

Soundness 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. The public-key model for interactive proofs has proved to be quite effective in improving protocol efficiency [CGGM00]. We argue, however, that its soundness notion is more subtle and complex than in the classical model, and that it should be better understood to avoid designing erroneous protocols. Specifically, for the public-key model, we

- identify four *meaningful* notions of soundness;
- prove that, under *minimal* complexity assumptions, these four notions are *distinct*;
- identify *the exact soundness notions* satisfied by prior interactive protocols; and
- identify the *round complexity* of some of the new notions.

1 Introduction

THE BARE PUBLIC-KEY MODEL FOR INTERACTIVE PROOFS. A novel protocol model, which we call the *bare public-key (BPK) model*, was introduced by Canetti, Goldreich, Goldwasser and Micali in the context of resettable zero-knowledge [CGGM00]. Although introduced with a specific application in mind, the BPK model applies to interactive proofs in general, regardless of their knowledge complexity. The model simply assumes that the verifier 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 him.

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. 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.)

Despite its apparent simplicity, the BPK model is quite powerful. While resettable zero-knowledge (RZK) protocols exist both in the standard and in the BPK models [CGGM00], only in the latter case can they be constant-round, at least in a black box sense (even the weaker notion of concurrent zero knowledge [DNS98] is not black-box implementable in a constant number of rounds [CKPR01]). Indeed, the BPK model was introduced precisely to improve the round efficiency of RZK protocols.

THE PROBLEM OF SOUNDNESS IN THE BARE PUBLIC-KEY MODEL. Despite its simple mechanics, we argue that the soundness property of the bare public-key model has not been understood, and indeed is more complex than in the classical case.

In the classical model for interactive proofs, soundness can be defined quite easily: essentially, there should be no efficient malicious prover \mathcal{P}^* that can convince \mathcal{V} of the verity of a false statement with non-negligible probability. This simple definition suffices regardless of whether \mathcal{P}^* interacts with the verifier only once, or several times in a sequential manner, or several times in a concurrent manner. The reason for this sufficiency is that, in the standard model, \mathcal{V} is polynomial-time and has no “secrets” (i.e., all of its inputs are known to \mathcal{P}^*). Thus, if there were a \mathcal{P}^* successful “against a multiplicity of verifiers,” then there would also be a malicious prover successful against a single verifier \mathcal{V} : it would simply let \mathcal{P}^* interact with \mathcal{V} while “simulating all other verifiers.”

In the BPK model, however, \mathcal{V} has a secret key SK , corresponding to its public key PK . Thus, \mathcal{P}^* could potentially gain some knowledge about SK from an interaction with \mathcal{V} , and this gained knowledge might help \mathcal{P}^* to convince \mathcal{V} of a false theorem in a subsequent interaction. Therefore,

in the BPK model, the soundness property may be affected by the type of interaction a malicious prover is entitled to have with the verifier, as well as the sheer number of these interactions.

In addition, other totally new issues arise in the BPK model. For example, should \mathcal{P}^* be allowed to determine the exact false statement of which it tries to convince \mathcal{V} before or after it sees PK ? Should \mathcal{P}^* be allowed to change that statement after a few interactions with \mathcal{V} ?

In sum, an increased use of the BPK model needs to be coupled with a better understanding of its soundness properties in order designing protocols that are unsound (and thus insecure) or “too sound” (and thus, potentially, less efficient than otherwise possible). This is indeed the process we start in this paper.

FOUR NOTIONS OF SOUNDNESS IN THE BARE PUBLIC-KEY MODEL. Having identified the above issues, we formalize four *meaningful* notions of soundness in the BPK model. (These notions correspond in spirit to the commonly used notions of zero knowledge in the standard model. That is, the ways in which a malicious prover is allowed to interact with the honest verifier correspond to

those in which a malicious verifier is allowed to interact with the honest prover in various notions of zero knowledge.) Roughly speaking, here are the four notions, each of which implies the previous one:

1. **one-time soundness**, when \mathcal{P}^* is allowed a single interaction with \mathcal{V} per theorem statement;
2. **sequential soundness**, when \mathcal{P}^* is allowed multiple but sequential interactions with \mathcal{V} ;
3. **concurrent soundness**, when \mathcal{P}^* is allowed multiple interleaved interactions with the same \mathcal{V} ; and
4. **resettable soundness**, when \mathcal{P}^* is allowed to reset \mathcal{V} with the same random tape *and* interact with it concurrently.

All four notions are meaningful. Sequential soundness (the notion implicitly used in [CGGM00]) is certainly a very natural notion, and concurrent and resettable soundness are natural extensions of it. As for one-time soundness, it is also quite meaningful when it is possible to enforce that a prover who fails to convince the verifier of the verity of a given statement S does not get a second chance at proving S . (E.g., the verifier may memorize the theorem statements for which the prover failed; or make suitable use of timestamps.)

These four notions of soundness apply both to interactive proofs (where a malicious prover may have unlimited power [GMR89]) and argument systems (where a malicious prover is restricted to polynomial time [BCC88]).

SEPARATING THE FOUR NOTIONS. We prove that the above four notions are not only meaningful, but also *distinct*. Though conceptually important, these separations are technically simple. They entail exhibiting three protocols, each satisfying one notion but not the next one; informally, we prove the following theorems.

Theorem 1. *If one-way functions exist, there is a compiler-type algorithm that, for any language L , and any interactive argument system for L satisfying one-time soundness, produces another interactive argument system for the same language L that satisfies one-time soundness but not sequential soundness.*

Theorem 2. *If one way functions exist, there is a compiler-type algorithm that, for any language L , and any argument system for L satisfying sequential soundness, produces another argument system for the same language L that satisfies sequential soundness but not concurrent soundness.*

Theorem 3. *There exists a compiler-type algorithm that, for any language L , and any interactive proof (or argument) system for L satisfying concurrent soundness, produces another interactive proof (respectively, argument) system for the same language L that satisfies concurrent soundness but not resettable soundness.*

Note that our separation theorems hold with complexity assumptions that are indeed minimal: the third theorem holds *unconditionally*; while the first and

second rely only on the existence of one-way functions. (This is why Theorems 1 and 2 only hold for bounded provers).

Realizing that there exist separate notions of soundness in the BPK model is crucial to avoid errors. By relying on a single, undifferentiated, and intuitive notion of soundness, one might design a BPK protocol sound in settings where malicious provers are limited in their interactions, while someone else might erroneously use it in settings where malicious provers have greater powers.

THE EXACT SOUNDNESS OF PRIOR PROTOCOLS IN THE BPK MODEL. Having realized that there are various notions of soundness and that it is important to specify which one is satisfied by any given protocol, a natural question arises: *what type of soundness is actually enjoyed by the already existing protocols in the BPK model?*

There are right now two such protocols: the original RZK argument proposed in [CGGM00] and the 3-round RZK argument of [MR01] (the latter holding in a BPK model with a counter). Thus we provide the following answers:

1. *The CGGM protocol is sequentially sound, and probably no more than that.* That is, while it is sequentially sound, we provide evidence that it is NOT concurrently sound.
2. *The MR protocol is exactly concurrently sound.* That is, while it is concurrently sound, we prove that it is NOT resettably sound.
(As we said, the MR protocol works in a stronger public-key model, but all our notions of soundness easily extend to this other model.)

THE ROUND COMPLEXITY OF SOUNDNESS IN THE BPK MODEL. Since we present four notions of soundness, each implying the previous one, one may conclude that only the last one should be used. However, we shall argue that achieving a stronger notion of soundness requires using more rounds. Since rounds perhaps are the most expensive resource in a protocol, our lowerbounds justify using weaker notions of soundness whenever possible.

To begin with, we adapt an older lowerbound of [GK96] to prove the following theorem.

