Non-black-box Simulation in the Fully Concurrent Setting, Revisited

Susumu Kiyoshima

NTT Secure Platform Laboratories, Japan.
susumu@kiyoshima.info

March 14, 2019

Abstract

We give a new proof of the existence of $O(n^\epsilon)$-round public-coin concurrent zero-knowledge arguments for $\mathcal{NP}$, where $\epsilon > 0$ is an arbitrary constant. The security is proven in the plain model under the assumption that collision-resistant hash functions exist. (The existence of such concurrent zero-knowledge arguments was previously proven by Goyal (STOC'13) in the plain model under the same assumption.) In the proof, we use a new variant of the non-black-box simulation technique of Barak (FOCS'01). An important property of our simulation technique is that the simulator runs in a "straight-line" manner in the fully concurrent setting. Compared with the simulation technique of Goyal, which also has such a property, the analysis of our simulation technique is (arguably) simpler.

This article is based on an earlier article: An Alternative Approach to Non-black-box Simulation in Fully Concurrent Setting, in Proceedings of TCC 2015, ©IACR 2015, DOI: 10.1007/978-3-662-46494-6_13.

1 Introduction

Zero-knowledge (ZK) *proofs* and *arguments* [GMR89] are interactive proof/argument systems with which the prover can convince the verifier of the correctness of a mathematical statement while providing *zero additional knowledge*. In the definition of ZK protocols,[1] this "zero additional knowledge" property is formalized thorough the *simulation paradigm*: An interactive proof/argument is said to be zero-knowledge if for any adversarial verifier there exists a *simulator* that can output a simulated view of the adversary. ZK protocols have been used as building blocks in many cryptographic protocols, and techniques developed for them have been used in many fields of cryptography.

Traditionally, the security of ZK protocols was proven via *black-box simulation*. That is, the zero-knowledge property was proven by showing a simulator that uses the adversary only as an oracle. Since black-box simulators use the adversaries as oracles, the only advantage they have is the ability to rewind the adversaries. Still, black-box simulation is quite powerful, and it can be used to obtain ZK protocols with a variety of additional properties, security, and efficiency.

However, black-box simulation has inherent limitations. For example, let us consider *public-coin* ZK *protocols* and *concurrent* ZK *protocols*, where the former is the ZK protocols such that the verifier sends only the outcome of its coin-tossing during the protocols, and the latter is the ZK protocols such that their zero-knowledge property holds even when they are concurrently executed many times. It is known that both of them can be constructed by using black-box simulation techniques [GMW91, RK99, KP01, PRS02]. However, it is also known that neither of them can be constructed by black-box simulation techniques if we additionally require round efficiency. Specifically, it was shown that constant-round public-coin ZK protocols and $o(\log n/\log\log n)$-round concurrent ZK protocols cannot be proven secure via black-box simulation [GK96, CKPR02]. Furthermore, it was also shown that no public-coin concurrent ZK protocol can be proven secure via black-box simulation irrespective to its round complexity [PTW09].

A natural question to ask is whether the ZK property can be proven by using *non-black-box simulation techniques*. In particular, whether the above impossibility results can be overcome by using non-black-box simulation techniques is a highly motivated question. Non-black-box simulation techniques are, however, significantly hard to develop. Specifically, non-black-box simulation seems to inherently involve "reverse engineering" of the adversaries, and such reverse engineering seems very difficult.

Barak [Bar01] made a breakthrough about non-black-box stimulation by proposing the first non-black-box simulation technique under a standard assumption, and showing that a black-box impossibility result can be overcome by using it. Specifically, Barak used his non-black-box simulation technique to obtain a constant-round public-coin ZK protocol under the assumption that a family of collision-resistant hash functions exists. (Recall that, as noted above, constant-round public-coin ZK protocols cannot be proven secure via black-box simulation.) The simulation technique of Barak is completely different from previous ones. Specifically, in his simulation technique, the simulator runs in a "straight-line" manner (that is, it does not "rewind" the adversary) and simulates the adversary's view by using the code of the adversary.[2]

Non-black-box simulation in the concurrent setting. Since Barak's non-black-box simulation technique allows us to overcome a black-box impossibility result, it is natural to ask whether we can overcome other black-box impossibility results as well by using Barak's technique. In particular, since Barak's simulation technique works in a straight-line manner and therefore completely removes the issue of *recursive rewinding* [DNS04] that arises in the setting of concurrent ZK, it is natural to

[1]We use "ZK protocols" to denote ZK proofs and arguments.

[2]The notion of "straight-line simulation" is, unfortunately, hard to formalize. I this paper, we use it only informally.

expect that Barak's simulation technique can be used to overcome the black-box impossibility results of $o(\log n/\log\log n)$-round concurrent ZK protocols and public-coin concurrent ZK protocols.

However, it turned out that Barak's non-black-box simulation technique is hard to use in the concurrent setting. In fact, although Barak's technique can be extended so that it can handle *bounded-concurrent execution* [Bar01] (i.e., concurrent execution where the protocol is concurrently executed a bounded number of times) and *parallel execution* [PRT13], it had been open for years to extend it so that it can handle fully concurrent execution. An important step toward obtaining non-black-box simulation in the fully concurrent setting was made by Deng, Goyal, and Sahai [DGS09], who used Barak's technique in the fully concurrent setting by combining it with a black-box simulation technique (specifically, with the recursive rewinding technique of Richardson and Kilian [RK99]). Another important step was made by Bitansky and Paneth [BP12, BP13, BP15], who developed a new non-black-box simulation technique (which is *not* based on that of Barak) that can handle fully concurrent execution when being combined with a black-box simulation technique (again, the recursive rewinding technique of [RK99]). The simulation techniques of these works are powerful enough to allow us to overcome another black-box impossibility result (the impossibility of *simultaneously resettable* ZK *protocols* [CGGM00, BGGL01]). However, they are not strict improvement over Barak's non-black-box simulation technique since they do not have some of the useful properties that Barak's technique do have, such as the public-coin property and the straight-line simulation property. As a result, they do not immediately allow us to overcome the black-box impossibility results of $o(\log n/\log\log n)$-round concurrent ZK protocols and public-coin concurrent ZK protocols.

Recently, several works showed that with a trusted setup or non-standard assumptions, Barak's simulation technique can be extended so that it can handle fully concurrent execution (without losing its public-coin property and straight-line simulation property). Furthermore, they showed that with their versions of Barak's technique, it is possible to overcome the black-box impossibility results of $o(\log n/\log\log n)$-round concurrent ZK protocols and public-coin concurrent ZK protocols. For example, Canetti et al. [CLP13a] constructed a public-coin concurrent ZK protocol in the *global hash function (GHF) model*, where a single hash function is used in all concurrent sessions. Also, Chung et al. [CLP13b] constructed a constant-round concurrent ZK protocol by assuming the existence of *$\mathcal{P}$-certificates* (i.e., "succinct" non-interactive proofs/arguments for $\mathcal{P}$), Pandey et al. [PPS15] constructed a constant-round concurrent ZK protocols by assuming the existence of *differing-input indistinguishability obfuscators*, and Chung et al. [CLP15] constructed a constant-round concurrent ZK protocols by assuming the existence of indistinguishability obfuscators.

Very recently, Goyal [Goy13] showed that Barak's non-black-box simulation technique can be extended so that it can handle fully concurrent execution *even in the plain model under standard assumptions*. Goyal then used his version of Barak's technique to obtain the first public-coin concurrent ZK protocol in the plain model under a standard assumption (the existence of a family of collision-resistant hash functions), where its round complexity is $O(n^{\epsilon})$ for an arbitrary constant $\epsilon > 0$. Like the original simulation technique of Barak (and many of its variants), the simulation technique of Goyal has a straight-line simulator; hence, Goyal's simulator performs *straight-line concurrent simulation*. Because of this straight-line concurrent simulation property, the simulation technique of Goyal has huge potential. In fact, it was shown subsequently that Goyal's technique can be used to obtain new results on concurrently secure multi-party computation and concurrent blind signatures [GGS15].

In summary, we currently have several positive results on non-black-box simulation in the concurrent setting, and in particular we have a one that has a straight-line concurrent simulator in the plain model under a standard assumption [Goy13]. However, the state-of-the-art is still not satisfactory and there are still many open problems to be addressed. For example, the simulation technique of Goyal [Goy13] requires the protocol to have $O(n^{\epsilon})$ rounds, so the problem of constructing $o(\log n/\log\log n)$-round concurrent ZK protocols in the plain model under standard assumptions is still open. Thus, studying more on non-black-box simulation and developing new non-black-box simulation techniques in the concurrent setting is still an important research direction.

1.1 Our Result

In this paper, we propose a new variant of Barak's non-black-box simulation technique, and use it to give a new proof of the following theorem, which was originally proven by Goyal [Goy13].

Theorem. *Assume the existence of a family of collision resistant hash functions. Then, for any constant $\epsilon > 0$, there exists an $O(n^\epsilon)$-round public-coin concurrent zero-knowledge argument of knowledge.*

Like the simulation technique of Goyal, our simulation technique can handle fully concurrent execution in the plain model under a standard assumption, and it has a simulator that runs in a straight-line manner in the fully concurrent setting. We emphasize that our simulation technique requires the same hardness assumption and the same round complexity as that of Goyal; hence, it does not immediately lead to improvement over the result of Goyal. Nevertheless, we believe that our simulation technique is interesting because it is different from that of Goyal and its analysis is (in our opinion) simpler than the analysis of Goyal's technique. (A comparison between our simulation technique and that of Goyal is given in Section 2.3.) We hope that our technique leads to further study on non-black-box simulation in the concurrent setting.

Brief overview of our technique. Our public-coin concurrent ZK protocol is based on the public-coin concurrent ZK protocol of Canetti, Lin, and Paneth (CLP) [CLP13a], which is secure in the global hash function model. Below, we give a brief overview of our technique under the assumptions that the readers are familiar with Barak's non-black-box simulation technique and CLP's techniques. In Section 2, we give a more detailed overview of our technique, including the explanation of the techniques of Barak and CLP.

