Resolving the Simultaneous Resettability Conjecture and a New Non-Black-Box Simulation Strategy

Vipul Goyal[*]
UCLA and MSR India
Email: vipul.goyal@gmail.com

Amit Sahai[†]
UCLA
Email: sahai@cs.ucla.edu

October 23, 2009

Abstract

Canetti, Goldreich, Goldwasser, and Micali (STOC 2000) introduced the notion of resettable zero-knowledge proofs, where the protocol must be zero-knowledge even if a cheating verifier can reset the prover and have several interactions in which the prover uses the same random tape. Soon afterwards, Barak, Goldreich, Goldwasser, and Lindell (FOCS 2001) studied the closely related notion of resettable soundness, where the soundness condition of the protocol must hold even if the cheating prover can reset the verifier to have multiple interactions with the same verifier's random tape. The main problem left open by this work was whether it is possible to have a single protocol that is simultaneously resettable zero knowledge and resettably sound. We resolve this question by constructing such a protocol.

At the heart of our construction is a new non-black-box simulation strategy, which we believe to be of independent interest. This new strategy allows for simulators which "marry" recursive rewinding techniques (common in the context of concurrent simulation) with non-black-box simulation. Previous non-black-box strategies led to exponential blowups in computational complexity in such circumstances, which our new strategy is able to avoid.

[*]Research supported in part by a Microsoft Research Graduate Fellowship and the grants of Amit Sahai mentioned below.

[†]Research supported in part from NSF grants 0830803, 0627781, 0716389, 0456717, and 0205594, a subgrant from SRI as part of the Army Cyber-TA program, an equipment grant from Intel, an Alfred P. Sloan Foundation Fellowship, and an Okawa Foundation Research Grant.

1 Introduction

A fundamental question in cryptography deals with understanding the role that randomness plays in cryptographic protocols and to what extent it is necessary. Progress on this question was made relatively early with the result of Goldreich and Oren [GO94] showing that zero knowledge protocols cannot exist in the setting where the parties do not have access to any randomness resource at all. While this work showed that randomness cannot be completely eliminated, it simultaneously motivated several natural questions studying the "extent" to which randomness is necessary. A rich line of work deals with studying the usage of imperfect randomness in various settings (see [KLRZ08, DOPS04] and the references therein). Another line of work (and the one dealt with in this paper) studies whether all the random choices can be made "offline" and be fixed once and for all. In other words, is it possible to design cryptographic protocols where a party can reuse the same random tape in multiple (or even all) executions.

The question of reusing randomness in cryptographic protocols was first considered in the context of zero knowledge by Canetti, Goldreich, Goldwasser, and Micali [CGGM00] who proposed the notion of *resettable zero knowledge*. In resettable zero knowledge, the zero knowledge property is required to hold even if a malicious verifier can "reset" the prover to the initial state and start a new interaction *where the prover uses the same random tape*. Canetti et al. [CGGM00] proposed constructions of resettable zero knowledge protocols based on standard cryptographic assumptions. Barak, Goldreich, Goldwasser, and Lindell [BGGL01] showed how to construct zero knowledge protocols for opposite setting (where soundness is required to hold even if the verifier uses the same random tape in multiple executions), which following Micali and Reyzin [MR01b][1] they call resettably sound (rS) zero-knowledge.

To summarize, there now exist zero knowledge protocols which allow one of the parties to reuse the same randomness while the other would be required to use fresh randomness in every execution. This leads to the following natural question (called the simultaneous resettability problem in [BGGL01]):

"Do there exist zero knowledge protocols which allow both parties to re-use their (respective) random tape in every execution?"

Since the works of [CGGM00, BGGL01], many questions about the security of resettable protocols have been addressed (see the subsection on related work). However, the above question of simultaneous resettability has remained open despite years of work.

Our Results. In this work, we settle this question in the positive by constructing the first simultaneous-resettable zero-knowledge protocol. The primary road block to answering this question had been a limitation in our understanding of non-black-box simulation strategies [Bar01], which are essential in this context (see below) and have been important in achieving a number of advanced cryptographic goals (e.g. [Bar01, Bar02, Pas04, PR05, BS05]). To get our result, we develop novel non-black-box simulation strategies that allow for efficient incorporation of recursive rewinding techniques [RK99, KP01, PRS02], which we believe to be of independent interest and of potentially wider applicability.

1.1 Discussion

Resettable zero knowledge. Resettable zero knowledge, where the prover may be reset, is closely related to concurrent zero knowledge (cZK) [DNS98], where the prover can be forced to interact in an unbounded number of concurrent executions of the protocol, with the interleaving at the control of the attacking verifier. Indeed, resettable zero-knowledge is a strictly stronger requirement than concurrent zero-knowledge; every resettable zero-knowledge protocol is also a concurrent zero-knowledge protocol, but many concurrent zero-knowledge

[1]Micali and Reyzin defined resettable soundness (and other soundness notions) in a public-key model, but did not consider the plain model, which is the focus of the present work.

protocols are not resettable zero-knowledge. Nevertheless, all known resettable zero-knowledge protocols (in the plain model) build upon concurrent zero-knowledge protocols [CGGM00] (see also [BGGL01]).

All known concurrent zero-knowledge protocols make use of *recursive rewinding* techniques for simulation [RK99, KP01, PRS02]. Since their introduction by Goldwasser, Micali, and Rackoff [GMR85], most zero-knowledge protocols have made use of rewinding – where the simulator "tries" to simulate the execution of the protocol, but sometimes "gets stuck", and in order to proceed, it rewinds the execution of the protocol to an earlier point, and tries again to "solve" the simulation (or tries to extract some information from the verifier that it needs to solve the simulation). In the setting of concurrent zero-knowledge, an additional problem arises: when the simulator rewinds the execution and tries again, the cheating verifier may schedule a new concurrent execution of the protocol – and this new execution of the protocol will itself need to be rewound in order to be solved. This leads to recursive rewinding strategies (with multiple "levels" of rewinding), and great care must be taken to prevent this recursion from leading to exponential-time simulations. When concurrent zero-knowledge protocols are modified to become resettable zero-knowledge, this recursive rewinding is inherited by the rZK simulators.

Resettable Soundness. Resettably sound zero knowledge, where the verifier may be reset by a cheating prover (who is trying to prove a false statement), presents a different kind of challenge. Indeed, if a zero-knowledge protocol is simulated by means of a rewinding strategy, then it seems that a cheating prover who can reset the verifier can implement the same strategy as the simulator (since rewinding is nothing more than resetting a party to an earlier state). This intuition is formalized by Barak et al. [BGGL01], who show that no resettably-sound zero-knowledge arguments exist for languages outside **BPP** if the simulator is black-box. Thus, non-black-box simulation, as pioneered by Barak [Bar01], is essential to building resettably sound zero-knowledge protocols. In non-black-box simulation, the actual code of the cheating verifier is used in order to simulate the protocol; this is something that is not available to a cheating prover who can only reset the verifier. Barak et al. [BGGL01] show how to use such non-black-box zero-knowledge protocols to achieve resettably sound zero-knowledge arguments.

1.2 Techniques

The central idea behind our result concerns a novel non-black-box simulation strategy. We first describe this simulation strategy in isolation and then discuss how it is useful in our context.

The New Non-black-box Simulation Strategy. To understand the simulation strategy, we first look at the non-black-box zero knowledge protocol of Barak [Bar01]. The core idea behind Barak's protocol is to have the prover commit in advance to a program that claims to predict (using an input of "small length") a string that is later randomly chosen by the verifier. The prover then must prove that either its committed program really can predict the verifier's string, or that the statement is true. In a real execution, the program is information theoretically extremely unlikely to be able to predict the verifier's random string no matter what the input string is. This is because the input string is not large enough to contain sufficient information about the verifier's random string. But in simulation (in the stand-alone setting), where the simulator can choose the verifier's random coins in advance and commit to these coins along with the verifier's code, the simulator can ensure that the program mimics the verifier's execution of the protocol and therefore correctly predict the verifier's string.

However, (for example) in the fully concurrent setting, there might be messages of several other sessions in between the two "key messages" (that is, the commitment from the prover and the random string from the verifier). Thus, the program that the simulator commits to must regenerate the transcript of the interaction of *all* the concurrent executions up until the point that the adversary in the *current* execution outputs his randomly chosen string. The simulation technique described above does not work in such a scenario and most natural approaches to try to extend it either lead to exponential time simulation or require the input string to the program to be too long (which in turn causes the soundness to fail).

Let us now consider what extra power can we provide to such a program without compromising on soundness:

1. We start with the observation that even if we allow the program to run in exponential time, the soundness of the protocol is still preserved. This is because even an exponential time program can't predict (output) a random value chosen after the program is fixed. Of course, this observation does not appear to lead to a useful simulation strategy because the running time of the simulator would be exponential.

2. Because of that, we instead consider polynomial time programs which however have *access to an outside oracle*. During execution, the program can query the oracle and get a response as many times as it wants. However, the key constraint is that there should only exist one valid response to a query. Definition of a valid response is fixed as part of the protocol specification. Further, given a response (and the corresponding query), it should be possible to determine in polynomial time if the response is valid. An example of such a system is when the program queries the oracle with $f(s)$ and expects s in return (where f is a one-way permutation).

   In other words, *the query string fixes the response string in an information theoretic sense*. Therefore, the program will only output one fixed value *no matter how the oracle is implemented*. This preserves the soundness of the protocol.

3. We now address the question of how the simulator implements the oracle. The simulator initially commits to a program which, during execution, might require responses to queries of a given format. The protocol execution continues and reaches the point when the simulator has to prove that the committed program can predict the verifier's challenge string. At this point, the simulator can anticipate the queries which this program would make (since the program was created by the simulator). Our protocol/simulation strategy will ensure that the simulator already has the responses to each of these queries by this point (possibly by additional means such as rewinding). Thus, the simulator's oracle simply consists of a polynomial number of query, response pairs.

4. To sum up, the simulator proves that there exists a string such that: (A) the committed program makes queries the responses to which can be found in the string, (B) all the query-response pairs satisfy the required format, and, (C) the committed program outputs the verifier's random string.

Here is another way of looking at the above idea. We allow the simulator to supply its program an input of any length. However, the program is only allowed to access this input in a controlled manner so as to not violate the soundness of the protocol.

Looking ahead, during simulation of our simultaneous resettable zero knowledge protocol, the program committed by the simulator would make queries consisting of a commitment (to some string) while the response would consist of the opening to the commitment. Assuming that the commitment scheme is one-to-one, there can only be one valid answer to any query. As mentioned earlier, our simulator would make use of rewinding techniques as well. The rewinding precisely accomplishes the goal of extracting from the verifier the opening to a commitment which can then be used in implementing the oracle later.

Using the above in Our Context. We will first discuss the simpler goal of building a resettably-sound *concurrent* zero-knowledge argument[2]. We begin by briefly recalling how all known concurrent zero-knowledge protocols work in the plain model, for proving "$x \in L$" where L is an NP-complete language [RK99, KP01, PRS02], at an informal level. The high level idea is this: First the verifier commits to a "secret". Then, the prover and verifier do the following many times sequentially (over many rounds of interaction): the prover makes a "challenge" to which the verifier responds, with the properties that: (1) a

[2]Indeed, we follow something similar to this in our actual technical approach, although our first goal is something slightly weaker than concurrent zero-knowledge. See below for a technical outline of our paper.

single challenge-response from each round reveals no information about the secret or the randomness used to commit to the secret, but (2) any two distinct challenge-response pairs *from the same round* reveal the secret and the randomness used to commit to the secret[3]. Then, the prover proves the following using an ordinary zero-knowledge (or WI) proof: that either the prover knows the verifier's secret, or that $x \in L$. This works because in real life, the prover only gets one response for each round, and therefore he cannot learn the verifier's secret. However, in a simulation, the simulator can "rewind" the verifier and try to get two challenge-response pairs for some round, and thereby learn the verifier's secret (and the randomness the verifier used to commit to his secret). As discussed earlier, the concurrent setting requires such a simulation to use a *recursive rewinding* strategy, in order to successfully "solve" every execution of the protocol as they arise.

As discussed earlier, such concurrent zero-knowledge protocols are certainly not resettably sound, since if a cheating prover could reset the verifier, it could use the same rewinding strategy to discover the verifier's secret and use it to cheat (just like the concurrent zero-knowledge simulator does). A simple idea to fix this problem is the following: Have the prover commit to all his challenges in advance, and then in the challenge-response phase, have the prover give a *resettably-sound* zero-knowledge argument that his challenges are the same as the ones that he committed to earlier. Now, the cheating prover can't cheat even if he can reset the verifier. But there seems to be a problem here: thinking back to the concurrent zero-knowledge simulator, in order to extract the verifiers' secrets, it needs to give different challenges from the ones it commits to, so it will need to simulate the resettably-sound zero-knowledge argument so that it can lie. Such a non-black-box simulation would make use of the ideas described earlier.

To reach from one point to the other in a thread of interaction, a rewinding simulator potentially goes through many other threads (in our case, to extract the opening to a commitment to the verifier's secret). If *the program committed* to by the simulator has to repeat through all the threads to reach (and output) the verifier's challenge string, it can be seen that the simulation becomes exponential time. Our new non-black-box simulation strategy avoids such a blowup as follows. Lets look at what happens between when the simulator committed to the program (point A) and when it actually needs to prove that its program can predict the verifier's string (point B). If there exists a session in which verifier's secret (and randomness used to commit to this secret) is required to move from point A to point B, simulator has already extracted this secret by the time it reaches point B by creating look-ahead threads (else the simulator would have aborted before reaching point B). Now the committed program, instead of creating these look-ahead threads again to move from point A to point B, gets these secrets from the simulator by making (unbounded) polynomial number of "commitment-breaking" oracle queries. In other words, the information extracted by the simulator in all "other" threads is already available to the committed program through oracle queries and hence these other threads need not be generated again by the program. Overall, this helps keep our simulation polynomial time.

