Constant-Round Concurrent Zero Knowledge in the Bounded Player Model

Vipul Goyal[1], Abhishek Jain[2], Rafail Ostrovsky[3], Silas Richelson[4], and Ivan Visconti[5]

[1] Microsoft Research, INDIA, vipul@microsoft.com
[2] MIT and Boston University, USA, abhishek@csail.mit.edu
[3] UCLA, USA, rafail@cs.ucla.edu
[4] UCLA, USA, sirichel@math.ucla.edu
[5] University of Salerno, ITALY, visconti@dia.unisa.it

Abstract. In [18] Goyal et al. introduced the bounded player model for secure computation. In the bounded player model, there are an a priori bounded number of players in the system, however, each player may execute any unbounded (polynomial) number of sessions. They showed that even though the model consists of a relatively mild relaxation of the standard model, it allows for round-efficient concurrent zero knowledge. Their protocol requires a super-constant number of rounds. In this work we show, constructively, that there exists a *constant-round* concurrent zero-knowledge argument in the bounded player model. Our result relies on a new technique where the simulator obtains a trapdoor corresponding to a player identity by putting together information obtained in multiple sessions. Our protocol is only based on the existence of a collision-resistance hash-function family and comes with a "straight-line" simulator.

We note that this constitutes the strongest result known on constant-round concurrent zero knowledge in the plain model (under well accepted relaxations) and subsumes Barak's constant-round bounded concurrent zero-knowledge result. We view this as a positive step towards getting constant round fully concurrent zero-knowledge in the plain model, without relaxations.

Keywords: concurrent zero knowledge, straight-line simulation, bounded player model.

1 Introduction

The notion of a zero-knowledge proof [17] is central in cryptography, both for its conceptual importance and for its wide ranging applications to the design of secure cryptography protocols. Initial results for zero-knowledge were in the so called stand-alone setting where there is a single protocol execution happening in isolation.

The fact that on the Internet an adversary can control several players motivated the notion of concurrent zero knowledge [15] (cZK). Here the prover is simultaneously involved in several sessions and the scheduling of the messages is

coordinated by the adversary who also keeps control of all verifiers. Concurrent zero knowledge is much harder to achieve than zero knowledge. Indeed, while we know how to achieve zero-knowledge in 4 rounds, a sequence of results [21,33,6] increased the lower bound on the round complexity of concurrent zero-knowledge with black-box simulation to almost logarithmic in the security parameter. In the meanwhile, the upper bound has been improved and now almost matches the logarithmic lower bound [31,20,30]. After almost a decade of research on this topic, the super-logarithmic round concurrent zero-knowledge protocol of Prabhakaran et al. [30] remains the best known in terms of round complexity.

Some hope for a better round complexity started from the breakthrough result of Barak [1] where non-black-box simulation under standard assumptions was proposed. His results showed how to obtain bounded-concurrent zero knowledge in constant rounds. This refers to the setting where there is an a priori fixed bound on the total number of concurrent executions (and the protocol may become completely insecure if the actual number of sessions exceed this bound). Unfortunately, since then, the question of achieving sub-logarithmic round complexity with unbounded concurrency using non-black-box techniques has remained open, and represents one of the most challenging open questions in the study of zero-knowledge protocols. [6]

Bounded player model. Recently, Goyal, Jain, Ostrovsky, Richelson and Visconti [18] introduced the so called bounded player model. In this model, it is only assumed that there is an a-priori (polynomial) upper-bound on the total number of players that may ever participate in protocol executions. There is no setup stage, or, trusted party, and the simulation must be performed in polynomial time. While there is a bound on the number of players, any player may join in at any time and may be subsequently involved in any unbounded (polynomial) number of concurrent sessions. Since there is no a priori bound on the number of sessions, it is a strengthening of the bounded-concurrency model used in Barak's result. The bounded player model also has some superficial similarities to the bare-public-key model of [5] which is discussed later in this section.

As an example, if we consider even a restriction to a single verifier that runs an unbounded number of sessions, the simulation strategy of [1] breaks down completely. Goyal et al. [18] gave a $\omega(1)$-round concurrent zero knowledge protocol in the bounded player model. The technique they proposed relies on the fact that the simulator has several choices in every sessions on where to spend computation trying to extract a trapdoor, and, its running time is guaranteed to be polynomial as long as the number of such choices is super-constant. Their technique fails inherently if constant round-complexity is desired.

We believe the eventual goal of achieving round efficient concurrent zero-knowledge (under accepted assumptions) is an ambitious one. Progress towards

[6] In this paper, we limit our discussion to results which are based on standard complexity-theoretic and number-theoretic assumptions. We note that constant round concurrent zero-knowledge is known to exist under non-standard assumptions such as a variation of the (non-falsifiable) knowledge of exponent assumption [19] or the existence of P-certificates [8].

this goal would not only impact how efficiently one can implement zero-knowledge (in the network setting), but also, will improve various secure computation protocol constructions in this setting (as several secure computation protocols use, e.g., PRS preamble [30] for concurrent input extraction). Bounded player model is somewhere between the standard model (where the best known protocols require super-logarithmic number of rounds), and, the bounded concurrency model (where constant round protocols are known). We believe the study of round complexity of concurrent zero-knowledge in the bounded player model might shed light on how to construct such protocols in the standard model as well.

Our Results. In this work, we give a constant-round protocol in the bounded player (BP) model. Our constructions inherently relies on non-black-box simulation. The simulator for our protocol does not rely on rewinding techniques and instead works in a "straight-line" manner (as in Barak [1]). Our construction is only based on the existence of a collision-resistant hash-function family.

Theorem 1. *Assuming the existence of a collision-resistance hash-function family, there exists a constant round concurrent zero-knowledge argument system with concurrent soundness in the bounded player model.*

We note that this constitutes the strongest result known on constant-round zero-knowledge in the concurrent setting (in the plain model). It subsumes Barak's result: now the total number of sessions no longer needs to be bounded; only the number of new players starting the interaction with the prover is bounded. A player might join in at anytime and may subsequently be involved in any unbounded (polynomial) number of sessions.

We further note that, as proved by Goyal et al. [18], unlike previously studied relaxations of the standard model (e.g., bounded number of sessions, timing assumptions, super- polynomial simulation), concurrent-secure computation is still impossible to achieve in the bounded player model. This gives evidence that the BP model is "closer" to the standard model than previously studied models, and study of this model might shed light on constructing constant-round concurrent zero-knowledge in the standard model as well. Moreover, despite the impossibility of concurrent-secure computation, techniques developed in the concurrent zero-knowledge literature have found applications in other areas in cryptography, including resettable security [5], non-malleability [14], and even in proving black-box lower bounds [27].

1.1 Technical Overview

In this section, first, we recall some observations by Goyal et al [18] regarding why simple approaches to extend the construction of Barak [1] to the bounded player model are bound to fail. We also recall the basic idea behind the protocol of [18]. Armed with this background, we then proceed to discuss the key technical ideas behind our constant round cZK protocol in the bounded player model. Initial parts of this section are borrowed verbatim from [18].