The protocol of CLP is similar to the ZK protocol of Barak except that it has multiple "slots" (i.e., pairs of a prover's commitment and a receiver's random-string message). A key observation by CLP is that given multiple slots, one can avoid the blow-up of the simulator's running time, which is the main obstacle to use Barak's simulation technique in the concurrent setting. More precisely, CLP's observation is that given multiple slots, the simulator can use any of these slots when generating the PCP proof in the universal argument (UA) of Barak's protocol, and therefore it can avoid the blow-up of its running time by using a good "proving strategy" that determines which slots to use in the generation of the PCP proofs in concurrent sessions. The proving strategy that CLP use is similar in spirit to the *oblivious rewinding strategy* [KP01, PRS02] of black-box concurrent ZK protocols. In particular, in the proving strategy of CLP, the simulator recursively divides the simulated transcript between honest provers and the cheating verifier into "blocks," and generates the PCP proofs only at the end of the blocks.

A problem that CLP encountered is that the simulator has only one opportunity to give the UA proof in each session, and thus it need to remember all previously generated PCP proofs if the adversary delays the execution of the UA proofs in all sessions. Because of this problem, the length of the PCP proofs can be rapidly blowing up in the concurrent setting, and the size of the simulator cannot be bounded by a polynomial. In [CLP13a], CLP solved this problem in the global hash function model by cleverly using the global hash function in UA.

To solve this problem in the plain model, we modify the protocol of CLP so that the simulator has multiple opportunities to give the UA proof in each session. We then show that by using a good proving strategy that also determines which opportunity the simulator takes to give the UA proof in each session, the simulator can avoid the blow-up of its size as well as that of its running time. Our proving strategy guarantees that a PCP proof generated at the end of a block is used only in its "parent block"; because of this guarantee, the simulator need to remember each PCP proof only for a limited time and therefore the length of the PCP proofs does not blow up. This proving strategy is the core of our simulation technique and the main deference between the simulation technique of ours and that of Goyal [Goy13]. (The simulator of Goyal also has multiple opportunities to give the UA proof in each session, but it determines which opportunity to take by using a proving strategy that is different from ours.) Interestingly, the strategy that we use is deterministic (whereas the strategy that Goyal uses is probabilistic). Because of the use of this deterministic strategy, we can analyze our simulator in a relatively simple way. In particular, when showing that every session is always successfully simulated, we need to use only a simple counting argument.

2 Overview of Our Technique

As mentioned in Section 1.1, our protocol is based on the protocol of Canetti et al. [CLP13a], which in turn is based on Barak's non-black-box zero-knowledge protocol [Bar01]. Below, we first recall the protocols of [Bar01, CLP13a] and then give an overview of our protocol.

2.1 Known Techniques

Barak's protocol. Roughly speaking, Barak's non-black-box zero-knowledge argument $\mathsf{BarakZK}$ proceeds as follows.

Protocol $\mathsf{BarakZK}$

1. The verifier V chooses a hash function $h \in \mathcal{H}_n$ and sends it to the prover P.
2. P sends $c \leftarrow \mathsf{Com}(0^n)$ to V, where Com is a statistically binding commitment scheme. (For simplicity, in this overview we assume that Com is non-interactive.) Then, V sends a random string $r \in \{0,1\}^n$ to P. In the following, the pair (c, r) is called a *slot*.
3. P proves the following statement by using a witness-indistinguishable argument.
   - $x \in L$, or
   - $(h, c, r) \in \Lambda$, where Λ is a language such that $(h, c, r) \in \Lambda$ holds if and only if there exists a machine Π such that (i) c is a commitment to $h(\Pi)$ and (ii) Π outputs r within $n^{\log\log n}$ steps.[3]

Since polynomial-time algorithms cannot check whether or not Π outputs r within $n^{\log\log n}$ steps, the statement proven in Step 3 is not in $\mathcal{NP}$. Thus, P proves this statement by using a witness-indistinguishable *universal argument* (WIUA), which is, roughly speaking, a witness-indistinguishable argument for $\mathcal{NEXP}$ such that a language whose witness relation is checkable in T steps can be proven in $\mathsf{poly}(T)$ steps.

Roughly speaking, the security of $\mathsf{BarakZK}$ is proven as follows. The soundness is proven by observing that even when a cheating prover P^* commits to $h(\Pi)$ for a machine Π, we have $\Pi(c) \neq r$ with overwhelming probability because r is chosen after P^* commits to $h(\Pi)$. The zero-knowledge property is proven by using a simulator that commits to a machine Π that emulates the cheating verifier V^*; since $\Pi(c) = V^*(c) = r$ from the definition, the simulator can give a valid proof in WIUA. Such a simulator runs in polynomial time since, from the property of WIUA, the running time of the simulator during WIUA is bounded by $\mathsf{poly}(t)$, where t is the running time of $\Pi(c)$.

[3]Here, $n^{\log\log n}$ can be replaced with any super-polynomial function. We use $n^{\log\log n}$ for concreteness.

Barak's protocol in the concurrent setting. A limitation of BarakZK is that we do not know how to prove its zero-knowledge property in the concurrent setting. Recall that in the concurrent setting, a protocol is executed many times concurrently; hence, to prove the zero-knowledge property of a protocol in the concurrent setting, we need to design a simulator against cheating verifiers that participate in many *sessions* of the protocol with honest provers. The above simulator for BarakZK, however, does not work against such verifiers since $V^*(c) = r$ does not hold when a verifier V^* participates in other sessions during a slot of a session (i.e., $V^*(c) \neq r$ holds when V^* first receives c in a session, next receives messages in other sessions, and then sends r in the first session).

A potential approach to proving the concurrent zero-knowledge property of BarakZK is to use a simulator $\mathcal{S}$ that commits to a machine that emulates $\mathcal{S}$ itself. The key observation behind this approach is the following: When V^* participates in other sessions during a slot of a session, all the messages that V^* receives in the other sessions are actually generated by $\mathcal{S}$; hence, if the committed machine Π can emulate $\mathcal{S}$, it can emulate all the messages between c and r for V^*, so $\Pi(c)$ can output r even when V^* receives many messages during a slot.[4]

This approach however causes a problem in the simulator's running time. For example, let us consider the following "nested concurrent sessions" schedule (see Figure 1).

- The $(i + 1)$-th session is executed in such a way that it is completely contained in the slot of the i-th session. That is, V^* starts the $(i + 1)$-th session after receiving c in the i-th session, and sends r in the i-th session after completing the $(i + 1)$-th session.

Let m be the number of sessions, and let t be the running time of $\mathcal{S}$ during the simulation of the m-th session. Then, to simulate the $(m - 1)$-th session, $\mathcal{S}$ need to run at least $2t$ steps—t steps for simulating the slot (which contains the m-th session) and t steps for simulating WIUA. Then, to simulate the $(m - 2)$-th session, $\mathcal{S}$ need to run at least $4t$ steps—$2t$ steps for simulating the slot and $2t$ steps for simulating WIUA. In general, to simulate the i-th session, $\mathcal{S}$ need to run at least $2^{m-i}t$ steps. Thus, the running time of $\mathcal{S}$ becomes super-polynomial when $m = \omega(\log n)$.

Protocol of Canetti et al. [CLP13a]. To avoid the blow-up of the simulator's running time, Canetti, Lin, and Paneth (CLP) [CLP13a] used the "multiple slots" approach, which was originally used in previous black-box concurrent zero-knowledge protocols [RK99, KP01, PRS02]. The idea is that if BarakZK has multiple sequential slots, $\mathcal{S}$ can choose any of them as a witness in WIUA, and therefore $\mathcal{S}$ can avoid the nested computations in WIUA by using a good *proving strategy* that determines which slot to use as a witness in each session. To implement this approach, CLP first observed that the four-round public-coin UA of Barak and Goldreich [BG09], from which WIUA can be constructed, can be divided into the *offline phase* and the *online phase* such that all heavy computations are done in the offline phase. Concretely, CLP divided the UA of [BG09] as follows. Let $x \in L$ be the statement to be proven in UA and w be a witness for $x \in L$.

Offline/online UA

- Offline Phase:
  1. V sends a random hash function $h \in \mathcal{H}_n$ to P.
  2. P generates a PCP proof π of statement $x \in L$ by using w as a witness, and then computes $\mathsf{UA}_2 := h(\pi)$. In the following, (h, π, UA_2) is called the *offline proof*.

[4]The circularity about the simulator committing to a machine that emulates the simulator itself can be avoided by separating it to the main simulator $\mathcal{S}$ and an auxiliary simulator aux-$\mathcal{S}$. Roughly speaking, aux-$\mathcal{S}$ takes a code of a machine Π as input and does simulation by committing to Π; then, $\mathcal{S}$ invokes aux-$\mathcal{S}$ with input $\Pi =$ aux-$\mathcal{S}$.

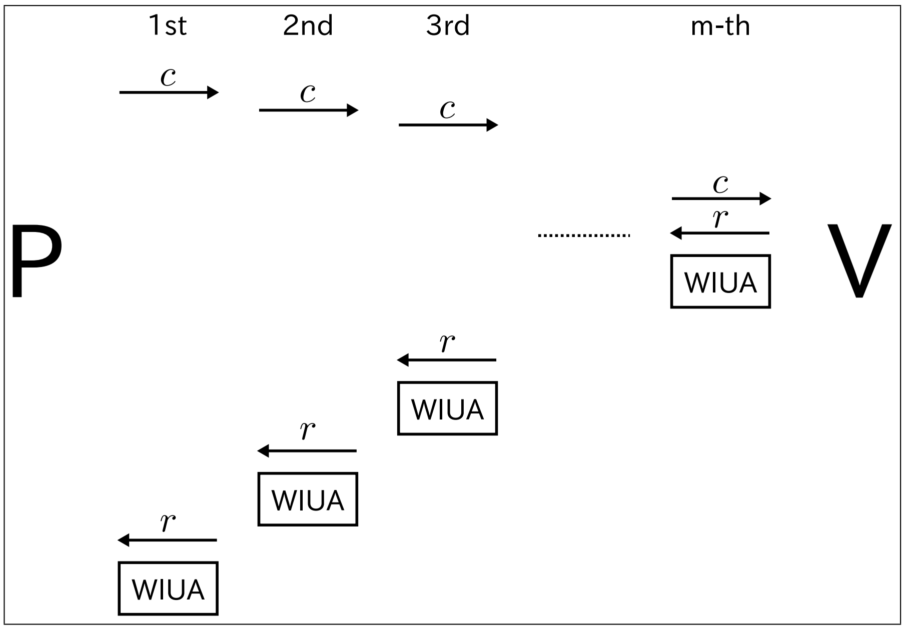

Figure 1: The "nested concurrent sessions" schedule.