To make this approach work, aside from the main idea above, we also make use of several other (new and old) ideas, including a new recursive rewinding technique inspired by [RK99]. At its core, our new non-black-box simulation strategy allows for protocols that make essential use of non-black-box simulation but that can also benefit from information learned using black-box recursive-rewinding simulation methods. Given that previous non-black-box simulation advances have had an impact on numerous advanced cryptographic research goals (e.g. [Bar01, Bar02, Pas04, PR05, BS05]), we believe that our new strategy will have other applications as well.

1.3 Related Work

Subsequent to the works of Canetti et al. [CGGM00] and Barak et al. [BGGL01] described above, a number of works have investigated the problem of security against resetting attacks for zero-knowledge protocols in the plain model. Barak, Lindell, and Vadhan [BLV03] constructed the first constant-round public-coin argument that is *bounded* resettable zero-knowledge. Deng and Lin [DL07a] showed a zero-knowledge argument system

[3]The requirement to be able to extract the randomness is actually not standard for concurrent zero-knowledge protocols from the literature, but it is important for our approach.

that is bounded resettable zero-knowledge and satisfies a weak form of resettable soundness; we make use of some ideas from [BLV03, DL07a] in this work.

A larger body of work has investigated the same problems in a relaxed setting, called the "bare public key" (BPK) model, introduced by [CGGM00], which assumes that parties must register (arbitrarily chosen) public keys prior to any attack taking place. We stress that our results hold in the plain model, not just in the BPK model, and the kinds of techniques used in the BPK model typically do not apply to the plain model. [CGGM00] presented a constant-round resettable zero-knowledge argument in the BPK model, the round complexity of which was improved by Micali and Reyzin [MR01b]. Micali and Reyzin [MR01b] also first investigated different notions of soundness in the BPK model, including the notion of resettable soundness. Di Crescenzo, Persiano, and Visconti [CPV04] described a resettable zero-knowledge protocol with concurrent soundness, and Deng and Lin [DL07b] improved the computational assumptions needed to obtain this result. Yung and Zhao [YZ07] also construct resettable zero-knowledge and concurrently sound arguments in the BPK model, using a general and efficient transformation. Micali and Reyzin [MR01a] also proposed a stronger variant of the BPK model for constructing bounded-secure protocols, and provided constant-round bounded resettable zero-knowledge arguments in this model; this result was strengthened by Zhao et al. [ZDLZ03] also in a bounded setting for resettable zero knowledge.

1.4 Technical Outline

Our technical approach to constructing a simultaneous-resettable argument for **NP** is as follows. Note that in all our theorems, we (sometimes implicitly) assume that trapdoor permutations exist and collision-resistant hash functions exist. We also discuss some building blocks that we use in Appendix A (these can all be instantiated based on the assumptions above).

1. In Appendix B, we provide formal definitions of arguments that are resettable zero knowledge and resettably sound.

   In the same section, by modifying definitions and arguments from [CGGM00, BGGL01], we define notions of *hybrid zero-knowledge* and *hybrid soundness* that are very similar to resettable zero-knowledge and resettable soundness, but somewhat easier to prove. Very informally, in a hybrid zero-knowledge protocol, all "important" messages of the verifier are fixed by its first message itself. Thus, to carry out a meaningful reset attack, the verifier has to change its first message whenever it resets the prover. The concept of hybrid soundness is defined analogously; the first message of the prover fixes all its subsequent "important" messages in the protocol.

   We then give transformations (in Appendix C) very similar to ones found in [CGGM00, BGGL01] that show how to modify any hybrid zero-knowledge hybrid sound argument into a resettable zero-knowledge resettably sound argument. The basic idea behind these transformation is have a party derive its own randomness using the first message of the other party. Thus, if the other party changes its first message (in trying to launch a successful reset attack), the randomness of the first party changes completely.

2. In Section 2, we describe our main protocol and its simulator which follows the intuition presented in the Techniques section above, but also incorporates several other ideas (these ideas are sketched in Sections 3.1 and 3.2). This protocol achieves hybrid soundness and a slightly relaxed notion of concurrent zero-knowledge, where in each new session scheduled by the adversary, the adversary must act as an honest verifier based on a fixed random tape (but the adversary can still schedule messages arbitrarily)[4]. These security properties are proven in Sections 3.4 and 4.

[4]Note that any such relaxed concurrent zero-knowledge protocol can be compiled into a fully concurrent zero-knowledge protocol using standard techniques (verifier commits to its randomness in the beginning and gives a zero-knowledge argument of correctness along with every outgoing message).

3. Finally, in Appendix 6, we present a compiler that transforms any relaxed concurrent zero-knowledge and hybrid sound argument into a hybrid zero-knowledge and hybrid sound argument. While our transformation is new, several techniques in this section are due to [DL07a, BLV03]. Taken together with the transformation mentioned above from Appendix C, this yields our main result:

Theorem 1 *Assume that trapdoor permutations and collision-resistant hash function families exist. Then there exists a resettably sound resettable zero-knowledge argument system for all languages in* ***NP****.*

2 Our Main Construction

We describe our construction of hybrid sound relaxed concurrent zero-knowledge arguments (which can be used to obtain resettably sound relaxed concurrent zero knowledge arguments using our hs-rs transformation) in this section. Recall that relaxed concurrent zero knowledge is formally defined in Appendix B. Appendix D contains a sketch of the high-level ideas of our construction.

Let $Com(s)$ denote a commitment to a string s using a non-interactive perfectly binding commitment scheme Com with unique opening (as described in Appendix A.1). Whenever we need to be explicit about the randomness, we denote by $Com(s;r)$ a commitment to a string s computed with randomness r.

The common input to P and V is x supposedly in the language $L \in NP$, and a security parameter n. The auxiliary input to P is an NP-witness w for $x \in L$. Our protocol proceeds as follows.

1. The prover P generates a set of $2n^2$ random challenge strings $\{ch_1, \ldots, ch_{2n^2}\}$ where for all i, $ch_i \in \{0,1\}^n$. P computes and sends commitments $\{Com(ch_1), \ldots, Com(ch_{2n^2})\}$ to the verifier V.

2. The verifier V sets a trapdoor string $trap = Com(1)$, generates a first verifier message σ of a rZAP system (see Appendix A.2) and sends $trap$ and σ to P. In addition, V computes the first message of the three round Blum Hamiltonian cycle protocol repeated in parallel $2n^3$ times for the statement: "$trap$ is a commitment to 1". In more detail, for every repetition, V generates a random permutation of the graph representing the above statement and sends to P the commitments to the permutation and each entry of the adjacency matrix of the permuted graph. This step requires V to use a witness relation such that a valid witness contains a tuple (s, r) such that $trap = Com(s; r)$.

3. For $i \in [2n^2]$, the protocol proceeds as follows:
   - the prover P sends the challenge string ch_i ($\in \{0,1\}^n$) to V.
   - P now proves to V that either: (a) ch_i is the right challenge string committed in the i^{th} commitment in step 2, or, (b) $x \in L$. This is done using our non black-box zero-knowledge argument compiled with the BGGL transformation as described in Figure 1 (having a novel trapdoor property to be used by the simulator).
   - The verifier V now responds to the challenge ch_i. Let $ch_i[j]$ denote the j^{th} bit of the challenge ch_i. For all j, V sends the appropriate commitment openings (as per the Blum Hamiltonian cycle protocol) for the $(ni + j)^{\text{th}}$ parallel repetition assuming the challenge bit to be $ch_i[j]$.

4. The prover P finally gives a rZAP to V proving either $x \in L$ or the string $trap$ is a commitment to 1 under the commitment scheme Com.

3 Proof of Relaxed Concurrent Zero Knowledge

3.1 Overview of Sim

We first informally describe the high-level structure of Sim highlighting the key issues (a more complete description will be given later on).

Common Input to P and V: x' supposedly in the language $L' \in NP$

Auxiliary input to P: An NP-witness w for $x' \in L'$

Protocol:

The zero knowledge argument proceeds as follows:

1. The verifier V chooses a random collision resistant hash function h from a function family $\mathcal{H}$ and sends it to P.
2. The prover P computes $z = Com(h(0))$ and sends it to V.
3. The verifiers V selects a string $r \overset{\$}{\leftarrow} \{0,1\}^{n^4}$ and sends it to P.
4. The prover P and the verifier V now execute a witness indistinguishable constant round public coin universal argument [BG02] where P proves to V that either $x' \in L'$ or the transcript of this zero knowledge argument τ (= (h, z, r)) is in a language Λ defined below. We require the communication complexity of this universal argument to be $O(n^2)$.

The language Λ is defined as follows. We say that $(h, z, r) \in \Lambda$ if there exists an oracle program Π s.t. $z = Com(h(\Pi))$ and there exist strings $y_1 \in \{0,1\}^{\leq n^3}$ and $y_2 \in \{0,1\}^{\leq n^{\log\log n}}$ with the following properties. The oracle program Π takes y_1 as input and outputs r within $n^{\log\log n}$ steps. Program Π may make calls to the oracle by producing a query of the form $trap$ and expecting (s, r) with $trap = Com(s; r)$ in return, such that, the tuple $(trap, s, r)$ is guaranteed to be found in the string y_2 (as per a suitable encoding of y_2). Thus, oracle calls by Π can be answered using y_2. If the program Π makes a query that cannot be answered using y_2, Π aborts and we have that $(h, z, r) \notin \Lambda$.

The above basic argument system is constant round as well as public coin. Applying the BGGL transformation [BGGL01], we obtain a new argument system (which is resettably sound if the above basic argument system is standalone sound).

Figure 1: Our new non black-box zero knowledge argument system

1. Sim generates the challenge strings in the first step randomly as described in the protocol and receives the reply from V^* (which contains the string $trap$).

2. Sim and V^* now execute the challenge response rounds (Step 3). In these rounds, the goal of Sim will be to rewind V^* (in the fully concurrent setting) and extract the "trapdoor witness" which V^* is using. Recall that there are $2n^2$ "slots". In each slot, the prover gives a challenge (along with a non black-box zero-knowledge argument of its correctness) and the verifier opens the appropriate commitments. Sim will attempt to rewind V^* in the concurrent setting and, in some slot, get its response for two different challenges.

   Indeed, rewinding strategies now exist which can achieve the above extraction goal even with $\omega(\log n)$ slots (see [PRS02]). However, the main non-triviality in our setting is that the prover is *committed to its challenge* in each slot ahead of time (this property will be crucial for achieving resettable soundness). Thus, in the look ahead threads, to give a challenge different from the one committed to in a slot, Sim will need to simulate the associated non black-box zero-knowledge argument in the fully concurrent setting (with no apriori bound on the number of executions). Our approach to solve this problem is as follows:

   - We first design a new rewinding strategy for Sim where every thread has the property that the simulator gives a randomly generated challenge (as opposed to the one committed to) to the verifier in at most a *constant number of slots across all sessions*. In all other slots, the simulator can continue to play honestly giving the challenge it committed to earlier. We stress none of the rewinding strategies in the previous works had this property.

- We then describe our novel non black-box simulation strategy using which the simulator can prove a false theorem in a constant number (across all sessions) of non black-box zero-knowledge arguments *in every thread* (in the fully concurrent setting). This allows our simulator to give a random challenge in a constant number of slots in every thread as required by our rewinding strategy.

Our non black box simulation strategy can potentially be extended to simulate any apriori bounded number of slots in each thread, however, this is not required by our rewinding strategy. We describe our rewinding and non black-box simulation strategies in detail in the following subsections.

3. Sim and V^* now execute the final rZAP. By this point, Sim has already extracted a witness to the statement "$trap$ is a commitment to 1" from the verifier. Hence, Sim uses this witness to execute this rZAP.

3.2 The Simulator Sim

Before going into the details of Sim, we first fix some terminology. We assume there are a total of m sessions (each session having $2n^2$ slots). The beginning of a slot is when the simulator gives the challenge, the end of the slot being when it receives the response. In between these two messages, apart from the non black-box zero knowledge argument associated with the challenge, there might be messages of other sessions. In any thread, we say a slot is being simulated if the simulator is giving a random challenge (as opposed to the one committed to) in that slot and is simulating the associated non black-box zero knowledge argument. Otherwise, if the challenge is being given honestly as committed to, we say that the slot is being honestly executed. We say that a session is being simulated if any slot in it is being simulated, otherwise we say the session is being honestly executed.

As with the strategy in [RK99] (and [PV08]), our rewinding schedule is "adaptive". In [RK99], at a very high level, whenever a slot s completes, the simulator may rewind s by calling itself recursively on s. That is, the simulator chooses another challenge for s and recursively executes until either it receives the response (and hence "solves" the session) or it observes that the verifier has started "too many" new sessions or has aborted. One case of special interest to us is when the simulator gets "stuck" on another unsolved session that started earlier than s. The simulator restarts rewinding s in such a case. Richardson and Kilian [RK99] observe that such a case can happen at most $m - 1$ times. This is because while the simulator is trying to rewind s, once it gets stuck on a session that started earlier, it will never get stuck on that session again. Such an analysis is problematic in our scenario where we can simulate only a constant (or a bounded) number of sessions (and hence in our scenario the challenges chosen for a slot in two different threads are not necessarily independent).

The key idea of our simulation technique is to completely avoid the scenario of simulator getting stuck on a session which started earlier. Whenever our simulator decides to rewind a slot s, it chooses a random challenge and recursively invokes itself by giving that random challenge in s (and simulating the associated non black-box zero knowledge argument). Going forward, challenges for all other slots are given honestly by default. In addition, as opposed to strategies in [RK99, PV08] where the simulator only rewinds slots of sessions that started at the current recursive level, our simulator is always "on the lookout" for opportunities to rewind and solve a session. More precisely, before the simulator reaches the final rZAP for a session (and hence potentially gets stuck on it), our choice of the number of slots guarantees that there would exist at least one recursive level which will have at least $2n$ slots of that session. Whenever the simulator observes $2n$ slots in one level, it would rewind those $2n$ slots and solve that session with high probability. This idea ensures that the simulator never gets stuck on a session that started earlier than the target slot s.