Theorem 4. *Any (resettable or not) black-box ZK protocol satisfying concurrent soundness in the BPK model for a language L outside of BPP requires at least four rounds.*

However, whether such an RZK protocol exists remains an open problem. A consequence of the above lowerbound is that, in any application in which four rounds are deemed to be too expensive, one needs either to adopt a stronger model (e.g., the public-key model with counter of [MR01]), or to settle for 3-round protocols satisfying a weaker soundness property¹. We thus provide such a protocol; namely, we prove the following theorem.

¹ It is easy to prove that one cannot obtain fewer rounds than three, using the theorem from [GO94] stating that, in the standard model, 2-round auxiliary-input ZK is impossible for non-trivial languages.

Theorem 5. *Assuming the security of RSA with large prime exponents against subexponentially-strong adversaries, for any $L \in \text{NP}$, there exists a 3-round black-box RZK protocol in the BPK model that possesses one-time, but not sequential, soundness.*

Whether the BPK model allows for 3-round, sequentially sound, ZK protocols remains an open problem. It is known that four rounds suffice in the standard model for ZK protocols [FS89], and therefore also in the BPK model. However, in the following theorem we show that in the BPK model four rounds suffice even for resettable ZK.

Theorem 6. *Assuming there exist certified trapdoor permutation families² secure against subexponentially-strong adversaries, for any $L \in \text{NP}$, there exists a 4-round black-box RZK protocol in the BPK model that possesses sequential soundness.*

2 Four Notions of Soundness

Note: For the sake of brevity, in this section we focus exclusively on arguments, rather than proofs (i.e., the malicious prover is limited to polynomial time, and soundness is computational). All the currently known examples of protocols in the BPK model are arguments anyway, because they enable a malicious prover to cheat if it can recover the secret key SK from the public key PK . Our definitions, however, can be straightforwardly modified for proofs. (Note that the BPK model does not rule out interactive proofs: in principle, one can make clever use of a verifier public key that has no secrets associated with it.)

In this section, we formally define soundness in the BPK model, namely that a malicious prover should be unable to get the verifier to accept a false statement.³ For the sake of brevity, we focus only on soundness. The notions of completeness (which is quite intuitive) and resettable zero-knowledgeness (previously defined in [CGGM00]) are provided in Appendix A

The Players

Before providing the definitions, we need to define the parties to the game: the honest \mathcal{P} and \mathcal{V} and the various malicious impostors. Let

² A trapdoor permutation family is *certified* if it is easy to verify that a given function belongs to the family.

³ It is possible to formalize the four notions of soundness by insisting that the verifier give zero knowledge to the (one-time, sequential, concurrent or resetting) malicious prover. This would highlight the correspondence of our soundness notions to the notions of zero-knowledge, and would be simpler to define, because the definitions of zero-knowledge are already well established. However, such an approach is an overkill, and would result in unnecessarily restrictive notions of soundness in the BPK model: we do not care if the prover gains knowledge so long as the knowledge does not allow the prover to cheat.

- 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.
- An (*honest*) *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 .
- An (*honest*) *verifier* \mathcal{V} be an interactive deterministic polynomial-time TM that works in two stages. In stage one (the *key-generation* stage), on input a security parameter 1^n and random tape r , \mathcal{V} outputs a public key PK and the corresponding secret key SK . In stage two (the *verification* stage), on input SK , and n -bit string x and a random string ρ , \mathcal{V} performs an interactive protocol with a prover, and outputs “accept x ” or “reject x .” For simplicity of exposition, fixing SK and ρ , one can view the verification stage of \mathcal{V} as a *non-interactive* TM that is given x and the entire history of the messages already received in the interaction, and outputs the next message to be sent, or “accept x ”/“reject x .” This view allows one to think of $\mathcal{V}(SK, \rho)$ as a simple deterministic oracle, which is helpful in defining the notion of resettable soundness below (however, we will use the interactive view of \mathcal{V} in defining one-time, sequential and concurrent soundness).
- A *s-sequential malicious prover* \mathcal{P}^* for a positive polynomial s be a probabilistic polynomial-time TM that, on first input 1^n , runs in at most $s(n)$ stages, so that
 1. In stage 1, \mathcal{P}^* receives a public key PK and outputs a string x_1 of length n .
 2. In every even stage, \mathcal{P}^* starts in the final configuration of the previous stage and performs a single interactive protocol: it outputs outgoing messages and receives incoming messages (the machine with which it performs the interactive protocol will be specified below, in the definition of sequential soundness). It can choose to abort an even stage at any point and move on to the next stage by outputting a special message.
 3. In every odd stage $i > 1$, \mathcal{P}^* starts in the final configuration of the previous stage and outputs a string x_i of length n .
- An *s-concurrent malicious prover* \mathcal{P}^* , for a positive polynomial s , be a probabilistic polynomial-time TM that, on inputs 1^n and PK , performs at most $s(n)$ interactive protocols as follows:
 1. If \mathcal{P}^* is already running $i - 1$ interactive protocols $1 \leq i - 1 < s(n)$, it can output a special message “Start x_i ,” where x_i is a string of length n .
 2. At any point it can output a message for any of its (at most $s(n)$) interactive protocols (the protocol is unambiguously identified in the outgoing message). It then immediately receives the party’s response and continues.
- An *s-resetting malicious prover* \mathcal{P}^* , for a positive polynomial s , be a probabilistic polynomial-time TM that, on inputs 1^n and PK , gets access to $s(n)$ oracles for the verifier (to be precisely specified below, in the definition of resettable soundness).

The Definitions

A pair $(\mathcal{P}, \mathcal{V})$ can satisfy one or more of the four different notions of soundness defined below. We note that each subsequent notion trivially implies the previous one.

For the purposes of defining one-time and sequential soundness, we consider the following procedure for a given s -sequential malicious prover \mathcal{P}^* , a verifier \mathcal{V} and a security parameter n .

Procedure Sequential-Attack

1. Run the key-generation stage of \mathcal{V} on input 1^n and a random string r to obtain PK, SK .
2. Run first stage of \mathcal{P}^* on inputs 1^n and PK to obtain an n -bit string x_1 .
3. For i ranging from 1 to $s(n)/2$:
 - 3.1 Select a random string ρ_i .
 - 3.2 Run the $2i$ -th stage of \mathcal{P}^* , letting it interact with the verification stage of \mathcal{V} with input SK, x_i, ρ_i .
 - 3.3 Run the $(2i + 1)$ -th stage of \mathcal{P}^* to obtain an n -bit string x_i .

Definition 1. $(\mathcal{P}, \mathcal{V})$ satisfies one-time soundness for a language L if for all positive polynomials s , for all s -sequential malicious provers \mathcal{P}^* , the probability that in an execution of Sequential-Attack, there exists i such that $1 \leq i \leq s(n)$, $x_i \notin L$, $x_j \neq x_i$ for all $j < i$ and \mathcal{V} outputs “accept x_i ” is negligible in n .

Sequential soundness differs from one-time soundness only in that the malicious prover is allowed to have $x_i = x_j$ for $i < j$.

Definition 2. $(\mathcal{P}, \mathcal{V})$ satisfies sequential soundness for a language L if for all positive polynomials s , for all s -sequential malicious provers \mathcal{P}^* , the probability that in an execution of Sequential-Attack, there exists i such that $1 \leq i \leq s(n)$, $x_i \notin L$, and \mathcal{V} outputs “accept x_i ” is negligible in n .

For the purposes of defining concurrent soundness, we consider the following procedure for a given s -concurrent malicious prover \mathcal{P}^* , a verifier \mathcal{V} and a security parameter n .

Procedure Concurrent-Attack

1. Run the key-generation stage of \mathcal{V} on input 1^n and a random string r to obtain PK, SK .
2. Run \mathcal{P}^* on inputs 1^n and PK .
3. Whenever \mathcal{P}^* outputs “Start x_i ,” select a fresh random string ρ_i and let the i -th machine with which \mathcal{P}^* interacts be the verification stage of \mathcal{V} on inputs SK, x_i, ρ_i .

