Non-Black-Box Simulation in the Fully Concurrent Setting

Vipul Goyal
Microsoft Research, India
Email: vipul@microsoft.com

Abstract

We present a new zero-knowledge argument protocol by relying on the non-black-box simulation technique of Barak (FOCS'01). Similar to the protocol of Barak, ours is public-coin, is based on the existence of collision-resistant hash functions, and, is not based on "rewinding techniques" but rather uses non-black-box simulation. However in contrast to the protocol of Barak, our protocol is secure even if there are any unbounded (polynomial) number of concurrent sessions.

This gives us the first construction of public-coin concurrent zero-knowledge. Prior to our work, Pass, Tseng and Wikström (SIAM J. Comp. 2011) had shown that using black-box simulation, getting a construction for even public-coin parallel zero-knowledge is impossible.

A public-coin concurrent zero-knowledge protocol directly implies the existence of a concurrent resettably-sound zero-knowledge protocol. This is an improvement over the corresponding construction of Deng, Goyal and Sahai (FOCS'09) which was based on stronger assumptions. Furthermore, this also directly leads to an alternative (and arguable cleaner) construction of a simultaneous resettable zero-knowledge argument system.

An important feature of our protocol is the existence of a "straight-line" simulator. This gives a fundamentally different tool for constructing concurrently secure computation protocols (for functionalities even beyond zero-knowledge).

The round complexity of our protocol is n^{ϵ} (for any constant $\epsilon > 0$), and, the simulator runs in strict polynomial time. The main technique behind our construction is *purely combinatorial* in nature.

1 Introduction

Zero-knowledge proofs have played a central role in the design of secure cryptographic schemes. Introduced in [GMR89], all initial zero-knowledge (ZK) protocols shared a simple structure (see e.g., [GMR89, GMW91, Blu86]): the messages of the verifier in the protocol were simply random coin tosses. This simple structure is quite appealing in and by itself beyond any applications. However over a period of time, this public coin property found applications in several (even seemingly unrelated) contexts. An (incomplete) list of such examples include: the Fiat-Shamir paradigm [FS86], zero-knowledge protocols for **IP** [BOGG$^+$88], constructing resettably sound protocols [BGGL01, PTW11], constructing covert computation protocols [CGOS07, GJ10], efficient parallel repetition theorems [PV07], constructing witness indistinguishable universal arguments [BG02], etc. Much of the early work on zero-knowledge was for the "stand-alone" setting where there is a single protocol execution running in isolation.

In a breakthrough work in 2001, Barak [Bar01] introduced non-black-box simulation techniques in cryptography. This was done by giving a protocol which was public-coin, constant rounds and secure with a bounded number of concurrent protocol executions. A key feature of the construction was that it did not rely on the traditional paradigm of rewinding the adversary (and instead the simulator was "straightline"). Barak's technique has since then been utilized in a variety of different contexts and has been used to get results provable impossible using the traditional black-box simulation based on

rewinding techniques (see e.g., [BGGL01, Bar02, BL02, Lin03, Pas04, PR05, BS05, GS09, DGS09, GJ10, GM11]).

The fact the protocol is secure only in the bounded concurrent setting has been an important limitation of Barak's construction [Bar01]. There has been a long line of works on developing simulation strategies based on rewinding that would work in the fully concurrent setting [RK99, KP01, PRS02, DGS09]. However all these works rely on the paradigm of "extracting some trapdoor" from the adversarial verifier. This necessarily means that the protocol must not be public-coin.

The existence of a public-coin concurrent zero-knowledge protocol has remained an intriguing question till now. All known protocols for concurrent zero-knowledge are private coin. A result by Pass, Tseng and Wikström [PTW11] shows that this is no coincidence: they show that *only languages in* ***BPP*** *can have public-coin concurrent zero-knowledge proven secure using black-box simulation.* In fact, their negative result even applies to the weaker case of parallel repetition. Subsequently, Pass, Rosen and Tseng [PRT11] proposed a construction of public-coin *parallel* zero-knowledge by relying on non-black-box simulation technique from [Bar01].

Our Contributions. In this work, we propose a new concurrent zero-knowledge protocol. Similar to Barak [Bar01], our construction is public-coin, is based on the existence of collision-resistant hash functions, and, is not based on "rewinding techniques" but rather uses non-black-box simulation. However in contrast to [Bar01], our protocol is secure even if there are any unbounded (polynomial) number of concurrent sessions. We note that our construction is not a strict improvement over the one in Barak [Bar01] because of higher round complexity: the round complexity of our protocol is n^{ϵ} (for any constant $\epsilon > 0$), while, it was only a constant in [Bar01]. This resolves the question of public-coin concurrent zero-knowledge in the affirmative.

Given any public-coin concurrent zero-knowledge protocol, getting a simultaneous resettable zero-knowledge protocol [BGGL01, DGS09] is easy using known techniques. First one can compile a public-coin concurrent ZK protocol into one which is a concurrent resettable-sound ZK protocol. This can be done using the PTW transformation [PTW11]. By applying this transformation on our protocol, we get the first concurrent resettably-sound ZK based solely on the existence of a CRHF family. This is an improvement over the corresponding result of Deng, Goyal, and Sahai [DGS09]. Further, by relying on the "concurrent resettably sound ZK to simultaneous resettable ZK transformation" of DGS [DGS09], we get a new construction of a simultaneous resettable ZK. The resulting construction has the same round complexity and is based on the same assumptions as the DGS construction [DGS09]. However the construction is quite different from DGS, is not based on "rewinding techniques", and, is arguably simpler and more modular. The main technique behind our construction is *purely combinatorial* in nature.

Applications to Concurrent Secure Computation. We believe that the potential of the techniques introduced in this paper goes *far beyond the above results.* Perhaps, the most important feature of our protocol is that it has a "straightline simulator". Indeed, the notion of a straightline simulation is hard to formalize and we do not attempt at such a formalization in the current work. Rather, this feature can directly be utilized in designing concurrently secure protocol for more complex functionalities (other than zero-knowledge). The intuition regarding this is the following.

A well established approach to constructing secure computation protocols in the standalone setting is to use the GMW compiler: take a semi-honest secure computation protocol and "compile" it with zero-knowledge arguments. The natural starting point in the concurrent setting is to follow the same principles: somehow compile a semi-honest secure computation protocol with a concurrent zero-knowledge protocol (for security in more demanding settings, compilation with concurrent non-malleable zero-knowledge [BPS06] may be required). Does such an approach (or minor variants)

already give us protocols secure according to the standard ideal/real world definition in the plain model?

The fundamental problem with this approach is the following. Prior to our work, the known concurrent zero-knowledge simulators in the fully concurrent setting[1] work by rewinding the adversarial parties. In the concurrent setting, the adversary is allowed to control the scheduling of the messages of different sessions. Then the following scenario might occur:

- Between two messages of a session s_1, there might exist another entire session s_2.
- When the simulator rewinds the session s_1, it may rewind past the beginning of session s_2. Hence throughout the simulation, the session s_2 may be executed multiple times from the beginning.
- Every time the session s_2 is executed, the adversary may choose a different input (e.g., the adversary may choose his input in session s_2 based on the entire transcript of interaction so far).
- In such a case, the simulator would have to query the ideal functionality for session s_2 more than once. However note that for every session, simulator gets to query the ideal functionality only once!