The formal description of our simulator is given below. We borrow some notation from [PV08].

- $d = \lceil \log_n(m \cdot 2n^2) \rceil$ will denote the maximum depth of recursion. Note that d is a constant since the number of sessions m is polynomial in the security parameter n. Our simulator will have the property that the total number of slots being simulated in any thread is bounded by d.
- slot(i, j) will denote slot j of session i.
- ℓ denotes the current depth of the recursion.

SOLVE$(x, \ell, h_{initial}, s, L)$

Let $h \leftarrow h_{initial}$
Repeat forever and update h after each step:

1. If the verifier aborts or the number of slots in h started after $h_{initial}$ (which we will call new slots) exceed $\frac{m \cdot 2n^2}{n^\ell}$, return h;
2. If the next message is the first prover message of some session, generate and commit random challenges honestly.
3. If the next message is the first verifier message of some session, continue;
4. If the next message is the final rZAP of some session then, as explained in step 6(c), we have already solved that session (else the simulator would have aborted by this point). In other words, that trapdoor witness for that session has been extracted. Use the extracted trapdoor witness (for the statement "$trap$ is a commitment to 1") to execute the final rZAP.
5. If the next message is a prover message for the beginning of a slot s', we have the following two possibilities:
   (a) If $s' \in L$, the slot s' is being simulated. The simulator uses the challenge specified in L. In addition, the simulator uses our non black-box simulator subroutine (described in the next subsection) to handle all the messages of the non black-box zero-knowledge argument associated with this message.
   (b) If $s' \notin L$, the simulator proceeds honestly to give the challenge. It also executes the associated non black-box zero-knowledge argument honestly.
6. If the next message is the end message of a slot $s' =$ slot(i', j'), proceed as follows:
   (a) If $s = s'$, we have succeeded in solving the target slot and hence the session. Return h;
   (b) Otherwise if the session i' has already been solved or the number of new slots (including s') of session i' in h started after $h_{initial}$ is less than $2n$, the simulator need not rewind this slot. Continue;
   (c) Otherwise, we have an unsolved session i' such that $2n$ of its slots (from slot$(i', j' - 2n + 1)$ to slot(i', j')) have appeared at the current level. The Sim will rewind each of these slots once and will solve session i' except with negligible probability. Observe that the depth d of the recursion is a constant and the total number of slots in a session is $2n^2$. This means just by the pigeonhole principle, for every session i', we would have this case at some level before we reach its final rZAP. For each slot s" in this list of $2n$ slots:
      i. Set $L" = L$. Add $s"$ to $L"$. In addition, select a random challenge for $s"$ and add it to $L"$.
      ii. Let $h"$ be the prefix of h which contains all messages up to but excluding the prover challenge for $s"$. Set $h^* \leftarrow$ SOLVE$(x, \ell + 1, h", s", L")$.

iii. If h^* contains an accepting execution for slot s", the simulator has succeeded in solving s" and hence session i'.

If after executing this step once for each such slot s", we have not yet solved session i', abort and output Ext_Fail.

$\underline{Sim(x, z)}$

Run SOLVE$(x, 0, \perp, \perp, \perp)$ and output the view returned by SOLVE, with the following exception. When the simulator generates random challenge for a simulated slot and it becomes equal to the real challenge for that slot or another simulated challenge generated previously in a different thread for the same slot, the simulator aborts and outputs $\perp$.

Looking ahead, the core of the analysis of this rewinding strategy can be found in Lemma 1 where we prove that the probability with which the simulator outputs Ext_Fail is negligible in n.

3.3 The Non Black-Box Simulation Subroutine

Recall that in a thread, whenever the simulator simulates a slot to give a random challenge, it was required to simulate the associated non black-box zero-knowledge argument. We describe our non black-box simulator subroutine in this subsection, and prove the completeness of the simulator's use of this subroutine to execute the non-black-box argument system.

First we describe the high level intuition behind how the the simulator uses its random tapes. The simulator has a pair of random tapes (R_A, R_B). The random tape R_A is sufficiently long so that it can be utilized to compute all messages of all threads *except* messages of a slot being simulated. R_A is formatted so as to allow "random access". That is, whenever the simulator needs to compute messages in a given thread, it can directly calculate what part of R_A should be used (thus, R_A need not be accessed sequentially). The random tape R_B is used to execute the slots which are being simulated. That is, simulator uses R_B to pick the random challenge for a slot being simulated and to complete the associated non-black-box zero-knowledge argument.

Formal details of the simulator random tape usage are as follow. Consider the "execution tree" of the simulation where each function call to SOLVE represents one node in the tree while each recursive call made by it represents one of its child nodes (see Section 5 for details on how this execution tree is defined). The random tape R_A has a (sufficiently long) portion for each such possible node in the execution tree. As shown in Section 5, this execution tree has depth upto d and degree upto $2mn^2$ (and hence only has a polynomial number of nodes). From this it follows that the length of R_A is only polynomial (since each execution of SOLVE can only utilize a polynomial amount of randomness). Note that a node in the tree at depth ℓ can be uniquely identified by a tuple $(S_1, \ldots, S_\ell)$ where $S_i \in [2mn^2]$. Hence, such a sequence also uniquely identifies the portion of the random tape R_A to be used for the execution of that node. We map each execution of SOLVE during a simulation to a node in the execution tree in the natural way by mapping the first call SOLVE$(x, 0, \perp, \perp, \perp)$ to the root node and mapping the i-th recursive call made by an execution of SOLVE to its i-th child node. This determines the randomness our simulator will use to complete that execution of SOLVE. The simulator uses the random tape R_B exclusively (in all threads) for picking random challenges in the slots being simulated and for executing non black box zero knowledge arguments which are being simulated. The simulator uses R_A for computing every other outgoing message (e.g., honest non black box zero knowledge arguments, the prover first message, final rZAP etc.). Such a separation of random tape is essential for our hybrid arguments to go through.

Denote the thread containing the slot to be simulated by T. The simulator sends a random challenge in this slot and uses the *trapdoor condition* of the associated non black-box zero knowledge argument to proceed. As the first step of the proof, the verifier sends a hash function h as usual. The simulator now constructs a program Π and sends $z = \text{Com}(h(\Pi))$.

The program Π, very roughly, is constructed using part of the current state of the simulator and the adversarial verifier such that it is able to go forward and produce their upcoming interaction transcript in the thread T (with some input and "help from outside"). Details of the functionality of Π are given later in this subsection.

After receiving the commitment from the simulator, the adversarial verifier may continue interaction in other concurrent sessions and finally produces a string r. The simulator now executes the universal argument with the verifier as follows. It first prepares a string y_1 containing the following:

1. The slot identifiers (in the form of tuples (i, j) containing the session and the slot numbers) for all the simulated slots in the thread T.

2. The randomly selected prover challenges for all the above slots.

3. For each slot above, a tuple uniquely identifying the corresponding node of the execution tree (note that each simulated slot can be mapped to an execution of SOLVE).

4. All prover messages which: (a) belong to a non black box zero knowledge argument being simulated, and, (b) lie between messages z and r in the thread T. This includes the message z.

5. The number of simulator-verifier steps of interaction in T between messages z and r.

Recall that the number of simulated slots in T is bounded by a constant $d(= \lceil \log_n(m \cdot 2n^2) \rceil)$. Furthermore, the size of each prover message included in y_1 is $O(n^2)$. From this, it can be shown that $|y_1| \leq n^3$.

The simulator additionally constructs a string y_2 as follows. Consider a session for which the final rZAP lies between messages z and r. Since the simulator executed the thread T without aborting till at least the message r, it follows that it must have extracted the trapdoor witness, i.e., a witness to the statement "$trap$ is a commitment to 1" (by getting a response for two different challenges in a slot across different threads) for that session. In other words if the trapdoor string for that session is $trap$, the simulator has obtained (s, r) with $trap = Com(s; r)$. The string y_2 simply contains such commitment strings $trap$ and their opening tuples (s, r). Now we discuss the functionality of Π in detail.

The program Π is constructed using two components: (a) the current state of the verifier, and, (b) a part of the current state of the simulator which has R_A (i.e., the first part of the simulator random tape). The goal of program Π is to regenerate the transcript of only thread T from z to r (z, r inclusive). It works as follows.

1. It takes y_1 as input and runs the inbuilt simulator and verifier machines from message z onwards.

2. Whenever Π needs to compute a prover message for a slot s (i.e., the challenge or a prover message of the associated non black-box zero knowledge argument), it checks y_1 to see if s is simulated. If so, Π uses the message specified in y_1. Otherwise, Π computes the message honestly using the appropriate part of R_A.

3. If the program Π has to execute the final rZAP of a session for which it does not have the required trapdoor, Π uses the allowed oracle calls. For such a session, let the description of its trapdoor string be denoted by $trap$. The program Π makes an oracle call with the string $trap$ in the query. This string $trap$ and its opening (s, r) with $trap = Com(s; r)$ is guaranteed to be found in y_2 by construction. Π obtains the required witness for the statement "$trap$ is a commitment to 1" in response and uses that to complete the final rZAP.

   Comment: Note that Π (i.e., the inbuilt simulator it has) may not have the required trapdoor when it has to execute the final rZAP of a session since Π *does not execute any other threads apart from* T. (This is crucial for the running time of our simulator to be polynomial.) See Figure 2. Here the thread shown in green and red is the thread T and its red portion is the part Π will regenerate. The look ahead threads (shown in black) are not regenerated by Π .

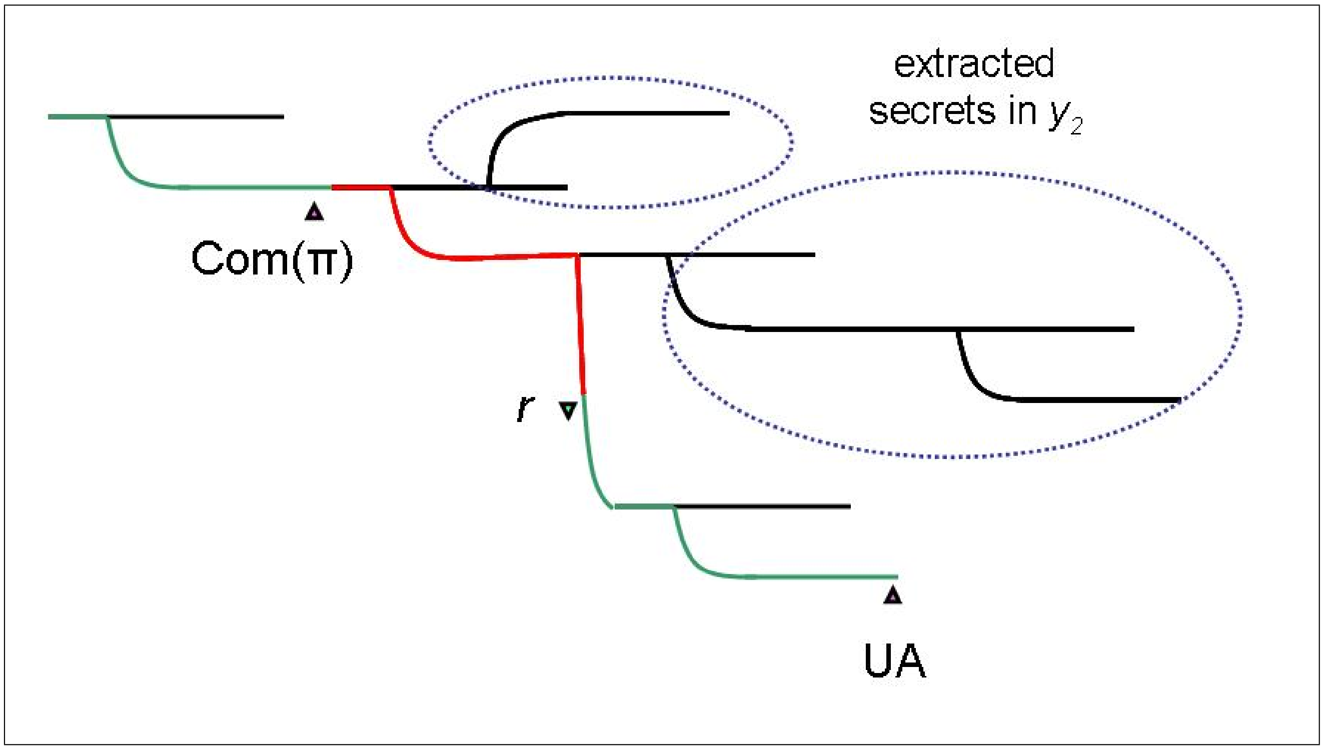

Figure 2: Π only re-generates the part of transcript shown in red. The secrets extracted in the look ahead threads are specified in y_2. This helps Π move forward using oracle calls without regenerating the look ahead threads.

4. After the program Π runs for the number of steps specified in the input y_1, it has regenerated the transcript in T between z and r. Π then halts outputting r.

To conclude, the opening to the commitment z and the pair (y_1, y_2) constitute a witness to the statement $(h, z, r) \in \Lambda$. The simulator uses this witness to execute the universal argument and hence complete the non black-box zero-knowledge argument, as long as y_2 contains all the necessary (s, r) pairs, which must be the case unless the simulator has already failed and output Ext_Fail.

3.4 Indistinguishability of the Real and Simulated Views

In this section, we prove:

Theorem 2 *Assume that trapdoor permutations and collision-resistant hash function families exist. Then the protocol in Section 2 is a relaxed concurrent zero-knowledge argument.*

PROOF. We will consider a series of hybrid experiments and show that the views of V^* in successive hybrids are indistinguishable from each other. Our initial experiment will be the actual protocol as executed by the prover and the verifier. Our final experiment will be the simulated protocol as described above.

Experiment H_0. This experiment corresponds to when the simulator Sim has the required witness and runs the protocol honestly with the malicious verifier V^* .