Of course, the multiple instances of \mathcal{V} are “unaware” and independent of each other, because they are started with fresh random strings.

Definition 3. $(\mathcal{P}, \mathcal{V})$ satisfies concurrent soundness for a language L if for all positive polynomials s , for all s -concurrent malicious provers \mathcal{P}^* , the probability that in an execution of Concurrent-Attack, \mathcal{V} ever outputs “accept x ” for $x \notin L$ is negligible in n .

Finally, for the purposes of defining resettable soundness, we consider the following procedure for a given s -resetting malicious prover \mathcal{P}^* , a verifier \mathcal{V} and a security parameter n .

Procedure Resetting-Attack

1. Run the key-generation stage of \mathcal{V} on input 1^n and a random string r to obtain PK, SK .
2. Run \mathcal{P}^* on inputs 1^n and PK .
3. Generate $s(n)$ random strings ρ_i for $1 \leq i \leq s(n)$.
4. Let \mathcal{P}^* interact with oracles for the second stage of the verifier, the i -th oracle having input SK, ρ_i .

Note that concurrent soundness and resettable soundness differ in one crucial aspect: for the former, every instance of \mathcal{V} is an interactive TM that keeps state between rounds of communication, and thus cannot be rewound; whereas for the latter, every instance of \mathcal{V} is just an oracle, and thus can effectively be rewound.

Definition 4. $(\mathcal{P}, \mathcal{V})$ satisfies resettable soundness for a language L if for all positive polynomials s , for all s -resetting malicious provers \mathcal{P}^* , the probability that in an execution of Resetting-Attack, \mathcal{P}^* ever receives “accept x ” for $x \notin L$ from any of the oracles is negligible in n .

3 Separating the Four Notions

The Common Idea

Given a protocol $(\mathcal{P}, \mathcal{V})$ that satisfies the i -th soundness notion (for $i = 1, 2$, or 3), we deliberately weaken the verifier to come up with a protocol $(\mathcal{P}', \mathcal{V}')$ that does not satisfy the $(i+1)$ -th soundness notion, but still satisfies the i -th. In each case, we add rounds at the beginning of the $(\mathcal{P}, \mathcal{V})$ (and sometimes information to the keys) that have nothing to do with the language or the theorem being proven. At the end of these rounds, either \mathcal{V}' accepts, or $(\mathcal{P}', \mathcal{V}')$ proceed with the protocol $(\mathcal{P}, \mathcal{V})$. In each case, it will be easy for a malicious prover for the $(i+1)$ -th notion of soundness to get \mathcal{V}' to accept at the end of these additional rounds.

To prove that the resulting protocol $(\mathcal{P}', \mathcal{V}')$ still satisfies the i -th notion of soundness, it will suffice to show that if a malicious prover \mathcal{P}'^* for $(\mathcal{P}', \mathcal{V}')$ exists, then it can be used to construct a malicious prover \mathcal{P}^* for $(\mathcal{P}, \mathcal{V})$. In each case, this is easily done: \mathcal{P}^* simply simulates the additional rounds to \mathcal{P}'^* (one also has to argue that \mathcal{V}' interacting with \mathcal{P}'^* is unlikely to accept during these additional rounds).

Finally, to ensure that zero-knowledgeness of $(\mathcal{P}, \mathcal{V})$ is not affected, during the additional rounds the honest \mathcal{P}' will simply send some fixed values to \mathcal{V}' and disregard the values sent by \mathcal{V}' .

Each of the subsections below described the specific additional information in the keys and the additional rounds. We do not provide the details of proofs, as they can be easily derived from the discussion above.

Proof of Theorem 1

Let F be a pseudorandom function [GGM86]; we denote by $F_s(x)$ the output of F with seed s on input x . Note that such functions exist assuming one-way functions exist [HILL99]. Let x denote the theorem that the prover is trying to prove to the verifier.

Add to Key Gen: Generate random n -bit seed s ; add s to the secret key SK .

Add \mathcal{P} Step: Set $\beta = 0$; send β to the verifier.

Add \mathcal{V} Step: If $\beta = F_s(x)$, accept and stop. Else send $F_s(x)$ to prover.

Note that a sequential malicious prover can easily get \mathcal{V}' to accept: it finds out the value of $F_s(x)$ in the first interaction, and sets $\beta = F_s(x)$ for the second. If, on the other hand, the malicious prover is not allowed to use the same x twice, then it cannot predict $F_s(x)$ before sending β , and thus cannot get \mathcal{V}' to accept.

Proof of Theorem 2

Let $(\text{SigKeyGen}, \text{Sign}, \text{Ver})$ be a signature scheme secure against adaptive chosen message attacks [GMR88]. Note that such a scheme exists assuming one-way functions exist [Rom90].

Add to Key Gen: Generate a key pair $(\text{SigPK}, \text{SigSK})$ for the signature scheme; add SigPK to the public key PK and SigSK to the secret key SK .

Add 1st \mathcal{P} Step: Set $M = 0$, and send M to the verifier.

Add 1st \mathcal{V} Step: 1. Send a signature s of M to the prover.
2. Let M' be random n -bit string; send M' to prover.

Add 2nd \mathcal{P} Step: Set $s' = 0$. Send s' to the verifier.

Add 2nd \mathcal{V} Step: If s' is a valid signature of M' , then accept and stop.

Note that a concurrent malicious prover can easily get \mathcal{V}' to accept. It starts a protocol with \mathcal{V}' , sends $M = 0$, receives M' from \mathcal{V} , and then pauses the protocol. During the pause, it starts a second protocol, and sends $M = M'$ to \mathcal{V}' to obtain a signature s of M' in first message from \mathcal{V}' . It then resumes the first protocol, and sends $s' = s$ to \mathcal{V}' as its second message, which \mathcal{V}' accepts.

Also note that a sequential malicious prover will most likely not be able to come up with a valid signature of M' , because of the signature scheme's security against adaptive chosen message attacks.

Proof of Theorem 3

Add \mathcal{P} Step: Set β be the string of n zeroes; send β to the verifier.

Add \mathcal{V} Step: Set α be a random string.

If $\beta = \alpha$, accept and stop. Else send α to the prover.

Note that a resetting malicious prover can easily get \mathcal{V}' to accept: it finds out the value of α in the first interaction, then resets \mathcal{V}' with the same random tape (and hence the same α , because α comes from \mathcal{V}' 's random tape) and sets $\beta = \alpha$ for the second interaction. A concurrent malicious prover, on the other hand, knows nothing about α when it determines β , and thus cannot get \mathcal{V}' to accept.

Note that this separation holds in the standard model as well—we never used the BPK model in this proof.

4 The “Exact” Soundness of Existing BPK Protocols

There are only two known protocols in the BPK model, the original one of [CGGM00] and the one of [MR01] (the latter actually working in a slightly stronger model). Thus we need to understand which notions of soundness they satisfy.

The CGGM Protocol is Sequentially but Probably Not Concurrently Sound

Although [CGGM00] did not provide formal definitions of soundness in the BPK model, their soundness proof essentially shows that their protocol is sequentially sound. However, let us (sketchily) explain why it will probably not be possible to prove their protocol concurrently sound.

The CGGM protocol begins with \mathcal{V} proving to \mathcal{P} knowledge of the secret key by means of parallel repetitions of a three-round proof of knowledge subprotocol. The subprotocol is as follows: in the first round, \mathcal{V} sends to \mathcal{P} a *commitment*; in the second round, \mathcal{P} sends to \mathcal{V} a one-bit *challenge*; in the third round, \mathcal{V} sends to \mathcal{P} a *response*. This is repeated k times in parallel in order to reduce the probability of \mathcal{V} cheating to roughly 2^{-k} .