Indeed, some such problem is rather inherent as indicated by various impossibility results [Lin08, BPS06]. Trying to solve this problem in various ways has inspired a number of different works. For example, Goyal, Jain and Ostrovsky [GJO10] proposed the multi-ideal-query model where, for every session in the real world, the simulator is allowed to query the functionality multiple times in the ideal world. In the plain model, recently Goyal [Goy12] was able to obtain *positive results for a large class of functionalities* in the concurrent setting by construct an "output predictor" to be utilized by the simulator during rewinds. In the resettable setting, Goyal and Sahai [GS09] (and more recently Goyal and Maji [GM11]) were able to obtain positive results by resetting the ideal functionality to obtain multiple outputs per session. Garg, Goyal, Jain and Sahai [GGJS12] were able to use this paradigm to construct protocols secure as per the notion of super-polynomial simulation by eliminating the extra outputs using super-polynomial time computations.

It is conceivable that a technique for straight-line simulation in the fully concurrent setting will give a *fundamentally new way of approaching the above problems.* Towards that end, we note that the techniques in this paper have already been utilized in the following further follow up works:

- **Resolving the bounded pseudoentropy conjecture:** Very recently, Goyal [Goy12] considered the so called "single input setting" where there is a party S which might interact with several other parties in any unbounded (polynomial) number of concurrent sessions. The party S holds a single input x which it uses in all the concurrent sessions. (An analogy is a server, holding a fixed database, and, interacting with various clients concurrently.) In this setting, Goyal obtained a positive result for many (or even most) functionalities in the plain model. Goyal left open the following (informally stated) conjecture which if resolved would give a more general and cleaner result:

  Bounded Pseudo-entropy Conjecture. Consider any ideal world experiment where the total amount of information learnt by the adversary in the ideal world (via calls to the trusted party) has bounded Yao pseudoentropy. In other words, given all the adversarial party inputs, there exists a string of a-priori bounded length using which it is possible to compute all the responses the trusted party provides to the adversary in the ideal world. Then there exists a real world protocol which securely realizes this ideal world.

[1] Note that in the bounded concurrent setting, general positive results for secure computation are already known [Lin03, PR03, Pas04].

Using our techniques, Goyal, Gupta and Sahai [GGS13] resolved the conjecture in the affirmative. Such a result subsumes several known positive results for concurrent secure computation in the plain model such as for zero-knowledge [RK99, KP01, PRS02], bounded concurrent computation [Lin03, PR03, Pas04], and, the positive results in the single input setting [Goy12]. The key technical tool used is a version of the protocol in this paper which additionally provides simulation extractability.

- **Improving the round complexity of protocols in the single input setting:** The round complexity of the construction of Goyal [Goy12] in the single input setting was a large polynomial depending not only upon the security parameter but also on the length of the input and the nature of the functionality. The construction of Goyal, Gupta and Sahai [GGS13] mentioned above only has n^{ϵ} round thus resulting in protocols in the single input setting with significantly better round complexity (depending only upon the security parameter).

- **Getting concurrent blind signatures:** The blind signature functionality is an interesting case in the paradigm of secure computation both from a theoretical as well as practical standpoint. The question of whether concurrent blind signatures (secure as per the ideal/real model simulation paradigm) exist is currently unresolved. Lindell [Lin03, Lin08] showed an impossibility result for concurrent blind signature based on *black-box simulation*. This result has also been used as a motivation to resort to weaker security notions or setup assumptions in various subsequent works (see e.g., [Fis06, Oka06, KZ06, HKKL07, GRS$^+$11]).

  Using our techniques, Goyal, Gupta and Sahai [GGS13] provide a new construction of concurrent blind signatures. Their construction is *secure in the plain model as the standard ideal/real world security notion* (i.e., no super-polynomial time simulation, no setup assumptions or random oracles, etc).

  This also gives the first "natural" example of a functionality which is impossible to realize using black-box simulation but can be securely realized using non-black-box simulation in the concurrent setting.[2] The only previous such example known [GM11] was for a reactive (and arguably rather contributed) functionality.

  Using techniques similar to that used in the impossibility of black-box concurrent blind signatures, Lindell also showed that concurrent oblivious transfer based on black-box simulation is impossible as well. A positive result or an unrestricted negative result for concurrent oblivious transfer was also an open problem until recently, Interestingly, this question was resolved in the *negative* recently by Agarwal et. al. [AGJ$^+$12], and, Garg et. al. [GKOV12].

Concurrent Independent Work. Independent of our work, Canetti, Lin and Paneth [CLP13] proposed a construction for public-coin concurrent zero-knowledge. Their construction is in the global setup model where there is a trusted party which publishes a fixed collision-resistant hash function (CRHF). Another interpretation of their model is that one can use a fixed function like SHA-3 (without having to assume a trusted party). In contrast to their work, our construction is in the plain model, and overall, use techniques quite different from theirs.

In the following, we describe a construction for public-coin concurrent zero-knowledge assuming the existence of a collision-resistant hash function family *and 1-to-1 one-way functions*. This represents our original result which was independent of the work of Canetti et al. [CLP13]. However, using a simple trick from [CLP13], our construction can be easily modified to yield such a result based only on the existence of a collision-resistant hash function family.

[2] Previous separations between the power of black-box and non-black-box simulation are known only if we place additional constraints on the design of the real world protocol (e.g., it should be public coin, or constant rounds, etc.)

Related Works. Barak's non-black-box simulation technique has been used in a variety of works such as on constructing resettably secure protocols [BGGL01, GS09, DGS09, GM11], non-malleable protocols [Bar02, Pas04, PR05], covert computation protocols [GJ10], etc. In a recent exciting work, Bitansky and Paneth [BP12] proposed an alternative non-black-box simulation strategy. Their technique was used to construct a resettably sound zero-knowledge protocol based on the existence of oblivious transfer.

There has also been a well-known line of works on studying (private-coin) concurrent zero-knowledge and its round complexity [RK99, KP01, PRS02, CKPR01].

1.1 Technical Challenges

Why Simple Approaches Fail. We highlight the main challenge involved in constructing a public-coin concurrent zero-knowledge protocol in this section. We assume familiarity with the construction of Barak [Bar01]. The description in this section is kept at an intuitive level and is not required to understand our construction in the subsequent technical sections. Barak's construction is a public-coin zero-knowledge argument system. However it is secure only with a bounded number of concurrent sessions.

The basic protocol of Barak works as follows (informally described). The verifier first chooses a function from a collision resistant hash function family and sends it to the prover. Next, the prover and the verifier execute a "slot": the prover sends a commitment z to the hash of a machine Π and the verifier replies back with a challenge string r. Then the prover and the verifier execute a universal argument (UA) [BG02] where the prover proves that either $x \in L$ or that the machine Π, upon execution, outputs the string r. Indeed, simulation in the standalone setting can be done by simply having the simulator commit to the code of the verifier as the machine Π. However in the concurrent setting, there may be messages of other sessions "in the slot " (i.e., between the prover message z and the verifier message r).

To simulate in the fully concurrent setting, following is a plausible approach: the simulator commits to "its own code" plus the code of the adversarial verifier as the machine Π. Now during simulation, when machine Π is run, it has the "entire state of the system". Hence, even if there are messages of other sessions before the string r appears in the current session, the machine Π can simply regenerate this entire transcript (by using the code of the simulator to compute prover messages and using code of adversary to compute verifier messages) and thus arrive at r. Thus, the simulation seems to work!

However, this "recomputation" of the transcript inside the slot now makes the simulator super-polynomial time. To see this, let the transcript of messages in the slot (between z and r) be denoted by τ. Let the time taken by the simulator to compute all the messages of τ be denoted by T. Now to execute the associated UA, the simulator needs execute the machine Π and have it output r (by recomputing the entire transcript τ). Hence, computing the UA messages itself requires further time T. This makes the UA messages quite "computationally heavy" capable of doubling the computation which has "happened so far". In more detail, consider the following scheduling. Inside the slot of a session, there occurs another full session. Furthermore, again inside the slot of this "inner" session, there is another full session and so on. This nesting goes upto depth $d = \omega(\log n)$. It can be seen that if the time taken to run the "innermost" session was T, the time taken to run the outermost session will be at least $2^d T$ (which is super-polynomial).