Why natural approaches fail. Recall that in the bounded player model, the only assumption is that the total number of players that will ever be present in the system is *a priori* bounded. Then, as observed by Goyal et al [18], the black-box lower-bound of Canetti et al. [6] is applicable to the bounded player model as well. Thus, it is clear that we must resort to non-black-box techniques. Now, a natural approach to leverage the bound on the number of players is to associate with each verifier V_i a public key pk_i and then design an FLS-style protocol [16] that allows the ZK simulator to extract, in a non-black-box manner, the secret key sk_i of the verifier and then use it as a "trapdoor" for "easy" simulation. The key intuition is that once the simulator extracts the secret key sk_i of a verifier V_i, it can perform easy simulation of *all* the sessions associated with V_i. Then, since the total number of verifiers is bounded, the simulator will need to perform non-black-box extraction only an *a priori* bounded number of times (once for each verifier), which can be handled in a manner similar to the setting of bounded-concurrency [1].

Unfortunately, as observed by Goyal et al. [18], the above intuition is misleading. In order to understand the problem with the above approach, let us first consider a candidate protocol more concretely. In fact, it suffices to focus on a preamble phase that enables non-black-box extraction (by the simulator) of a verifier's secret key since the remainder of the protocol can be constructed in a straightforward manner following the FLS approach. Now, consider the following candidate preamble phase (using the non-black-box extraction technique of [3]): first, the prover and verifier engage in a coin-tossing protocol where the prover proves "honest behavior" using a Barak-style non-black-box ZK protocol [1]. Then, the verifier sends an encryption of its secret key under the public key that is determined from the output of the coin-tossing protocol [18].

In order to analyze this protocol, we will restrict our discussion to the simplified case where only one verifier is present in the system (but the total number of concurrent sessions are unbounded). At this point, one may immediately object that in the case of a single verifier identity, the problem is not interesting since the bounded player model is identical to the bare-public key model, where one can construct four-round cZK protocols using rewinding based techniques. However, simulation techniques involving rewinding do not "scale" well to the case of polynomially many identities (unless we use a large number of rounds) and fail. In contrast, our simulation approach is "straight-line" for an unbounded number of sessions and scales well to a large bounded number of identities. Therefore, in the forthcoming discussion, we will restrict our discussion to straight-line simulation. In this case, we find it instructive to focus on the case of a single identity to explain the key issues and our ideas to resolve them.

We now turn to analyze the candidate protocol. Now, following the intuition described earlier, one may think that the simulator can simply cheat in the coin-tossing protocol in the "inner-most" session in order to extract the secret key, following which all the sessions can be simulated in a straight-line manner, without performing any additional non-black-box simulation. Consider, however, the following adversarial verifier strategy: the verifier schedules an unbounded

number of sessions in such a manner that the coin-tossing protocols in all of these sessions are executed in a "nested" manner. Furthermore, the verifier sends the ciphertext (containing its secret key) in each session only *after* all the coin-tossing protocols across all sessions are completed. Note that in such a scenario, the simulator would be forced to perform non-black-box simulation in an unbounded number of sessions. Unfortunately, this is a non-trivial problem that we do not know how to solve.

The approach of Goyal et al. [18]. In an effort to bypass the above problem, Goyal et al. use multiple ($\omega(1)$, to be precise) preamble phases (instead of only one), such that the simulator is required to "cheat" in only one of these preambles. This, however, immediately raises a question: in which of the $\omega(1)$ preambles should the simulator cheat? This is a delicate question since if, for example, we let the simulator pick one of preambles uniformly at random, then with non-negligible probability, the simulator will end up choosing the first preamble phase. In this case, the adversary can simply perform the same attack as it did earlier playing only the first preamble phase, but for many different sessions so that the simulator will still have to cheat in many of them. Indeed, it would seem that any randomized oblivious simulation strategy can be attacked in a similar manner by simply identifying the first preamble phase where the simulator would cheat with a non-negligible probability.

The main idea in [18] is to use a specific probability distribution such that the simulator cheats in the first preamble phase with only negligible probability, while the probability of cheating in the later preambles increases gradually such that the "overall" probability of cheating is 1 (as required). Further, the distribution is such that the probability of cheating in the i^{th} preamble is less than a fixed polynomial factor of the total probability of cheating in one of the previous $i-1$ blocks. This allows them (by a careful choice of parameters) to ensure that the probability of the simulator failing in more than a given polynomially bounded number of sessions w.r.t. any given verifier is negligible (and then rely on the techniques from the bounded-concurrency model [1] to handle the bounded number of non-black-box simulations).

Our Construction. The techniques used in our work are quite different and unrelated to the techniques in the work of Goyal et al. [18]. As illustrated in the discussion above, the key issue is the following. Say that a slot of the protocol completes. Then, the simulator starts the non-black-box simulation and computes the first "heavy" universal argument message, and, sends it across. However, before the simulator can finish this simulation successfully (and somehow learn a trapdoor from the verifier which can then be used to complete other sessions without non-black-box simulation), the verifier switches to another session. Then, in order to proceed, the simulator would have to perform non-black-box simulation and the heavy computation again (resulting in the number of sessions where non-black-box simulation is performed becoming unbounded). So overall, the problem is the "delay" between the heavy computation, and, the point at which the simulator extracts the verifier trapdoor (which can then be

used to quickly pass through other sessions with this particular verifier without any heavy computation or non-black-box simulation).

Our basic approach is to "construct the trapdoor slowly as we go along": have any heavy computation done in any session (with this verifier) contribute to the construction of a trapdoor which can then be used to quickly pass through other sessions. To illustrate our idea, we shall focus on the case of a single verifier as before. The description below is slightly oversimplified for the sake of readability.

To start with, in the very first session, the verifier is supposed to choose a key pair of a signature scheme (this key pair remains the same across all sessions involving this verifier). As in Barak's protocol [1], we will just have a single slot followed by a universal argument (UA). However, now once a slot is complete, the verifier is required to immediately send a signature[7] on the transcript of the slot (i.e., on the prover commitment, and, the verifier random string) to the prover. This slot now constitutes a "hard statement" certified by the verifier: it could be used by the prover in any session (with this verifier). If the prover could prove that he has a signed slot such that the machine committed to in this slot could output the verifier random string in this slot, the verifier would be instructed to accept. Thus, the simulator would now simply take the first slot that completes (across all sessions), and, would prove the resulting "hard statement" in the universal arguments of all the sessions. This would allow him to presumably compute the required PCP only once and use it across all sessions. Are we done? Turns out that the answer is no.

Even if the prover is executing the UA corresponding to the same slot (on which he has obtained a signature) in every session, because of the interactive nature of UAs, the (heavy) computation the prover does in a session cannot be entirely used in another session. This is because the challenge of the verifier would be different in different sessions. To solve this problem and continue the construction of a single trapdoor (useful across all sessions), we apply our basic idea one more time. The prover computes and sends the first UA message. The verifier is required to respond with a random challenge and a signature on the UA transcript so far. The prover can compute the final UA message, and, the construction of the trapdoor is complete: the trapdoor constitutes of a signed slot, an accepting UA transcript (proving that the machine committed to in the slot indeed outputs the random string in that slot), and, a signature on the first two UA messages (proving that the challenge was indeed generated by the verifier after getting the first UA message). To summarize, the simulator would use the following two sessions for the construction of the trapdoor: the first session where a slot completes, and, the first session where the verifier sends the UA random challenge.