In order to prove soundness against a malicious prover \mathcal{P}^* , these parallel repetitions of the subprotocol need to be simulated to \mathcal{P}^* (by a simulator that does not know the secret key). The best known simulation techniques for this general type of proof of knowledge run in time roughly 2^k . This exponential in k simulation time is not a concern, because of their use of “complexity leveraging” in the proof of soundness. Essentially, the soundness of their protocol relies on an underlying much harder problem: for instance, one that is assumed to take more than 2^{3k} time to solve. Thus, the soundness of the CGGM protocol is proved by contradiction: by constructing a machine from \mathcal{P}^* that runs in time $2^k < 2^{3k}$ and yet solves the underlying harder problem.

A concurrent malicious prover \mathcal{P}^* , however, may choose to run L parallel copies of \mathcal{V} . Thus, to prove soundness against such a \mathcal{P}^* , the proof-of-knowledge

subprotocol would have to be simulated Lk times in parallel, and this simulation would take roughly 2^{Lk} time. If $L > 3$, then we will not be able to solve the underlying hard problem in time less than 2^{3k} , and thus will not be able to derive any contradiction.

Thus, barring the emergence of a polynomial-time simulation for parallel repetitions of 3-round proofs of knowledge (or a dramatically new proof technique for soundness), the CGGM protocol is not provably concurrently sound.

The MR Protocol is Concurrently but Not Resettably Sound

The protocol in [MR01] extends the BPK model with a *counter*. Namely, there is an a-priori polynomial bound B that limits the total number of times the verifier executes the protocol, and the verifier maintains state information from one interaction to the next via a counter (that can be tested and incremented in a single atomic operation).

Our soundness notions easily extend to the MR model as well, and their soundness proof can be easily modified to yield that their protocol is concurrently sound in the new model. However, let us (sketchily) prove here that the MR protocol is not resettably sound.

In the MR protocol, verifier \mathcal{V} publishes a public key for a trapdoor commitment scheme, and then proves knowledge of the trapdoor using non-interactive zero-knowledge proof of knowledge (NIZKPK), relative to a jointly generated string σ . It is easy to see that in the MR protocol, if \mathcal{P}^* could learn \mathcal{V} 's trapdoor, then he could force \mathcal{V} to accept a false theorem. The knowledge-extraction requirement of the NIZKPK system guarantees that, by properly selecting σ , one could extract the trapdoor from the proof. Now, a malicious resetting prover \mathcal{P}^* has total control over σ . Indeed, in the MR protocol σ is the exclusive-or of two strings: $\sigma_{\mathcal{P}}$ provided by the prover in the first round, and $\sigma_{\mathcal{V}}$ provided by the verifier in the second round. Thus, \mathcal{P}^* simply finds out $\sigma_{\mathcal{V}}$ by running the protocol once, then resets \mathcal{V} and provides $\sigma_{\mathcal{P}}$ such that the resulting $\sigma = \sigma_{\mathcal{V}} \oplus \sigma_{\mathcal{P}}$ will equal the string that allows \mathcal{P}^* to extract the trapdoor.

5 The Cost of Soundness in Zero-Knowledge Proofs

The BPK model was introduced to save rounds in RZK protocols, but has itself introduced four notions of soundness. We have already shown that these notions are formally separated. Now, we show that they also have quite different algorithmic requirements: namely, stronger notions of soundness for ZK protocols require more rounds to be implemented. More precisely, we show a lowerbound, namely that concurrently sound black-box ZK requires four or more rounds, and two upperbounds, namely that one-time-sound RZK can be achieved in three rounds (which can be shown optimal using the standard-model lowerbound of [GO94]), and that sequential RZK can be achieved in four rounds.

Note that our lowerbound in the BPK model is not contradicted by the existence of the 3-round concurrently-sound protocol of [MR01], which is in a stronger model, where the verifier has a counter.

We derive our lowerbound in the BPK model, where there are different notions of soundness, from the older one of Goldreich and Krawczyk [GK96] for black-box ZK in the *standard* model, where one-time, sequential and concurrent soundness coincide. Thus, somehow, their proof can be extended to verifiers that have public and secret keys, though (as clear from our upperbound) *this extension fails to apply to some types of soundness*. This point is important to understanding soundness in the BPK model, and we’ll try to highlight it when sketching the lowerbound proof below.

Our bounds are not tight: we do not know whether 4-round concurrently sound RZK protocols exist, nor whether 3-round sequentially sound ZK protocols exist. Before our work, however, the gap was even wider: the CGGM — sequentially sound— RZK protocol had 8 rounds without preprocessing, though it could be easily reduced to 5 rounds.

5.1 No Concurrent Soundness for Black-Box ZK in Three Rounds

Theorem 4 *Any (resettable or not) black-box ZK protocol satisfying concurrent soundness in the BPK model for a language L outside of BPP requires at least four rounds.*

Proof Sketch. The Goldreich and Krawczyk’s proof that, for languages outside of BPP, there are no three-round protocols that are black-box zero-knowledge in the standard model, proceeds by contradiction. Assuming the existence of a black-box zero-knowledge simulator M , it constructs a BPP machine \bar{M} for L . Recall that M interacts with a verifier in order to output the verifier’s view. On input x , \bar{M} works essentially as follows: it simply runs M on input x , simulating a verifier to it. For this simulation, \bar{M} uses the algorithm of the honest verifier \mathcal{V} and the messages supplied by M , but ignores the random strings supplied by M and uses its own random strings (if the same message is given twice by M , then \bar{M} uses the same random string—thus making the verifier appear deterministic to M). If the view that M outputs at the end is accepting, then \bar{M} concludes that $x \in L$. Otherwise, it concludes that $x \notin L$.

To show that \bar{M} is a BPP machine for L , Goldreich and Krawczyk demonstrate two statements: that if $x \in L$, \bar{M} is likely to output an accepting conversation, and that if $x \notin L$, \bar{M} is unlikely to output an accepting conversation. The first statement follows because, by zero-knowledgeness, M ’s output is indistinguishable from the view generated by the true prover and the true verifier on input x , and, by completeness, this view is accepting. The second statement follows from soundness: if \bar{M} can output an accepting conversation for $x \notin L$, then one can construct a malicious prover \mathcal{P}^* that can convince \mathcal{V} of the false statement “ $x \in L$.” Such a \mathcal{P}^* needs in essence to “execute \bar{M} ” and simply let it interact with \mathcal{V} .

Having \mathcal{P}^* execute \bar{M} requires some care. At first glance, because simulator M is capable of resetting the verifier, it would seem that, in order to execute \bar{M} , also \mathcal{P}^* should have this capability. However, for 3-round protocols only, [GK96] show that

- (*) \mathcal{P}^* can execute M without resetting \mathcal{V} , so long as it has one-time access to \mathcal{V} .

Notice that by the term “one-time access” we make retroactive use of our modern terminology: [GK96] make no mention of one-time provers, because they work in the standard model. However, this terminology allows us to separate their proof of (*) into two distinct steps:

- (*') \mathcal{P}^* can execute M so long as it has concurrent access to \mathcal{V} ; and
 (*'') losing only a polynomial amount of efficiency, concurrent access to \mathcal{V} is equivalent to one-time access.

Tedious but straightforward analysis shows that (*') and the rest of their proof — except for (*'') — carries through in the BPK model (where the 3-round protocol is modified to include verifier key generation, and public and secret verifier keys are then involved). Step (*''), however, only holds in the standard model (where, as we pointed out, one-time, sequential and concurrent soundness coincide).

In sum, therefore, once verifier keys are introduced, one is left with a concurrent prover. \square

5.2 One-Time Sound RZK in Three Rounds

Theorem 5 *Assuming the security of RSA with large prime exponents against subexponentially-strong adversaries, for any $L \in \text{NP}$, there exists a 3-round black-box RZK protocol in the BPK model that possesses one-time, but not sequential, soundness.*

Proof Sketch. The proof of the theorem is constructive: we demonstrate such a protocol $(\mathcal{P}, \mathcal{V})$.