One could try the following construction: instead of a single slot, we have multiple slots in the protocol and then prove in the UA that in one the slots, the committed machine outputs the challenge of the verifier.[3] However even this protocol runs into similar problems and succumbs to a slightly more sophisticated scheduling. Suppose, the number of slots per session are n. Then at each "level" (up to depth d), we will have n sessions running in parallel (instead of just 1). At level i, each (of the

[3]This protocol is due to Sahai and to Ostrovsky and Visconti who proposed it to us independently.

n) sessions will have n sequential slots. However the slots of different sessions will be run in parallel, and hence, slots of different sessions share the same "inner transcripts" (resulting in only n inner transcripts). Now each of the n parallel sessions will have a heavy UA. If the computation of inner transcripts for each slot required time T, the total computation in the previous level was nT (because there are n such transcripts required to make each slot "heavy"). Now the total computation after this level would become $2nT$. This is because of the addition of n UA's each requiring at least T units of computation. Thus, the computation is still doubling at each level (at the next level, the n heavy UA's can be used to make the n slots heavy and so on).

Thus, the main challenge will be regenerate transcripts and output the adversary challenge strings while still keeping the simulation polynomial time.

Our Techniques. Observe that the basic problem in our setting is that the UA become "computationally heavy" during simulation. Such a UA can be put inside the slots of other sessions. This, in turn, would make the UA's of those sessions even heavier, and so on.

Our key idea to tackle this problem is to *hide where the heavy messages occur* in the sessions transcript. The basic structure of our protocol will be as follows. The protocol will have several slots (each consisting of a commitment to a machine and a verifier challenge string) *as well as several UA executions*. Any of the UA's in the session may be picked for simulation. If a UA is picked for simulation, to make our analysis go through, there is a further choice of which slot will be picked for transcript regeneration (e.g., the simulator might prefer the slots with a computationally lighter and shorter transcript). Otherwise, the simulator can just send random messages in place of having an accepting UA execution. Of course, the constraint is that in each of the sessions, there must be at least one UA picked for simulation.

Indeed, there are several issues in turning this idea into a protocol. If the adversary knows which UA was picked for simulation, the location of the heavy message is revealed and our starting premise is lost. To resolve this, we instead have "encrypted" executions of UAs (by relying on the public-coin property of the UA). Thus, a UA picked for simulation is indistinguishable from all other UAs. At the end of the protocol, we now have a (standard) zero-knowledge argument of knowledge proving that at least one of the encrypted UA transcripts was accepting.

Given such a protocol, the key technical challenge is to come up with a strategy to select which of the UA's to simulate and which to let go (as we move forward in the simulation). Towards that end, we come up with a combinatorial recursive *marking strategy* to mark each outgoing UA message either simulate or blank . Our strategy somewhat resembles the oblivious rewinding strategies employed in [KP01, PRS02]. Indeed, one can view simulating a slot as some equivalent of rewinding a slot in the setting of [KP01, PRS02]. However, the key difference in our setting is that, instead of the heavy computation happening *at the point* where the slot occurs, our heavy computation instead may occur at an arbitrary later point in the transcript (namely where the universal argument regenerating this slot occurs). Hence, despite the (arguably superficial) resemblance of the strategies, the ideas involved in the technical analysis of our marking strategy are quite unrelated to those involved in analyzing rewinding strategies in [KP01, PRS02].

Somewhat surprisingly, our overall construction and the analysis is quite short and simple (arguably simpler than the black-box concurrent rewinding strategies [RK99, KP01, PRS02]). Our technical analysis is modular and is divided into a purely combinatorial part (which relates to the development of a marking strategy), and, a "cryptographic part" which includes the hybrid argument. Along the way, we also introduce new techniques to execute hybrid arguments under non-black-box simulation settings which necessitate committing to (part of) the code of a simulator as well. This is done by relying on *committing an Oracle machine paradigm* introduced by Deng, Goyal, and, Sahai [DGS09]. We believe this to be of independent interest.

2 Preliminaries

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-to-one one-way function 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.

As mentioned earlier, we can remove the 1-to-1 one-way function assumption in our work by relying on a simple trick from [CLP13]. In particular, [CLP13] uses a notion of forward-secure pseudo-random functions (based only on one-way functions) which can also be employed in our construction (as opposed to non-interactive commitment schemes with a unique decommitment).

Three-round public-coin universal-arguments. Universal arguments [BG02] are used in order to provide efficient proofs to statements of the form $y = (M, x, t)$, where y is considered to be a true statement if M is a non-deterministic machine that accepts x within t steps. We shall make use of public-coin universal-arguments from [BG02] which is only 3-rounds assuming the prover and the verifier have agreed upon a function from a CRHF family before the protocol starts. In addition, we shall make use of the fact that the universal argument system from [BG02] is a weak proof of knowledge.

Security notions. For definitions of established cryptographic notions like concurrent zero-knowledge, universal arguments, zero-knowledge argument of knowledge, and, various notions of resettability, we refer the reader to previous works [RK99, Bar01, BG02, DGS09].

3 Our Protocol.

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 Section 2). 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. Unless other specified, all commitments on the protocol are executed using this commitment scheme.

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 verifier V chooses a random collision resistant hash function h from a function family $\mathcal{H}$ and sends it to P.
2. For $i \in [n^6]$, the protocol proceeds as follows:[4]
   - The prover P computes $z_i = Com(h(0))$ and sends it to V.
   - The verifiers V selects a string $r_i \xleftarrow{\$} \{0,1\}^{n^2}$ and sends it to P. The above two message (consisting of the prover commitment and the verifier string) are refereed to as a "slot".
   - The prover P and the verifier V will now start a three-round public coin universal argument (of knowledge) [BG02] where P proves to V that *there exists j, s.t., τ_j $(= (h, z_j, r_j))$ is in a language Λ* (defined in figure 1). The three messages of this UA protocol are called as the *first UA message*, *verifier UA challenge*, and, the *last UA message*.

[4]Note that the round complexity of our protocol can be made n^ϵ using standard techniques involving "scaling down" the security parameter.

Observe that the UA does not just refer the slot immediately preceding it but rather has a choice of using *any of the slots that have completed* in the protocol so far.

- The prover computes the first UA message and sends a *commitment* to this message to the verifier. The honest prover will simply commit to a random string of appropriate size.
- The verifier now sends the UA challenge message.
- The prover computes the last UA message and again sends only a *commitment* to this message to the verifier. The honest prover will simply commit to a random string of appropriate size.

3. Finally, the prover has to prove either: (1) $x \in L$, or, (2) There exists i such that the i-th UA execution was "convincing". In more detail, the prover and the verifier run a witness-indistinguishable argument of knowledge protocol where the prover proves the following. There exists an $i \in [n^6]$ such that there exists an opening to the prover first and last UA messages such that the verifier would have accepted the transcript of the UA execution. Observe that a witness to the above statement would be the opening of the commitments to the UA first and last messages. Hence, the size of the witness is fixed and depends only upon the communication complexity of the 3-round UA system being used.

   The honest prover simply uses the witness to the condition $x \in L$ to complete the witness-indistinguishable argument of knowledge protocol.

4 Security Analysis: The Combinatorial Part

The description of our simulator can be separated into two parts: the first module selects which of the universal arguments to "simulate", while, the second module actually computes all the outgoing messages (based on the selections of the first module). The first (and key) module of the simulator is purely combinatorial. In this section, we set up the combinatorial problem and provide details on how our simulator solves it.