- Online Phase:
  1. P sends UA_2 to V.
  2. V chooses randomness ρ for the PCP-verifier and sends $\mathsf{UA}_3 := \rho$ to P.
  3. P computes a PCP-query Q by executing the PCP-verifier with statement $x \in L$ and randomness ρ, and then sends $\{\pi_i\}_{i\in Q}$ to V (i.e., partially reveals π according to the locations that are specified by Q) while proving that $\{\pi_i\}_{i\in Q}$ is correctly computed w.r.t. the string it hashed in UA_2. (Such a proof can be generated efficiently if P computes $\mathsf{UA}_2 = h(\pi)$ by tree hashing.)
  4. V first verifies the correctness of the revealed bits $\{\pi_i\}_{i\in Q}$, and next verifies the PCP proof by executing the PCP-verifier on $\{\pi_i\}_{i\in Q}$.

Note that the only heavy computations—the generation of π and the computation of $h(\pi)$—are performed in the offline phase; the other computations can be performed in a fixed polynomial time. (For simplicity, here we assume that P has random access to π.[5]) Thus, in the online phase, the running time of P can be bounded by a fixed polynomial in n. In the offline phase, the running time of P is bounded by a fixed polynomial in t, where t is the time needed for verifying $x \in L$ with witness w. The length of the offline proof is also bounded by a polynomial in t.

CLP then considered the following protocol (which is an over-simplified version of their final protocol). Let N_{slot} be a parameter that is determined later.

Protocol BasicCLP

Stage 1. V chooses a hash function $h \in \mathcal{H}_n$ and sends it to P.

Stage 2. For each $i \in [N_{\text{slot}}]$ in sequence, P and V do the following.

- P sends $C_i \leftarrow \mathsf{Com}(0^n)$ to V. Then, V sends a random string $r_i \in \{0,1\}^n$ to P.

[5]This assumption is used only in this overview.

Stage 3. P and V execute the special-purpose WIUA of Pass and Rosen [PR05] with the UA system of Barak and Goldreich [BG09] being used as the underlying UA system. Concretely, P and V do the following.

1. P sends $D_{\mathsf{UA}} \leftarrow \mathsf{Com}(0^n)$ to V.
2. V sends a third-round UA message UA_3 to P (i.e., V sends a random string of appropriate length).
3. P proves the following statement by using a witness-indistinguishable proof of knowledge (WIPOK).
   - $x \in L$, or
   - there exist $i \in [N_{\mathrm{slot}}]$ and a second- and a fourth-round UA message $\mathsf{UA}_2, \mathsf{UA}_4$ such that D_{UA} is a commitment to UA_2 and $(h, \mathsf{UA}_2, \mathsf{UA}_3, \mathsf{UA}_4)$ is an accepting proof for the statement $(h, C_i, r_i) \in \Lambda$.

Recall that the idea of the multiple-slot approach is that $\mathcal{S}$ avoids nested computations in WIUA by using a good proving strategy that determines which slots to use as witnesses. Based on this idea, CLP designed a proving strategy as well as a simulator. First, their simulator works roughly as follows: $\mathcal{S}$ commits to a machine in each slots, where the committed machines emulate $\mathcal{S}$ as mentioned above; $\mathcal{S}$ then computes an offline proof (including a PCP proof) w.r.t. a slot that is chosen according to the proving strategy; $\mathcal{S}$ then commits to the second-round UA message (i.e., the hash of the PCP proof) in Stage 3-1 and gives a WIPOK proof in Stage 3-3 using the offline proof as a witness. Second, their proving strategy works roughly as follows. As in the *oblivious rewinding strategy* of black-box concurrent zero-knowledge protocols [KP01, PRS02], the proving strategy of CLP recursively divides the entire transcript between honest provers and the cheating verifier into "blocks." Let M be the total number of messages and q be a parameter called the *splitting factor*. Assume for simplicity that M is a power of q, i.e., $M = q^d$ for $d \in \mathbb{N}$.

- The level-d block is the entire transcript. Thus, the level-d block contains $M = q^d$ messages.
- Then, the level-d block is divided into q sequential blocks, where each of them contains q^{d-1} messages. These blocks are called the level-$(d-1)$ blocks.
- Similarly, each level-$(d-1)$ block is divided into q sequential blocks, where each of them contains q^{d-2} messages. These blocks are called the level-$(d-2)$ blocks.
- In this way, each block is continued to be divided into q blocks until the level-0 blocks are obtained. A level-0 block contains only a single message.

Then, the proving strategy of CLP specifies that at the end of each block in each level, $\mathcal{S}$ computes offline proofs w.r.t. each slot that is contained in that block. Note that the offline proofs are computed only at the end of the blocks, and the maximum level of the blocks (i.e., d) is constant when $q = n^{\epsilon}$ for a constant ϵ. We therefore have at most constant levels of nesting in the executions of WIUA. Furthermore, it was shown by CLP that when $N_{\mathrm{slot}} = \omega(q) = \omega(n^{\epsilon})$, the simulator does not "get stuck," i.e., in every session, the simulator obtains an offline proof before Stage 3 starts.

The protocol $\mathsf{BasicCLP}$ is, however, not concurrent zero-knowledge in the plain model. Roughly speaking, this is because the size of $\mathcal{S}$'s state can become super-polynomial. Recall that $\mathcal{S}$ generates an offline proof in Stage 2 and uses it in Stage 3 in each session. Then, since V^* can choose any concurrent schedule (and in particular can delay the execution of Stage 3 arbitrarily), in general, $\mathcal{S}$ need to remember every previously generated offline proof during its execution. This means that each committed machine also need to contain every previously generated offline proof (otherwise they cannot emulate the simulator), and therefore an offline proof (which is generated by using a committed machine as a witness) is as long as the total length of all the offline proofs that are generated previously. Thus, the length of the offline proofs can be rapidly blowing up and the size of $\mathcal{S}$'s state cannot be bounded by a polynomial.

A key observation by CLP is that this problem can be solved in the *global hash model*, in which a global hash function is shared by all sessions. Roughly speaking, CLP avoided the blow-up of the simulator's size by considering machines that contain only the hash values of the offline proofs; then, to guarantee that the simulation works with such machines, they modified $\mathsf{BasicCLP}$ in such a way that P proves in WIUA that $x \in L$ or the committed machine outputs r given access to the *hash-inversion oracle*; in the simulation, $\mathcal{S}$ commits to a machine that emulates $\mathcal{S}$ by recovering offline proofs from their hash values using the hash-inversion oracle. In this modified protocol, the soundness is proven by using the fact that the same hash function is used across all the sessions.

In this way, CLP obtained a public-coin concurrent zero-knowledge protocol in the global hash model. Since $q = n^{\epsilon}$ and $N_{\mathrm{slot}} = \omega(q)$, the round complexity is $O(n^{\epsilon'})$ for a constant ϵ'. (Since ϵ is an arbitrary constant, ϵ' can be an arbitrary small constant.) CLP also showed that by modifying the protocol further, the round complexity can be reduced to $O(\log^{1+\epsilon} n)$.

2.2 Our Techniques

We obtain our $O(n^{\epsilon})$-round protocol by modifying $\mathsf{BasicCLP}$ of Canetti et al. [CLP13a] so that its concurrent zero-knowledge property can be proven without using global hash functions. Recall that a global hash function is used in [CLP13a] to avoid the blow-up of the simulator's state size. In particular, a global hash function is used so that the simulation works even when the committed machines do not contain any previously computed offline proof. Below, we first introduce the machines that our simulator commits to in the slots. They do not contain any previously generated offline proof and therefore their sizes are bounded by a fixed polynomial. We then explain our protocol and simulator, which are designed so that the simulation works even when the committed machines do not contain any previously computed offline proof. In the following, we set $q := n^{\epsilon}$, $N_{\mathrm{slot}} := \omega(q)$, and $N_{\mathrm{col}} := \omega(1)$.

The machines to be committed. Our first observation is that if the committed machines emulate a larger part of the simulation, they generate more offline proofs by itself, and they are more likely to be able to output r even when they contain no offline proof. For example, let us consider an extreme case that the committed machines emulate the simulator from the beginning of the simulation (rather than from the beginning of the slots in which they are committed to). In this case, the committed machines generate every offline proof by themselves, so they can output r even when they contain no offline proof. A problem of this case is that the running time of each committed machine is too long and the running time of the simulator becomes super-polynomial. We therefore need to design machines that emulate a large, but not too large, part of the simulation.

Based on this observation, we consider machines that emulate the simulator from the beginning of the "current blocks," i.e., machines that emulate the simulator from the beginning of the blacks that contain the commitments in which they are committed to. More precisely, we first modify $\mathsf{BasicCLP}$ so that P gives N_{col} parallel commitments in each slot. Then, our simulator commits to machines in each slot as follows. Below, the i-th *column* ($i \in [N_{\mathrm{col}}]$) of a slot is the i-th commitment of the slot, and the *current level-ℓ block* ($\ell \in [d]$) at a point during the interaction with V^* is the level-ℓ block that will contain the next-scheduled message (see Figure 2).

