing*, 1998.

[DP92] A. De Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction. In *33rd Annual Symposium on Foundations of Computer Science*, 1992.

[DPP97] I. B. Damgård, T. P. Pedersen, and B. Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. *Journal of Cryptology*, 10(3):163–194, Summer 1997.

[DS98] C. Dwork and A. Sahai. Concurrent zero-knowledge: Reducing the need for timing constraints. In H. Krawczyk, ed., *Advances in Cryptology—CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, 1998.

[FLS99] U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29(1):1–28, 1999.

[FS89] U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In Brassard [Bra89], pages 526–545.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[GK96] O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, February 1996.

[GMR88] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18:186–208, 1989.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.

[HILL99] J. Håstad, R. Impagliazzo, L.A. Levin, and M. Luby. Construction of pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HM96] S. Halevi and S. Micali. Practical and provably-secure commitment schemes from collision-free hashing. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 201–215. Springer-Verlag, 18–22 August 1996.

[KP00] J. Kilian and E. Petrank. Concurrent zero-knowledge in polylogarithmic rounds. Technical Report 2000/013, Cryptology ePrint Archive, `http://eprint.iacr.org`, 2000.

[KPR98] J. Kilian, E. Petrank, and C. Rackoff. Lower bounds for zero-knowledge on the Internet. In *39th Annual Symposium on Foundations of Computer Science*, pages 484–492, Los Alamitos, California, November 1998. IEEE.

[Mer89] R. C. Merkle. A certified digital signature. In Brassard [Bra89], pages 218–238.

[Mic] Silvio Micali. CS proofs. SIAM Journal on Computing, to appear.

[MR01] S. Micali and L. Reyzin. Soundness in the public-key model. Unpublished manuscript, 2001.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science*, pages 458–467, Miami Beach, Florida, 20–22 October 1997. IEEE.

[RK99] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In Jacques Stern, editor, *Advances in Cryptology—*

EUROCRYPT '99, volume 1592 of *Lecture Notes in Computer Science*, pages 415–431. Springer-Verlag, 2–6 May 1999.

A Merkle Trees

The description below is almost verbatim from [Mic].

Recall that a binary tree is a tree in which every node has at most two children, hereafter called the *0-child* and the *1-child*. A *Merkle tree* [Mer89] with security parameter n is a binary tree whose nodes store values, some of which are computed by means of a collision-free hash function $H : \{0,1\}^* \to \{0,1\}^n$ in a special manner. A leaf node can store any value, but each internal node should store a value that is the one-way hash of the concatenation of the values in its children. That is, if an internal node has a 0-child storing the value u and a 1-child storing a value v, then it stores the value $H(u \circ v)$. Thus, because H produces n-bit outputs, each internal node of a Merkle tree, including the root, stores an n-bit value. Except for the root value, each value stored in a node of a Merkle tree is said to be a 0-value, if it is stored in a node that is the 0-child of its parent, a 1-value otherwise.

The crucial property of a Merkle tree is that, unless one succeeds in finding a collision for H, *it is computationally hard to change any value in the tree (and, in particular, a value stored in a leaf node) without also changing the root value.* This property allows a party A to commit to L values, $v_1, \ldots, v_L$ (for simplicity assume that L is a power of 2 and let $d = \log L$), by means of a single n-bit value. That is, A stores value v_i in the i-th leaf of a full binary tree of depth d, and uses a collision-free hash function H to build a Merkle tree, thereby obtaining an n-bit value, R, stored in the root. This root value R "implicitly defines" what the L original values were. Assume in fact that, as some point in time, A gives R, but not the original values, to another party B. Then, whenever, at a later point in time, A wants to "prove" to B what the value of, say, v_i was, A may just reveal all L original values to B, so that B can recompute the Merkle tree and the verify that the newly computed root-value indeed equals R. More interestingly, A may "prove" what v_i was by revealing just $d+1$ (that is, just $1 + \log L$) values: v_i together with its *authenticating path*, that is, the values stored in the siblings of the nodes along the path from leaf i (included) to the root (excluded), $w_1, \ldots, w_d$. Party B verifies the received alleged leaf-value v_i and the received alleged authenticating path $w_1, \ldots, w_d$ as follows. She sets $u_1 = v_i$ and, letting $i_1, \ldots, i_d$ be the binary expansion of i, computes the values $u_2, \ldots, u_d$ as follows: if $i_j = 0$, she sets $u_{j+1} = H(w_j \circ u_j)$; else, she sets $u_{j+1} = H(u_j \circ w_j)$. Finally, B checks whether the computed n-bit value u_d equals R.

B Pseudorandom Functions

A pseudorandom function family, introduced by Goldreich, Goldwasser and Micali [GGM86] is a keyed family of efficiently computable functions, such that a function picked at random from the family is indistinguishable (via oracle access) from a truly random function with the same domain and range. More formally, let $\mathrm{PRF}(\cdot,\cdot) : \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$ be an efficiently computable function. Our definition below is quite standard, except that it requires security against subexponentially strong adversaries.

Definition 6. *We say that* PRF *is a* pseudorandom function *if* $\exists\ \alpha > 0$ *such that for all sufficiently large* n *and all* 2^{n^α}*-gate adversaries* ADV*, the following difference is negligible in* n*:*

$$\mathrm{PROB}[\mathit{PRFKey} \overset{R}{\leftarrow} \{0,1\}^n : \mathrm{ADV}^{\mathrm{PRF}(\mathit{PRFKey},\cdot)} = 1] - \\ \mathrm{PROB}[F \overset{R}{\leftarrow} (\{0,1\}^n)^{\{0,1\}^n \times \{0,1\}^*} : \mathrm{ADV}^{F(\cdot)} = 1]$$

We call α *the* pseudorandomness constant.

Pseudorandom functions can be constructed based on a variety of assumption. We refer the reader to [GGM86,NR97] (and references therein) for details.