Observe that there are n^6 slots each session of our protocol will have. The simulator will need to "simulate each session" which will amount to choosing at least one of the UA's in session for simulation. We first define a "measure of heavyness" for each slot and each UA in the entire execution.

- A UA which is not selected for simulation is called a 0-heavy UA. Both prover messages of this UA are considered to be 0-heavy UA messages.
- A slot which *does not* contain a message of any UA selected for simulation is called a 0-heavy slot[5].
- A slot which contains a message (either first or last) of a i-heavy UA, but, does not contain a message of a j-heavy UA for $j > i$ is called a i-heavy slot.
- A UA which is selected for simulation is called a i-heavy UA, if, informally, the "lightest" slot of the session so far is $(i-1)$-heavy. More precisely, i is such that the following holds. Look at all the slots in this session which concluded prior to the start of the execution of this UA. i is the maximum integer such that all these slots are j-heavy for $j + 1 \geq i$. Both prover messages of this UA are considered to be i-heavy UA messages. In particular this means that a UA which is selected for simulation and has a 0-heavy slot preceding it in the session becomes a 1-heavy UA.

[5]By this, we mean that the messages of the other sessions between the prover and the verifier messages of the slot does not contain any simulated UA message.

The following notation will be convenient. A (i+)-heavy slot (resp. UA) is a j-heavy slot (resp. UA) for $j \geq i$. Similarly, a (i-)-heavy slot (resp. UA) is a j-heavy slot (resp. UA) for $j \leq i$

Assume w.l.g. that the transcript of the interaction with the adversary will have exactly n^c messages (counting both prover and verifier messages in all concurrent sessions) where c is a constant. Our goal will be construct a simulation strategy with the following constraints.

- The simulator, when required to send a UA message (either first or last), marks this message either simulate or blank . If the UA message is marked blank , it remains 0-heavy and becomes i-heavy otherwise for $i \geq 1$ as discussed above.
- In all concurrent sessions, we would require that no UA become c-heavy. This in particular would also mean that no UA becomes (c+1)-heavy and so on. This is refereed to as the *runtime property.*
- In each session, we would require at least one pair of UA message to be marked simulate . In other words, at least for one UA in each session, both the first and the last message should be marked simulate . This is referred to as the *coverage property.*

Rest of this section would be devoted to constructing such a marking strategy.

The marking strategy. Consider an complete n-ary tree of depth $c+1$. Each node of the tree will have exactly n children, and hence, the total number of leaf nodes will be n^c. We call the leaf nodes to be at level 0 of the tree while the root node is said to be at level c.

Consider the full transcript of interaction between the simulator and the adversary consisting of n^c messages. This transcript be said to be a block of level c and is denoted by B. The block B corresponds to the root node of the n-ary tree. Now we call the recursive function $Mark(c-1, B)$ given below (used to create blocks at level $i-1$ given a block at level i).

$Mark(i, B)$

{

- Divide the given block B into n equal blocks of level i denoted by $B_1, \ldots, B_n$. The first $1/n$ fraction of the messages in B form the block B_1, and so on.
- The blocks $B_1, \ldots, B_n$ now correspond to the n children of the tree node corresponding to the block B. If $i = 0$, return, else continue to the next step.
- Randomly mark exactly one of the resulting n blocks as blank . Each UA prover message of that block is marked blank $-i$. Jumping ahead, this means that this UA message will not be allowed to become $(i+)$-heavy in the simulation.
- Invoke $Mark(i-1, B_1), \ldots, Mark(i-1, B_n)$.

}

Now, our marking strategy works as following. Consider a UA prover message (either first or last). Say if the UA message was to be selected for simulation, it would become i-heavy for some i. We mark this message simulate if and only if it has not been marked blank $-j$ for any $j \leq i$ *and* $i \neq c$. Otherwise we mark this UA message as blank .

Observe that since we know the exact number of messages in advance, roughly speaking, this marking can be done even before the interaction with the adversary starts at all. That is, whether or not a UA message is marked simulate or blank depends only upon where it lies in the transcript of interaction (and not on the content / scheduling of the messages so far).

Now we prove that the above marking strategy satisfies the properties we outlined earlier. The *runtime property* follows by construction. We do not allow any UA message to become c-heavy (and hence no message is $(c+)$-heavy as well). The *coverage property* is proved in the lemma below.

Lemma 1 *At least for one UA in each session, both the first and the last prover messages are marked* simulate .

Proof. The proof of this lemma relies upon the following key claim.

Claim 1 *Suppose that a session has at least n^4 slots contained in a block of level i. Then at least one of those slots will be marked* blank *-j for $j \leq i-1$.*

Proof. First we claim that there exists $j \leq i-1$ such that there are at least n^2 distinct blocks at level j each containing a slot of this session fully.

To see this, observe the following.

- Consider the subtree whose root is the block in question at level i. This subtree has exactly 1 node at level i, n nodes at level $i-1$, and, n^i nodes at level 0. Denote by $SUM(j)$ the number of slots (out of these n^4) which are fully contained in a block corresponding to a node at level j of this subtree. Hence, we are given that $SUM(i) \geq n^4$.
- At level 0, since each block just consists of a single message, no slot is contained in any level 0 block. Hence, $SUM(0) = 0$.
- This means that there must exists a j s.t. $SUM(j) - SUM(j-1) \geq n^3$. This is because i is a constant (there are only a constant number of levels).
- If a block fully contains m slots of a session, then there would exists at least $m-n$ slots which will be fully contained in at least one of its n child blocks (as defined by the n-ary tree). This is because the only slots that are contained in the parent block but not in any of the child blocks (called lost slots) are such that the start of the slot (the prover message) and the end of the slot (the verifier message) lie in different child blocks. Since there are only n child blocks, there can only be at most $n-1$ such slots.
- This means that there are at least n^2 distinct blocks at level j each containing a full slot of this session. This is because since each block when divided into n blocks can lead to a loss of a maximum of n slots, the total number of slots lost otherwise would be less than n^3.

Now we argue that w.h.p., at least one of these n^2 blocks will be marked blank *-j*. This is because the probability of the first of these blocks being marked blank *-j* is exactly $1/n$. Furthermore, conditioned on all previous blocks not marked blank *-j*, the probability of the next block being marked blank *-j* is also *at least* $1/n$. Hence the claim follows.

Our next claim is the following.

Claim 2 *Consider any i-heavy UA of a session. The probability of either of the two prover messages of this UA being marked* blank *is at most $2c/n$.*

Proof. The proof of this claim is simple. Each message in the transcript lies in at most c blocks (excluding the block corresponding to the root node). The probability of a block at level i being marked blank *-i* individually is exactly $1/n$. Hence, by union bound, the probability of a UA message being marked blank *-i* for any i is at most c/n. Hence, the overall probability of a UA message being marked blank is at most c/n.

Again by union bound, the probability of *any* of the two prover messages of the UA being marked blank is at most $2c/n$.

Completing the proof. For any given session, now fix an index i such that there are at least n^5 consecutive UAs which, if simulated, would become i-heavy. First we argue that there must exist such an i. There exists at least n^5 UAs which, if simulated, would become i-heavy for some i because there are a total of n^6 UAs and only constant number of levels (of heavyness). Also, all these UAs must be consecutive because, by our marking strategy, a UA can never be "heavier" than an earlier UA of that session.