- In the i-th column ($i \in [d]$) of a slot, our simulator commits to a machine Π_i that emulates the simulator from the beginning of the current level-i block, where Π_i does not contain any offline

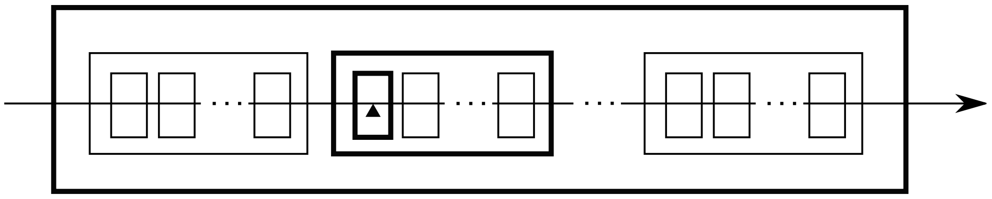

Figure 2: An illustration of the current blocks. When the next scheduled message is located on the place specified by the triangle, the current blocks are the ones described with the thick lines.

proofs, and it terminates with output fail if the emulation fails due to the lack of the offline proofs.

Now, we observe that the simulator's running time does not blow-up when the simulator commits to machines as above. Assume that, as in the proving strategy of CLP, the simulator computes the offline proofs only at the end of the blocks. Specifically, assume that the simulator compute the offline proofs at the end of the blocks as follows.

- At the end of a level-ℓ block b ($\ell \in [d]$), the simulator finds all the slots that are contained in block b, and generates offline proofs w.r.t. those slots by using the machine that are committed to in their ℓ-th columns. Note that those committed machines emulate the simulator from the beginning of block b, so the simulator can indeed use them as witness when generating the offline proofs.

Let t_i be the maximum time needed for simulating a level-i block ($i \in \{0, \ldots, d\}$). Recall that a level-i block consists of q level-$(i-1)$ blocks, and at most $m := \mathsf{poly}(n)$ offline proofs are generated at the end of each level-$(i-1)$ block. Then, since each offline proof at the end of a level-$(i-1)$ block can be computed in $\mathsf{poly}(t_{i-1})$ steps, we have

$$t_i \leq q \cdot (t_{i-1} + m \cdot \mathsf{poly}(t_{i-1})) \leq \mathsf{poly}(t_{i-1}) \ .$$

Recall that we have $t_0 = \mathsf{poly}(n)$ (this is because a level-0 block contains only a single message), and the maximum level $d = \log_q M$ is constant. We therefore have $t_d = \mathsf{poly}(n)$, so the running time of the simulator is bounded by a polynomial in n.

We note that although the above machines do not contain any previously generated offline proof, they do contain every previously generated WIPOK witness (i.e., UA_2 and UA_4).[6] As explained below, allowing the committed machines to contain every previously generated WIPOK witness is crucial to obtain our protocol and simulator.

Our protocol and simulator. When the simulator commits to the above machines, the simulation does not work if the committed machines output fail. In particular, the simulation fails if there exists a block in which the simulator uses an offline proof that are generated before the beginning of that block. (If such a block exists, the machines that are committed in this block output fail since they cannot emulate the simulator due to the lack of the offline proof.) Thus, to guarantee successful simulation, we need to make sure that in each block, the simulator uses only the offline proofs that are generated in that block. Of course, we also need to make sure that the simulator does not "get stuck," i.e., we need to guarantee that in each session, the simulator obtains a valid witness before WIPOK starts.

[6]Since the length of the WIPOK witnesses is bounded by a fixed polynomial, the sizes of the committed machines do not blow up even when they contain every previously generated WIPOK witness.

To avoid the simulation failure, we first modify BasicCLP as follows. As noted in the previous paragraph, we need to construct a simulator such that in each block, it uses only the offline proofs that are generated in that block. In BasicCLP, it is hard to construct such a simulator since offline proofs may be used long after they are generated. (Recall that during the simulation, the offline proofs are generated in Stage 2 and they are used in Stage 3 to compute WIPOK witnesses, and V^* can delay the execution of Stage 3 arbitrarily.) Thus, we modify BasicCLP so that the simulator can use the offline proofs soon after generating them; in particular, we modify BasicCLP so that Stage 3 can be executed "in the middle of" Stage 2. Concretely, after each slot in Stage 2, we add another slot, *a* UA-*slot*, that can be used for executing Stage 3-1 and Stage 3-2. That is, we consider the following protocol. (As stated before, we also modify BasicCLP so that P gives N_{col} parallel commitments in each slot.)

Protocol OurZK

Stage 1. V chooses a hash function $h \in \mathcal{H}_n$ and sends it to P.

Stage 2. For each $i \in [N_{\text{slot}}]$ in sequence, P and V do the following.

Π-slot: P sends $C_{i,1} \leftarrow \mathsf{Com}(0^n), \ldots, C_{i,N_{\text{col}}} \leftarrow \mathsf{Com}(0^n)$ to V. Then, V sends a random string $r_i \in \{0,1\}^{n^2}$ to P.

UA-slot: P sends $D_{i,1} \leftarrow \mathsf{Com}(0^n), \ldots, D_{i,N_{\text{col}}} \leftarrow \mathsf{Com}(0^n)$ to V. Then, V sends a random string ω_i to P.

Stage 3. P proves the following statement with WIPOK.

- $x \in L$, or
- there exist $i_1, i_2 \in [N_{\text{slot}}]$, $j \in [N_{\text{col}}]$, and a second- and a fourth-round UA message UA_2 and UA_4 such that $D_{i_2,j}$ is a commitment to UA_2 and $(h, \mathsf{UA}_2, \omega_{i_2}, \mathsf{UA}_4)$ is an accepting proof of the statement $(h, C_{i_1,j}, r_{i_1}) \in \Lambda$.

We then consider the following simulator. Recall that, as explained above, our simulator commits to machines that emulate the simulation from the beginning of the current blocks, and its running time can be bounded by a polynomial if the offline proofs are computed only at the end of the blocks. Recall also that we need to make sure that (i) in each block the simulator uses only the offline proofs that are generated in that block (so that each committed machine does not output fail due to the lack of the offline proofs), and (ii) the simulator does not get stuck.

Roughly speaking, our simulator does the following in each block (see Figure 3). Consider any level-$(i+1)$ block b ($i \in [d-1]$), and recall that b consists of q level-i blocks. The goal of our simulator in block b is to compute offline proofs by using the machines that emulate the simulation from the beginning of those level-i blocks, and find opportunities to use them before block b completes. Therefore, for each session s, our simulator first tries to find a level-i block that contains a Π-slot of session s. If it finds such a level-i block and a Π-slot, it computes an offline proof at the end of that level-i block by using the machine that is committed to in the i-th column of that Π-slot, and commits to this offline proof in the i-th column of the UA-slots of session s in the subsequent level-i blocks. If a subsequent level-i block contains a UA-slot of session s, it computes a WIPOK witness from this offline proof (i.e., by using the third-round UA message in that UA-slot, it computes a fourth-round UA message from that offline proof).

More precisely, we consider the following simulator. In what follows, for each $i \in \{0, \ldots, d-1\}$, we say that two level-i blocks are *sibling* if they are contained by the same level-$(i+1)$ block.

- In the i-th column ($i \in [d]$) of a Π-slot of a session s, our simulator commits to a machine that emulates the simulator from the beginning of the current level-i block.

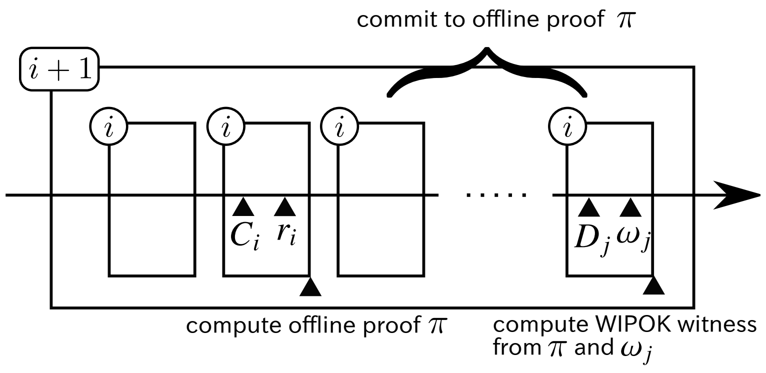
commit to offline proof π
$i+1$
i
i
i
i
C_i r_i
D_j ω_j
compute offline proof π
compute WIPOK witness from π and ω_j

Figure 3: Our simulator's strategy.

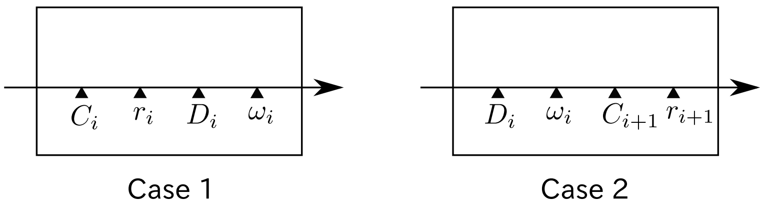
C_i r_i D_i ω_i
Case 1
D_i ω_i C_{i+1} r_{i+1}
Case 2

Figure 4: If a block contains two slots of a session, it contains both a Π-slot and a UA-slot.

- In the i-th column ($i \in [d]$) of a UA-slot of a session s, our simulator commits to 0^n if no prior sibling of the current level-i block contains a Π-slot of session s; if a prior sibling contains a Π-slot of session s, an offline proof w.r.t. such a Π-slot was computed at the end of that prior sibling (see below), so our simulator commits to that offline proof instead of 0^n.
- When WIPOK starts, our simulator does the following. If it already obtained a valid witness (see below), it gives a proof by using this witness. If it does not have a valid witness, it aborts with output stuck.
- At the end of a level-i block b ($i \in [d-1]$), our simulator does the following. For each Π-slot that is contained in block b, it computes an offline proof by using the machine that is committed to in the i-th column of that Π-slot. Also, for each UA-slot that is contained in block b, if an offline proof is committed to in the i-th column of that UA-slot, it computes a WIPOK witness by using that offline proof.