Experiment H_1. In this experiment, Sim starts creating the look-ahead threads as per our rewinding strategy but with the following two modifications. For a slot being simulated in any look-ahead thread, Sim chooses a challenge at random as usual. However, the first modification is that Sim executes the associated non black-box zero knowledge argument in such a slot using a witness to the statement "$x \in L$" as opposed to simulating using the trapdoor condition $(h, z, r) \in \Lambda$ (recall that in the non-black-box zero knowledge arguments, the prover proves that either the challenge string is the correct one it committed to or $x \in L$). Secondly, Sim does not use the extracted witnesses in any way in any thread (i.e, Sim continues to use the witness for the statement $x \in L$ to complete the final rZAP in all threads), just as the honest prover would do in an honest execution. The only difference between the view of V^* in H_0 and H_1 is that Sim might abort the interaction (and hence the main thread) because of one of the following 2 reasons (for reference, see the description of Sim in the previous section):

1. A random challenge, ch, picked by Sim for a simulated slot becomes equal to either the real challenge or another simulated challenge generated previously in a different thread for the same slot. Now since $ch \overset{\$}{\leftarrow} \{0,1\}^n$ and, as shown in section 5, there are only a polynomial number of threads, the probability of this event happening is bounded by $\frac{\text{poly}(n)}{2^n}$ (which is negligible in n).

2. Sim fails to extract a valid witness in one of the sessions (i.e outputs Ext_Fail). We show in Lemma 1 that the probability of this event happening is negligible.

Taken together, the above points imply that the view of V^* in experiment H_1 is indistinguishable from that in experiment H_0.

Lemma 1 *The probability with which the simulator Sim outputs Ext_Fail in experiment H_1 is negligible.*

PROOF. We prove the above claim by contradiction. Assume that this probability is noticeable. Now since there are a polynomial number of threads, a polynomial number of sessions in each thread and a polynomial number of slots per session, there would exist a thread[5] T, a session i and a slot j (of session i) such that Sim outputs Ext_Fail with a noticeable probability after trying to solve session i by rewinding slots j to $j + 2n - 1$ appearing at some level in thread T (see description of Sim in Section 3.2). We call this event Fail in experiment H_1 and all subsequent experiments given below. Now consider the following hybrid experiments:

Experiment G_0. This experiment is the same as hybrid experiment H_1 with the following exceptions. The simulator Sim only runs the thread T. In addition, Sim runs the look ahead threads forking off T from the beginning of the slots j to $j + 2n - 1$ of session i if they all appear at the same level. Event Fail occurs if the simulator runs these $2n$ look ahead threads but fails to extract a witness for session i. In other words, Sim only generates the interaction transcript for thread T by not executing any threads from the execution tree except T and (if slots j to $j + 2n - 1$ occur at the same level) the $2n$ look ahead threads.

Note that Pr[Fail] may only increase in this experiment. This is because Sim now outputs Ext_Fail at least with the probability with which Sim outputs Ext_Fail in session i when started rewinding from slot j in thread T in the experiment H_1 (since the thread T and $2n$ look ahead threads in this experiment provide a perfect emulation of the corresponding threads in H_1 to the point they appear in the execution tree in H_1). Thus, Pr[Fail] is still noticeable in this experiment.

Experiment G_1. This experiment is identical to the previous one except that now Sim starts using a witness for the statement $x \in L$ to complete the non-black-box zero-knowledge argument in each of these $2n$ slots in thread T (as opposed to completing it honestly) . The witness indistinguishability of the universal argument scheme directly implies the indistinguishability of the view of V^* in this experiment from that in the previous one. Hence, following the previous experiment, we have that Pr[Fail] is still noticeable.

[5]A thread T is defined by a slot number (from 1 to $m \cdot 2n^2$) and an attempt number (from 1 to n) for every recursive level. This is to uniquely pick the path from the root to a leaf node in the execution tree (see Section 5). Of course, it could be the case that in a particular execution of the simulator, the thread T doesn't appear or appears only partially. This is because not every slot at every level occurs and is rewound n times by the simulator in every execution.

Experiment G_2**.** This experiment is identical to the previous one except that now Sim gives a random challenge in each of these $2n$ slots in thread T (as opposed to giving the ones it committed to). Observe that at this point, Sim is not using the openings of the commitments to these challenges any time during the simulation (since the associated non black-box zero-knowledge argument is being executed using a witness to the statement $x \in L$). Hence, the computational hiding property of the scheme Com directly implies that the view of V^* in this experiment is indistinguishable from that in the previous one. Hence, as in the previous experiment, we have that Pr[Fail] is still noticeable.

If in a given execution, the $2n$ slots (from j to $j + 2n - 1$) occur at the same level, let ℓ denote that level. Since the total number of new slots starting and ending at level ℓ is bounded by $\frac{m \cdot 2n^2}{n^\ell}$, at least n of these $2n$ slots have a maximum of $\frac{m \cdot 2n^2}{n^{\ell+1}}$ new slots between their start and finish messages. **We call these slots** $(\ell + 1)$**-good**.

Now consider a slot k from these $2n$ slots and the look ahead threads for this slot. Clearly the transcript of slot k in thread T (i.e., the transcript in thread T from the start of k to its end) is identically distributed to the transcript of k that appears in the look ahead thread for k (forking off thread T). Now observe that when the event Fail occurs: (a) all the $2n$ slots (from j to $2n + j - 1$) occur at the same level, and, (b) at least n of these $2n$ slots in thread T are $(\ell+1)$-good while none of the slots in the look ahead threads are (since otherwise Sim would have extracted a witness for session i).

Experiment G_3**.** Sim now proceeds as follows. Sim runs the thread T as in experiment G_2 till the point when the first of these $2n$ slot begins (i.e., the challenge for the first of these slots is due). Sim now creates 2 threads from this point onwards and continues each of them until the point this slot finishes or verifier aborts. Sim now randomly selects one of these threads to be part of the thread T and treats the remaining as the look ahead thread trying to "solve" this slot. Sim now continues this (extended) thread T till the point when the second of the $2n$ slots begins and proceeds from there similarly as for the first slot (by creating 2 threats as before and selecting one of them to add to T). Sim continues this process to generate the entire thread T and the look ahead threads for the $2n$ slots. Sim outputs $\mathsf{Ext\_Fail}$ if the $2n$ slots all occur at the same level ℓ and *none of the* $2n$ *look ahead threads are* $(\ell + 1)$*-good.* Thus, exactly as in the previous experiment, when event Fail occurs: (a) all the $2n$ slots (from j to $2n + j - 1$) occur at the same level, and, (b) at least n of these $2n$ slots in thread T are $(\ell + 1)$-good while none of the slots in the look ahead threads are. The only effective difference from the previous experiment is that in this experiment, some of the look ahead threads are being generated as the thread T is being generated (as opposed to generating the thread T first and then all the look ahead threads as in the previous experiment). Hence it is easy to see that the probability of event Fail occurring remains identical to that in experiment G_2. Now we try to bound this probability.

To analyze the probability of event Fail in this experiment, we consider an experiment in which part of the random tape of Sim is supplied by an external party. In detail, as it goes forward, Sim generates 2 threads for each of these $2n$ slots and randomly chooses one of them to be part of the thread T as described except for the following. Whenever exactly one of the generated 2 threads is $(\ell + 1)$-good, Sim queries the external party with these thread transcripts asking it to randomly choose the thread to be part of the thread T. The probability with which the external party chooses the thread which is $(\ell + 1)$-good is exactly $\frac{1}{2}$. Fail occurs only if Sim makes at least n queries to the external party and each time the external party chooses the only $(\ell + 1)$-good thread. This implies that in this experiment

$$\Pr[\mathsf{Fail}] \leq \frac{1}{2^n}$$

However this contradicts the fact that Pr[Fail] is noticeable in this experiment. Hence Lemma 1 follows. □

Experiment H_2**.** This experiment is defined exactly as the previous one with the exception that now Sim uses the trapdoor witness (extracted from V^*) to complete the final rZAP in all threads. Again, by relying on the witness indistinguishability property of rZAPs (in the concurrent setting), it can be shown that the view of V^* in this experiment is indistinguishable from that in the previous one.

Experiment H_3**.** This experiment is identical to the previous one except that now in the non black-box zero-knowledge argument for each of the simulated slots (with a random challenge as opposed to the one committed to) in every thread, Sim, instead of committing to $h(0)$, commits to $h(\Pi)$ where Π is a program to predict the string r and is computed as described in the previous section. Note that none of the programs Π's contain the randomness required to generate these commitments (since it comes from the second part of the random tape of Sim) and at this point, the opening of these commitments is not being used anywhere by the Sim (since the corresponding universal arguments are being executed using a witness to the statement $x \in L$). Hence, the computational hiding property of the commitment scheme Com directly implies the indistinguishability of the view of V^* in this experiment from that in the previous one.

Note that at this point, in the non-black-box zero knowledge arguments in a simulated slot, the verifier V^* sends a hash function h and the prover sends $z = Com(h(\Pi))$ where Π is a program which can compute the forthcoming verifier challenge string r (with the appropriate input and Oracle calls as described in section 3.3). Hence we have that the trapdoor condition $(h, z, r) \in \Lambda$ in the associated universal argument is true (even though the universal argument is being executed using a witness to the statement $x \in L$ at this point).

Experiment H_4**.** This experiment is identical to the previous one except that now Sim starts using a witness for the trapdoor condition $(h, z, r) \in \Lambda$ to complete the universal arguments in each of the simulated slots.

The indistinguishability of this hybrid from the previous one relies on the adversary being relaxed concurrent. First we observe that since the randomness required to execute these universal arguments is not committed to as part of any of the programs Π, they are essentially being executed with "off the record" randomness. Secondly, since the adversary is relaxed concurrent, it follows that even in the presence of multiple threads, a universal argument does not have multiple different continuations across different threads with the same prefix. That is, if it thread with a partially executed universal argument gets "forked off" in multiple different threads, all of these threads will have the same continuation of this universal argument. Hence, given a V^* which can distinguish between the distribution of the view in experiment H_3 from that in experiment H_4, it is possible for Sim to have an external universal argument prover and distinguish the case from when it is using a witness to $x \in L$ from the case when it is using a witness to the statement $(h, z, r) \in \Lambda$. Hence, the witness indistinguishable property of the universal arguments implies the indistinguishability of the view of V^* in this experiment from that in the previous one.

Note that the simulator in the experiment H_4 is our actual simulator. Thus, the output of the simulator Sim is computationally indistinguishable from the distribution of the transcript of a real interaction. Aside from showing that the simulation in polynomial-time (shown in the next subsection), this completes our proof. □

4 Proof of Hybrid Soundness

In this section, we first prove that our protocol is hybrid sound (i.e hs, see Definition 8). We then apply our hs-rs transformation to obtain a resettably sound protocol (which is a still relaxed concurrent zero knowledge). As a first step, we analyze the (standalone) soundness of our new non black box zero knowledge argument system.

Lemma 2 *The zero knowledge argument system described in Figure 1 is computationally sound in the standalone setting.*

PROOF. Recall that in the protocol, after receiving h from the verifier, the (possibly malicious)- prover sends $z = Com(h(\Pi))$ where Π could be any arbitrary program. We first analyze the probability of such a program being able to output the verifier random string r ($\in_R \{0,1\}^{n^4}$) given the input $y_1 \in \{0,1\}^{\leq n^3}$ and access to the oracle queries which are answered using $y_2 \in \{0,1\}^{\leq n^{\log\log n}}$ as described in the specification of language Λ. Now when Π is executed, there are a number of possibilities of the output depending upon what the input y_1 is and how the oracle queries are answered. Since Π only makes queries of the form $trap$ expecting (s, r) in return with $trap = Com(s; r)$, by the unique opening property of the commitment scheme Com (See Appendix A.1), the answer to the query is information theoretically fixed given the query itself. Hence the input y_1 alone information theoretically determines the output of Π. Since $y_1 \in \{0,1\}^{n^3}$, there are a total of 2^{n^3} possible outputs of Π. Denote by S the set of these possible outputs. Now the probability of a string $r \in_R \{0,1\}^{n^4}$ being an element of this set is bounded by $2^{n^4 - n^3}$ which is negligible in n. The above argument still does not imply that $(h, z, r) \notin \Lambda$ since $z (= Com(h(\Pi)))$ does not information theoretically fix the program Π.

The rest of our soundness proof is along the lines of the one in [Bar01]. Assume $x' \notin L'$ and a malicious prover P^* is still able to successfully complete the protocol such that a honest verifier V outputs accept with a noticeable probability ϵ. We can assume P^* is deterministic without loss of generality. Call the first verifier message h to be the prefix for the rest of the protocol. Now it has to be the case that for atleast a fraction $\frac{\epsilon}{2}$ of the prefixes, the probability (over rest of the verifier random coins) that P^* will succeed is atleast $\frac{\epsilon}{2}$. We call such prefixes good. Now the verifier executes the protocol with P^* and invokes the weak knowledge extractor associated with the universal argument system [BG02]. The probability (over all verifier random coins) of the prefix being good and the extractor succeeding given that the prefix is good is atleast $\frac{\epsilon}{2} \cdot p(\frac{\epsilon}{2})$ where p is a polynomial (recall that the probability of success of the extractor is polynomially related to the probability of success of the prover). Now if the extractor succeed and extracted a program (say Π_1), the verifier restarts the execution, sends the same first message h and receiving the same $z = Com(h(\Pi_1))$ and continues from there on with independent random coins and running the knowledge extractor (the verifier in particular chooses an independent random string $r \in_R \{0,1\}^{n^4}$). As argued in the previous section, if S_{Π_1} is the set of all possible outputs of Π_1, the probability that $r \in S_{\Pi_1}$ is negligible. If the extraction succeeds again, the verifier has obtained another program Π_2. As argued before, except with negligible probability, Π_1 could not have predicted r and hence $\Pi_1 \neq \Pi_2$. However since $h(\Pi_1) = h(\Pi_2)$, we have obtained a collision in the hash function. The probability of this event COLL can be computed as follows:

$$\begin{aligned}\Pr[\text{COLL}] &\geq \Pr[\text{The prefix } h \text{ is good}] \cdot \Pr[\text{Extractor succeeds in two independent executions with prefix } h] \\ &\quad - \Pr[\Pi_1 = \Pi_2] \\ &\geq \frac{\epsilon}{2} \cdot \left(p\left(\frac{\epsilon}{2}\right)^2\right) - \text{negl}(n)\end{aligned}$$

which is still noticeable. This violates the collision resistance property of the function family $\mathcal{H}$. Hence the lemma follows. □

Since our argument system is constant round and public coin in addition, the application of the BGGL transformation (proposition 3.5 of [BGGL01]) results in a resettably sound argument system. The hybrid soundness of our construction will crucially depend upon the resettable soundness of our non black box zero knowledge argument system.