Now by claim 1, no block of level i can fully contain more than $n^4 + 1$ of these consecutive UAs (otherwise this would mean that this block contains at least n^4 consecutive $(i-1)$+-heavy slots of this session which is a contradiction to the statement of the claim). This means that there are at least $n/2$ blocks of level i each having at least one UA prover message out of these n^5 consecutive UAs. This also implies that $i \neq c$ since there is only one block at level c.

Now from these $n/2$ blocks at level i, we pick $n/4$ UA pairs (picking both first and last prover message in each pair) such that none of these blocks contains a message from *more than one* of these UA pairs. This can be done by simply picking either one UA pair from each block (if a pair exists in that block) or otherwise picking just a single UA prover message such that the matching UA prover message (which lies in a different block) is also picked.

It follow that we have picked at least $n/4$ pairs of UA messages (with each pair consisting of the first and last UA prover message) out of these n^5 UAs. By claim 2, the probability of each UA pair being picked for simulation (i.e., probability of both messages marked simulate) is at least $1 - 2c/n \geq 1/2$. Also, the probability of picking each UA pair for simulation is independent. This is because any block can have message from at most 1 of these $n/4$ pairs of UA. Hence, the probability of no UA pair picked for simulation is negligible in n. Thus, the claim follows.

5 Completing the Security Analysis

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 $y_1 \in \{0,1\}^{\leq n}$ 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 str and expecting $(value, r)$ with $str = Com(value; r)$ in return, such that, the tuple $(str, value, r)$ 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 $(h, z, r) \notin \Lambda$.

Figure 1: The DGS Universal Argument Language [DGS09]

Description of the Simulator. Our simulator will be divided into two parts: one referred to as *the Oracle*, and, the other referred to as the next message machine (or NMM in short). Recall that the entire transcript will have exactly n^c message. We assume that the random tape of the NMM is divided into n^c strings of equal lengths (a pseudo-random generator may be applied on a string to expand it if required). The i-th string will be used as randomness to generate the i-th message of the transcript (or verify the i-th message in case it is an incoming message). Denote these string by $s_1, \ldots, s_{n^c}$.

As the first step, *the Oracle generates these strings* and produces n^c commitments to these strings denoted as $C_1, \ldots, C_{n^c}$. The strings as well as the randomness used to generate these commitments is

retained by the Oracle while the commitments themselves are given to the NMM (which then stores these commitments). Denote by $C = (C_1, \ldots, C_{n^c})$. The combinatorial module (described in the previous section) to mark a UA prover message either blank or simulate is a part of the NMM and has its own random tape s. The string s is generated and stored by the NMM.

Throughout the simulation, the NMM will not have the randomness required to compute the next message. This critical missing information is stored by the Oracle. To compute the next outgoing message, the NMM may access the Oracle through a clearly defined interface: make a call to the oracle by producing a query of the form str and expecting $(value, r)$ with $str = Com(value; r)$ in return. Hence, the NMM slowly learns the required random tapes and the associated decommitment information as we proceed in the simulation.

The intuition behind such a separation of the simulator into two parts will only be clear once we go through the hybrid experiments. However we give a rough justification in the following. To complete the simulation, the simulator will be required to commit to a machine which can predict the challenge r of the adversary. One option is to just commit to the entire state of the simulator (and the verifier) as the machine. However, if all the randomness for all the messages is committed, the changes to various messages required as part of the hybrid argument seem difficult to perform. To solve this issue, we commit to a machine which does not have these random tapes for the future messages. This machine can still regenerate the required transcript with the help of an external Oracle. The next message machine NMM represents the part of the simulator that will be committed as the machine while the Oracle represents the external uncommitted information that will be available to this machine while trying to regenerate the transcript.

Now we describe our simulator in detail. The NMM handles each outgoing message as described below. In each of these steps, the randomness required by the NMM is taken from the string s_i (assuming that the message will be the i-th message exchanged between the adversary and the NMM). Indeed, the NMM does not have the required string s_i but only a commitment c_i to it. To get s_i, it just queries the Oracle with c_i.

1. **The slot begin message z:** The NMM defines a machine Π which simply consists of all the information the NMM has as part of its own state plus the current state of the (adversarial) verifier machine. However, the machine Π does not have the information that the Oracle stores (which is not yet learnt by the NMM). The NMM then computes $z = Com(h(\Pi))$ and sends z to the adversary.

2. **A UA message marked blank :** The NMM runs the combinatorial module to see if the UA message is marked blank . If so, the NMM simply generates a random string of appropriate size and sends a commitment to that string to the adversary.

3. **A UA message marked simulate :** If the combinatorial module marks a UA prover message as simulate , the NMM proceeds as follows. If it is a UA first message, the NMM constructs a witness for the statement $(h, z, r) \in \Lambda$ and uses that to compute the UA first message (details of how the witness is constructed is given later on). The NMM now commits to the resulting UA first message. The resulting UA first message and the opening to the commitment are now stored by the NMM.

   Note that the NMM does not store any other intermediate information resulting from this step. In particular, it discards the witness and other information required to compute the UA first message (such as the PCP). This is done to ensure that the state of the NMM (and hence the machine Π it commits to when a slot begins) is bounded by a fixed polynomial. This is crucial to keep the simulation polynomial time.

   If the message is a last UA prover message such that the corresponding UA first message was also marked simulate , then the NMM *again reconstructs* the witness for the appropriate statement

$(h, z, r) \in \Lambda$, reconstructs the UA first message, and then, computes the UA last message depending upon the UA challenge given by the adversary. Now the NMM computes a commitment to this UA last message and sends it to the adversary. The computed message and the opening to the commitment are again stored by the NMM. Note that, as earlier, the NMM discards any intermediate information computed in this step.

If the message is a UA last message where the UA first message was marked blank , the NMM simply generates a random string of appropriate size and sends a commitment to that string to the adversary.

4. **Witness-indistinguishable argument of knowledge:** The NMM uses a witness to the statement: "there exists i such that the i-th UA execution was convincing". By lemma 1, before we reach the WI stage of a session, there must exist an i such that both the UA first and last message were marked simulate except with negligible probability (if this doesn't happen, the simulation is aborted). This means that for that i, the NMM must have computed the UA first and last message such that the UA execution becomes convincing. A witness to this statement is simply the openings to the commitments to the UA first and last messages.

   Observe that, by the properties of the UA system, the computation required in this step is similar for each session and, in particular, does not "grow" as the simulation proceeds (since it depends only on the computational complexity of the UA verifier and not on that of the UA prover).

Constructing the witness. Now we show that for any universal argument which is selected for simulation, there exists a witness for the statement $(h, z, r) \in \Lambda$. This can be seen by construction of the NMM and the Oracle. We observe the following:

- Consider the point till which the NMM had generated the message z in the transcript. From this point onwards, the NMM, given only queries to the Oracle, was able to continue generating the transcript and arrive at the challenge r. Thus, the machine Π can perform the same execution since it starts from the same internal state *assuming* it gets access to the same Oracle query responses as well.

- Now consider the point when the first UA message is due. Let the index of the message r in the transcript be j and let $y_1 = j$. Furthermore, let y_2 contain all the Oracle queries the NMM made while going from the message z to r (both inclusive). Now observe that any Oracle queries Π makes (before it outputs r) can be answered using y_2. This is because its execution would be identical to that of the NMM.

- Now the witness simply consists of Π, y_1, y_2, and, the opening of the commitment z. The NMM already has all of these as part of its internal state by the time it reaches r (and in particular, by the time the first UA message is due). Thus, the NMM has computed the witness required to complete the universal argument.

The Hybrid Argument.

Experiment H_0: This corresponds to the real world execution between the prover and the verifier. The simulator has the required witness in every session and plays honestly according to the protocol.