In the simulation by our simulator, the committed machines never fail due to the lack of the offline proofs. This is because in each block, our simulator uses only the offline proofs that are generated in that block.

Thus, it remains to show that our simulator does not get stuck, i.e., in each session our simulator has a valid witness when WIPOK starts. Below, we use the following terminologies.

- For any session s, a block is *good* w.r.t. s if it contains at least two slots of session s and does not contain the first prover message of WIPOK of session s. Here, we use "slots" to refer to both Π-slots and UA-slots. Hence, if a block is good w.r.t. session s, it contains both a Π-slot and a UA-slot of session s (see Figure 4).
- For each $i \in [d]$, we say that a level-$(i-1)$ block is a *child* of a level-i block if the former is contained by the latter. (Thus, each block has q children.)

From the construction, our simulator does not get stuck if for any session s that reaches WIPOK, there exists a block such that at least two of its children are good w.r.t. session s. (If such a block exists, an offline proof is computed at the end of the first good child, and a WIPOK witness is computed at the end of the second good child, so the simulator obtains a WIPOK witness before WIPOK starts in session s.) Thus, we show that if a session s reaches WIPOK, there exists a block such that at least two of its children are good w.r.t. session s. To prove this, it suffices to show that if a session s reaches WIPOK, there exists a block such that at least three of its children contain two or more slots of session s. (This is because at most one child contains the first message of WIPOK of session s.) Assume for contradiction that there exists a session s^* such that s^* reaches WIPOK but every block has at most two children that contain two or more slots of s^*. Let $\Gamma(i)$ be the maximum number of the slots that belong to s^* and are contained by a level-i block. Then, since in each block b,

- at most two children of b contain two or more slots of s^*, and the other children contain at most a single slot of s^*, and
- s^* has at most $q-1$ slots that are contained by block b but are not contained by its children (see Figure 5),

we have

$$\Gamma(i) \leq 2 \cdot \Gamma(i-1) + (q-2) \cdot 1 + q - 1 = 2\Gamma(i-1) + 2q - 3 \ .$$

Then, since $\Gamma(0) = 0$ (this is because a level-0 block contains only a single message), and the maximum level d is constant, we have

$$\Gamma(d) \leq 2^d \Gamma(0) + \sum_{i=0}^{d-1} 2^i (2q-3) = O(q) \ .$$

This means that there are at most $O(q)$ slots of s^* in the entire transcript. This is a contradiction since we have $N_{\text{slot}} = \omega(q)$ and assume that s^* reaches WIPOK. Thus, if a session reaches WIPOK, there exists a block such that at least two of its children are good w.r.t. that session. Thus, the simulator does not get stuck.

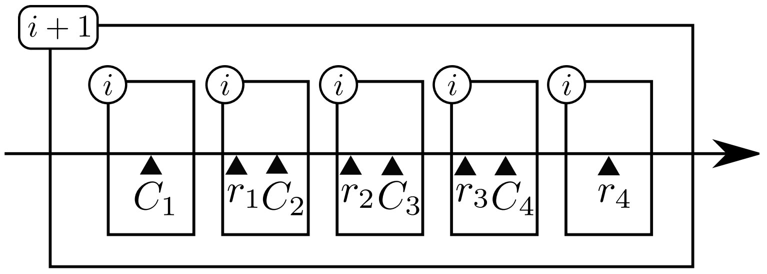

Figure 5: An example that a session has $q-1$ slots that are contained by a block but are not contained by its children. (For simplicity, only Π-slots are illustrated.)

Since $q = O(n^{\epsilon})$ and $N_{\text{slot}} = \omega(q)$, the round complexity of our protocol is $O(n^{\epsilon'})$ for a constant $\epsilon' > \epsilon$. Since ϵ is an arbitrary constant, ϵ' can be an arbitrary small constant.

Toward the final protocol. To obtain a formal proof of security, we need to add a slight modification to the above protocol. In particular, as pointed out in previous work [Goy13, CLP13a, CLP13b, PPS15], when the code of the simulator is committed in the simulation, we have to take special care to the randomness of the simulator.[7] Fortunately, the techniques used in the previous work can also be used here to overcome this problem. In this work, we use the technique of [CLP13a, CLP13b], which uses forward-secure pseudorandom generators (which can be obtained from one-way functions).

2.3 Comparison with the Simulation Technique of Goyal [Goy13]

In this section, we compare the simulation technique of ours with that of Goyal [Goy13], which is the only known simulation technique that realizes straight-line concurrent simulation in the plain model under standard assumptions.

First of all, our protocol is almost identical with that of Goyal. The only difference is that the prover gives $\omega(1)$ commitments in each slot in our protocol whereas it gives only a single commitment in each slot in Goyal's protocol.

Our simulation technique is also very similar to Goyal's simulation technique. For example, in both simulation techniques, the simulator commits to machines that emulate itself, and it has multiple opportunities to give UA proof and determines which opportunities to take by using the blocks.

However, there are also differences between the two simulation techniques. A notable difference is how the simulator determines which opportunities to take to give UA proofs. Recall that, in the simulation technique of ours, the strategy that the simulator uses to determine whether it embeds a UA message in a slot is deterministic (the simulator checks whether a prior sibling of the current block contains a Π-slot; see Figure 3 in Section 2.2). In contrast, in the simulation technique of Goyal, the strategy that the simulator uses is probabilistic (the simulator uses a probabilistic procedure that performs the "marking" of the blocks and the UA messages). Since in the simulation technique of ours the simulator uses a deterministic strategy, the analysis of our simulator is simple: We use only a simple counting argument (and no probabilistic argument) to show that the simulator will not get stuck.

3 Preliminary

We assume familiarity to the definition of basic cryptographic primitives and protocols, such as collision-resistant hash functions and commitment schemes.

3.1 Notations

We use n to denote the security parameter, and PPT as an abbreviation of "probabilistic polynomial time." For any $k \in \mathbb{N}$, let $[k] \overset{\text{def}}{=} \{1, \ldots, k\}$. For any randomized algorithm Algo, we use $\mathsf{Algo}(x; r)$ to denote the execution of Algo with input x and randomness r, and $\mathsf{Algo}(x)$ to denote the execution of Algo with input x and uniformly chosen randomness. For any two interactive Turing machines A, B and three strings $x, y, z \in \{0, 1\}^*$, we use $\langle A(y), B(z)\rangle(x)$ to denote the output of B in the interaction between $A(x, y)$ and $B(x, z)$.

[7]When the code of the simulator is committed, the randomness used for generating this commitment is also committed; thus, if a protocol is designed naively, we need a commitment scheme such that the committed value is hidden even when it contains the randomness used for the commitment.

3.2 Tree Hashing

In this paper, we use a family of collision-resistant hash functions $\mathcal{H} = \{h_\alpha\}_{\alpha \in \{0,1\}^*}$ that satisfies the following properties.

- For any $h \in \mathcal{H}_n \overset{\text{def}}{=} \{h_\alpha \in \mathcal{H} : \alpha \in \{0,1\}^n\}$, the domain of h is $\{0,1\}^*$ and the range of h is $\{0,1\}^n$.
- For any $h \in \mathcal{H}_n$, $x \in \{0,1\}^{\leq n^{\log\log n}}$, and $i \in \{1, \dots, |x|\}$, one can compute a short certificate $\mathsf{auth}_i(x) \in \{0,1\}^{n^2}$ such that given $h(x)$, x_i, and $\mathsf{auth}_i(x)$, anyone can verify that the i-th bit of x is indeed x_i.

Such a collision-resistant hash function family can be obtained from any (standard) length-halving collision-resistant hash function family by using Merkle's tree-hashing technique. We notice that when $\mathcal{H}$ is obtained in this way, $\mathcal{H}$ satisfies an additional property that we can find a collision of the underlying hash function from two pairs $(x_i, \mathsf{auth}_i(x))$ and $(x'_i, \mathsf{auth}_i(x'))$ such that $x_i \neq x'_i$; furthermore, finding such a collision takes only time polynomial in the size of the hash value (i.e., $|h(x)| = n$).

3.3 Naor's Commitment Scheme

In our protocol, we use Naor's two-round statistically binding commitment scheme Com, which can be constructed from one-way functions [Nao91, HILL99]. A nice property of Com is that its security holds even when the same first-round message $\tau \in \{0,1\}^{3n}$ is used in multiple commitments. For any $\tau \in \{0,1\}^{3n}$, we use $\mathsf{Com}_\tau(\cdot)$ to denote an algorithm that, on input $m \in \{0,1\}^*$, computes a commitment to m by using τ as the first-round message.

3.4 Interactive Proofs and Arguments

We recall the definitions of *interactive proofs* and *interactive arguments*, and the definitions of their *witness indistinguishability* and *proof-of-knowledge* property [GMR89, BCC88, FS90, BG92].