The above idea indeed is oversimplified and ignores several problems. Firstly, since an honest prover executes each concurrent session oblivious of others, any correlations in the prover messages across different sessions (in particular, send-

[7] Signatures of committed messages computed by a verifier where previously used in [12] to allow the simulator to get through rewindings one more signature in order to cheat in the main thread. Here instead we insist with straight-line simulation.

ing the same UA first message) would lead to the simulated transcript being distinguishable from the real one. Furthermore, the prover could be proving a different overall statement to the verifier in every session (and hence even a UA first message cannot be reused across different sessions). The detailed description of our construction is given in Section 3.

1.2 Related Work

Bare public key and other related models. The bare public key model was proposed in [5] where, before any interaction starts, every player is required to declare a public key and store it in a public file (which never changes once the sessions start). In this model it is known how to obtain constant-round concurrent zero knowledge with concurrent soundness under standard assumptions [13,35,36,34]. This model has also been used for constant-round concurrent non-malleable zero knowledge [25] and various constant-round resettable and simultaneously resettable protocols [22,39,11,9,10,38,37,7].

As discussed in [18], the crucial restriction of the BPK model is that all players who wish to ever participate in protocol executions must be fixed during the preprocessing phase, and new players cannot be added "on-the-fly" during the proof phase. We do *not* make such a restriction in our work and, despite superficial resemblance, the techniques useful in constructing secure protocols in the BPK model have limited relevance in our setting. In particular, constant round cZK is known to exist in the BPK model using only black-box simulation, while in our setting, non-black-box techniques are *necessary* to achieve sublogarithmic-round cZK.

In light of the above discussion, since the very premise of the BPK model (that all players are fixed ahead of time and declare a key) does not hold in the bounded player model, we believe that the bounded player model is much closer in spirit (as well as technically) to the bounded concurrency model of Barak. The bounded player model is a strict generalization of the bounded concurrency model. Thus, our constant-round construction is the first strict improvement to Barak's bounded concurrent ZK protocol. We stress that we improve the achieved security under concurrent composition, still under standard assumptions and without introducing any setup/weakness. Summing up, ours is a construction which is the closest known to achieving constant-round concurrent zero knowledge in the plain model.

Round efficient concurrent zero-knowledge is known in a number of other models as well (which do not seem to be directly relevant to our setting) such as the common-reference string model, the super-polynomial simulation model, etc. We refer the reader to [18] for a more detailed discussion.

2 Preliminaries and Definitions

Notation. We will use the symbol "||" to denote the concatenation of two strings appearing respectively before and after the symbol.

2.1 Bounded Player Model

We first recall the *bounded player model* for concurrent security, as introduced in [18]. In the bounded player model, there is an a-priori (polynomial) upper bound on the total number of player that will ever be present in the system. Specifically, let n denote the security parameter. Then, we consider an upper bound $N = \text{poly}(n)$ on the total number of players that can engage in concurrent executions of a protocol at any time. We assume that each player P_i ($i \in N$) has an associated unique identity id_i, and that there is an established mechanism to enforce that party P_i uses the same identity id_i in each protocol execution that it participates in. Note, however, that such identities do not have to be established in advance. In particular, new players can join the system with their own (new) identities, as long as the number of players does not exceed N. We stress that there is not bound on the number of protocol executions that can be started by each party.

The bounded player model is formalized by means of a functionality F_{bp}^N that registers the identities of the player in the system. Specifically, a player P_i that wishes to participate in protocol executions can, at any time, register an identity id_i with the functionality F_{bp}^N. The registration functionality does not perform any checks on the identities that are registered, except that each party P_i can register at most one identity id_i, and that the total number of identity registrations are bounded by N. In other words, F_{bp}^N refuses to register any new identities once N number of identities have already been registered. The functionality F_{bp}^N is formally defined in Figure 1.

Functionality F_{bp}^N

F_{bp}^N initializes a variable *count* to 0 and proceeds as follows.

- **Register commands:** Upon receiving a message $(\textbf{register}, sid, \mathsf{id}_i)$ from some party P_i, the functionality checks that no pair (P_i, id_i') is already recorded and that $count < N$. If this is the case, it records the pair (P_i, id_i) and sets $count = count + 1$. Otherwise, it ignores the received message.
- **Retrieve commands:** Upon receiving a message $(\mathsf{retrieve}, sid, P_i)$ from some party P_j or the adversary A, the functionality checks if some pair (P_i, id_i) is recorded. If this the case, it sends $(sid, P_i, \mathsf{id}_i)$ to P_j (or A). Otherwise, it returns $(sid, P_i, \perp)$.

Fig. 1. The Bounded Player Functionality F_{bp}^N.

In our constructions we will explicitly work in the setting where the identity of each party is a tuple (h, vk), where $h \leftarrow \mathcal{H}_n$ is a hash function chosen from a

family $\mathcal{H}_n$ of collision resistant hash functions, and vk is a verification key for a signature scheme.

2.2 Concurrent Zero Knowledge in Bounded Player Model

In this section, we formally define concurrent zero knowledge in the bounded player model. The definition given below, is an adaptation of the one of [30] to the bounded player model, by also considering non-black-box simulation. Some of the text below is taken verbatim from [30].

Let PPT denote probabilistic-polynomial time. Let $\langle P, V \rangle$ be an interactive argument for a language L. Consider a concurrent adversarial verifier V^* that, given input $x \in L$, interacts with an unbounded number of independent copies of P (all on the same common input x and moreover equipped with a proper witness w), without any restriction over the scheduling of the messages in the different interactions with P. In particular, V^* has control over the scheduling of the messages in these interactions. Further, we say that V^* is an N-bounded concurrent adversary if it assumes at most N verifier identities during its (unbounded) interactions with P.[8]

The transcript of a concurrent interaction consists of the common input x, followed by the sequence of prover and verifier messages exchanged during the interaction. We denote by $\mathsf{view}^P_{V^*}(x, z, N)$ the random variable describing the content of the random tape of the N-bounded concurrent adversary V^* with auxiliary input z and the transcript of the concurrent interaction between P and V^* on common input x.

Definition 1 (Concurrent Zero Knowledge in Bounded Player Model). *Let $\langle P, V \rangle$ be an interactive argument system for a language L. We say that $\langle P, V \rangle$ is* concurrent zero-knowledge *in the bounded player model if for every N-bounded concurrent non-uniform* PPT *adversary V^*, there exists a* PPT *algorithm $\mathcal{S}$, such that the following ensembles are computationally indistinguishable, $\{\mathsf{view}^P_{V^*}(x, z, N)\}_{x \in L, z \in \{0,1\}^*}$ and $\{\mathcal{S}(x, z, N)\}_{x \in L, z \in \{0,1\}^*}$.*

As a final note, we remark that following previous work in the BPK model and in the BP model, we will consider the notion of concurrent soundness where the malicious prover is allowed to play any concurrent number of sessions with the same verifier. Indeed, this is notion is strictly stronger than sequential soundness.

2.3 Building Blocks

In this section, we discuss the main building blocks that we will use in our cZK construction.

[8] Thus, V^* can open multiple sessions with P for every unique verifier identity.