Experiment H_1: In this experiment, the simulator is divided into the Oracle and the next message machine (NMM). The Oracle generates the strings $s_1, \ldots, s_{n^c}$, computes the commitments $C = (C_1, \ldots, C_{n^c})$ and gives C to the NMM. In addition, the NMM generates s required to run the combinatorial module. To compute the next message, the NMM queries the Oracle to get the appropriate random string and computes the required message using this as the random tape (still continuing to behave honestly). Furthermore, the NMM runs the combinatorial module (using the random tape s) and internally marks each UA message either simulate or blank .

The transcript of interaction in this experiment is identical to that in the previous one.

Intuition behind rest of the proof. Next, we shall consider n^c sets of intermediate hybrid experiments $H_2[i,1], H_2[i,2], H_2[i,3]$ for $i \in [n^c]$. Some basic intuition behind what happens in these hybrids is as follows. The basic problem we deal with in these hybrid experiments occurs because of non-black-box simulation. We are using some cryptographic primitives in our protocol such as a commitment scheme (providing hiding property), and, a WIAOK system. We would go through the hybrids by changing what we commit to (and then rely on the hiding property of the commitment scheme), and, by changing the witness we use in the WIAOK system (and then rely on the witness indistinguishability).

Now, say that the NMM starts using the opening of the commitment C_i (to the string s_i) while either preparing the machine it commits to (as part of the slot begin message), or, while completing a simulated universal argument. Then, if a primitive is executed using randomness s_i, we note that its security properties (which we rely on), may no longer hold. For example, if the commitment is prepared using randomness s_i, we can no longer rely on the hiding property of this commitment.

To solve this problem, we shall move forward in the transcript and make changes in a sequential manner. When we are in the i-th set of hybrids, we will maintain the following invariants: (a) the NMM will only use the opening of the commitments C_j for $j < i$, and, (b) from this hybrid onwards, we will only change messages indexed i and higher. We must have already made all the required changes in messages before the i-th message.

Experiment $H_2[i,1]$: This experiment is identical to the previous one except in how the message with index i in the transcript is handled by the simulator. In this hybrid, if the i-th message is a slot being message z, the NMM works as follows. Instead of committing to $h(0)$, the NMM now constructs a machine Π (which consists of the internal state of NMM and that of the adversarial verifier), computes the commitment $z = Com(h(\Pi); s_i)$ using the appropriate randomness s_i and sends it to the adversary.

Proof of indistinguishability. We now claim that the view of the adversary in this experiment is indistinguishable from the previous one by the (computational) hiding of the commitment scheme Com. The only reason it is non-trivial is that the string C contains a commitment c_i to the randomness s_i required to produce the commitment z. However, we argue that the opening to the commitment c_i is not used by the NMM in any non-trivial way in the interaction with the adversary.

Consider a modified experiment $H_2'[i,1]$ in which the Oracle, upon receiving a query str, only returns $value$ s.t. $str = Com(value; r)$ if $str = c_j$ for $j \geq i$. Even given access to such a modified Oracle, the NMM could produce a transcript which is identically distributed. This is because the openings given by the Oracle to the NMM are used by NMM only to either construct a witness for a simulated UA, or, to commit to a machine Π when a slot begins. However, there is no simulated UA or a commitment to a machine after the message i of the transcript in this experiment. Thus, for a message $j \geq i$, the NMM is not using the opening of the commitment c_j in any way in the experiment.

Hence, in this experiment, the string s_i is semantically secure and hence any commitment using s_i as randomness also is semantically secure. Thus, the claim follows.

Experiment $H_2[i, 2]$: This experiment is identical to the previous one again except in how the message with index i in the transcript is handled by the simulator. In this hybrid, if the i-th message is a UA message which is marked simulate , the way the NMM handles this machine is identical to the final NMM (described earlier). In particular, the NMM starts using the witness guaranteed by the construction of Π for computing this UA messages (as opposed to just committing to random strings). The intermediate information resulting from this step is discarded (as described earlier).

Proof of indistinguishability. The indistinguishability from the previous experiment relies on the computational hiding of the commitment scheme Com and is very similar to the previous indistinguishability proof. The difference between this hybrid and the previous one is only in the computation of the i-th message (in case it is a UA prover message marked as simulate). However, similar to the previous indistinguishability argument, we argue that the randomness used to compute the commitment to this UA prover message is not used by the NMM in anyway in the interaction with the adversary.

Towards that end, consider a modified experiment $H_2'[i, 2]$ in which the Oracle, when given a query of the form str by the NMM, only returns $value$ s.t. $str = Com(value; r)$ if $str = c_j$ for $j \geq i$. Even given access to such a modified Oracle, the NMM would produce a transcript which is identically distributed. This is because, as argued earlier, the openings given by the Oracle to the NMM are used by NMM only to either construct a witness for a simulated UA, or, to commit to a machine when a slot begins. However, there is no simulated UA or a commitment to a machine after the message i of the transcript in this experiment.

Hence, in this experiment, the random string used to compute the commitment to this UA prover message is semantically secure. Hence the commitment using this random string is also semantically secure. Hence the claim follows.

Experiment $H_2[i, 3]$: This experiment in identical to the previous one except that now in all the sessions "solved" so far, the NMM starts using the alternative witness (the opening to the simulated UA messages) to complete the WIAOK. In more detail, consider any session in which up to (and including) the i-th message of the transcript, there exists a UA both of whose prover messages are marked simulate . For such session, the NMM now starts using the openings of these UA messages as the witness to complete the WIAOK execution.

Proof of indistinguishability. This is very similar to the previous proof of indistinguishability. The only changes made in this hybrid occur after the message with index i in the interaction transcript. However, as argued earlier, the random tape used to compute such a message is still semantically secure. Hence, by the witness indistinguishability of the WIAOK protocol, the claim follows.

Experiment H_3: This is the final hybrid in which the Oracle and the next message machine are identical to as in our simulator. We note that this hybrid is identical to the hybrid experiment $H_2[2^c, 3]$. □

Relaxing the assumptions. We can relax the requirement of a 1-to-1 one-way function by using a trick from [CLP13]. Canetti et al [CLP13] use a form of forward-secure PRFs based only on one-way functions to avoid "committing the entire randomness" of the simulation as part of the machine Π. In particular, the machine Π can be given as input a seed s_i which would enable it to recover random tapes for all messages up to (and including) message i. However, the random tapes of all messages indexed j with $j > i$ would remain computationally hidden. This idea can be employed in a straightforward manner in our construction as well. The resulting construction would be based on just the existence of CRHFs. The details regarding this will be provided in the full version.

5.1 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 the adversary is polynomial.

Theorem 1 *The simulator described in the previous section runs in time which is polynomial in the security parameter n.*

PROOF. To start with, we note that for Sim, computing the next message takes fixed polynomial time for all messages other than ones which are UA prover messages marked simulate (for which a more complex argument will be required). In particular, this is true for the slot begin message: the machine committed to simply consists of the internal state of the verifier and the NMM. The verifier is guaranteed to be polynomial time. In addition, the NMM stores (s, C) (the starting state), the transcript so far, the list of queries made to the Oracle so far, the computed UA first and last messages, and, some intermediate information for completing the WIAOK messages (since WIAOK is an interactive argument system). Given the number of messages n^c, this can be upper bounded (and in particular does not "grow recursively").

Let n^{c_p} denote the bound on the time taken by the simulator to compute the next such prover message (i.e, a prover message not marked simulate). Let the running time of the verifier's next message function be bounded by n^{c_v}. Say n^{c_1} represent the maximum of n^{c_p} and n^{c_v}. Then, n^{c_1} represents the bound on the time taken by the simulator to compute the next message (either prover or verifier) unless the next message is a UA prover message marked simulate .