Theorem 3 *The construction presented in Section 2 is hybrid sound.*

PROOF. We first show that our protocol is a verifier admissible proof system. We divide the messages of the prover into main and authenticator messages naturally as follows. The first prover message (i.e the commitment to the challenges) is considered to be a main (and the determining) message. All the prover messages of the (resettable sound) non black box zero knowledge argument system are considered to be authenticator messages for the associated challenge which is considered to be a main message. The final rZAP is regarded as authenticator message for the (implicit message) "$x \in L$". Each incarnation of the verifier V is divided into (V_1, V_2). We view the random tape of V_1 as a tuple $\omega_1^{(1)}, \omega_1^{(2)}$ where $\omega_1^{(1)}$ is used for emulating the verifier of the non black box zero knowledge argument system whenever required while $\omega_1^{(2)}$ is used for emulating the verifier of the rZAP system. V_1 simply forwards the first main message received from P to V_2 who sends back the reply. Now in each slot, V_1 receives the challenge string and acts as the verifier of the non black box zero knowledge argument system on its own (using random tape $\omega_1^{(1)}$) to verify correctness of the challenge string. If the argument completes successfully, i.e V_1 accepts, it forwards the challenge string to V_2 who sends back the response. Finally when P sends the final rZAP, V_1 verifies it on its own (using randomness $\omega_1^{(2)}$) and forwards the message "$x \in L$" by convention. V_2 outputs accept if it receives such a message from V_1.

To summarize, P sends three types of main messages: the determining message (i.e, the commitment to the challenges), the challenges themselves in various slots, and the final (implicit) message "$x \in L$". Now consider an arbitrary resetting PPT prover P^*. For the same determining message M, it follows from the soundness of our non black box zero knowledge that except with negligible probability, P^* is unable to generate two different challenge strings in a slot such that V_1 accepts both. This is because since the first main message contains a perfectly binding commitment to the challenge strings, doing so would amount to proving a false theorem (and in particular, given such a P^* one could construct a resetting prover for a resettable sound zero knowledge argument system which can prove an adaptively chosen false theorem to an honest verifier with noticeable probability). Finally, we observe that there is only one possibility for the final main message. Hence it follows that our protocol is a verifiable admissible proof system.

All that remains to be shown now is that our protocol is sound in the hybrid model. This is shown by focusing on the view of one incarnation of V_2. V_2 interacts with V_1 and handles a single execution with fresh randomness without getting reset. We focus on one such incarnation of V_2 (while playing honestly in others) and, very roughly, (a) make a look ahead thread to learn all the challenges committed by the verifier, (b) rewind back to the point it has to generate a trapdoor string and generate a false trapdoor string (i.e a string $trap$ which is a commitment to 0), and, (c) still complete the Blum hamiltonian cycle protocol since it already knows the challenges of P^*. Now if P^* still manages to complete the proof successfully, it violates the resettable soundness of the rZAP system. More details follow.

Suppose $x \notin L$ and a malicious prover P^* still manages to complete the protocol successfully (such that V outputs accept) in the hybrid model with a noticeable probability ϵ. Since P^* interacts directly only with V_1 which in turn interacts with V_2, we view $V_1(P^*)$ as a single machine which interacts with V_2 as described before. We can assume that $V_1(P^*)$ is deterministic without loss of generality. Call the first prover message M consisting of the commitment to the challenge string to be the prefix for the rest of the protocol. Now it has be the case that for atleast a fraction $\frac{\epsilon}{2}$ of prefixes, the probability (over the random coins of V_2) that $V_1(P^*)$ will succeed is atleast $\frac{\epsilon}{2}$. We call such prefixes good. Now, V_2 executes the protocol with $V_1(P^*)$ honestly. The probability (over the random coins of V_2) of the prefix being good and $V_1(P^*)$ succeeding given that the prefix is good is atleast $\frac{\epsilon^2}{4}$. Now if $V_1(P^*)$ succeeded (which means that V_2 learns all that challenge strings committed to), V_2 restarts the execution and receives the same prefix M from $V_1(P^*)$. V_2 however generates a false trapdoor string $trap$ this time (i.e., $trap = Com(0)$). By the resettable soundness of the non black box zero knowledge argument as before, all the challenge strings given by $V_1(P^*)$ in this execution would be identical to the ones in the previous execution. Hence, V_2 already has all the challenge strings of $V_1(P^*)$. It now follows from standard techniques that V_2 can still successfully complete the Blum hamiltonian cycle protocol as executed between V_2 and $V_1(P^*)$. Simply by relying on the computationally hiding property of the commitment scheme, Com, it follows that $V_1(P^*)$ would still succeed in this execution with probability $\frac{\epsilon}{2} - \text{negl}(n)$ if the prefix M is good. This would violate the resettable soundness of the rZAP system (since $x \notin L$ and $trap = Com(0)$). The probability of this event VIO can be computed as follows:

$$\begin{aligned}
\Pr[\text{VIO}] &\geq \Pr[\text{The prefix } M \text{ is good}] \cdot \Pr[\ V_1(P^*) \ \text{ succeeds in both executions with prefix } M] \\
&\geq \frac{\epsilon^2}{4} \cdot \left(\frac{\epsilon}{2} - \text{negl}(n)\right)
\end{aligned}$$

which is still noticeable. This violates the resettable soundness property of the rZAP system. Hence the theorem follows. □

5 The Running Time of Our Simulation

In this section, we analyze the running time of our simulator Sim and prove that it is polynomial in the security parameter n assuming the running time of V^* is polynomial. It is easy to see that for Sim, computing the next message takes polynomial time for all messages other than those belonging to the universal argument of a *simulated* non black-box zero-knowledge argument. Let n^{c_1} denote the bound on the time taken by the simulator to compute the next such prover message (i.e, a prover message not part of a universal argument of a simulated non black box zero knowledge argument). Now consider messages of a "simulated universal argument" (i.e, universal argument of a simulated non black box zero knowledge argument). To execute this universal argument, the simulator uses a witness to the statement $(h, z, r) \in \Lambda$. Observe that verifying this statement given the witness (which constitutes of the committed program Π, strings (y_1, y_2) and the opening of the commitment z) requires running the program Π to regenerate the protocol transcript between the messages z and r. Lets consider the time taken to compute all prover messages between z and r. This time is clearly $O(mn^2 \cdot n^{c_1})$ since there can only be $O(mn^2)$ such messages between z and r. All the prover messages belonging to a simulated slot are given as input as part of string y_1 (and only need to be "read" rather than computed). Again, the time taken to "read" such a message is also bounded by n^{c_1}. Hence, overall the time taken to generate such prover messages between z and r is $O(mn^2 \cdot n^{c_1})$. If the running time of the verifier's next message function is bounded by n^{c_2}, it follows that Π can regenerate the transcript between z and r in time $O(mn^2 \cdot n^{c_1+c_2})$. In other words, the theorem statement $(h, z, r) \in \Lambda$ can be verified in time $O(mn^2 \cdot n^{c_1+c_2})$. Then by the properties of universal arguments [BG02], it can be shown that the time taken to execute a simulated universal argument is bounded by $p(mn^2 \cdot n^{c_1+c_2})$ where p is a polynomial.

Hence, we conclude that Sim takes polynomial time to compute each next outgoing message to V^* . In other words, each query from Sim to V^* (where a query is defined to be one round of communication between Sim and V^* : computing the next prover message and the verifier's reply) takes polynomial time. All that remains to be shown now is that Sim makes only a polynomial number of queries to V^* .

To bound the number of queries Sim makes to V^* , we consider the recursive execution tree (of constant depth) resulting out of Sim rewinding V^* . Each call to the function SOLVE$(\cdot, \cdot, \cdot, \cdot, \cdot)$ will represent one node in the execution tree. The nodes resulting from all further recursive calls to SOLVE will be treated as children of this node. Thus, the root node (at depth 0) is the call SOLVE$(x, 0, \cdot, \cdot, \cdot)$ made by $Sim(x, z)$. This call results in the main thread while recursive calls give rise to the look ahead threads.

Now consider the transcript generated by a function call representing a node at depth ℓ (excluding the transcripts generated by any further recursive calls). The number of new slots in this transcript is bounded by $m \cdot 2n^2$ (in fact $\frac{m \cdot 2n^2}{n^\ell}$). Now, each of these slots may have up to one look ahead thread resulting in a total of up to $2mn^2$ children for this node. Hence, the execution tree is a tree of depth up to d and degree up to $2mn^2$. Hence, the total number of nodes is bounded by $(2mn^2)^{d+1}$. The transcript of each node contains up to $O(2mn^2)$ queries. Hence, the total number of queries Sim makes to V^* is $O((2mn^2)^{d+2})$ which is a polynomial (since d is a constant). This concludes our analysis.

6 Getting Resettably Sound Resettable Zero Knowledge

In this section, our goal is to construct a general compiler to transform any resettably sound relaxed concurrent zero knowledge argument Σ into one that is resettably sound resettable zero knowledge argument. Combining this with the protocol in our previous section, gives us our main result.

As an intermediate step, we first present a compiler to transform any given resettably sound relaxed concurrent zero knowledge protocol Σ into one that is hZK and hs (see Definition 6 and Definition 8). Once we have a protocol that is both hZK and hs, we can immediately obtain one that is rZK and rs by applying our hZK-rZK and hs-rs transformations. Given a protocol Σ, our compiler works as follows.

The common input to P and V is x supposedly in the language $L \in NP$, and a security parameter n. The auxiliary input to P is an NP-witness w for $x \in L$. The compiled protocol proceeds as follows:

1. The prover P generates a random string R_p (of appropriate length to be used to emulate the verifier of a resettably sound zero knowledge argument system). P further generates the first verifier message σ_p of a rZAP system and sends $Com(R_p)$ and σ_p to the verifier V.

2. The verifier V similarly generates a first verifier message σ_v of a rZAP system. V further generates a string $trap$ such that $trap = Com(0)$. V sends σ_v and $trap$ to P.

3. The prover P and the verifier V now execute a protocol in which V proves to P that the string $trap$ is a commitment to 0 (in other words, there exists a r s.t. $trap = Com(0; r)$). This protocol is a resettably sound zero knowledge argument [BGGL01] in which P uses the random tape R_p to emulate the verifier. *In addition*, P sends a rZAP along with every message of this argument proving that either:

   (a) The message is "honestly computed" and is consistent with R_p. More precisely, this message is what an honest verifier using the random tape R_p (as committed in the first step) would have sent given the transcript of the resettably sound zero knowledge argument so far, or,

   (b) $x \in L$

4. Let τ denote the string consisting of the protocol transcript so far except the prover messages of the rZAP system (i.e., the rZAPs in step 3). The verifier V chooses a function $f : \{0,1\}^{\leq \text{poly}(n)} \rightarrow \{0,1\}^{\leq \text{poly}(n)}$ from an ensemble of pseudorandom functions. All further random coins required by V in the protocol will come from random tape $f(\tau)$.

   The verifier V now generates a random string R_v (of appropriate length to emulate the verifier in the protocol Σ). V then sends $Com(R_v)$ and $trap$ to P. (Sending $trap$ is only required for technical reasons for our hZK-rZK transformation to be applicable.)

5. The prover P and the verifier V now execute the protocol Σ in which P proves to V that $x \in L$. In this protocol, V uses the random tape R_v to emulate the verifier. *In addition*, V sends a rZAP along with every message of this argument proving that either:

   (a) The message is "honestly computed" and is consistent with R_v. More precisely, this message is what an honest verifier using the random tape R_v (as committed in step 4) would have sent given the transcript of the protocol Σ so far, or,

   (b) the string $trap$ is a commitment to 1. That is, there exists a r such that $trap = Com(1; r)$.

   The verifier V outputs accept if the verifier of Σ outputs accept.

Theorem 4 *Assuming that the protocol Σ is relaxed concurrent zero knowledge, the compiled protocol described above is hZK.*

PROOF. We start by proving that the compiled protocol is a prover-admissible proof system.

Lemma 3 *Assuming that the protocol Σ is relaxed concurrent zero knowledge, the compiled protocol is a prover-admissible proof system as per Definition 5.*

PROOF. We partition the prover P into (P_1, P_2) as follows. The first main verifier message (i.e., the determining message) is the one containing $Com(R_v)$ (See step 4 of our protocol). All verifier messages before the determining message are considered to be authenticator messages to be handled by P_1. In case the resettably sound zero knowledge argument is successful in step 3, P_1 forwards the first round main message ($Com(R_v), trap$) to P_2. Next, during the execution of the protocol, all rZAP messages are considered to be authenticator messages associated with a verifier message of Σ which is considered to be a main message itself. P_1 verifies the rZAP on its own and if successful, forwards the associated verifier message of protocol Σ to P_2. We view the random tape of P_1 as a tuple $(\omega_1^{(1)}, \omega_1^{(2)}, \omega_1^{(3)}, \omega_1^{(4)})$. $\omega_1^{(1)}$ is used to emulate the prover of the rZAP system in step 3, $\omega_1^{(2)}$ is identical to R_p and used when emulating the verifier of the resettably sound zero knowledge argument system in step 3, $\omega_1^{(3)}$ is used to emulate the verifier of the rZAP in step 5 and $\omega_1^{(4)}$ is used for the remainder of the tasks.