Statistically binding commitment schemes. In our constructions, we will make use of a statistically binding string commitment scheme, denoted **Com**. For simplicity of exposition, we will make the simplifying assumption that **Com** is a non-interactive perfectly binding commitment scheme. In reality, **Com** would be taken to be a standard 2-round commitment scheme, e.g. [24]. Unless stated otherwise, we will simply use the notation $\mathbf{Com}(x)$ to denote a commitment to a string x, and assume that the randomness (used to create the commitment) is implicit. We will denote by $\mathbf{Com}(x; r)$ a commitment to a string x with randomness r.

Witness indistinguishable arguments of knowledge. We will also make use of a witness-indistinguishable proof of knowledge (WIPOK) for all of $\mathcal{NP}$ in our construction. Such a scheme can be constructed, for example, by parallel repetition of the 3-round Blum's protocol for Graph Hamiltonicity [4]. We will denote such an argument system by $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}} \rangle$.

The universal argument of [2]. In our construction, we will use the 4-round universal argument system (UA), denoted pUA presented in [2] and based on the existence of collision-resistant hash functions. We will assume without loss of generality that the initial commitment of the PCP sent by the prover in the second round also contains a commitment of the statement. We notice that such an argument system is still sound when the prover is required to open the commitment of the statement in the very last round.

Signature schemes. We will use a signature scheme $(\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify})$ that is unforgeable against chosen message attacks. Note that such signatures schemes are known based on one way functions [32].

3 A Constant-Round Protocol

In this section, we describe our constant-round concurrent zero-knowledge protocol in the bounded player model.

Relation R_{sim}. We first recall a slight variant of Barak's [1] $\mathbf{NTIME}(T(n))$ relation R_{sim}, as used previously in [28]. Let $T : \mathbb{N} \to \mathbb{N}$ be a "nice" function that satisfies $T(n) = n^{\omega(1)}$. Let $\{\mathcal{H}_n\}_n$ be a family of collision-resistant hash functions where a function $h \in \mathcal{H}_n$ maps $\{0,1\}^*$ to $\{0,1\}^n$, and let **Com** be a perfectly binding commitment scheme for strings of length n, where for any $\alpha \in \{0,1\}^n$, the length of $\mathbf{Com}(\alpha)$ is upper bounded by $2n$. The relation R_{sim} is described in Figure 2.

Remark 1. The relation presented in Figure 2 is slightly oversimplified and will make Barak's protocol work only when $\{\mathcal{H}_n\}_n$ is collision-resistant against "slightly" super-polynomial sized circuits. For simplicity of exposition, in this manuscript, we will work with this assumption. We stress, however, that as discussed in prior works *[2,26,29,28,18]*, this assumption can be relaxed by using

Instance: A triplet $\langle h, c, r\rangle \in \mathcal{H}_n \times \{0,1\}^n \times \{0,1\}^{\mathrm{poly}(n)}$.
Witness: A program $\Pi \in \{0,1\}^*$, a string $y \in \{0,1\}^*$ and a string $s \in \{0,1\}^{\mathrm{poly}(n)}$.
Relation: $R_{\mathsf{sim}}(\langle h, c, r\rangle, \langle \Pi, y, s\rangle) = 1$ if and only if:

1. $|y| \leq |r| - n$.
2. $c = \mathbf{Com}(h(\Pi); s)$.
3. $\Pi(y) = r$ within $T(n)$ steps.

Fig. 2. R_{sim} - A variant of Barak's relation [28]

a "good" error-correcting code ECC (with constant distance and polynomial-time encoding and decoding procedures), and replacing the condition $c = \mathbf{Com}(h(\Pi); s)$ with $c = \mathbf{Com}(\mathsf{ECC}(h(\Pi)); s)$.

Our protocol. We are now ready to present our concurrent zero knowledge protocol, denoted $\langle P, V\rangle$. Let P and V denote the prover and verifier respectively. Let N denote the bound on the number of verifiers in the system. In our construction, the identity of a verifier V_i corresponds to a verification key vk_i of a secure signature scheme and a hash function $h_i \in \mathcal{H}_n$ from a family $\mathcal{H}_n$ of collision-resistant hash functions. Let $(\mathbf{KeyGen}, \mathbf{Sign}, \mathbf{Verify})$ be a secure signature scheme. Let $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}}\rangle$ be a witness-indistinguishable argument of knowledge system. Let pUA be the universal argument (UARG) system of [2] that we discussed previously; the transcript is composed by four messages $(h, \beta, \gamma, \delta)$ where h is a collision-resistant hash function.

The protocol $\langle P, V\rangle$ is described in Figure 3. For our purposes, we set the length parameter $\ell(N) = N \cdot P(n) + n$, where $P(n)$ is a polynomial upper bound on the total length of the prover messages in the UARG pUA plus the output length of a hash function $h \in \mathcal{H}_n$. For simplicity we omit some standard checks (e.g., the prover needs to check that vk and h are recorded, the prover needs to check that the signatures is valid).

The completeness property of $\langle P, V\rangle$ follows immediately from the construction. Next, in Section 3.2, we prove *concurrent soundness* of $\langle P, V\rangle$, i.e., we show that a computationally-bounded adversarial prover who engages in multiple concurrent executions of $\langle P, V\rangle$ (where the scheduling across the sessions is controlled by the adversary) can not prove a false statement in any of the executions, except with negligible probability. As observed in [18], "stand-alone" soundness does not imply concurrent soundness in the bounded player model. Informally speaking, this is because the standard approach of reducing concurrent soundness to stand-alone soundness by "internally" emulating all but one verifier does not work since the verifier's keys are private.[9]

[9] Indeed, Micali and Reyzin [23] gave concrete counter-examples to show that stand-alone soundness does not imply concurrent soundness in the bare public key model. It is not difficult to see that their results immediately extend to the bounded player model.

Parameters: Security parameter n, number of players $N = N(n)$, length parameter $\ell(N)$.
Common Input: $x \in \{0,1\}^{\mathrm{poly}(n)}$.
Private Input to P: A witness w s.t. $R_L(x,w) = 1$.
Private Input to V: A key pair $(sk, vk) \xleftarrow{R} \mathbf{KeyGen}(1^n)$, and a hash function $h \xleftarrow{R} \mathcal{H}_n$.

Stage 1 (Preamble Phase):
$V \to P$: Send vk, h.
$P \to V$: Send $c = \mathbf{Com}(0^n)$.
$V \to P$: Send $r \xleftarrow{R} \{0,1\}^{\ell(N)}$, and $\sigma = \mathbf{Sign}_{sk}(c\|r)$.
$P \to V$: Send $c' = \mathbf{Com}(0^n)$.
$V \to P$: Send $\gamma \xleftarrow{R} \{0,1\}^n$, and $\sigma' = \mathbf{Sign}_{sk}(c'\|\gamma)$.
Stage 2 (Proof Phase):
$P \leftrightarrow V$: An execution of WIPOK $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}} \rangle$ to prove the OR of the following statements:
1. $\exists w \in \{0,1\}^{\mathrm{poly}(|x|)}$ s.t. $R_L(x,w) = 1$.
2. $\exists \langle c, r, \sigma \rangle$, and $\langle \beta, \gamma, \delta, c', t, \sigma' \rangle$ s.t.
   - $\mathbf{Verify}_{vk}(c\|r;\sigma) = 1$, and
   - $c' = \mathbf{Com}(\beta; t)$, and $\mathbf{Verify}_{vk}(c'\|\gamma;\sigma') = 1$, and
   - $(h, \beta, \gamma, \delta)$ is an accepting transcript for a UARG pUA proving the following statement: $\exists \langle \Pi, y, s \rangle$ s.t. $R_{\mathsf{sim}}(\langle h, c, r \rangle, \langle \Pi, y, s \rangle) = 1$.