Say that the complexity of verifying the theorem statement $(h, z, r) \in \Lambda$ is n^k. Then by the properties of universal arguments [BG02], it can be shown that the time taken to compute a simulated universal argument message (proving this statement) is bounded by n^{k+c_2} where c_2 is a constant. Recall that the entire transcript has n^c messages. Now we prove the following claim.

Claim 3 *Consider a UA prover message which is marked simulate . We claim that if the message become i-heavy upon simulation, the time taken to compute such a message would be bounded by $n^{i\cdot c+c_1+i\cdot c_2}$.*

This would conclude the analysis for the simulator running time since i is only a constant in our simulation. We prove this statement by induction. The statement is clearly true for $i = 0$ since as argued before, n^{c_1} represents the bound on the time taken by the simulator to compute a UA prover message which is not simulated (and hence is 0-heavy).

Now consider messages of a UA prover message marked simulate which would become i-heavy upon simulation. To execute this UA message, 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 messages between z and r. This time is bounded by $n^c \cdot n^{(i-1)\cdot c+c_1+(i-1)\cdot c_2}$. This is because there can only be $O(n^c)$ messages between z and r. Furthermore, all of these messages are either $(i-1)$-heavy UA prover messages or messages whose computation takes time lower than that required for a $(i-1)$-heavy UA prover message. Thus, the taken time to compute each of these messages may be bounded by $n^{(i-1)\cdot c+c_1+(i-1)\cdot c_2}$. It follows that Π can regenerate the transcript between z and r in time $n^c \cdot n^{(i-1)\cdot c+c_1+(i-1)\cdot c_2}$. Then by the properties of universal arguments [BG02], it can be shown that the time taken to compute the required simulated universal argument message is bounded by $n_2^c \cdot n^c \cdot n^{(i-1)\cdot c+c_1+(i-1)\cdot c_2}$ which is $n^{i\cdot c+c_1+i\cdot c_2}$. This proves the above claim.

Hence, we conclude that Sim takes polynomial time to compute each next outgoing message to the adversary. Since the total number of outgoing messages in the transcript is also a polynomial, we conclude that the overall running time of our simulator is a (strict) polynomial. □

5.2 Soundness and Argument of Knowledge

Theorem 2 *The interactive argument system described in Section 3 is computationally sound. In fact, the interactive argument system is also an argument of knowledge.*

PROOF. The proof of this claim extends ideas from a similar proof of soundness in [DGS09]. 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 ($\in_R \{0,1\}^{n^2}$) given the input $y_1 \in \{0,1\}^{\leq n}$ 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 *str* expecting $(value, r)$ in return with $str = Com(value; r)$, by the unique opening property of the commitment scheme Com (See Section 2), 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 $y_1 \in \{0,1\}^n$, there are a total of 2^n possible outputs of Π. Denote by S the set of these possible outputs. Now the probability of a string $r \in_R \{0,1\}^{n^2}$ being an element of this set is bounded by 2^{n^2-n} 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.

Now assume $x \notin L$ and a malicious prover P^* is still able to successfully complete the protocol such that an honest verifier V outputs accept with a noticeable probability. We can assume P^* is deterministic without loss of generality. Then, there must exists an index i such the following happens with a noticeable probability ϵ: (a) P^* is able to successfully complete the protocol such that an honest verifier V outputs accept, *and*, (b) upon running extractor of the WIAOK system, we recover the transcript of the i-th universal argument. We denote this event by complete-i. Thus, probability of complete-i is ϵ.

Now we define prefix1 and prefix2 of the protocol as follows. Call prefix1 to be the point where the prover has given a commitment z (i.e., has given the slot begin message) in the i-th slot and is now expecting the verifier message r. Call prefix2 to be point where the prover has given the first UA prover message of the i-th UA execution and is now expecting the verifier challenge.

Now it has to be the case that for atleast a fraction $\frac{\epsilon}{2}$ of prefix1 executions, the probability (over rest of the verifier random coins) that the event complete-i happens is atleast $\frac{\epsilon}{2}$. We call such prefix1 executions as good1. Furthermore, conditioned on prefix1 being good1, at least for $\frac{\epsilon}{4}$ fraction of prefix2 executions, the probability that the event complete-i happens is atleast $\frac{\epsilon}{4}$. We call such prefix2 executions as good2.

Now the verifier executes in two phases as follows. In phase 1:

- The verifier honestly executes the full protocol with the prover and runs the extractor associated with the WIAOK system.
- If the event complete-i happens, the verifier rewinds the adversarial prover to prefix2 (i.e., the point where the prover expects the UA challenge in the i-th UA execution). The verifier aborts otherwise.
- Now the verifier starts an external verifier of the 3-round universal argument system. By this point, the verifier has already extracted the opening of the commitment to the UA first prover message received in the i-th UA execution with the prover (as part of the witness extracted from the WIAOK system). The verifier forwards this UA first message to the external UA verifier.
- The verifier receives the UA challenge from the external UA verifier and simply passes it on to the prover.

- Now the verifier honestly completes the rest of the protocol with the prover and runs the extractor associated with the WIAOK system. If the event complete-i happens again, the verifier has extracted the the last UA prover message (and aborts otherwise). It then forwards this message to the external UA verifier.
- The probability of this entire phase 1 getting complete without aborting is at least Pr[prefix1 is good1] $\cdot$ Pr[prefix2 is good2 — prefix1 is good1] $\cdot$ Pr[Event complete-i happens in two independent trials starting from prefix2]. This comes out to be $\frac{\epsilon}{2}\cdot\frac{\epsilon}{4}\cdot\frac{\epsilon}{4}\cdot\frac{\epsilon}{4}$ which is $\frac{\epsilon^4}{128}$.
- Now employing the weak knowledge extractor associated with the universal argument system [BG02], we can extract the witness from the UA with probability at least $p(\frac{\epsilon^4}{128})$ 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). This is because $\frac{\epsilon^4}{128}$ is the probability with which the external UA verifier gets a complete accepting UA execution. Thus, we have extracted a machine Π such the prover commitment z is a commitment to $h(\Pi)$.

Now in phase 2 of the experiment, the verifier rewinds the execution to prefix1 and completes the protocol execution honestly with fresh random coins (and in particular, giving a fresh challenge r to complete the slot). Rest of this phase is identical to phase 1. If the event complete-i happens, rewind the prover to prefix2, give the UA first message to the external UA verifier, get the UA challenge and pass it on to the verifier. Now the verifier honestly completes the rest of the protocol with the prover and if the event complete-i happens again, the verifier forwards the UA last message to the external UA verifier. The knowledge extractor associated with the UA system is again employed to recover the UA witness.

Note that conditioned on prefix1 being good1, phase 2 is completed without aborting with probability at least $\frac{\epsilon}{4}\cdot\frac{\epsilon}{4}\cdot\frac{\epsilon}{4}$ which is $\frac{\epsilon^3}{64}$. Hence, with probability at least $p(\frac{\epsilon^3}{64})$, we have extracted another machine Π' such the prover commitment z is a commitment to $h(\Pi')$

Now observe that the fresh challenge r (in phase 2) was chosen by the verifier after it received the program Π in phase 1. As argued in the previous paragraph, if S_Π is the set of all possible outputs of Π, the probability that $r \in S_\Pi$ is negligible. If phase 2 succeeds, the verifier has obtained another program Π'. As argued before, except with negligible probability, Π could not have predicted r and hence $\Pi \neq \Pi'$. However since $h(\Pi) = h(\Pi')$, we have obtained a collision in the hash function. The probability of this event COLL can be computed as follows:

$$\begin{aligned}\Pr[\text{COLL}] &\geq \Pr[\text{Phase 1 succeeds in extracting } \Pi]\cdot\Pr[\text{Phase 2 succeeds in extracting } \Pi'] \\ &\quad -\Pr[\Pi = \Pi'] \\ &\geq \frac{\epsilon^4}{128}\cdot\frac{\epsilon^3}{64} - \text{negl}(n)\end{aligned}$$

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

This means that for every i, the probability of the event complete-i occurring is negligible. This must mean that the extractor of the WIAOK instead outputs a witness for $x \in L$. This proves soundness as well as argument of knowledge property.