We prove that our protocol is a prover-admissible proof system by contradiction. Suppose that a resetting adversarial verifier V^* can interact with (an incarnation of) the prover $P = (P_1, P_2)$ and with a noticeable probability ϵ produces two different main messages, both accepted by P_1, for some round ℓ in two different interactions with P with the same determining message $(trap, Com(R_v))$. We say, as a short hand, that V^* violates property of the prover-admissible proof system with probability ϵ. We consider the following hybrid experiment:

Experiment H_1. This experiment is identical to the actual proof system as described above with the exception that P_2 uses a witness to the statement $x \in L$ to complete all the rZAPs in step 3. By the resettable witness indistinguishability of the rZAP system, it follows that V^* still violates property 3 of the prover-admissible proof system with probability $\epsilon - \text{negl}_1(n)$.

Experiment H_2. This experiment is identical to the previous one with the exception that P_2 starts sending $Com(0)$ as opposed to $Com(R_p)$ in step 1. This in particular means that, P_2 is using "off the record" randomness to emulate the verifier of the resettably sound zero knowledge argument system in step 3. Since the message sent by P_2 to V^* in the first step is always fixed (across resets), from the computation hiding property of the commitment scheme Com it follows that V^* still violates property 3 of the prover-admissible proof system with probability $\epsilon - \text{negl}_2(n)$.

It can be shown that in experiment H_2, the string $trap$ is a commitment to 0 except with negligible probability. This directly follows from the resettable soundness of the argument system being used in step 3. Now, the determining message of V^* contains $Com(R_v)$ such that V^* has to give a rZAP with every following main message (i.e., a verifier message of Σ) essentially proving its consistency with R_p (since $trap$ is a commitment to 0). Thus, violating property 3 (of the prover-admissible proof system) in our protocol amounts to violating the resettable soundness of the rZAP system. This contradicts the fact that ϵ is noticeable. □

All that remains to be shown now is that our protocol is zero knowledge in the hybrid model. To prove that, we focus on the view of all incarnations of P_2. Let $\mathcal{M}$ denote the machine resulting from the combination of all incarnations of P_2 (i.e., $\mathcal{M}$ simply has the code of each incarnation of P_2 and handles all their interactions). $\mathcal{M}$ interacts with various incarnations of P_1 and handles multiple concurrent executions each with fresh randomness (without getting reset). Since V^* interacts directly only with an incarnation of P_1 which in turn interacts with the corresponding incarnation of P_2, we view $P_1(V^*)$ as a single machine constructed using the code of V^* and each incarnation of P_1. This machine in turn interacts with $\mathcal{M}$ as described before.

To show zero knowledge, we construct a simulator which interacts with the machine $P_1(V^*)$. This simulator $\mathcal{S}$ is essentially identical to the relaxed concurrent zero knowledge simulator $\mathcal{S}_\Sigma$ of protocol Σ. Note that the first message of $P_1(V^*)$ in a session is $(Com(R_v), trap)$ which can be seen as $P_1(V^*)$ writing on a special tape before beginning the session. Further, $P_1(V^*)$ just serves as the verifier of the protocol Σ (after sending the first message). From the resettable soundness of the rZAP system, except with negligible probability, all messages of $P_1(V^*)$ in Σ are consistent with R_v (where $Com(R_v)$ appears on the special tape). Hence during the entire simulation, the view of S is statistically indistinguishable from the view when $\mathcal{S}$ was interacting with a relaxed concurrent adversary $\mathcal{A}$. Now recall that for relaxed concurrent adversary $\mathcal{A}$, the advantage of a distinguisher in distinguishing between the output of $\mathcal{S}_\Sigma$ and the output of $\mathcal{A}$ in a real interaction with $\mathcal{M}$ is negligible. Thus it follows that such a statement holds even when $\mathcal{A}$ is instantiated with $P_1(V^*)$. This concludes the proof that our protocol is hZK. $\square$

Theorem 5 *Assuming that the protocol Σ is resettably sound, the compiled protocol is hs (i.e., hybrid sound).*

PROOF. We start by showing that our protocol is a verifiable admissible proof system. We divide the verifier V into V_1 and V_2 as follows. The first message sent by P^* (containing a commitment to R_p) is considered to be the determining message. V_2 sends the reply to this message as in step 2. In step 3, each verifier message of P^* contains a verifier message of a resettably sound zero knowledge argument system which is considered to be a main message. The accompanying rZAP is considered to be an authenticator message handled by V_1 which verifies it and forwards the associated main message to V_2 (which then sends the next prover message of the resettably sound zero knowledge argument system). Finally, step 4 and 5 are handled by V_1 alone which sends a message "$x \in L$" by convention to V_2 if P^* completes Σ successfully. In other words, all the messages of P^* in step 4 and 5 are authenticator messages for the (implicit) main message "$x \in L$". We view the random tape of V_1 as a tuple $(\omega_1^{(1)}, \omega_1^{(2)})$ where $\omega_1^{(1)}$ is used to emulate the verifier of the rZAP system (in step 3) while $\omega_1^{(2)}$ is identical to the pseudorandom function f used to produce the random tape used by V_1 in step 4 and 5. We define two parts of V_1: V_1^{part1} and V_1^{part2}. V_1^{part1} has randomness $\omega_1^{(1)}$ and executes step 1 to 3 of our protocol. V_1^{part2} has the description of f and executes step 4 and 5.

To see that our protocol is a verifiable admissible proof system, the first message of P^* contains a commitment to R_p. In step 3, all main messages accompanied by a rZAP proving their consistency with R_p. Finally, there is only one possible main message in step 4 and 5 (i.e., $x \in L$). Hence from the resettable soundness of the rZAP system, it follows that our protocol is a verifiable admissible proof system.

All that remains to be shown now is that our protocol is sound in the hybrid model. To prove this, we focus on one incarnation I of $V = (V_1, V_2)$. We first define machines $M_1(P^*)$ and M_2 as follows. $M_1(P^*)$ is made of a combination of P^*, all incarnations of V except I and V_1^{part1} of incarnation I. Machine M_2 is made of a combination of V_1^{part2} and V_2 of incarnation I. In other words, $M_1(P^*)$ interacts with P^* and honestly handles all the messages of interaction with it internally which either V_1^{part1} of incarnation I or any verifier V of any of the incarnation are supposed to handle. For the remaining messages of incarnation I, $M_1(P^*)$ interacts with M_2 (which is capable of executing V_1^{part2} and V_2 of incarnation I).

Suppose $x \notin L$ and the malicious prover P^* still manages to complete the protocol successfully with incarnation of verifier $V = (V_1^{\text{part1}}, V_1^{\text{part2}}, V_2)$ with a noticeable probability ϵ in the hybrid model. We will focus on the view of M_2. Now consider the following hybrid experiments:

Experiment H_1. This experiment is identical to our protocol (as described in the terminology of $M_1(P^*)$ and M_2 as above) except for the following. The machine M_2, to emulate V_1^{part2}, uses a separate random tape R as opposed to using the output of f applied on τ. Note that τ is exactly the transcript of interaction between $M_1(P^*)$ and V_2 which is encapsulated inside M_2 (V_2 gets the last "$x \in L$" message from V_1^{part2} rather than $M_1(P^*)$). Observe that this interaction between $M_1(P^*)$ and V_2 would have concluded by the time V_1^{part2} starts any interaction with $M_1(P^*)$.

Now recall that the random tape used by V_2 is independent of any random tape used by any incarnation of V_2 encapsulated in machine $M_1(P^*)$. From this it can be shown that with high probability, the string τ is different from any string τ' on which $f(\tau')$ is evaluated by any other incarnation of V_2 encapsulated in $M_1(P^*)$. Thus, just by the pseudorandomness property of the function f, the view of P^* (inside $M_1(P^*)$) in this experiment remains indistinguishable from the one in the real protocol. This in particular means that V_2 (inside M_2) still outputs accept with probability at least $\epsilon - \text{negl}_1(n)$.

Observe that at this point both V_2 and V_1^{part2} are using random coins independent of any random coins used by $M_1(P^*)$. V_2 cannot be reset by $M_1(P^*)$ while V_1^{part2} might be. Further, V_2 completes step 3 of the protocol before V_1^{part2} is invoked by M_2 for any interaction.

Experiment H_2**.** This experiment is identical to the previous one except for the following. V_2 sets string $trap = Com(1)$ in step 2. Furthermore, V_2 runs the simulator $\mathcal{S}_{rs}$ associated with the resettably sound zero knowledge argument system to complete step 3. Note that it is sufficient for $\mathcal{S}_{rs}$ to only work in the standalone setting. By the computationally hiding property of commitment scheme Com and the indistinguishability of the view generated by $\mathcal{S}_{rs}$ from when this argument was honestly executed, it follows that the view of $M_1(P^*)$ in this experiment is indistinguishable from the one in the previous experiment. This in particular means that V_2 still outputs accept with probability at least $\epsilon - \text{negl}_2(n)$.

Experiment H_3**.** This experiment is identical to the previous one except for the following. V_1^{part2}, instead of sending $Com(R_v)$ in its first outgoing message, sends $Com(0)$. Further, V_1^{part2} uses the trapdoor condition "string $trap$ is a commitment to 1" to compute the rZAP it is required to send along with every verifier message of the protocol Σ (a witness for the statement "$trap$ is a commitment to 1" is passed from V_2 on to V_1^{part2} by machine M_2). Note that the commitment in the first outgoing message of V_1^{part2} (i.e., $Com(R_v)$ in the previous experiment, $Com(0)$ in this one) is identical across all sessions resulting from the resets. Thus the computational hiding property of the commitment scheme Com along with the resettable witness indistinguishability of rZAP system implies that the view of $M_1(P^*)$ in this experiment is indistinguishable from the one in the previous experiment. This in particular means that V_2 still outputs accept with probability at least $\epsilon - \text{negl}_3(n)$.

Note that in experiment H_3, V_1^{part2} is using "off the record" random tape R_v to emulate the verifier of protocol Σ. Furthermore, for V_2 to output accept, V_1^{part2} should accept in Σ (and send the message "$x \in L$" to V_2). Hence the fact that ϵ is noticeable contradicts the resettable soundness of the protocol Σ. Thus, the theorem follows. $\square$

References

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *FOCS*, pages 106–115, 2001.

[Bar02] Boaz Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *Proc. 43rd FOCS*. IEEE, 2002. Preliminary full version available on `http://www.math.ias.edu/~boaz`.

[BG02] Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *IEEE Conference on Computational Complexity*, pages 194–203, 2002.

[BGGL01] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. In *FOCS*, pages 116–125, 2001.

[BLV03] Boaz Barak, Yehuda Lindell, and Salil P. Vadhan. Lower bounds for non-black-box zero knowledge. In *FOCS*, pages 384–393, 2003.

[BS05] Boaz Barak and Amit Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *FOCS*, pages 543–552. IEEE, 2005.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *STOC*, pages 235–244, 2000.

[CPV04] Giovanni Di Crescenzo, Giuseppe Persiano, and Ivan Visconti. Constant-round resettable zero knowledge with concurrent soundness in the bare public-key model. In *CRYPTO*, pages 237–253, 2004.

[DL07a] Yi Deng and Dongdai Lin. Instance-dependent verifiable random functions and their application to simultaneous resettability. In Naor [Nao07], pages 148–168.

[DL07b] Yi Deng and Dongdai Lin. Resettable zero knowledge with concurrent soundness in the bare public-key model under standard assumption. In *Inscrypt*, pages 123–137, 2007.

[DN00] Cynthia Dwork and Moni Naor. Zaps and their applications. In *FOCS*, pages 283–293, 2000.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *STOC*, pages 409–418, 1998.

[DOPS04] Yevgeniy Dodis, Shien Jin Ong, Manoj Prabhakaran, and Amit Sahai. On the (im)possibility of cryptography with imperfect randomness. In *FOCS*, pages 196–205. IEEE Computer Society, 2004.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proc. 17th STOC*, pages 291–304. ACM, 1985.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1):1–32, 1994.

[KLRZ08] Yael Tauman Kalai, Xin Li, Anup Rao, and David Zuckerman. Network extractor protocols. In *FOCS*, pages 654–663. IEEE Computer Society, 2008.

[KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in poly-loalgorithm rounds. In *STOC*, pages 560–569, 2001.

[MR01a] Silvio Micali and Leonid Reyzin. Min-round resettable zero-knowledge in the public-key model. In *EUROCRYPT*, pages 373–393, 2001.

[MR01b] Silvio Micali and Leonid Reyzin. Soundness in the public-key model. In *CRYPTO*, pages 542–565, 2001.

[Nao07] Moni Naor, editor. *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*. Springer, 2007.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *Proc. 36th STOC*, pages 232–241. ACM, 2004.

[PR05] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In *Proc. 37th STOC*. ACM, 2005.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS*, pages 366–375, 2002.

[PV08] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. On constant-round concurrent zero-knowledge. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 553–570. Springer, 2008.

[RK99] Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In *EUROCRYPT*, pages 415–431, 1999.

[YZ07] Moti Yung and Yunlei Zhao. Generic and practical resettable zero-knowledge in the bare public-key model. In Naor [Nao07], pages 129–147.

[ZDLZ03] Yunlei Zhao, Xiaotie Deng, Chan H. Lee, and Hong Zhu. Resettable zero-knowledge in the weak public-key model. In *EUROCRYPT*, pages 123–139, 2003.

A Building Blocks

A.1 Non-interactive Perfectly Binding Commitment Scheme with a Unique Decommitment

In our protocol, we shall use a non-interactive perfectly binding commitment scheme with the properties that every commitment has a unique decommitment and the verification of the decommitment is deterministic. An example of such a scheme is the scheme that commits to the bit b by $Com(b;(r,x)) = r||\pi(x)||(x \cdot r) \oplus b$ where π is a one-way permutation on the domain $\{0,1\}^k$, $x \cdot y$ denotes the inner-product of x and y over $GF(2)$, and $x, r \leftarrow U_k$. We denote this commitment scheme by Com.