BASIC TOOLS. The protocol $(\mathcal{P}, \mathcal{V})$ relies on three techniques: a pseudorandom function PRF [GGM86], a verifiable random functions VRF [MRV99], and a non-interactive zero-knowledge (NIZK) proof system (NIP, NIV) [BFM88, BDMP91]. Note that both PRFs and NIZKs can be constructed using general assumptions [HILL99, FLS99], and it is only for VRFs that we need the specific RSA assumption (which is formally stated in Appendix B.3).

The definitions of NIZKs and VRFs are recalled recalled in Appendix B. Here we briefly introduce the notation:

- The keys $VRFPK$, $VRFSK$ for VRF are produced by VRFGen. The evaluation is performed by VRFEval, and the proof is computed by VRFProve. The verification is performed by VRFVer.
- The NIZK proof with security parameter n requires a shared random string σ of length $\text{NI}\sigma\text{Len}(n)$. The proof is computed by NIP and verified by NIV. The shared string and the proof together can be simulated by NIS.

The construction works for any language L for which an NIZK proof system exists, and, therefore, for all of NP.

This construction also uses “complexity leveraging” [CGGM00], although in a somewhat unusual way. Namely, let α be the pseudorandomness constant for VRF (that is, the output of the VRF Eval is indistinguishable from random for circuits of size 2^{k^α} , where k is VRF the security parameter). Let γ_1 be the following constant: for all sufficiently large n , the length of the NIZK proof Π for $x \in L$ of length n is upper bounded by n^{γ_1} . Let γ_2 be the following constant: for all sufficiently large n , the length of the NP-witness y for $x \in L$ of length n is upper bounded by n^{γ_2} . We then set $\gamma = \max(\gamma_1, \gamma_2)$, and $\epsilon > \gamma/\alpha$. We use NIZK with security parameter n and VRF with a (larger) security parameter $k = n^\epsilon$. This ensures that one can enumerate all potential NIZK proofs Π , or all potential NP-witnesses y , in time 2^{n^γ} , which is less than the time it would take to break the residual pseudorandomness of VRF (because $2^{n^\gamma} < 2^{k^\alpha}$).

THE PROTOCOL. For a security parameter n , \mathcal{V} generates a key pair for the VRF with output length $\text{NI}\sigma\text{Len}(n)$ and security parameter $k = n^\epsilon$. VRFSK is \mathcal{V} 's secret key, and VRFPK is \mathcal{V} 's public key.

Public File: A collection F of records (id, VRFPK_{id}) , where VRFPK_{id} is allegedly the output of $\text{VRF Gen}(1^k)$

Common Input: An element $x \in L$

\mathcal{P} Private Input: The NP-witness y for $x \in L$; \mathcal{V} 's id and the file F ;
a random string ω

\mathcal{V} Private Input: A secret key SK

\mathcal{P} Step One:

1. Using the string ω as a seed for PRF, generate a string $\sigma_{\mathcal{P}}$ of length $\text{NI}\sigma\text{Len}(n)$ from the inputs x, y and id .
2. Send $\sigma_{\mathcal{P}}$ to \mathcal{V} .

\mathcal{V} Step One:

1. Compute a string $\sigma_{\mathcal{V}}$ of length $\text{NI}\sigma\text{Len}(n)$ as $\sigma_{\mathcal{V}} = \text{VRF Eval}(\text{VRFSK}, x)$, and the VRF proof $pf = \text{VRF Prove}(\text{VRFSK}, x)$.
2. Send $\sigma_{\mathcal{P}}$ and pf to \mathcal{P} .

\mathcal{P} Step Two:

1. Verify that $\sigma_{\mathcal{V}}$ is correct by invoking $\text{VRF Ver}(\text{VRFPK}, x, \tau, pf)$. If not, abort.
2. Let $\sigma = \sigma_{\mathcal{V}} \oplus \sigma_{\mathcal{P}}$. Using $\text{NIP}(\sigma, x, y)$, compute and send to \mathcal{V} the proof Π of the statement “ $x \in L$.”

\mathcal{V} Step Two:

1. Let $\sigma = \sigma_{\mathcal{V}} \oplus \sigma_{\mathcal{P}}$. Using $\text{NIV}(\sigma, x, \Pi)$, verify if Π is valid. If so, accept. Else reject.

As far as we know, the above protocol is the first application of VRFs. The very strong properties of this new tool yield surprisingly simple proofs of one-time soundness and resettable zero-knowledgeness.

COMPLETENESS AND RZK. As usual, completeness of our protocol is easily verified. The RZK property can be shown in a way similar to (and simpler than)

the way is shown in [CGGM00] and [MR01]. One simply builds an RZK simulator who finds out $\text{VRF Eval}(VRF SK, x)$ for every pair $(VRF PK, x)$ that \mathcal{V}^* is likely to input to \mathcal{P} , and then rewinds and uses the NIZK simulator $\text{NIS}(x)$ just like the sequential malicious prover described above.

SOUNDNESS. First of all, note that soundness of our protocol is provably not sequential, because $\sigma_{\mathcal{V}}$ depends only on the input x , and hence will repeat if \mathcal{V} is run with the same x twice. Thus, once a sequential malicious prover \mathcal{P}^* knows $\sigma_{\mathcal{V}}$, it can run the NIZK simulator $\text{NIS}(x)$ to obtain (σ', Π') , restart with the same x , and use $\sigma'_{\mathcal{P}} = \sigma' \oplus \sigma_{\mathcal{V}}$ as its first message and Π' as its second message.

To show one-time soundness, first assume (for simplicity) that \mathcal{P}^* interacts with \mathcal{V} only once (we will deal with the general case later). Then we will construct a machine $T = (T_J, T_E)$ to break the residual pseudorandomness of the VRF (see the definition of VRF in Appendix B). Namely, given the public key $VRF PK$ of a VRF with security parameter k , T_J runs the first stage of \mathcal{P}^* on input $VRF PK$ to receive a string x . It then checks if $x \in L$ by simply enumerating all potential NP witnesses y in time $2^{n^{72}}$. If it is, then T_J outputs $(x, state)$, where $state = 0$. Otherwise, it runs the second stage of \mathcal{P}^* to receive $\sigma_{\mathcal{P}}$, and outputs $(x, state)$, where $state = (x, \sigma_{\mathcal{P}})$.

Now, T_E receives v , and T_E 's job is to find out whether v is a random string or $\text{VRF Eval}(VRF SK, x)$. If $state = 0$, then T_E simply guesses at random. Otherwise, $state = (x, \sigma_{\mathcal{P}})$. Let $\sigma = \sigma_{\mathcal{P}} \oplus v$. If v is a random string, then σ is also random, so most likely there is no NIZK proof Π of the statement “ $x \in L$ ” with respect to σ (by soundness of the NIZK proof system). Otherwise, $v = \sigma_{\mathcal{V}}$, so, if \mathcal{P}^* has a better than negligible probability of success, then there is a better than negligible probability that Π exists with respect to σ . Thus, T_E simply searches whether a proof Π exists (in time $2^{n^{71}}$) to determine whether v is random or the output of VRF Eval .

Complexity leveraging is crucial here: we are using the fact that the VRF is “stronger” than the non-interactive proof system. Otherwise, the output of VRF Prove (which the prover gets, but T does not) could help a malicious prover find Π . By using a stronger VRF, we are ensuring that such Π will most likely not even exist.

Now we address the general case, when \mathcal{P}^* is allowed $s(n)$ sequential interactions with \mathcal{V} , and wins if \mathcal{V} accepts at least one of them (say, the i -th one) for $x_i \notin L$. Then T_J simply guesses, at random, the conversation number i for which \mathcal{P}^* will succeed, and simulates conversations before the i -th one by querying VRF Eval and VRF Prove on x_j for $j < i$ (it is allowed to do so, because, in one-time soundness, $x_j \neq x_i$). \square

5.3 Sequentially Sound RZK in Four Rounds

Theorem 6 *Assuming there exist certified trapdoor permutation families⁴ secure against subexponentially-strong adversaries, for any $L \in \text{NP}$, there exists*

⁴ A trapdoor permutation family is *certified* if it is easy to verify that a given function belongs to the family.

a 4-round black-box RZK protocol in the BPK model that possesses sequential soundness.