□

Getting Simultaneous Resettable Zero-Knowledge Arguments. Once we get a public-coin concurrent zero-knowledge argument of knowledge system, getting a simultaneous resettable zero-knowledge argument is possible by a direct application of known results. As the first step, We use the following theorem from [PTW11].

Theorem 3 *[PTW11] Let $\Sigma = \langle P, V \rangle$ be a public-coin argument of knowledge for an **NP** language L with negligible soundness error. Let the round complexity of Σ be $k = poly(n)$. Define $\tilde{\Sigma}^m = \langle P^m, \tilde{V}^m \rangle$ to be m parallel repetitions of Σ with the following modification: $\tilde{V}^m$ will sample a pseudo-random function f at the beginning of the protocol, and construct each verifier message by applying f to the prover messages received so far. Then, whenever $m \geq k^2 \log n$, $\tilde{\Sigma}^m$ is a resettably-sound argument of knowledge.*

Applying the above transformation to our protocol, we get a construction for a concurrent resettably-sound zero-knowledge protocol.

Theorem 4 *There exists a concurrent resettably-sound zero-knowledge argument with round complexity n^ϵ assuming only the existence of a collision-resistant hash function family.*

The above theorem is an improvement over the corresponding result by Deng, Goyal and Sahai [DGS09] where such a construction was presented assuming the existence of a CRHF family, 1-to-1 one-way functions and two-round zaps.

Next, we rely on the following lemma from [DGS09].

Lemma 2 *[DGS09] Suppose there exists a k-round concurrent resettably-sound zero-knowledge protocol. Then, there exists a $O(k)$-round simultaneous resettable zero-knowledge argument system assuming the existence of a collision-resistant hash function family and two-round zaps.*

This gives us a new construction of a simultaneous resettable ZK argument system. The resulting construction has the same round complexity as the DGS construction [DGS09]. However the construction is quite different from DGS, is not based on "rewinding techniques" and is arguably simpler and more modular.

More details will be provided in a future version.

Acknowledgements. Thanks to the nice staff at Cafe Coffee Day, Lavelle Road where much of this research was carried out. Thanks to Divya Gupta for useful comments on many parts of the paper.

References

[AGJ+12] Shweta Agrawal, Vipul Goyal, Abhishek Jain, Manoj Prabhakaran, and Amit Sahai. New impossibility results on concurrently secure computation and a non-interactive completeness theorem for secure computation. In *CRYPTO*, 2012.

[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. 2002.

[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.

[BL02] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. Cryptology ePrint Archive, Report 2002/043, 2002. Extended abstract appeared in STOC' 02.

[Blu86] Manuel Blum. How to prove a theorem so no one else can claim it. *Proceedings of the International Congress of Mathematicians*, 1986.

[BOGG+88] Michael Ben-Or, Oded Goldreich, Shafi Goldwasser, Johan Håstad, Joe Kilian, Silvio Micali, and Phillip Rogaway. Everything provable is provable in zero-knowledge. In Shafi Goldwasser, editor, *CRYPTO*, volume 403 of *Lecture Notes in Computer Science*, pages 37–56. Springer, 1988.

[BP12] Nir Bitansky and Omer Paneth. From the impossibility of obfuscation to a new non-black-box simulation technique. In *FOCS*, 2012.

[BPS06] Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *FOCS*, pages 345–354, 2006.

[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 Computer Society, 2005.

[CGOS07] Nishanth Chandran, Vipul Goyal, Rafail Ostrovsky, and Amit Sahai. Covert multi-party computation. In *FOCS*, pages 238–248. IEEE Computer Society, 2007.

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires omega∼(log n) rounds. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *STOC*, pages 570–579. ACM, 2001.

[CLP13] Ran Canetti, Huijia Lin, and Omer Paneth. Public coin concurrent zero-knowledge with a global hash. In *TCC*, 2013.

[DGS09] Yi Deng, Vipul Goyal, and Amit Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *FOCS*, pages 251–260. IEEE Computer Society, 2009.

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 60–77. Springer, 2006.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[GGJS12] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Bringing people of different beliefs together to do uc. In *Eurocrypt*, 2012.

[GGS13] Vipul Goyal, Divya Gupta, and Amit Sahai. Bounded pseudoentropy conjecture: Resolution and applications. In *In preparation*, 2013.

[GJ10] Vipul Goyal and Abhishek Jain. On the round complexity of covert computation. In Leonard J. Schulman, editor, *STOC*, pages 191–200. ACM, 2010.

[GJO10] Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Password-authenticated session-key generation on the internet in the plain model. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 277–294. Springer, 2010.

[GKOV12] Sanjam Garg, Abishek Kumarasubramanian, Rafail Ostrovsky, and Ivan Visconti. Impossibility results for static input secure computation. In *CRYPTO*, 2012.

[GM11] Vipul Goyal and Hemanta K. Maji. Stateless cryptographic protocols. In Rafail Ostrovsky, editor, *FOCS*, pages 678–687. IEEE, 2011.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38(3):691–729, 1991.

[Goy12] Vipul Goyal. Positive results for concurrently secure computation in the plain model. In *FOCS*, 2012.

[GRS+11] Sanjam Garg, Vanishree Rao, Amit Sahai, Dominique Schröder, and Dominique Unruh. Round optimal blind signatures. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 630–648. Springer, 2011.

[GS09] Vipul Goyal and Amit Sahai. Resettably secure computation. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 54–71. Springer, 2009.

[HKKL07] Carmit Hazay, Jonathan Katz, Chiu-Yuen Koo, and Yehuda Lindell. Concurrently-secure blind signatures without random oracles or setup assumptions. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 323–341. Springer, 2007.

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

[KZ06] Aggelos Kiayias and Hong-Sheng Zhou. Concurrent blind signatures without random oracles. In Roberto De Prisco and Moti Yung, editors, *SCN*, volume 4116 of *Lecture Notes in Computer Science*, pages 49–62. Springer, 2006.

[Lin03] Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *STOC*, pages 683–692. ACM, 2003.

[Lin08] Yehuda Lindell. Lower bounds and impossibility results for concurrent self composition. *J. Cryptology*, 21(2):200–249, 2008.

[Oka06] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 80–99. Springer, 2006.

[Pas04] Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. pages 232–241, 2004.

[PR03] Rafael Pass and Alon Rosen. Bounded-concurrent secure two-party computation in a constant number of rounds. 2003.

[PR05] Rafael Pass and Alon Rosen. Concurrent non-malleable commitments. In *FOCS*, pages 563–572, 2005.

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

[PRT11] Rafael Pass, Alon Rosen, and Wei-Lung Dustin Tseng. Public-coin parallal zero-knowledge for np. *Journal of Cryptology*, 2011.

[PTW11] Rafael Pass, Wei-Lung Dustin Tseng, and Douglas Wikström. On the composition of public-coin zero-knowledge protocols. *SIAM J. Comput.*, 40(6):1529–1553, 2011.

[PV07] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. An efficient parallel repetition theorem for arthur-merlin games. In David S. Johnson and Uriel Feige, editors, *STOC*, pages 420–429. ACM, 2007.

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