Fig. 3. Protocol $\langle P, V \rangle$

We now turn to prove that protocol $\langle P, V \rangle$ is concurrent zero-knowledge in the bounded player model.

3.1 Proof of Concurrent Zero Knowledge

In this section, we prove that the protocol $\langle P, V \rangle$ described in Section 3 is concurrent zero-knowledge in the bounded player model. Towards this end, we will construct a non-black-box (polynomial-time) simulator and then prove that the concurrent adversary's view output by the simulator is indistinguishable from the real view. We start by giving an overview of the proof and then proceed to give details.

Overview. Recall that unlike the bounded concurrency model, the main challenge in the bounded player model is that the total number of sessions that a concurrent verifier may schedule is not a priori bounded. Thus, one can not directly employ Barak's simulation strategy of committing to a machine that takes only a bounded-length input y (smaller than the challenge string r) and outputs the next message of the verifier. Towards this end, the crucial observation in [18] is that in the bounded player model, once the simulator is able to "solve" the identity of a specific verifier, then it does not need to be perform any more

"expensive" (Barak-style) non-black-box simulation for that identity. Then, the main challenge remaining is to ensure that the expensive non-black-box simulations that need to be performed *before* the simulator can solve a particular identity, can be a-priori bounded, regardless of the number of concurrent sessions opened by the verifier. Indeed, [18] use a randomized simulation strategy (that crucially relies on a super-constant number of rounds) to achieve this effect.

In our case, we also build on the same set of observations. However, we crucially follow a different strategy to a-priori bound the number of expensive non-black-box simulations that need to performed in order to solve a given identity. In particular, unlike [18], where the "trapdoor" for a given verifier simply corresponds to its secret key, in our case, the trapdoor consists of a signed statement and a corresponding universal argument proof transcript (where the signature is computed by the verifier using the signing key corresponding to its identity). Further, and more crucially, unlike [18], where the simulator makes a "disjoint" effort in each session corresponding to a verifier to extract the trapdoor, in our case, the simulator gradually builds the trapdoor by making "joint" effort across the sessions. In fact, our simulator only performs *one* expensive non-black-box simulation per identity; as such, the a-priori bound on the number of identities immediately yields us the desired effect. Indeed, this is why we can perform concurrent simulation in only a constant number of rounds.

The Simulator. We now proceed to describe our simulator $\mathcal{S}$. Let N denote the a priori bound on the number of verifiers in the system. Then, the simulator $\mathcal{S}$ interacts with an adversary $V^* = (V_1^*, \ldots, V_N^*)$ who controls verifiers $V_1, \ldots, V_N$. V^* interacts with $\mathcal{S}$ in m sessions, and controls the scheduling of the messages. $\mathcal{S}$ is given non-black-box access to V^*.

The simulator $\mathcal{S}$ consists of two main subroutines, namely, $\mathcal{S}_{\mathsf{easy}}$ and $\mathcal{S}_{\mathsf{heavy}}$. As the name suggests, the job of $\mathcal{S}_{\mathsf{heavy}}$ is to perform the "expensive" non-black-box simulation operations, namely, constructing the transcripts of universal arguments, which yield a trapdoor for every verifier V_i. On the other hand, $\mathcal{S}_{\mathsf{easy}}$ computes the actual (simulated) prover messages in both the preamble phase and the proof phase, by using the trapdoors. We now give more details.

Simulator $\mathcal{S}$. Throughout the simulation, $\mathcal{S}$ maintains the following three data structures, each of which is initialized to $\perp$:

1. a list $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_N)$, where each π_i is either $\perp$ or is computed to be $h_i(\Pi)$. Here, h_i is the hash function corresponding to V_i and Π is the augmented machine code that is used for non-black-box simulation. We defer the description of Π to below.
2. a list $\mathsf{trap}^{\mathsf{heavy}} = (\mathsf{trap}_1^{\mathsf{heavy}}, \ldots, \mathsf{trap}_N^{\mathsf{heavy}})$, where each $\mathsf{trap}_i^{\mathsf{heavy}}$ corresponds to a tuple $\langle h_i, c, r, \Pi, y, s\rangle$ s.t. $R_{\mathsf{sim}}(\langle h_i, c, r\rangle, \langle \Pi, y, s\rangle) = 1$.
3. a list $\mathsf{trap}^{\mathsf{easy}} = (\mathsf{trap}_1^{\mathsf{easy}}, \ldots, \mathsf{trap}_N^{\mathsf{easy}})$, where each $\mathsf{trap}_i^{\mathsf{easy}}$ corresponds to a tuple $\langle c, r, \sigma, \beta, \gamma, \delta, c', t, \sigma'\rangle$ s.t.
   - $\mathbf{Verify}_{vk_i}(c \| r; \sigma) = 1$, and
   - $c' = \mathbf{Com}(\beta; t)$, and $\mathbf{Verify}_{vk_i}(c' \| \gamma; \sigma') = 1$, and

– $(h_i, \beta, \gamma, \delta)$ is an accepting transcript for a UARG pUA proving the following statement: $\exists \langle \Pi, y, s \rangle$ s.t. $R_{\mathsf{sim}}(\langle h_i, c, r \rangle, \langle \Pi, y, s \rangle) = 1$.

Augmented machine Π. The augmented machine code Π simply consists of the code of the adversarial verifier V^* and the code of the subroutine $\mathcal{S}_{\mathsf{easy}}$ (with a sufficiently long random tape hardwired, to compute the prover messages in each session) , i.e., $\Pi = (V^*, \mathcal{S}_{\mathsf{easy}})$. The input y to the machine Π consists of the lists $\boldsymbol{\pi}$ and $\mathsf{trap}^{\mathsf{easy}}$, i.e., $y = (\boldsymbol{\pi}, \mathsf{trap}^{\mathsf{easy}})$. Note that it follows from the description that $|y| \leq \ell(N) - n$.

We now describe the subroutines $\mathcal{S}_{\mathsf{easy}}$ and $\mathcal{S}_{\mathsf{heavy}}$, and then proceed to give a formal description of $\mathcal{S}$. For simplicity of exposition, in the discussion below, we assume that the verifier sends the first message in the WIPOK $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}} \rangle$.