A.2 Resettably Sound Resettable Zaps

Zaps are two round public coin witness indistinguishable proofs introduced by Dwork and Naor [DN00]. Zaps further have the special property that the first message (sent by the prover) can be reused for multiple proofs.

As noted in [BGGL01], any ZAP system already has the property of resettable soundness. Furthermore, resettable witness indistinguishability property can be obtained by applying the transformation in [CGGM00]. We refer to the resulting system as an rZAP system having the property of resettable soundness as well as resettable witness indistinguishability (see Theorem 1.5 in [BGGL01] for more details).

A.3 Resettable Sound Zero-Knowledge Arguments

Resettable sound zero-knowledge (rsZK) arguments were studied by Barak et al [BGGL01]. As in resettable zero-knowledge [CGGM00], rsZK arguments deal with the zero-knowledge functionality but consider the setting when the *verifier* is resettable by the prover. Barak et al [BGGL01] gave a construction of rsZK arguments relying on the non-black techniques introduced by Barak [Bar01]. They also ruled out rsZK arguments having a black-box simulator (except for languages in $\mathcal{BPP}$) thus showing that usage of non-black box techniques is inherent. In our constructions, we rely crucially on the fact that rsZK arguments (as defined by [BGGL01]) have the property that soundness holds even if the verifier can use the same random string in multiple zero-knowledge argument executions even for different statements.

Barak et al [BGGL01] in fact also presented a general transformation from any constant round public coin zero knowledge argument system to a resettably sound zero knowledge argument system. We make use of this transformation in our constructions and refer to it as the BGGL transformation.

B The Model and Definitions

B.1 Relaxed Concurrent Zero Knowledge

Informally speaking, in relaxed concurrent zero knowledge, we only quantify over *relaxed concurrent adversaries*. We first define relaxed concurrent adversaries in the setting of zero knowledge.

Definition 1 (Relaxed Concurrent Adversary:) *An adversary $\mathcal{A}$ who interacts with the prover P concurrently in multiple sessions is a relaxed concurrent adversary if it has the following property. Before $\mathcal{A}$ starts a new session, it writes a string s to a special tape such that:*

- *There exists a function f (not necessarily polynomial time computable) fixed before the start of any interactions such that $r = f(s)$, and,*
- *All messages of the adversary $\mathcal{A}$ (playing as the verifier) in this session are consistent with the messages of an honest verifier using the random tape r.*

Informally speaking, the next message of the adversary $\mathcal{A}$ in a session is information theoretically fixed given the special tape of $\mathcal{A}$ for this session and the transcript of interaction between P and $\mathcal{A}$ in *this session alone*. In particular, very roughly, if the simulator rewinds $\mathcal{A}$ and changes a prover message (in another session) which appears in the interaction transcript after this session started, the transcript of interaction of this session would remain unchanged.

Definition 2 (Relaxed Concurrent Zero Knowledge:) *A protocol Σ is called relaxed concurrent zero knowledge if it remains zero knowledge with respect to a relaxed concurrent adversary $\mathcal{A}$. In other words, for every relaxed concurrent adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that the distribution of the view of $\mathcal{A}$ in interaction with the prover P is indistinguishable from the distribution of the output of $\mathcal{S}$ (which is given only the common input).*

B.2 Resettably Sound Resettable Zero Knowledge Arguments

In this section, we recall the definition of the properties *resettable zero knowledge* and *resettably sound arguments* from the works in [CGGM00, BGGL01]. Our goal would be to construct an interactive proof system which satisfies both of these properties. The definitions below are taken almost verbatim from [CGGM00, BGGL01].

Definition 3 (rZK [CGGM00]:) *An interactive proof system (P, V) for a language L is said to be* resettable zero-knowledge *if for every probabilistic polynomial-time adversary V^* there exists a probabilistic polynomial-time simulator M^* so that the distribution ensembles D_1 and D_2 described below are computationally indistinguishable: Let each distribution be indexed by a sequence of* distinct *common inputs $\bar{x} = x_1, \ldots, x_{poly(n)} \in L \cap \{0,1\}^n$ and a corresponding sequence of prover's auxiliary-inputs $\bar{y} = y_1, \ldots, y_{poly(n)}$.*

Distribution D_1 *is defined by the following random process which depends on P and V^*.*

1. *Randomly select and fix $t = poly(n)$ random-tapes $\omega_1, \ldots, \omega_t$ for P, resulting in deterministic strategies $P^{(i,j)} = P_{x_i,y_i,\omega_j}$ defined by $P_{x_i,y_i,\omega_j}(\alpha) = P(x_i, y_i, \omega_j, \alpha)$, for $i, j \in \{1, \ldots, t\}$. Each $P^{(i,j)}$ is called an* incarnation of P.
2. *Machine V^* is allowed to run polynomially-many sessions with the $P^{(i,j)}$'s. Throughout these sessions, V^* is required to complete its current interaction with the current copy of $P^{(i,j)}$ before starting a new interaction with any $P^{(i',j')}$, regardless if $(i, j) = (i', j')$ or not. Thus, the activity of V^* proceeds in rounds. In each round it selects one of the $P^{(i,j)}$'s and conducts a complete interaction with it.*
3. *Once V^* decides it is done interacting with the $P^{(i,j)}$'s it (i.e, V^*) produces an output based on its view of these interactions. This output is denoted by $\langle P(\bar{y}), V^* \rangle(\bar{x})$ and is the output of the distribution.*

Distribution D_2:
The output of $M^*(\bar{x})$.

Definition 4 (rs [BGGL01]:) *A* resetting attack *of a cheating prover P^* on a* resettable verifier *V is defined by the following two-step random process, indexed by a security parameter n.*

1. *Uniformly select and fix $t = \mathrm{poly(n)}$ random-tapes, denoted $r_1, \ldots, r_t$, for V, resulting in deterministic strategies $V^{(j)}(x) = V_{x,r_j}$ defined by $V_{x,r_j}(\alpha) = V(x, r_j, \alpha)$, where $x \in \{0,1\}^n$ and $j \in [t]$. Each $V^{(j)}(x)$ is called an* incarnation of V.
2. *On input 1^n, machine P^* is allowed to initiate poly(n)-many interactions with the $V^{(j)}(x)$'s. The activity of P^* proceeds in rounds. In each round P^* chooses $x \in \{0,1\}^n$ and $j \in [t]$, thus defining $V^{(j)}(x)$, and conducts a complete session with it.*

Let P and V be some pair of interactive machines, and suppose that V is implementable in probabilistic polynomial-time. We say that (P, V) is a resettably-sound proof system for L (resp., resettably-sound argument system for L) *if the following two conditions hold:*

- Resettable-completeness: *Consider an arbitrary resetting attack (resp., polynomial-size resetting attack), and suppose that in some session after selecting an incarnation $V^j(x)$, the attacker follows the strategy P. The, if $x \in L$ then $V^{(j)}(x)$ accepts with negligible probability.*
- Resettable-soundness: *For every resetting attack (resp., polynomial size resetting attack), the probability that in some session the corresponding $V^{(j)}(x)$ has accepted and $x \notin L$ is negligible.*

C The hZK and rs Definitions and Transformations

Canetti et al [CGGM00] proposed a general paradigm for constructing resettable zero knowledge protocols. This paradigm was further generalized by Barak et al [BGGL01]. They defined a class of proof systems called *admissible proof systems* and a new model called the *hybrid model* which is a strengthening of the concurrent model. They proposed a general transformation applicable to an admissible proof system and showed that if the original (admissible) proof system was zero knowledge in the hybrid model, the transformed proof system would be resettable zero knowledge. A similar result was shown for the case of witness indistinguishability. These results were obtained by showing that *any fully resetting adversary can be emulated by an adversary in the hybrid model*.

In this section, we modify the class of admissible proof systems and the corresponding hybrid model slightly to fit our requirements. We show that even for our definition, the transformation of [CGGM00, BGGL01] still works and a proof of this fact is very similar to the corresponding proofs in [BGGL01]. Furthermore, we define an analogous transformation for the case of resettable soundness. That is, we define the class of admissible proof systems and the hybrid model for this case and then present a transformation to convert an admissible proof system that is sound in the hybrid model to a resettably sound proof system. This is done in a manner very similar to the case of resettable zero knowledge by showing that a fully resetting prover can be emulated by a prover in the hybrid model.

We now discuss our transformations in detail. Our transformations are modifications of the transformations found in [CGGM00, BGGL01]

Definition 5 (Prover-admissible proof-system) *A proof-system (P, V) is called* prover-admissible *if the following requirements hold:*

1. *The prover P consists of two parts P_1, P_2. Similarly, the prover's random input ω is partitioned into two disjoint parts, $\omega^{(1)}, \omega^{(2)}$, where $\omega^{(i)}$ is given to P_i*
2. *A message sent by the verifier may either be labeled as* main *message or an* authenticator *message. The first main message sent by the verifier in the protocol is called the* determining *message. Each verifier message is first received by P_1. In each round of interaction, the verifier and P_1 exchange a number of messages in which exactly one of the messages is a main message while the rest are authenticator messages. At the end of an interaction round, P_1 decides whether to accept the (only) main message received based on the transcript of interaction round itself and the transcript of the verifier main messages and the corresponding replies of P_2 so far. If P_1 accepts, it forwards the main message to P_2 who generates the reply.*
3. *Let V^* be an arbitrary (*deterministic*) polynomial-size circuit, where V^* may execute a resetting-attack on P (as described in Distribution 1 of Definition 3). Let V^* interact with some incarnation of $P = (P_1, P_2)$. Then, except with negligible probability, V^* is unable to generate two different main messages, both accepted by P_1, for some round ℓ in two different interactions with P with the same determining message.*

The Hybrid Model Loosely speaking, in our hybrid model, as in [CGGM00, BGGL01], the verifier is given the ability to "partially reset" the prover (while otherwise interacting in the concurrent setting). More precisely, in the prover admissible proof system, each incarnation of the prover is identified by three indices: $P^{(i,j,k)} = P_{x_i,y_i,\omega_{j,k}}$ where $\omega_{j,k} = (\omega_j^{(1)}, \omega_k^{(2)})$. The string $\omega_j^{(1)}$ represents the random input to P_1 while $\omega_k^{(2)}$ represents the random input to P_2. The verifier can interact with these incarnations in the concurrent setting where it is allowed to have a *single* session with each incarnation. However, the verifier is not allowed to interact with two different incarnations $P^{(i,j,k)}$ and $P^{(i',j',k')}$ such that $k = k'$. Furthermore, the verifier is given the power to request P_1 (in an incarnation) to restart the interaction from the beginning while leaving P_2 in the same state as it was (we prefer not to use the term "reset P_1" since such a restart does not completely erase its memory)[6]. After being restarted, P_1 operates as usual using the same random tape $\omega^{(1)}$ except for the following. P_1 aborts if the verifier sends a different determining message in the first round of interaction after P_1 restarts. Furthermore, in a round of interaction with the verifier, P_1 does not forward the received main message to P_2 (even if it is accepted) in case a main message in that round has been forwarded earlier (before P_1 was requested to restart the interactions) regarding of whether the message is equal to the previous one or not. Instead, P_1 simply sends accept or reject to the verifier (without sending any other reply) depending upon whether it accepted or rejected the main message in that round. P_1 then waits for the verifier messages for the next round as if P_2 had sent the same reply it sent earlier in that round before P_1 restarted the interaction. Intuitively, such a setting ensures that P_1 and P_2 do not go out of sync even though only P_1 restarts the interaction (with the same randomness).

Definition 6 (hZK) *A* hybrid cheating verifier V^* *works against prover-admissible proof systems in the hybrid model as described above. A proof system is hZK if it is prover admissible and satisfies Definition 3 with respect to hybrid cheating verifiers.*

The transformation below is identical to the ones in [CGGM00, BGGL01].

Transformation hZK-rZK Given a prover-admissible proof system (P, V), where $P = (P_1, P_2)$, and a collection f of pseudorandom functions, we define a new proof system $(\mathbf{P}, \mathbf{V})$ as follows.

The new verifier V is identical to V.

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

Theorem 6 *Suppose that (P, V) is prover-admissible, and let $\mathbf{P}$ be the prover strategy obtained from P by applying Transformation hZK-rZK. Then:*

Assuming that pseudorandom functions exist, for every probabilistic polynomial-time resetting cheating verifier $\mathbf{V}^*$ (as in Definition 3) there exists a probabilistic polynomial-time hybrid cheating verifier W^* (as in Definition 6) such that $\langle P(\bar{y}), W^* \rangle(\bar{x})$ is computationally indistinguishable from $\langle \mathbf{P}(\bar{y}), \mathbf{V}^* \rangle(\bar{x})$.

PROOF. The proof of the above theorem easily follows from the corresponding proof in [BGGL01]. Consider a fully resetting verifier V^*. We will construct an adversary W^* in the hybrid model which would be able to simulate the view of V^*. Consider an incarnation of the prover $P^{i,j,k} = (P_1^{i,j,k}, P_2^{i,j,k})$. W^* simply

[6]Note that such a power was not available to the verifier in the hybrid world of Barak et al [BGGL01]. Giving the verifier such power makes the proof in our setting only easier while allowing us to generalize the class of (prover) admissible proof systems.

relays messages between V^* and $P^{i,j,k}$ until the point V^* resets $P^{i,j,k}$. At this point, W^* starts interaction with another incarnation of the prover $P^{i,j,k'}$ with $k \neq k'$ and simply relays messages of the first round of interaction between V^* and $P^{i,j,k'}$. At the completion of the first round (i.e., assuming that the interaction is not already aborted by this point), W^* forwards V^* the reply of $P_2^{i,j,k'}$ only if the determining message of V^* was different from the one in the interaction with $P^{i,j,k}$ (and if so, continues to relay messages between V^* and $P^{i,j,k'}$). Otherwise, if the determining message was the same, W^* aborts the interaction with $P^{i,j,k'}$ and forwards to V^* the reply received earlier from $P^{i,j,k}$ in the first round. Further, W^* requests $P_1^{i,j,k}$ to restart and then executes the same (first) round of interaction with $P_1^{i,j,k}$ as V^* did with $P_1^{i,j,k'}$(this is possible since the random tapes of $P_1^{i,j,k}$ and $P_1^{i,j,k'}$ are identical) to get "in sync". From the second round onwards, W^* again simply relays messages between V^* and $P^{i,j,k}$ with the following exception. In case a reply in the current round was already given by $P_2^{i,j,k}$ earlier, W^* would not receive a reply in that case. However, by the property of a prover admissible proof system, except with negligible probability, the transcript of interaction of this session consisting of V^* main message and the replies of $P_2^{i,j,k}$ is identical to the earlier one (since the determining message is identical). Hence, W^* forwards the reply received earlier in that case.