Proof Sketch. The proof is, again, constructive. The construction is a modification of the CGGM protocol (which has 8 rounds, and can easily be modified to have 5 by combining the first three rounds with later rounds).

MAIN IDEAS. The CGGM protocol starts with a three-round proof of knowledge subprotocol in which \mathcal{V} proves to \mathcal{P} knowledge of the secret key. After that, \mathcal{P} proves to \mathcal{V} that a graph is three-colorable using a five-round protocol.

Our main idea is to replace the five-round protocol with a single round using non-interactive zero-knowledge. The first three rounds are then used both for the proof of knowledge and for agreeing on a shared random auxiliary string σ needed for the NIZK proof. To agree on σ , \mathcal{V} sends to \mathcal{P} an encryption of a random string $\sigma_{\mathcal{V}}$, \mathcal{P} sends to \mathcal{V} its own random string $\sigma_{\mathcal{P}}$, and then \mathcal{V} reveals $\sigma_{\mathcal{V}}$ (and the coins used to encrypt it). The string σ is computed as $\sigma_{\mathcal{P}} \oplus \sigma_{\mathcal{V}}$.

Thus, \mathcal{V} 's key pair is simply a key pair for an encryption scheme. The protocol is zero-knowledge essentially for the same reasons that the CGGM protocol is zero-knowledge: because the simulator can extract the decryption key from the proof of knowledge and thus find out $\sigma_{\mathcal{V}}$ before needing to submit $\sigma_{\mathcal{P}}$. This will allow it to select σ as it wishes and thus use the NIZK simulator.

The protocol is sequentially sound because if the theorem is false, then with respect to only a negligible portion of the possible strings σ does a NIZK proof of the theorem exist. Thus, if a malicious prover \mathcal{P}^* , after seeing only an encryption of $\sigma_{\mathcal{V}}$, is able to come up with $\sigma_{\mathcal{P}}$ such that the NIZK proof exists with respect to the resulting string $\sigma = \sigma_{\mathcal{P}} \oplus \sigma_{\mathcal{V}}$, then one can use \mathcal{P}^* to break the security of the encryption scheme.

The computational assumption for our protocol follows from the fact that trapdoor permutations are sufficient for encryption [GM84, Yao82, GL89], certified trapdoor permutations are sufficient for NIZKs [FLS99], one-way permutations are sufficient for the proof of knowledge [Blu86] (which is the same as in the CGGM protocol) and one-way functions are sufficient for PRFs [HILL99].

DETAILS OF THE CONSTRUCTION. This construction, like the previous one, works for any languages L for which an NIZK proof system exists. Hence it works for all $L \in NP$.

The protocol below relies on parallel executions of three-round proofs of knowledge, which are performed in exactly the same way as in [CGGM00]. We also use “complexity leveraging,” in a way similar to our three-round one-time-sound construction. Namely, let α be the indistinguishability constant for the encryption scheme (that is, the encryptions of two different strings are indistinguishable from each other for circuits of size 2^{k^α} , where k is the security parameter). Let γ_1 be the following constant: for all sufficiently large n , the length of the NIZK proof Π for x of length n is upper bounded by n^{γ_1} . Let γ_2 be following constant: n parallel repetitions of the proof-of-knowledge subprotocol can be simulated in time less than $2^{n^{\gamma_2}}$. We then set $\gamma = \max(\gamma_1, \gamma_2)$, and $\epsilon > \gamma/\alpha$.

We use NIZK with security parameter n and perform n parallel repetitions of the proof-of-knowledge subprotocol, while the encryption scheme has a (larger) security parameter $k = n^\epsilon$. This ensures that one can enumerate all potential NIZK proofs Π and simulate the proof of knowledge subprotocol in time 2^{n^γ} , which is less than the time it would take to break the indistinguishability of the encryption scheme (because $2^{n^\gamma} < 2^{k^\alpha}$).

THE PROTOCOL. For a security parameter n , the verifier \mathcal{V} generates a pair $(EncPK, EncSK)$ of keys for the encryption scheme with security parameter $k = n^\epsilon$. $EncSK$ is \mathcal{V} 's secret key, and $EncPK$ is \mathcal{V} 's public key.

Public File: A collection F of records $(id, EncPK_{id})$, where $EncPK_{id}$ is allegedly the output of \mathcal{V} 's key generation

Common Inputs: An element $x \in L$

\mathcal{P} Private Input: The NP-witness y for $x \in L$; \mathcal{V} 's id and the file F ;
a random string ω

\mathcal{V} Private Input: A secret key $EncSK$; a random string ρ

\mathcal{V} Step One: 1. Generate a random string $\sigma_{\mathcal{V}}$ of length $NI\sigma Len(n)$.
2. Encrypt $\sigma_{\mathcal{V}}$, using a portion ρ_E of the input random string ρ , to get a ciphertext c . Send c to \mathcal{P} .
3. Generate and send to \mathcal{P} the first message of the n parallel repetitions of the proof of knowledge of $EncSK$.

\mathcal{P} Step One: 1. Using the input random string ω as a seed for PRF, generate a sufficiently long "random" string from the input to be used in the remaining computation by \mathcal{P} .
2. Generate and send to \mathcal{V} random string $\sigma_{\mathcal{P}}$ of length $NI\sigma Len(n)$.
3. Generate and send to \mathcal{V} the second message of the n parallel repetitions of the proof of knowledge of $EncSK$.

\mathcal{V} Step Two: 1. Send $\sigma_{\mathcal{V}}$ and the coins ρ_E used to encrypt it to \mathcal{P} .
2. Generate and send the third message of the n parallel repetitions of the proof of knowledge of $EncSK$.

\mathcal{P} Step Two: 1. Verify that $\sigma_{\mathcal{V}}$ encrypted with coins ρ_E produces ciphertext c .
2. Verify the n parallel repetitions proof of knowledge of $EncSK$.
3. If both verifications hold, let $\sigma = \sigma_{\mathcal{V}} \oplus \sigma_{\mathcal{P}}$. Using the NIZK prover $NIP(\sigma, x, y)$, compute and send to \mathcal{V} the proof Π of the statement " $x \in L$."

\mathcal{V} Step Three: Let $\sigma = \sigma_{\mathcal{V}} \oplus \sigma_{\mathcal{P}}$. Using the NIZK verifier $NIV(\sigma, x, \Pi)$, verify if Π is valid. If so, accept. Else reject.

COMPLETENESS AND RZK. Completeness of this protocol is, as usual, easily verified. The proof of resettable zero-knowledgeness is very similar to that of

[CGGM00]: once the simulator recovers SK from the proof of knowledge, it can find out $\sigma_{\mathcal{V}}$ before having to send $\sigma_{\mathcal{P}}$, and thus can run the NIZK simulator to get (σ, Π) and set $\sigma_{\mathcal{P}} = \sigma \oplus \sigma_{\mathcal{V}}$.

SOUNDNESS. Sequential soundness can be shown as follows. Suppose \mathcal{P}^* is a malicious prover that can make \mathcal{V} accept a false theorem with probability $p(n)$ (where the probability is taken over the coin tosses of the \mathcal{V} and \mathcal{P}^*). First, assume (for simplicity) that \mathcal{P}^* interacts with \mathcal{V} only once (we will deal with the general case of a sequential malicious prover later).

We will use \mathcal{P}^* to construct an algorithm A that breaks the encryption scheme. A is given, as input, the public key PK for the encryption scheme. Its job is to pick two strings τ_0 and τ_1 , receive an encryption of τ_b for a random bit b and tell whether $b = 0$ or $b = 1$. It picks τ_0 and τ_1 simply as random strings of length $\text{NI}\sigma\text{Len}(n)$. Let c be the encryption of τ_b . Then A publishes PK as its public key, runs the first stage of \mathcal{P}^* to receive x , and initiates a protocol with the second stage of \mathcal{P}^* .