Algorithm $\mathcal{S}_{\mathsf{easy}}(i, \mathsf{msg}_j^V, \boldsymbol{\pi}, \mathsf{trap}^{\mathsf{easy}}; z)$. The algorithm $\mathcal{S}_{\mathsf{easy}}$ prepares the (simulated) messages of the prover P in the protocol. More specifically, when executed with input $(i, \mathsf{msg}_j^V, \boldsymbol{\pi}, \mathsf{trap}^{\mathsf{easy}}; z)$, $\mathcal{S}_{\mathsf{easy}}$ does the following:

1. If msg_j^V is the first verifier message of the preamble phase from V_i in a session, then $\mathcal{S}_{\mathsf{easy}}$ parses $\boldsymbol{\pi}$ as $\pi_1, \ldots, \pi_N$. It computes and outputs $c = \mathbf{Com}(\pi_i; z)$.
2. If msg_j^V is the second verifier message of the preamble phase from V_i in a session, then $\mathcal{S}_{\mathsf{easy}}$ computes and outputs $c = \mathbf{Com}(\beta; z)$, where β is the corresponding (i.e., fourth) entry in $\mathsf{trap}_i^{\mathsf{easy}} \in \mathsf{trap}^{\mathsf{easy}}$.
3. If msg_j^V is a verifier message of the WIPOK from V_i in the proof phase of a session, then if $\mathsf{trap}_i^{\mathsf{easy}} = \bot$, then $\mathcal{S}_{\mathsf{easy}}$ aborts and outputs $\bot$, otherwise $\mathcal{S}_{\mathsf{easy}}$ simply runs the code of the honest P_{WI} to compute the response using randomness z and the trapdoor witness $\mathsf{trap}_i^{\mathsf{easy}}$.

Algorithm $\mathcal{S}_{\mathsf{heavy}}(i, j, \gamma, \mathsf{trap}^{\mathsf{heavy}})$. The algorithm $\mathcal{S}_{\mathsf{heavy}}$ simply prepares *one* UARG transcript for every verifier V_i, which in turn is used as a trapdoor by the algorithm $\mathcal{S}_{\mathsf{easy}}$. More concretely, when executed with input $(i, j, \gamma, \mathsf{trap}^{\mathsf{heavy}})$, $\mathcal{S}_{\mathsf{heavy}}$ does the following:

1. If $j = 1$, then $\mathcal{S}_{\mathsf{heavy}}$ parses the i^{th} entry $\mathsf{trap}_i^{\mathsf{heavy}}$ in $\mathsf{trap}^{\mathsf{heavy}}$ as (h_i, c, r, Π, y, s). It runs the honest prover algorithm P_{UA} and computes the first message β of a UARG for the statement: $\exists \langle \Pi, y, s \rangle$ s.t. $R_{\mathsf{sim}}(\langle h_i, c, r \rangle, \langle \Pi, y, s \rangle) = 1$. $\mathcal{S}_{\mathsf{heavy}}$ saves its internal state as state_i and outputs β.[10]
2. If $j = 2$, then $\mathcal{S}_{\mathsf{heavy}}$ uses state_i and γ to honestly compute the final prover message δ for the UARG with prefix (h_i, β, γ). It outputs δ.

Algorithm $\mathcal{S}$. Given the above subroutines, the simulator $\mathcal{S}$ works as follows. We assume that every time $\mathcal{S}$ updates the lists $\boldsymbol{\pi}$ and $\mathsf{trap}^{\mathsf{easy}}$, it also automatically updates the entry corresponding to y (i.e., the fifth entry) in each $\mathsf{trap}_i^{\mathsf{heavy}} \in \mathsf{trap}^{\mathsf{heavy}}$. For simplicity of exposition, we do not explicitly mention this below.

Preamble phase:

[10] For simplicity of exposition, we describe $\mathcal{S}_{\mathsf{heavy}}$ as a stateful algorithm.

1. On receiving the first message $\mathsf{msg}_1^V = (vk_i, h_i)$ from V^* on behalf of V_i in the preamble phase of a session, $\mathcal{S}$ first checks whether $\pi_i = \perp$ (where π_i is the i^{th} entry in the list $\boldsymbol{\pi}$); if the check succeeds, then $\mathcal{S}$ updates $\pi_i = h_i(\Pi)$. Next, $\mathcal{S}$ samples fresh randomness s from its random tape and runs $\mathcal{S}_{\mathsf{easy}}$ on input $(i, \mathsf{msg}_1^V, \boldsymbol{\pi}, \mathsf{trap}^{\mathsf{easy}}; s)$. $\mathcal{S}$ sends the output string c from $\mathcal{S}_{\mathsf{easy}}$ to V^*. Further, $\mathcal{S}$ adds $(h_i, c, \cdot, \Pi, y, s)$ to $\mathsf{trap}_i^{\mathsf{heavy}}$ and $(c, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot)$ to $\mathsf{trap}_i^{\mathsf{easy}}$.
2. On receiving the second message message $\mathsf{msg}_2^V = (r, \sigma)$ from V^* on behalf of V_i in the preamble phase of a session, $\mathcal{S}$ first verifies the validity of the signature σ w.r.t. vk_i. If the check fails, $\mathcal{S}$ considers this session aborted (as the prover would do) and ignores any additional message for this session. Otherwise, $\mathcal{S}$ checks whether the entries corresponding to r and σ (i.e., 2nd and 3rd entries) in $\mathsf{trap}_i^{\mathsf{easy}}$ are $\perp$. If the check succeeds, then:
   - $\mathcal{S}$ sets r as 3rd entry of $\mathsf{trap}_i^{\mathsf{heavy}}$ and r, σ as second and third entries of $\mathsf{trap}_i^{\mathsf{easy}}$.
   - Further, $\mathcal{S}$ runs $\mathcal{S}_{\mathsf{heavy}}$ on input[11] $(i, 1, \perp, \mathsf{trap}^{\mathsf{heavy}})$ to compute the message β of a UARG for the statement: $\exists \langle \Pi, y, s \rangle$ s.t. $R_{\mathsf{sim}}(\langle h_i, c, r \rangle, \langle \Pi, y, s \rangle) = 1$. Here h_i, c, r, Π, y, s are such that $\mathsf{trap}_i^{\mathsf{heavy}} = \langle h_i, c, r, \Pi, y, s \rangle$.
   - On receiving the output message β, $\mathcal{S}$ sets to β the fourth slot of $\mathsf{trap}_i^{\mathsf{easy}}$.

   Next, $\mathcal{S}$ samples fresh randomness t and runs $\mathcal{S}_{\mathsf{easy}}$ on input $(i, \mathsf{msg}_2^V, \boldsymbol{\pi}, \mathsf{trap}^{\mathsf{easy}}; t)$. On receiving the output string c' from $\mathcal{S}_{\mathsf{easy}}$, $\mathcal{S}$ forwards it to V^*. Further, $\mathcal{S}$ sets to (c', t) the 7th and 8th slot of $\mathsf{trap}_i^{\mathsf{easy}}$.
3. Finally, on receiving the last message $\mathsf{msg}_{\mathsf{fin}}^V = (\gamma, \sigma')$ from V^* on behalf of V_i in the preamble phase of a session, $\mathcal{S}$ first verifies the validity of the signature σ' w.r.t. vk_i. If the check fails, $\mathcal{S}$ considers this session aborted (as the prover would do) and ignores any additional message for this session. Otherwise, $\mathcal{S}$ checks whether the entries corresponding to γ and σ' in $\mathsf{trap}_i^{\mathsf{easy}}$ are $\perp$. If the check succeeds, then:
   - $\mathcal{S}$ sets to γ and σ' the 5th and 9th slot of $\mathsf{trap}_i^{\mathsf{easy}}$.
   - Further, $\mathcal{S}$ runs $\mathcal{S}_{\mathsf{heavy}}$ on input $(i, 2, \gamma, \mathsf{trap}^{\mathsf{heavy}})$ to compute the final prover message δ of the UARG with prefix (h_i, β, γ), where (β, γ) are the corresponding entries in $\mathsf{trap}_i^{\mathsf{easy}}$.
   - On receiving the output message δ, $\mathcal{S}$ sets to δ the 6th slot of $\mathsf{trap}_i^{\mathsf{easy}}$.