Now observe that the main difference between the view of V^* between when it is interacting with W^* in such a setting and when it is directly interacting with prover incarnations and resetting them is the following. W^* "switches" the interaction of V^* from $P^{i,j,k}$ to another incarnation $P^{i,j,k'}$ whenever V^* resets $P^{i,j,k}$ and starts a session with another determining message. The indistinguishability of the views in these two cases then follows from the pseudorandomness of the function f. $\square$

We now define an analogous class of admissible proof systems, hybrid model and a transformation for the case of resettable soundness. This case is symmetric to the case of resettable zero knowledge and the proof again follows from the fact that a fully resetting adversary can be emulated by an adversary in the hybrid model. For completeness, we give some of the details in the following.

Definition 7 (Verifier-admissible proof-system) *A proof-system (P, V) is called* verifier-admissible *if the following requirements hold:*

1. *The verifier V consists of two parts V_1, V_2. Similarly, the verifier's random input ω is partitioned into two disjoint parts, $\omega^{(1)}, \omega^{(2)}$, where $\omega^{(i)}$ is given to V_i*

2. *A message sent by the prover may either be labeled as* main *message or an* authenticator *message. The first main message sent by the prover in the protocol is called the* determining *message. Each prover message is first received by V_1. In each round of interaction, the prover and V_1 exchange a number of messages in which exactly one of the messages is a main message while the rest are authenticator messages. At the end of an interaction round, V_1 decides whether to accept the (only) main message received based on the transcript of interaction round itself and the transcript of the prover main messages and the corresponding replies of V_2 so far. If V_1 accepts, it forwards the main message to V_2 who generates the reply.*

3. *Let P^* be an arbitrary (*deterministic*) polynomial-size circuit, where P^* may execute a resetting-attack on V (see Definition 4). Let P^* interact with some incarnation of $V = (V_1, V_2)$. Then, except with negligible probability, P^* is unable to generate two different main messages, both accepted by V_1, for some round ℓ in two different interactions with V with the same determining message.*

The hybrid model for this case is exactly symmetric to the hybrid model for the case of prover admissible proof system (with the roles of the prover and the verifier exchanged).

Definition 8 (hs) *A* hybrid cheating prover *P^* works against verifier-admissible proof systems in the hybrid model as described above. A proof system is hs if it is verifier admissible and satisfies Definition 4 with respect to hybrid cheating provers.*

Transformation hs-rs Given a verifier-admissible proof system (P, V), where $V = (V_1, V_2)$, and a collection f of pseudorandom functions, we define a new proof system $(\mathbf{P}, \mathbf{V})$ as follows.

The new prover P is identical to P.

The new verifier V: The new verifier $\mathbf{V}$'s randomness is viewed as a pair $(\omega^{(1)}, f)$, where $\omega^{(1)} \in \{0,1\}^{\text{poly}(n)}$ is of length adequate for the random-tape of V_1 and $f : \{0,1\}^{\leq \text{poly}(n)} \to \{0,1\}^{\text{poly}(n)}$ is a description of a function taken from an ensemble of pseudorandom functions. For convenience, we describe the new verifier $\mathbf{V}$ as a pair $\mathbf{V} = (\mathbf{V}_1, \mathbf{V}_2)$. $\mathbf{V}_1$ is identical to V_1 with random-tape $\omega^{(1)}$; $\mathbf{V}_2$ emulates the actions of V_2 with a random tape that is determined by applying f to the input, the random coins $\omega^{(1)}$ and the determining message. That is, upon receiving the determining message, denote *msg*, $\mathbf{V}_2$ sets $\omega^{(2)} = f(x, \omega^{(1)}, \text{msg})$ and runs V_2 with random input $\omega^{(2)}$. From this step on, $\mathbf{V}_2$ emulates the actions of V_2 using $\omega^{(2)}$ as V_2's random-tape.

Theorem 7 *Suppose that (P, V) is verifier-admissible, and let $\mathbf{V}$ be the verifier strategy obtained from V by applying Transformation hs-rs. Then:*

Assuming that pseudorandom functions exist, for every probabilistic polynomial-time resetting cheating prover $\mathbf{P}^*$ (as in Definition 4) there exists a probabilistic polynomial-time hybrid cheating prover W^* (as in Definition 8) such that $\langle W^*, V\rangle(\bar{x})$ is computationally indistinguishable from $\langle \mathbf{P}^*, \mathbf{V}\rangle(\bar{x})$.

PROOF. The proof of the above theorem easily follows from the proof of Theorem 6 and the techniques in [CGGM00, BGGL01]. Consider a fully resetting prover P^*. We will construct an adversary W^* in the hybrid model which would be able to simulate the view of P^*. Consider an incarnation of the verifier $V^{i,j,k} = (V_1^{i,j,k}, V_2^{i,j,k})$. W^* simply relays messages between P^* and $V^{i,j,k}$ until the point P^* resets $V^{i,j,k}$. At this point, W^* starts interaction with another incarnation of the verifier $V^{i,j,k'}$ with $k \neq k'$ and simply relays messages of the first round of interaction between P^* and $V^{i,j,k'}$. At the completion of the first round (i.e., assuming that the interaction is not already aborted by this point), W^* forwards P^* the reply of $V_2^{i,j,k'}$ only if the determining message of P^* was different from the one in the interaction with $V^{i,j,k}$ (and if so, continues to relay messages between P^* and $V^{i,j,k'}$). Otherwise, if the determining message was the same, W^* aborts the interaction with $V^{i,j,k'}$ and forwards to P^* the reply received earlier from $V^{i,j,k}$ in the first round. Further, W^* requests $V_1^{i,j,k}$ to restart and then executes the same (first) round of interaction with $V_1^{i,j,k}$ as P^* did with $V_1^{i,j,k'}$(this is possible since the random tapes of $V_1^{i,j,k}$ and $V_1^{i,j,k'}$ are identical) to get "in sync". From the second round onwards, W^* again simply relays messages between P^* and $V^{i,j,k}$ with the following exception. In case a reply in the current round was already given by $V_2^{i,j,k}$ earlier, W^* would not receive a reply in that case. However, by the property of a verifier admissible proof system, except with negligible probability, the transcript of interaction of this session consisting of P^* main message and the replies of $V_2^{i,j,k}$ is identical to the earlier one (since the determining message is identical). Hence, W^* forwards the reply received earlier in that case.

Now observe that the main difference between the view of P^* between when it is interacting with W^* in such a setting and when it is directly interacting with verifier incarnations and resetting them is the following. W^* "switches" the interaction of P^* from $V^{i,j,k}$ to another incarnation $V^{i,j,k'}$ whenever P^* resets $V^{i,j,k}$ and starts a session with another determining message. The indistinguishability of the views in these two cases then follows from the pseudorandomness of the function f. □

We remark that our hZK-rZK transformation (resp., hs-rs transformation) only specifies a new prover (resp., verifier) strategy without changing the verifier (resp., prover) strategy. Hence, if the original proof system is resettably sound (resp., resettable zero knowledge), so is the transformed proof system. Similar statement holds for the case of hybrid soundness and concurrent soundness (resp., hybrid zero knowledge, concurrent zero knowledge and relaxed concurrent zero knowledge).

D High-Level Idea of the Protocol

The central idea behind our result concerns a novel non-black-box simulation strategy. To understand this idea (and why it is useful), we will discuss the simpler goal of building a resettably-sound *concurrent* zero-knowldge argument[7]. We begin by briefly recalling how all known concurrent zero-knowledge protocols work in the plain model, for proving "$x \in L$" where L is an NP-complete language [RK99, KP01, PRS02], at an informal level. The high level idea is this: First the verifier commits to a "secret". Then, the prover and verifier do the following many times sequentially (over many rounds of interaction): the prover makes a "challenge" to which the verifier responds, with the properties that: (1) a single challenge-response from each round reveals no information about the secret or the randomness used to commit to the secret, but (2) any two distinct challenge-response pairs *from the same round* reveal the secret and the randomness used to commit to the secret[8]. Then, the prover proves the following using an ordinary zero-knowledge (or WI) proof: that either the prover knows the verifier's secret, or that $x \in L$. This works because in real life, the prover only gets one response for each round, and therefore he cannot learn the verifier's secret. However, in a simulation, the simulator can "rewind" the verifier and try to get two challenge-response pairs for some round, and thereby learn the verifier's secret (and the randomness the verifier used to commit to his secret). As discussed earlier, the concurrent setting requires such a simulation to use a *recursive rewinding* strategy, in order to successfully "solve" every execution of the protocol as they arise.

As discussed earlier, such concurrent zero-knowledge protocols are certainly not resettably sound, since if a cheating prover could reset the verifier, it could use the same rewinding strategy to discover the verifier's secret and use it to cheat (just like the concurrent zero-knowledge simulator does). A simple idea to fix this problem is the following: Have the prover commit to all his challenges in advance, and then in the challenge-response phase, have the prover give a *resettably-sound* zero-knowledge argument that his challenges are the same as the ones that he committed to earlier. Now, the cheating prover can't cheat even if he can reset the verifier. But there seems to be a circularity here: thinking back to the concurrent zero-knowledge simulator, in order to extract the verifiers' secrets, it needs to give different challenges from the ones it commits to, so it will need to simulate the resettably-sound zero-knowledge argument so that it can lie. But we are in the concurrent setting, so it seems that we will need a resettably-sound *concurrent* zero-knowledge argument for this. Indeed, in general this is the case, and it may appear that we haven't made any progress.

To resolve this situation, we can try to take a look inside the guts of the resettably-sound zero-knowledge argument of Barak et al. [BGGL01]. As described earlier, the idea is to use the non-black-box zero-knowledge protocol of Barak [Bar01]. The core idea behind Barak's protocol is to have the prover commit in advance to a program that can predict a string that is later randomly chosen by the verifier. The prover then must prove that either its committed program really can predict the verifier's string, or that the statement is true (in our case, that the prover's challenge is what he committed to earlier). In a real execution, the program is information theoretically extremely unlikely to be able to predict the verifier's random value. But in simulation (in the stand-alone setting), where the simulator can choose the verifier's random coins in advance and commit to these coins and the verifier's code, the simulator can ensure that the program mimics the verifier's execution of the protocol and therefore correctly predicts the verifier's string.

However, Barak's protocol is not fully concurrent zero-knowledge, and most natural approaches to try to extend it to the fully concurrent scenario either cause soundness to fail or lead to exponential-time simulation. Let us look at one approach, in our context: Recall that the program that the simulator commits to must regenerate the transcript of the interaction of *all* the concurrent executions up until the point that the adversary in the *current* execution outputs his randomly chosen string (within Barak's protocol). Let us consider having the simulator commit to an *exponential-time* program, one that runs the adversary's code to regenerate the

[7]Indeed, we follow something similar to this in our actual technical approach, although our first goal is something slightly weaker than concurrent zero-knowledge. See below for a technical outline of our paper.

[8]The requirement to be able to extract the randomness is actually not standard for concurrent zero-knowledge protocols from the literature, but it is important for our approach.

transcript, but whenever the adversary commits to a verifier secret in one of the concurrent executions of the overall protocol, then the program uses exponential time to break the commitment and recover the secret. This secret can then be used by the program to simulate any other protocol executions that arise before the program is able to predict the desired verifier string. While this doesn't make sense for us yet (since it requires exponential time), it puts us on our path to solving the problem, based on the following crucial points:

- Soundness will still hold with regard to this protocol, because even an exponential-time deterministic program can't predict a random value chosen after the program is fixed.

- Our program's only use of exponential time is to break the verifier's commitments to his secrets. In other words, our program is actually a polynomial-time program that needs oracle-access to a commitment-breaking oracle.

- We're now trying to eliminate the exponential-time requirement in this non-black-box simulation. Recall that if our idea will work, then by the property of the overall recursive rewinding strategy, when the simulator actually needs to prove that its program can predict the verifier's string, the simulator will have already extracted the verifiers' secrets (and randomness used to commit to those secrets) for all the verifiers that have appeared in any concurrent executions between when we committed to the program and the time when we need to complete the proof. Therefore, the simulator already knows the secrets and randomness corresponding to all the commitment-breaking oracle queries that our (now polynomial-time) program will ever make.

  So, instead of implementing the oracle with an exponential-time machine, we can have the simulator implement the oracle by providing a list of all the verifier secrets and randomness that it has learned so far. When the program makes an oracle query, the list is inspected to see if the right response to the query is in the list. If it is not, then the program halts and fails. Assuming that the commitment scheme is one-to-one, there can only be one correct answer to any query. Therefore, the program will only output one fixed value (or halt and fail), no matter what list the simulator specifies. Thus, except for the additional possibility that the program halts and fails, it will behave exactly as the original exponential-time program did. This preserves the soundness of the protocol.

To make this approach work, aside from the main idea above, we also make use of several other (new and old) ideas, including a new recursive rewinding technique inspired by [RK99]. At its core, our new non-black-box simulation strategy allows for protocols that make essential use of non-black-box simulation but that can also benefit from information learned using black-box recursive-rewinding simulation methods. Given that previous non-black-box simulation advances have had an impact on numerous advanced cryptographic research goals (e.g. [Bar01, Bar02, Pas04, PR05, BS05]), we believe that our new strategy will have other applications as well.