Definition 1 (Interactive Proof System). *For an $\mathcal{NP}$ language L with witness relation $\boldsymbol{R}_L$, a pair of interactive Turing machines $\langle P, V\rangle$ is an* interactive proof *for L if it satisfies the following properties.*

- ***Completeness:*** *For every $x \in L$ and $w \in \boldsymbol{R}_L(x)$,*

$$\Pr\left[\langle P(w), V\rangle(x) = 1\right] = 1 \ .$$

- ***Soundness:*** *For every computationally unbounded Turing machine P^*, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $x \notin L$ and $z \in \{0,1\}^*$,*

$$\Pr\left[\langle P^*(z), V\rangle(x) = 1\right] < \mathsf{negl}(|x|) \ .$$

If the soundness condition holds only against every PPT *Turing machine, the pair $\langle P, V\rangle$ is an* interactive argument. ◇

Definition 2 (Witness Indistinguishability). *An interactive proof (or argument) system $\langle P, V\rangle$ for an $\mathcal{NP}$ language L with witness relation $\boldsymbol{R}_L$ is said to be* witness indistinguishable *if for every* PPT *Turing machine V^* and for every two sequences $\{w^1_x\}_{x\in L}$ and $\{w^2_x\}_{x\in L}$ such that $w^1_x, w^2_x \in \boldsymbol{R}_L(x)$ for every $x \in L$, the following ensembles are computationally indistinguishable.*

- $\left\{\langle P(w^1_x), V^*(z)\rangle(x)\right\}_{x\in L, z\in\{0,1\}^*}$

- $\left\{\langle P(w_x^2), V^*(z)\rangle(x)\right\}_{x\in L, z\in\{0,1\}^*}$ $\diamond$

Definition 3 (Proof of Knowledge)**.** *An interactive proof system $\langle P, V\rangle$ for an $\mathcal{NP}$ language L with witness relation $\boldsymbol{R}_L$ is said to be* proof of knowledge *if there exists an expected* PPT *oracle machine E (call the* extractor*) such that the following holds: For every computationally unbounded Turing machine P^*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $x \in \{0,1\}^*$ and $z \in \{0,1\}^*$,*

$$\Pr\left[\exists w \in \boldsymbol{R}_L(x) \text{ s.t. } E^{P^*(x,z)}(x) = w\right] > \Pr\left[\langle P^*(z), V\rangle(x) = 1\right] - \mathsf{negl}(|x|) \ .$$

If the above condition holds only against every PPT *Turing machine P^*, the pair $\langle P, V\rangle$ is said to be* argument of knowledge. $\diamond$

A four-round witness-indistinguishable proof of knowledge system WIPOK can be obtained from one-way functions by executing Blum's Hamiltonian-cycle protocol in parallel [Blu86].

3.5 Concurrent Zero-Knowledge Proofs/Arguments

We recall the definition of the *concurrent zero-knowledge* property of interactive proofs and arguments [RK99]. For any polynomial $m(\cdot)$, *m-session concurrent cheating verifier* is a PPT Turing machine V^* such that on input (x, z), V^* concurrently interacts with $m(|x|)$ independent copies of P. The interaction between V^* and each copy of P is called *session*. There is no restriction on how V^* schedules messages among sessions, and V^* can abort some sessions. Let $\mathsf{view}_{V^*}\langle P(w), V^*(z)\rangle(x)$ be the view of V^* in the above concurrent execution, where $x \in L$ is the common input, $w \in \mathbf{R}_L(x)$ is the private input to P, and z is the non-uniform input to V^*.

Definition 4 (Concurrent Zero-Knowledge)**.** *An interactive proof (or argument) $\langle P, V\rangle$ for an $\mathcal{NP}$ language L is* concurrent zero-knowledge *if for every polynomial $m(\cdot)$ and every m-session concurrent cheating verifier V^*, there exists a* PPT *simulator $\mathcal{S}$ such that for any sequence $\{w_x\}_{x\in L}$ such that $w_x \in \boldsymbol{R}_L(x)$, the following ensembles are computationally indistinguishable.*

- $\{\mathsf{view}_{V^*}\langle P(w_x), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$
- $\{\mathcal{S}(x, z)\}_{x\in L, z\in\{0,1\}^*}$ $\diamond$

Remark 1. As in previous work (e.g., [RK99, KP01, PRS02]), we consider the setting where the same statement x is proven in all the sessions. We comment that our protocol and its security proof work even in a slightly generalized setting where predetermined statements $x_1, \ldots, x_m$ are proven in the sessions. (However, they do not work if the statements are chosen adaptively by the cheating verifier.)

3.6 PCP and Universal Argument

We recall the definitions of *probabilistically checkable proof* (PCP) systems and *universal argument* systems [AS98, BG09].

3.6.1 Universal Language $L_{\mathcal{U}}$.

For simplicity, we show the definitions of PCPs and universal arguments only w.r.t. the membership of a single "universal" language $L_{\mathcal{U}}$. For triplet $y = (M, x, t)$, we have $y \in L_{\mathcal{U}}$ if non-deterministic machine M accepts x within t steps. (Here, all components of y, including t, are encoded in binary.) Let $\mathbf{R}_{\mathcal{U}}$ be the witness relation of $L_{\mathcal{U}}$, i.e., $\mathbf{R}_{\mathcal{U}}$ is a polynomial-time decidable relation such that for any $y = (M, x, t)$, we have $y \in L_{\mathcal{U}}$ if and only if there exists $w \in \{0,1\}^{\leq t}$ such that $(y, w) \in \mathbf{R}_{\mathcal{U}}$. Note that every language $L \in \mathcal{NP}$ is linear-time reducible to $L_{\mathcal{U}}$. Thus, a proof system for $L_{\mathcal{U}}$ allows us to handle all $\mathcal{NP}$ statements.[8]

[8]In fact, every language in $\mathcal{NEXP}$ is polynomial-time reducible to $L_{\mathcal{U}}$.

3.6.2 PCP System.

Roughly speaking, a PCP system is a PPT verifier that can decide the correctness of a statement $y \in L_{\mathcal{U}}$ given access to an oracle π that represents a proof in a redundant form. Typically, the verifier reads only few bits of π in the verification.

Definition 5 (PCP system—basic definition)**.** *A probabilistically checkable proof (PCP) system (with a negligible soundness error) is a* PPT *oracle machine V (called a verifier) that satisfies the following.*

- ***Completeness:*** *For every $y \in L_{\mathcal{U}}$, there exists an oracle π such that*

$$\Pr\left[V^{\pi}(y) = 1\right] = 1 \ .$$

- ***Soundness:*** *For every $y \notin L_{\mathcal{U}}$ and every oracle π, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\Pr\left[V^{\pi}(y) = 1\right] < \mathsf{negl}(|y|) \ .$$

$\diamond$

In this paper, PCP systems are used as a building block in the universal argument UA of Barak and Goldreich [BG09]. To be used in UA, PCP systems need to satisfy four auxiliary properties: relatively efficient oracle construction, non-adaptive verifier, efficient reverse sampling, and proof of knowledge. The definitions of the first two properties are required to understand this paper; for the definitions of the other properties, see [BG09].

Definition 6 (PCP system—auxiliary properties)**.** *Let V be a PCP-verifier.*

- ***Relatively efficient oracle construction:*** *There exists an algorithm P (called a prover) such that, given any $(y, w) \in \boldsymbol{R}_{\mathcal{U}}$, algorithm P outputs an oracle π_y that makes V always accept (i.e., as in the completeness condition). Furthermore, there exists a polynomial $p(\cdot)$ such that on input (y, w), the running time of P is $p(|y| + |w|)$.*
- ***Non-adaptive verifier:*** *The verifier's queries are determined based only on the input and its internal coin tosses, independently of the answers given to previous queries. That is, V can be decomposed into a pair of algorithms Q and D such that on input y and random tape r, the verifier makes the query sequence $Q(y, r, 1), Q(y, r, 2), \ldots, Q(y, r, p(|y|))$, obtains the answers $b_1, \ldots, b_{p(|y|)}$, and decides according to $D(y, r, b_1 \cdots b_{p(|y|)})$, where p is some fixed polynomial.*

$\diamond$

3.6.3 Universal Argument.

Universal arguments [BG09], which are closely related to the notion of CS poofs [Mic00], are "efficient" arguments of knowledge for proving the membership in $L_{\mathcal{U}}$. For any $y = (M, x, t) \in L_{\mathcal{U}}$, let $T_M(x, w)$ be the running time of M on input x with witness w, and let $\mathbf{R}_{\mathcal{U}}(y) \overset{\text{def}}{=} \{w : (y, w) \in \mathbf{R}_{\mathcal{U}}\}$.

Definition 7 (Universal argument)**.** *A pair of interactive Turing machines $\langle P, V\rangle$ is a* universal argument *system if it satisfies the following properties.*

- ***Efficient verification:*** *There exists a polynomial p such that for any $y = (M, x, t)$, the total time spent by (probabilistic) verifier strategy V on inputs y is at most $p(|y|)$.*

- ***Completeness by a relatively efficient prover:*** *For every $y = (M, x, t) \in L_{\mathcal{U}}$ and $w \in \boldsymbol{R}_{\mathcal{U}}(y)$,*

$$\Pr\left[\langle P(w), V\rangle(y) = 1\right] = 1 \ .$$

  Furthermore, there exists a polynomial q such that the total time spent by P, on input (y, w), is at most $q(|y| + T_M(x, w)) \leq q(|y| + t)$.

- ***Computational Soundness:*** *For every* PPT *Turing machine P^*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every $y = (M, x, t) \notin L_{\mathcal{U}}$ and $z \in \{0,1\}^*$,*

$$\Pr\left[\langle P^*(z), V\rangle(y) = 1\right] < \mathsf{negl}(|y|) \ .$$

- ***Weak Proof of Knowledge:*** *For every polynomial $p(\cdot)$ there exists a polynomial $p'(\cdot)$ and a* PPT *oracle machine E such that the following holds: For every* PPT *Turing machine P^*, every sufficiently long $y = (M, x, t) \in \{0,1\}^*$, and every $z \in \{0,1\}^*$, if $\Pr\left[\langle P^*(z), V\rangle(y) = 1\right] > 1/p(|y|)$, then*

$$\Pr_r\left[\exists w = w_1 \cdots w_t \in \boldsymbol{R}_{\mathcal{U}}(y) \text{ s.t. } \forall i \in [t], E_r^{P^*(y,z)}(y, i) = w_i\right] > \frac{1}{p'(|y|)} \ ,$$

  where $E_r^{P^*(y,z)}(\cdot,\cdot)$ denotes the function defined by fixing the randomness of E to r, and providing the resulting E_r with oracle access to $P^*(y, z)$. ◇

The weak proof-of-knowledge property of universal arguments only guarantees that each individual bit w_i of a witness w can be extracted in probabilistic polynomial time. However, for any $y = (M, x, t) \in L_{\mathcal{U}}$, since the witness $w \in \mathbf{R}_{\mathcal{U}}(y)$ is of length at most t, there exists an extractor (called the *global extractor*) that extracts the whole witness in time polynomial in $\mathsf{poly}(|y|) \cdot t$; we call this property the *global proof-of-knowledge property* of a universal argument.