Proof phase: On receiving any message msg_j^V from V^* on behalf of V_i, $\mathcal{S}$ runs $\mathcal{S}_{\mathsf{easy}}$ on input $(i, \mathsf{msg}_j^V, \boldsymbol{\pi}, \mathsf{trap}^{\mathsf{easy}})$ and fresh randomness. $\mathcal{S}$ forwards the output message of $\mathcal{S}_{\mathsf{easy}}$ to V^*.

This completes the description of $\mathcal{S}$ and the subroutines $\mathcal{S}_{\mathsf{easy}}$, $\mathcal{S}_{\mathsf{heavy}}$. It follows immediately from the above description that $\mathcal{S}$ runs in polynomial time and outputs $\perp$ with probability negligibly close to an honest prover.

We now show through a series of hybrid experiments the simulator's output is computationally indistinguishable from the output of the adversary when interacting with honest provers. Our hybrid experiments will be H_i for $i = 0, \ldots, 3$. We write $H_i \approx H_j$ if no V^* can distinguish (except with negligible probability) between its interaction with H_i and H_j.

[11] For simplicity of exposition, we assume that randomness is hardwired in $\mathcal{S}^{\mathsf{heavy}}$ and do not mention it explicitly.

Hybrid H_0. Experiment H_0 corresponds to the honest prover. That is, in every session $j \in [m]$, H_0 sends c and c' as commitments to the all zeros string in the preamble phase. We provide H_0 with a witness that $x \in L$ which it uses to complete the both executions of the WIPOK $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}} \rangle$ played in each session.

Hybrid H_1. Experiment H_1 is similar to H_0, except the following. For every $i \in [N]$, for every session corresponding to verifier V_i, the commitment c in the preamble phase is prepared as a commitment to $\pi_i = h_i(\Pi)$, where h_i is the hash function in the identity of V_i and Π is the augmented machine code as described above.

The computational hiding property of **Com** ensures that $H_1 \approx H_0$.

Hybrid H_2. Experiment H_2 is similar to H_1, except the following. For every $i \in [N]$, for every session corresponding to verifier V_i, the commitment c' in the preamble phase is prepared as a commitment to the string β with randomness t, where β is the first prover message of a UARG computed by $\mathcal{S}_{\mathsf{heavy}}$, in the manner as described above.

The computational hiding property of **Com** ensures that $H_2 \approx H_1$.

Hybrid H_3. Experiment H_3 is similar to H_2, except the following. For every $i \in [N]$, for every session corresponding to verifier V_i, the WIPOK $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}} \rangle$ in the proof phase is executed using the trapdoor witness $\mathsf{trap}_i^{\mathsf{easy}}$, in the manner as described above. Note that this is our simulator $\mathcal{S}$.

The witness indistinguishability property of $\langle P_{\mathsf{WI}}, V_{\mathsf{WI}} \rangle$ ensures that $H_3 \approx H_2$.

3.2 Proof of Concurrent Soundness

Consider the interaction between a cheating P^* and an honest V. Suppose that P^* fools V into accepting a false proof in some session with non-negligible probability. We show how to reduce P^* to an adversary that breaks the security of one of the used ingredients. We will first consider P^* as a sequential malicious prover. We will discuss the issues deriving from a concurrent attack later.

First of all, notice that by the proof of knowledge property of the second WIPOK, we have that with non-negligible probability, an efficient adversary E can simply run as a honest verifier and extract a witness from that WIPOK of session l where the false statement is proved. Since the statement is false, the witness extracted will therefore be $(c, r, \sigma, \beta, \gamma, \delta, c', t, \sigma')$ such that $\mathbf{Verify}_{vk}(c \| r; \sigma) = 1$, $c' = \mathbf{Com}(\beta; t)$, $\mathbf{Verify}_{vk}(c' \| \gamma; \sigma') = 1$, and $(h, \beta, \gamma, \delta)$ is an accepting transcript for a UARG pUA proving the statement $\exists \langle \Pi, y, s \rangle$ s.t. $R_{\mathsf{sim}}(\langle h, c, r \rangle, \langle \Pi, y, s \rangle) = 1$, and h is the hash function corresponding to the verifier run by E in session l.

By the security of the signature scheme, it must be the case that signatures σ and σ' were generated and sent by E during the experiment (the reduction is standard and omitted).

Therefore we have that with non-negligible probability there is a session i where h and γ were played honestly by E, $(h, \beta, \gamma, \delta)$ is an accepting transcript

for the UARG for $R_{\mathsf{sim}}(\langle h, c, r\rangle, \langle \Pi, y, s\rangle) = 1$, and a commitment to β was given before γ was sent. Moreover, there is a session j where c and r were played as commitment and challenge. Remember that the session l is the one where the false statement is proved.

We can now complete the proof by relying almost verbatim on the same analysis of [1,2]. Indeed, by rewinding the prover and changing the challenge r in session j, with another random string, we would have an execution identically distributed with respect to the previous one. Therefore it will happen with non-negligible probability that the prover succeeds in session l, still relying on the information obtained in sessions i and j. The analysis of [1,2] by relying on the weak proof of knowledge property of the UA, shows that this event can be reduced to finding a collision that contradicts the collision resistance of h.

We finally discuss the case of a concurrent adversarial prover. Such an attack is played by a prover aiming at obtaining from concurrent sessions some information to be used in the target session where the false theorem must be proved. In previous work in the BPK model and in the BP model this was a major problem because the verifier used to give a proof of knowledge of its secret key, and the malleability of such a proof of knowledge could be exploited by the malicious prover. Our protocol however bypasses this attack because our verifier does not give a proof of knowledge of the secret key of the signature scheme, but only gives signatures of specific messages. Indeed the only point in which the above proof of soundness needs to be upgraded is the claim that by the security of the signature scheme, it must be the case that signatures σ and σ' where generated and sent by E during the experiment. In case of sequential attack, this is true because running the extractor of the WIPOK in session l does not impact on other sessions since they were played in full either before or after session l. Instead, in case of a concurrent attack, while rewinding the adversarial prover, new sessions could be started and more signatures could be needed. As a result, it could happen that in such new sessions the prover would ask precisely the same signatures that are then extracted from the target session. We can conclude that this does not impact on the proof for the following two reasons. First, in the proof of soundness it does not matter if those signatures appear in the transcript of the attack, or just in the transcript of a rewinded execution. Second, the reduction on the security of the signature scheme works for any polynomial number of signatures asked to the oracle, therefore still holds in case of a concurrent attack. Indeed, the work of E is performed in polynomial time even when rewinding a concurrent malicious prover, therefore playing in total (i.e., summing sessions in the view of the prover and sessions played during rewinds) a polynomial number of sessions, and therefore asking a polynomial number of signatures only to the signature oracle.

Further details on the proof of soundness. Given a transcript $(h, UA1, UA2, UA3)$ for the universal argument of [2], we stress that soundness still works when the prover sends the statement to the verifier only at the 4th round, opening a commitment played in the second round. The proof of concurrent soundness of our

protocol goes through a reduction to the soundness of the universal argument of [2] and goes as follows.