In the first message, A sends c for the encryption of $\sigma_{\mathcal{V}}$ (for the proof-of-knowledge subprotocol, A uses the simulator, which runs in time $2^{n^{72}}$). It then receives $\sigma_{\mathcal{P}}$ from \mathcal{P}^* , computes $\sigma_i = \sigma_{\mathcal{P}} \oplus \tau_i$ and determines (by exhaustive search, which takes time $2^{n^{71}}$) if there exists an NIZK proof Π_i for the statement $x \in L$ with respect to σ_i (for $i = 0, 1$). If Π_i exists and Π_{1-i} does not, then A outputs $b = i$. If neither Π_0 nor Π_1 exists, or if both exist, then A outputs a random guess for b .

We now need to compute the probability that A correctly guessed b . Of course, by construction,

$$\Pr[A \text{ outputs } b] = \Pr[\exists \Pi_b \text{ and } \nexists \Pi_{1-b}] + \Pr[\exists \Pi_b \text{ and } \exists \Pi_{1-b}]/2 + \Pr[\nexists \Pi_b \text{ and } \nexists \Pi_{1-b}]/2.$$

Note that $\Pr[\exists \Pi_b \text{ and } \exists \Pi_{1-b}] + \Pr[\nexists \Pi_b \text{ and } \nexists \Pi_{1-b}] = 1 - (\Pr[\exists \Pi_b \text{ and } \nexists \Pi_{1-b}] + \Pr[\nexists \Pi_b \text{ and } \exists \Pi_{1-b}])$. Therefore,

$$\Pr[A \text{ outputs } b] = 1/2 - \Pr[\nexists \Pi_b \text{ and } \exists \Pi_{1-b}]/2 + \Pr[\exists \Pi_b \text{ and } \nexists \Pi_{1-b}]/2.$$

Note that the either of the events $\nexists \Pi_b$ and $\nexists \Pi_{1-b}$ can occur only if $x \notin L$, by completeness of the NIZK system. Therefore,

$$\begin{aligned} \Pr[A \text{ outputs } b] &= 1/2 - \Pr[\nexists \Pi_b \text{ and } \exists \Pi_{1-b} \text{ and } x \notin L]/2 + \\ &\quad \Pr[\exists \Pi_b \text{ and } \nexists \Pi_{1-b} \text{ and } x \notin L]/2 \\ &= 1/2 - \Pr[\nexists \Pi_b \text{ and } \exists \Pi_{1-b} \text{ and } x \notin L]/2 \\ &\quad + \Pr[\exists \Pi_b \text{ and } x \notin L]/2 - \Pr[\exists \Pi_b \text{ and } \exists \Pi_{1-b} \text{ and } x \notin L]/2 \\ &\geq 1/2 + p(n)/2 - \Pr[\exists \Pi_{1-b} \text{ and } x \notin L]. \end{aligned}$$

However, τ_{1-b} is picked uniformly at random and \mathcal{P}^* receives no information about it, so the string $\sigma_{1-b} = \sigma_{\mathcal{P}} \oplus \tau_{1-b}$ is distributed uniformly at random,

so, by soundness of NIZK, $\Pr[\exists \Pi_{1-b} \text{ and } x \notin L]$ is negligible in n . Thus, A 's advantage is only negligibly less than $p(n)/2$.

Now we address the case of sequential malicious provers. Suppose \mathcal{P}^* is an s -sequential malicious prover. Then \mathcal{P}^* initiates at most $s(n)$ sequential conversations and wins if \mathcal{V} accepts at least one of them for $x \notin L$. Then A simply guesses, at random, the conversation for which \mathcal{P}^* will succeed, and simply simulates the other conversations by using the simulator for the proof of knowledge and honestly encrypting random strings. Only for that conversation does it use the procedure described above. This reduces A 's advantage by a polynomial factor of at most $s(n)$. \square

References

- [BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, October 1988.
- [BDMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, December 1991.
- [BFM88] Manuel Blum, Paul Feldman, and Silvio 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, Chicago, Illinois, 2–4 May 1988.
- [Blu86] Manuel Blum. How to prove a theorem so no one else can claim it. In *Proc. of the International Congress of Mathematicians, Berkeley, CA*, pages 1444–1451, 1986.
- [CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, Portland, Oregon, 21–23 May 2000. Updated version available at the Cryptology ePrint Archive, record 1999/022, <http://eprint.iacr.org/>.
- [CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon 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*, Crete, Greece, 6–8 July 2001.
- [DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero knowledge. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 409–418, Dallas, Texas, 23–26 May 1998.
- [FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM Journal on Computing*, 29(1):1–28, 1999.
- [FS89] Uriel Feige and Adi Shamir. Zero knowledge proofs of knowledge in two rounds. In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 526–545. Springer-Verlag, 1990, 20–24 August 1989.
- [GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.
- [GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, February 1996.

- [GL89] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, 15–17 May 1989.
- [GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.
- [GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.
- [GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18:186–208, 1989.
- [GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 1994.
- [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.
- [MR01] Silvio Micali and Leonid Reyzin. Min-round resettable zero knowledge in the public-key model. In Birgit Pfitzmann, editor, *Advances in Cryptology—EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 373–393. Springer-Verlag, 6–10 May 2001.
- [MRV99] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science*, pages 120–130, New York, October 1999. IEEE.
- [Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 387–394, Baltimore, Maryland, 14–16 May 1990.
- [Yao82] A. C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 November 1982. IEEE.

A Definitions of Completeness and RZK

Completeness for a pair $(\mathcal{P}, \mathcal{V})$ is defined the usual way. Consider the following procedure for $(\mathcal{P}, \mathcal{V})$, a string $x \in L$ of length n and a string y .

Procedure Normal-Interaction

1. Run the key-generation stage of \mathcal{V} on input 1^n and a random string r to obtain PK, SK .
2. Pick any id , and let F be a public file that contains the record (id, PK) .
3. Pick strings ω and ρ at random and run \mathcal{P} on inputs $1^n, x, y, id, \omega$, and the verification stage of \mathcal{V} on inputs SK, x, ρ , letting them interact.

Definition 5. A pair $(\mathcal{P}, \mathcal{V})$ is complete for an NP-language L if for all n -bit strings $x \in L$ and their NP-witnesses y , the probability that in an execution of Normal-Interaction \mathcal{V} outputs “accept” differs from 1 by a quantity negligible in n .

The notion of resettable zero-knowledgeness is a bit harder to define. We do not describe the motivation and intuition behind RZK and instead refer the reader to the original exposition of [CGGM00]. Also, note that here we define only black-box RZK (because it is the notion most relevant to this paper). That is, we demand that there exist a single simulator that works for all malicious verifiers \mathcal{V}^* (given oracle access to \mathcal{V}^*).

We introduce a few more players before formally stating the definition. Let

- An *honest prover* \mathcal{P} , for the purposes of defining RZK, be viewed as a *non-interactive* TM that is given, in addition to the inputs given in Section 2, the entire history of the messages already received in the interaction, and outputs the next message to be sent. Fixing all inputs, this view allows one to think of $\mathcal{P}(1^n, x, y, F, id, \omega)$ as a simple deterministic oracle that outputs the next message given the history of the interaction.
- An *(s, t) -resetting malicious verifier* \mathcal{V}^* , for any two positive polynomials s and t , 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 $x_1, \dots, x_{s(n)} \in L$ of length n each, and outputs an arbitrary public file F and a list of $s(n)$ identities $id_1, \dots, id_{s(n)}$.
 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 its “view” of the interactions: its random string and the messages received from the provers.
 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 $x_1, \dots, x_{s(n)} \in L$ as input.