In this paper, we use the public-coin four-round universal argument system UA of Barak and Goldreich [BG09] (Figure 6). As in [CLP13a], the construction of UA below is separated into an *offline stage* and an *online stage*. In the offline stage, the running time of the prover is bounded by a fixed polynomial in $n + T_M(x, w)$.

3.7 Forward-secure PRG

We recall the definition of *forward-secure pseudorandom generators* (PRGs) [BY03]. Roughly speaking, a forward-secure PRG is a pseudorandom generator such that

- It periodically updates the seed. Hence, we have a sequence of seeds $(\sigma_1, \sigma_2, \ldots)$ that generates a sequence of pseudorandomness $(\rho_1, \rho_2, \ldots)$.
- Even if the seed σ_t is exposed (and thus the "later" pseudorandom sequence $\rho_{t+1}, \rho_{t+2}, \ldots$ is also exposed), the "earlier" sequence $\rho_1, \ldots, \rho_t$ still remains pseudorandom.

In this paper, we use a simple variant of the definition by [CLP13b]. We notice that in the following definition, the indices of the seeds and pseudorandomness are written in the reverse order because we use them in the reverse order in the analysis of our concurrent zero-knowledge protocol.

Definition 8 (Forward-secure PRG). *We say that a polynomial-time computable function* $\mathsf{f\text{-}PRG}$ *is a* forward-secure pseudorandom generator *if on input a string σ and an integer $\ell \in \mathbb{N}$, it outputs two sequences $(\sigma_\ell, \ldots, \sigma_1)$ and $(\rho_\ell, \ldots, \rho_1)$ that satisfy the following properties.*

- **Input:** The common input is $y = (M, x, t) \in L_{\mathcal{U}}$, and the private input to P is $w \in \mathbf{R}_{\mathcal{U}}(y)$. Let $n \stackrel{\text{def}}{=} |y|$.
- **Offline Phase:**
  1. V sends a random hash function $h \in \mathcal{H}_n$ to P.
  2. P generates a PCP proof π of statement $y \in L_{\mathcal{U}}$ by using w as a witness. Then P computes $\mathsf{UA}_2 := h(\pi)$. The tuple (h, π, UA_2) is called the *offline proof.*
- **Online Phase:**
  1. P sends UA_2 to V.
  2. V chooses randomness $\omega \in \{0,1\}^{n^2}$ for the PCP-verifier and sends $\mathsf{UA}_3 := \omega$ to P.
  3. P computes queries Q by executing the PCP-verifier with statement $y \in L_{\mathcal{U}}$ and randomness ω. Then, P sends $\mathsf{UA}_4 := \{(i, \pi_i, \mathsf{auth}_i(\pi))\}_{i \in Q}$ to V, where π_i is the i-th bit of π and $\mathsf{auth}_i(\pi)$ is a certificate that the i-th bit of π is indeed π_i.
  4. V verifies the correctness of all the certificates and checks whether the PCP-verifier accepts on input $(y, \{(i, \pi_i)\}_{i \in Q})$ with randomness ω.

Figure 6: Online/offline UA system of [BG09, CLP13a].

- ***Consistency:*** *For every $n, \ell \in \mathbb{N}$ and $\sigma \in \{0,1\}^n$, if* $\mathsf{f\text{-}PRG}(\sigma, \ell) = (\sigma_\ell, \dots, \sigma_1, \rho_\ell, \dots, \rho_1)$*, then it holds* $\mathsf{f\text{-}PRG}(\sigma_\ell, \ell - 1) = (\sigma_{\ell-1}, \dots, \sigma_1, \rho_{\ell-1}, \dots, \rho_1)$.
- ***Forward Security:*** *For every polynomial $\ell(\cdot)$, the following ensembles are computationally indistinguishable.*
  - $\{\sigma \leftarrow U_n; (\sigma_{\ell(n)}, \dots, \sigma_1, \rho_{\ell(n)}, \dots, \rho_1) := \mathsf{f\text{-}PRG}(\sigma, \ell(n)) : (\sigma_{\ell(n)}, \rho_{\ell(n)})\}_{n \in \mathbb{N}}$
  - $\{\sigma \leftarrow U_n; \rho \leftarrow U_n : (\sigma, \rho)\}_{n \in \mathbb{N}}$

  Here, U_n is the uniform distribution over $\{0,1\}^n$. ◇

Any (traditional) PRG implies the existence of a forward-secure PRG. Thus from the result of [HILL99], the existence of forward-secure PRGs are implied by the existence of one-way functions.

4 Our Public-Coin Concurrent Zero-Knowledge Argument

In this section, we prove our main theorem.

Theorem 1. *Assume the existence of a family of collision resistant hash functions. Then, for any constant $\epsilon > 0$, there exists an $O(n^\epsilon)$-round public-coin concurrent zero-knowledge argument of knowledge system.*

Proof. Our $O(n^\epsilon)$-round public-coin concurrent zero-knowledge argument of knowledge, cZKAOK, is shown in Figure 7. In cZKAOK, we use the following building blocks.

- Naor's two-round statistically binding commitment scheme Com, which can be constructed from one-way functions (see Section 3.3).
- A four-round public-coin witness-indistinguishable proof of knowledge system WIPOK, which can be constructed from one-way functions (see Section 3.4).

- Four-round public-coin universal argument UA of Barak and Goldreich [BG09], which can be constructed from collision-resistant hash functions (see Section 3.6.3).

Clearly, cZKAOK is public-coin and its round complexity is $4N_{\text{slot}} + 5 = O(n^{\epsilon})$. Thus, Theorem 1 follows from the following lemmas.

Lemma 1. cZKAOK *is concurrent zero-knowledge.*

Lemma 2. cZKAOK *is argument of knowledge.*

Lemma 1 is proven in Section 4.1, and Lemma 2 is proven in Section 4.2. □

Input: The input to the prover P is (x, w), where $x \in L$ and $w \in \mathbf{R}_L(x)$. The input to the verifier V is x. Let $n \overset{\text{def}}{=} |x|$.

Parameter: Integers $N_{\text{slot}} = O(n^{\epsilon})$ and $N_{\text{col}} = \omega(1)$, where ϵ is an arbitrary constant.

Stage 1: V chooses a random hash function $h \in \mathcal{H}_n$ and sends it to P. Additionally, V sends a first-round message $\tau \in \{0,1\}^{3n}$ of Com to P.

Stage 2: For each $i \in [N_{\text{slot}}]$ in sequence, P and V do the following.

Π-slot:

1. P computes $C_{i,j} \leftarrow \mathsf{Com}_{\tau}(0^n)$ for each $j \in [N_{\text{col}}]$. Then, P sends $\overline{C}_i := (C_{i,1}, \ldots, C_{i,N_{\text{col}}})$ to V.
2. V sends random $r_i \in \{0,1\}^{n^2}$ to P.

UA-slot:

3. P computes $D_{i,j} \leftarrow \mathsf{Com}_{\tau}(0^n)$ for each $j \in [N_{\text{col}}]$. Then, P sends $\overline{D}_i := (D_{i,1}, \ldots, D_{i,N_{\text{col}}})$ to V.
4. V computes a third-round message of UA. (Recall that a third-round message of UA is just a random string with appropriate length.) Then, V sends this UA message, ω_i, to P.

Stage 3: P proves the following statement by using WIPOK.

- $x \in L$, or
- $(h, \tau, \overline{C}_1, r_1, \overline{D}_1, \omega_1, \ldots, \overline{C}_{N_{\text{slot}}}, r_{N_{\text{slot}}}, \overline{D}_{N_{\text{slot}}}, \omega_{N_{\text{slot}}}) \in \Lambda_1$, where the language Λ_1 is defined in Figure 8.

Figure 7: Public-coin concurrent zero-knowledge argument cZKAOK.

Remark 2. The languages Λ_2 in Figure 8 is slightly over-simplified and will make cZKAOK work only when $\mathcal{H}$ is collision resistant against $\mathsf{poly}(n^{\log\log n})$-time adversaries. We can make it work under standard collision resistance by using a trick by Barak and Goldreich [BG09], which uses a "good" error-correcting code ECC (i.e., with constant relative distance and with polynomial-time encoding and decoding). More details are given in Section 4.2.

Language Λ_1: (Statement for WIPOK)

$(h, \tau, \overline{C}_1, r_1, \overline{D}_1, \omega_1, \ldots, \overline{C}_{N_{\text{slot}}}, r_{N_{\text{slot}}}, \overline{D}_{N_{\text{slot}}}, \omega_{N_{\text{slot}}}) \in \Lambda_1$ if and only if there exist

- $i_1, i_2 \in [N_{\text{slot}}]$ and $j \in [N_{\text{col}}]$ such that $i_1 \le i_2$
- a second- and a fourth-round UA message $\mathsf{UA}_2 \in \{0,1\}^n$ and $\mathsf{UA}_4 \in \{0,1\}^{\mathsf{poly}(n)}$
- randomness $R \in \{0,1\}^{\mathsf{poly}(n)}$ for Com

such that

- $D_{i_2,j} = \mathsf{Com}_\tau(\mathsf{UA}_2; R)$, and
- $(h, \mathsf{UA}_2, \omega_{i_2}, \mathsf{UA}_4)$ is an accepting proof for $(h, \tau, C_{i_1,j}, r_{i_1}) \in \Lambda_2$.

Language Λ_2: (Statement for UA)

$(h, \tau, C, r) \in \Lambda_2$ if and only if there exist

- a machine Π (with some of the inputs being hardwired) such that $|\Pi| \le n^{\log\log n}$
- randomness $R \in \{0,1\}^{\mathsf{poly}(n)}$ for Com
- a string y such that $|y| = n$

such that

- $C = \mathsf{Com}_\tau(h(\Pi); R)$, and
- $\Pi(y)$ outputs a string that has r as a substring, and $\Pi(y)$ outputs it within $n^{\log\log n}$ steps.