Let P^*_{ua} be the adversarial prover that we construct against the universal argument of [2], by making use of the adversary P^* of our protocol. Let V_{ua} be the honest verifier of the universal argument of [2]. P^*_{ua} gets "h" from V_{ua} and plays it in a random session s of the experiment (it could therefore be played in a rewinding thread) with P^*. Later on, since by contradiction P^* is successful, UA messages $(UA1, UA2, UA3)$ are extracted and with noticeable probability they correspond to session s. Therefore P^*_{ua} sends $UA1$ to V_{ua} and gets back $UA2'$. Then P^*_{ua} rewinds P* to the precise point where $UA2$ was played. Now P^*_{ua} plays $UA2'$. Again, later on, since by contradiction P^* is successful, P^*_{ua} will again extract from P^* and with noticeable probability (still because the number of sessions played in the experiment is polynomial), it will get an accepting transcript $(UA1, UA2', UA3^*)$ for the same statement (this is guaranteed by the security of the signature scheme and the binding of the commitment). Then P^*_{ua} can send $UA3^*$ to V_{ua} therefore proving a false statement.

4 Acknowledgments

Work supported in part by NSF grants 0830803, 09165174, 1065276, 1118126 and 1136174, US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, MIUR Project PRIN "GenData 2020" and Lockheed-Martin Corporation Research Award. This material is based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014−11−1−0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

1. Barak, B.: How to go beyond the black-box simulation barrier. In: FOCS. pp. 106–115 (2001)
2. Barak, B., Goldreich, O.: Universal arguments and their applications. In: IEEE Conference on Computational Complexity. pp. 194–203 (2002)
3. Barak, B., Lindell, Y.: Strict polynomial-time in simulation and extraction. In: STOC. pp. 484–493 (2002)
4. Blum, M.: How to prove a theorem so no one else can claim it. In: Proceedings of the International Congress of Mathematicians. pp. 1444–1451 (1987)
5. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable zero-knowledge (extended abstract). In: STOC. pp. 235–244 (2000)
6. Canetti, R., Kilian, J., Petrank, E., Rosen, A.: Black-box concurrent zero-knowledge requires $\widetilde{\Omega}(\log n)$ rounds. In: STOC. pp. 570–579 (2001)
7. Cho, C., Ostrovsky, R., Scafuro, A., Visconti, I.: Simultaneously resettable arguments of knowledge. In: TCC. Lecture Notes in Computer Science, Springer-Verlag (2012)

8. Chung, K.M., Lin, H., Pass, R.: Constant-round concurrent zero knowledge from p-certificates. In: FOCS. IEEE Computer Society (2013)
9. Deng, Y., Lin, D.: Instance-dependent verifiable random functions and their application to simultaneous resettability. In: EUROCRYPT. pp. 148–168 (2007)
10. Deng, Y., Lin, D.: Resettable zero knowledge with concurrent soundness in the bare public-key model under standard assumption. In: Inscrypt. pp. 123–137 (2007)
11. Di Crescenzo, G., Persiano, G., Visconti, I.: Constant-round resettable zero knowledge with concurrent soundness in the bare public-key model. In: Advances in Cryptology – Crypto '04. Lecture Notes in Computer Science, vol. 3152, pp. 237–253. Springer-Verlag (2004)
12. Di Crescenzo, G., Persiano, G., Visconti, I.: Improved setup assumptions for 3-round resettable zero knowledge. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 3329, pp. 530–544. Springer (2004)
13. Di Crescenzo, G., Visconti, I.: Concurrent zero knowledge in the public-key model. In: Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005. Lecture Notes in Computer Science, vol. 3580, pp. 816–827. Springer (2005)
14. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM J. Comput. 30(2), 391–437 (2000)
15. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: STOC. pp. 409–418 (1998)
16. Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In: FOCS. pp. 308–317 (1990)
17. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: STOC. pp. 291–304 (1985)
18. Goyal, V., Jain, A., Ostrovsky, R., Richelson, S., Visconti, I.: Concurrent zero knowledge in the bounded player model. In: TCC. pp. 60–79 (2013)
19. Gupta, D., Sahai, A.: On constant-round concurrent zero-knowledge from a knowledge assumption. IACR Cryptology ePrint Archive 2012, 572 (2012)
20. Kilian, J., Petrank, E.: Concurrent and resettable zero-knowledge in poly-loalgorithm rounds. In: STOC. pp. 560–569 (2001)
21. Kilian, J., Petrank, E., Rackoff, C.: Lower bounds for zero knowledge on the internet. In: FOCS. pp. 484–492 (1998)
22. Micali, S., Reyzin, L.: Soundness in the public-key model. In: Advances in Cryptology – Crypto '01. Lecture Notes in Computer Science, vol. 2139, pp. 542–565. Springer-Verlag (2001)
23. Micali, S., Reyzin, L.: Soundness in the public-key model. In: CRYPTO. pp. 542–565 (2001)
24. Naor, M.: Bit commitment using pseudorandomness. J. Cryptology 4(2), 151–158 (1991)
25. Ostrovsky, R., Persiano, G., Visconti, I.: Constant-round concurrent non-malleable zero knowledge in the bare public-key model. In: Proc. of ICALP '08. Lecture Notes in Computer Science, vol. 5126, pp. 548–559. Springer (2008)
26. Pass, R.: Bounded-concurrent secure multi-party computation with a dishonest majority. In: STOC. pp. 232–241 (2004)
27. Pass, R.: Limits of provable security from standard assumptions. In: STOC. pp. 109–118 (2011)
28. Pass, R., Rosen, A.: Concurrent non-malleable commitments. In: FOCS. pp. 563–572 (2005)
29. Pass, R., Rosen, A.: New and improved constructions of non-malleable cryptographic protocols. In: STOC. pp. 533–542 (2005)

30. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: FOCS. pp. 366–375 (2002)
31. Richardson, R., Kilian, J.: On the concurrent composition of zero-knowledge proofs. In: EUROCRYPT. pp. 415–431 (1999)
32. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: STOC. pp. 387–394 (1990)
33. Rosen, A.: A note on the round-complexity of concurrent zero-knowledge. In: CRYPTO. pp. 451–468 (2000)
34. Scafuro, A., Visconti, I.: On round-optimal zero knowledge in the bare public-key model. In: EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 7237, pp. 153–171. Springer (2012)
35. Visconti, I.: Efficient zero knowledge on the internet. In: Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Part II. Lecture Notes in Computer Science, vol. 4052, pp. 22–33. Springer (2006)
36. Yao, A.C.C., Yung, M., Zhao, Y.: Concurrent knowledge extraction in the public-key model. In: ICALP (1). pp. 702–714 (2010)
37. Yi, D., Feng, D., Goyal, V., Lin, D., Sahai, A., Yung, M.: Resettable cryptography in constant rounds - the case of zero knowledge. In: ASIACRYPT (2011)
38. Yung, M., Zhao, Y.: Generic and practical resettable zero-knowledge in the bare public-key model. In: EUROCRYPT. pp. 129–147 (2007)
39. Zhao, Y.: Concurrent/resettable zero-knowledge with concurrent soundness in the bare public-key model and i ts applications. Cryptology ePrint Archive, Report 2003/265 (2003), `http://eprint.iacr.org/`