Now we can formally define the resettable-zero-knowledgeness property.

Definition 6. *(\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 $x_1, \dots, x_{s(n)} \in L$ and their corresponding NP-witnesses $y_1, \dots, y_{s(n)}$, the following probability distributions are indistinguishable (in time polynomial in n):*

1. *The output of \mathcal{V}^* obtained from the experiment of choosing $\omega_1, \dots, \omega_{s(n)}$ 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 $1 \leq i, j, k \leq s(n)$.*
2. *The output of M with input $x_1, \dots, x_{s(n)}$ interacting with \mathcal{V}^* .*

B Tools

B.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 . The set of strings having a positive probability in $A(x)$ will be denoted by “ $\{A(x)\}$ ”. If S is a probability space, then “ $x \stackrel{R}{\leftarrow} S$ ” denotes the algorithm which assigns to x an element randomly selected according to S . If F is a finite set, then the notation “ $x \stackrel{R}{\leftarrow} F$ ” denotes the algorithm that chooses x uniformly from F .

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

B.2 Non-Interactive Zero-Knowledge Proofs

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 $\text{NI}\sigma\text{Len}(n)$, where $\text{NI}\sigma\text{Len}$ is a fixed, positive polynomial.

Let us quickly recall their definition, adapted for polynomial-time provers.

Definition 7. *Let NIP (non-interactive prover) and NIV (non-interactive verifier) be two probabilistic polynomial-time algorithms, and let $\text{NI}\sigma\text{Len}$ 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 $\text{NI}\sigma\text{Len}(n)$, and NP-witness y for x ,

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

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

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

is negligible in n .

3. Zero-Knowledgeness. *There exists 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 polynomial-time adversary:*

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

The authors of [BDMP91] show that non-interactive zero-knowledge proofs exist for all NP languages under the quadratic residuosity assumption. The authors of [FLS99] show the same under a general assumptions: namely, that certified trapdoor permutations exist (a family of trapdoor permutations is *certified* if it is easy to tell that a given function belongs to the family). We refer the reader to these papers for details.

B.3 Verifiable Random Functions

A family of verifiable random functions (VRFs), as proposed in [MRV99], is essentially a pseudorandom function family with the additional property that the correct value of a function on an input can not only be computed by the owner of the seed, but also *proven* to be the unique correct value. The proof can be verified by anyone who knows the public key corresponding to the seed.

More precisely, a VRF is a quadruple of functions. The function VRFGen generates a key pair $(\text{VRFSK}, \text{VRFPK})$. The function $\text{VRFEval}(\text{VRFSK}, x)$ computes the pseudorandom output v ; the function $\text{VRFProve}(\text{VRFSK}, x)$ computes pf_x , the proof that v is correct. This proof can be verified by anyone who knows the VRFPK by using $\text{VRFVer}(\text{VRFPK}, x, v, pf_x)$; moreover, no matter how maliciously VRFPK is constructed, for each x , there exists at most one v for which a valid proof pf_x exists. The pseudorandomness requirement states that, for all the points for which no proof has been provided, the function $\text{VRFEval}(\text{VRFSK}, \cdot)$ remains indistinguishable from random. The following formal definition is almost verbatim from [MRV99].

Definition 8. *Let VRFGen , VRFEval , VRFProve , and VRFVer be polynomial-time algorithms (the first and last are probabilistic, and the middle two are deterministic). Let $a: \mathbb{N} \rightarrow \mathbb{N} \cup \{*\}$ and $b: \mathbb{N} \rightarrow \mathbb{N}$ be any two functions that are computable in time $\text{poly}(n)$ and bounded by a polynomial in n (except when a takes on the value $*$).*

We say that $(\text{VRFGen}, \text{VRFEval}, \text{VRFProve}, \text{VRFVer})$ is a verifiable pseudorandom function (VRF) with input length $a(n)$,⁵ and output length $b(n)$ if the following properties hold:

1. *The following two conditions hold with probability $1 - 2^{-\Omega(n)}$ over the choice of $(\text{VRFPK}, \text{VRFSK}) \stackrel{R}{\leftarrow} \text{VRFGen}(1^n)$:*
 - (a) *(Domain-Range Correctness): $\forall x \in \{0, 1\}^{a(n)}$, $\text{VRFEval}(\text{VRFSK}, x) \in \{0, 1\}^{b(n)}$.*
 - (b) *(Complete Provability): $\forall x \in \{0, 1\}^{a(n)}$, if $v = \text{VRFEval}(\text{VRFSK}, x)$ and $pf = \text{VRFProve}(\text{VRFSK}, x)$, then*

$$\text{PROB}[\text{VRFVer}(\text{VRFPK}, x, v, pf) = \text{YES}] > 1 - 2^{-\Omega(k)}$$

(this probability is over the coin tosses of VRFVer).

⁵ When $a(n)$ takes the value $*$, it means that the VRF is defined for inputs of all lengths. Specifically, if $a(n) = *$, then $\{0, 1\}^{a(n)}$ is to be interpreted as the set of all binary strings, as usual.

2. (*Unique Provability*): For every $VRFPK$, x , v_1 , v_2 , pf_1 , and pf_2 such that $v_1 \neq v_2$, the following holds for either $i = 1$ or $i = 2$:

$$\text{PROB}[\text{VRFVer}(VRFPK, x, v_i, pf_i) = \text{YES}] < 2^{-\Omega(k)}$$

(this probability is also over the coin tosses of VRFVer).

3. (*Residual Pseudorandomness*): Let $\alpha > 0$ be a constant. Let $T = (T_E, T_J)$ be any pair of algorithms such that $T_E(\cdot, \cdot)$ and $T_J(\cdot, \cdot, \cdot)$ run for a total of at most 2^{n^α} steps when their first input is 1^n . Then the probability that T succeeds in the following experiment is at most $1/2 + 1/2^{n^\alpha}$:
- (a) Run $\text{VRFGGen}(1^n)$ to obtain $(VRFPK, VRFSK)$.
 - (b) Run $T_E^{\text{VRF Eval}(VRFSK, \cdot), \text{VRF Prove}(VRFSK, \cdot)}(1^n, VRFPK)$ to obtain the pair (x, state) .
 - (c) Choose $r \stackrel{R}{\leftarrow} \{0, 1\}$.
 - i. if $r = 0$, let $v = \text{VRF Eval}(VRFSK, x)$.
 - ii. if $r = 1$, choose $v \stackrel{R}{\leftarrow} \{0, 1\}^{b(n)}$.
 - (d) Run $T_J^{\text{VRF Eval}(VRFSK, \cdot), \text{VRF Prove}(VRFSK, \cdot)}(1^n, v, \text{state})$ to obtain guess.
 - (e) $T = (T_E, T_J)$ succeeds if $x \in \{0, 1\}^{a(n)}$, guess = r , and x was not asked by either T_E or T_J as a query to $\text{VRF Eval}(VRFSK, \cdot)$ or $\text{VRF Prove}(VRFSK, \cdot)$.

We call α the pseudorandomness constant.

The authors of [MRV99] show how to construct VRFs based on the following variant of the RSA assumption. (We refer the reader to that paper for details of the construction.) Let PRIMES_n be the set of the n -bit primes, and RSA_n be the set of composite integers that are the product of two primes of length $\lfloor (n-1)/2 \rfloor$.

The RSA' Subexponential Hardness Assumption: There exists a constant α such that, if A is any probabilistic algorithm which runs in time 2^{n^α} when its first input is 1^n , then,

$$\text{PROB}[m \stackrel{R}{\leftarrow} \text{RSA}_n ; x \stackrel{R}{\leftarrow} \mathbb{Z}_m^* ; p \stackrel{R}{\leftarrow} \text{PRIMES}_{n+1} ; y \stackrel{R}{\leftarrow} A(1^n, m, x, p) : y^p = x \pmod{m}] < 1/2^{n^\alpha}.$$