Figure 8: Languages used in cZKAOK.

4.1 Concurrent Zero-knowledge Property

Proof of Lemma 1. Let V^* be any cheating verifier. Since V^* takes an arbitrary non-uniform input z, we assume without loss of generality that V^* is deterministic. Let $m(\cdot)$ be a polynomial such that V^* invokes $m(n)$ concurrent sessions during its execution. (Recall that $n \stackrel{\text{def}}{=} |x|$.) Let $q \stackrel{\text{def}}{=} n^{\epsilon/2}$. We assume without loss of generality that in the interaction between V^* and provers, the total number of messages across all the sessions is always the power of q (i.e., it is q^d for an integer d). Since the total number of messages is at most $M \stackrel{\text{def}}{=} (4N_{\text{slot}} + 5) \cdot m$, we have $d = \log_q M = \log_q(\mathsf{poly}(n)) = O(1)$.

4.1.1 Simulator $\mathcal{S}$

In this section, we describe our simulator. We first give an informal description; a formal description is given after the informal one. We recommend the readers to browse the overview of our techniques in Section 2.2 before reading this section. In the informal description, we use some terminologies that we introduced in Section 2.2.

Informal Description of $\mathcal{S}$

Our simulator, $\mathcal{S}$, simulates the view of V^* by using an auxiliary simulator algorithm $\mathsf{aux}\text{-}\mathcal{S}$, which simulates the transcript between V^* and honest provers by recursively executing itself. The input to $\mathsf{aux}\text{-}\mathcal{S}$ is the recursion level ℓ and the transcript trans that is simulated so far. $\mathsf{aux}\text{-}\mathcal{S}$ is also given oracle access to tables $\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}$ (which $\mathsf{aux}\text{-}\mathcal{S}$ can freely read and update), where T_Π contains the hash values of the machines that should be committed to in the Π-slots, and T_{UA} and T_{W} contain the second-round UA messages and the WIPOK witnesses that are computed so far. The goal of $\mathsf{aux}\text{-}\mathcal{S}$, on input (ℓ, trans) and access to $\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}$, is to add the next q^ℓ messages to trans while updating the tables $\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}$. More details about $\mathsf{aux}\text{-}\mathcal{S}$ are described below.

On level 0 (i.e., when $\ell = 0$), $\mathsf{aux}\text{-}\mathcal{S}$ adds a single message to the simulated transcript as follows. If the next message is a verifier message, $\mathsf{aux}\text{-}\mathcal{S}$ simulates it by simply receiving it from V^*. If the next message is a prover message $(C_1, \ldots, C_{N_{\mathrm{col}}})$ in a Π-slot, $\mathsf{aux}\text{-}\mathcal{S}$ finds the values to be committed from T_Π and generates commitments to them by using Com. Similarly, if the next message is a prover message $(D_1, \ldots, D_{N_{\mathrm{col}}})$ in a UA-slot, $\mathsf{aux}\text{-}\mathcal{S}$ finds appropriate second-round UA messages from T_{UA} and generates commitments to them by using Com. (If appropriate UA messages cannot be found, $\mathsf{aux}\text{-}\mathcal{S}$ generates commitments to 0^n.) If the next message is a prover message of WIPOK, $\mathsf{aux}\text{-}\mathcal{S}$ computes it honestly by using a witness that is stored in T_{W}. (If the stored witness is not a valid witness, $\mathsf{aux}\text{-}\mathcal{S}$ aborts.)

On level $\ell > 0$, $\mathsf{aux}\text{-}\mathcal{S}$ simulates the next q^ℓ messages by recursively executing itself q times in sequence, where each recursive execution simulates $q^{\ell-1}$ messages. More precisely, $\mathsf{aux}\text{-}\mathcal{S}$ first updates T_Π by storing the hash values of its own code (with the inputs and the entries of the tables being hardwired), where the hash functions of all the existing sessions are used for computing these hash values, and each hash value is stored as the value to be committed in the ℓ-th commitment in Π-slots. (By requiring $\mathsf{aux}\text{-}\mathcal{S}$ to store its own code in T_Π in this way, we make sure that when $\mathsf{aux}\text{-}\mathcal{S}$ simulates a Π-slot, it commits to its own code in the Π-slot.) Then, $\mathsf{aux}\text{-}\mathcal{S}$ recursively executes itself q times in sequence with level $\ell - 1$; at the same time, $\mathsf{aux}\text{-}\mathcal{S}$ updates $\mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}$ at the end of each recursive execution in the following way.

- If a Π-slot (both the prover message and the verifier message) of a session is simulated by the recursive execution that has just been completed, $\mathsf{aux}\text{-}\mathcal{S}$ computes a second-round UA message about such a Π-slot and stores it in T_{UA}.

  (A machine that emulates this recursive execution must be committed in the $(\ell - 1)$-th commitment of such a Π-slot (this is because the recursively executed $\mathsf{aux}\text{-}\mathcal{S}$ must have stored its own code in T_Π at the beginning of its execution), and this machine can be used as a witness for generating a UA proof about this Π-slot.)

- If a UA-slot of a session is simulated by the recursive execution that has just been completed, and a second-round UA message for this session was stored in T_{UA} before this recursive execution, $\mathsf{aux}\text{-}\mathcal{S}$ computes a WIPOK witness for this session and stores it in T_{W}.

  (If such a UA-slot and a second-round UA message exist, that second-round UA message must be committed to in that UA-slot, and they can be used as a WIPOK witness.)

Finally, $\mathsf{aux}\text{-}\mathcal{S}$ outputs the q^ℓ messages that are simulated by these q recursive executions.

We remark that for technical reasons, the formal description of $\mathcal{S}$ below is a bit more complex.

- To avoid the circularity that arises when $\mathsf{aux}\text{-}\mathcal{S}$ uses it own code, we use a technique by Chung, Lin, and Pass [CLP13b]. Roughly speaking, $\mathsf{aux}\text{-}\mathcal{S}$ takes the code of a machine Π as input and uses this code rather than its own code; we then design $\mathcal{S}$ and $\mathsf{aux}\text{-}\mathcal{S}$ in such a way that when $\mathsf{aux}\text{-}\mathcal{S}$ is invoked, we always have $\Pi = \mathsf{aux}\text{-}\mathcal{S}$.

- To avoid the circularity issue about randomness that we sketched in Section 2.2, we use a technique of [CLP13a, CLP13b] that uses a forward-secure PRG $\mathsf{f\text{-}PRG}$. Roughly speaking, $\mathsf{aux}\text{-}\mathcal{S}$ takes a seed σ of $\mathsf{f\text{-}PRG}$ as input, computes a sequence of pseudorandomness $\rho_{q^\ell}, \ldots, \rho_2, \rho_1$ (notice that the indices are written in the reverse order), and simulates the prover messages in such a way that the i-th message in the transcript is simulated with randomness ρ_i.

Formal Description of $\mathcal{S}$

The input to $\mathcal{S}$ is (x, z), and the input to $\mathsf{aux}\text{-}\mathcal{S}$ is $(x, z, \ell, \Pi, \mathsf{trans}, \sigma)$ such that:

- $\ell \in \{0, \ldots, d\}$.
- Π is a code of a machine. (In what follows, we always have $\Pi = \mathsf{aux}\text{-}\mathcal{S}$.)
- $\mathsf{trans} \in \{0,1\}^{\mathsf{poly}(n)}$ is a prefix of a transcript between $V^*(x, z)$ and honest provers.
- $\sigma \in \{0,1\}^n$ is a seed of $\mathsf{f\text{-}PRG}$.

The auxiliary simulator $\mathsf{aux}\text{-}\mathcal{S}$ is also given oracle access to three tables $\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}$ such that:

- $\mathsf{T}_\Pi = \{v_{s,j}\}_{s\in[m], j\in[N_{\mathrm{col}}]}$ is a table of the hash values of some machines.
- $\mathsf{T}_{\mathsf{UA}} = \{\mathsf{ua}_{s,j}\}_{s\in[m], j\in[N_{\mathrm{col}}]}$ is a table of second-round UA messages.
- $\mathsf{T}_{\mathsf{W}} = \{w_s\}_{s\in[m]}$ is a table of WIPOK witnesses.

We allow $\mathsf{aux}\text{-}\mathcal{S}$ to read and update the entries in $\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}$ freely.

Simulator $\mathcal{S}(x, z)$:

1. Choose a random seed $\sigma_{q^d+1} \in \{0,1\}^n$ of $\mathsf{f\text{-}PRG}$. Initialize T_Π, T_{UA}, T_{W} by $v_{s,j} := 0^n$, $\mathsf{ua}_{s,j} := 0^n$, $w_s := \bot$ for every $s \in [m]$ and $j \in [N_{\mathrm{col}}]$.
2. Compute $\mathsf{trans} := \mathsf{aux}\text{-}\mathcal{S}^{\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}}(x, z, d, \mathsf{aux}\text{-}\mathcal{S}, \varepsilon, \sigma_{q^d+1})$, where ε is the empty string.
3. Output (x, z, trans).

Auxiliary Simulator $\mathsf{aux}\text{-}\mathcal{S}^{\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}}(x, z, \ell, \Pi, \mathsf{trans}, \sigma)$:

Step 1.
`/* Preparing randomness for simulating the next` q^ℓ `messages */`
Let $\kappa := |\mathsf{trans}|$ (i.e., κ be the number of the messages that are included in trans). Then, compute

$$(\sigma_{\kappa+q^\ell}, \ldots, \sigma_1, \rho_{\kappa+q^\ell}, \ldots, \rho_1) := \mathsf{f\text{-}PRG}(\sigma, \kappa + q^\ell) \ .$$

Step 2a: Simulation (base case). If $\ell = 0$, do the following.

1. If the next-scheduled message msg is a verifier message, feed trans to $V^*(x, z)$ and receive msg from V^*.

   If the next-scheduled message msg is a prover message of the s-th session ($s \in [m]$), do the following with randomness $\rho_{\kappa+1}$. (If necessary, $\rho_{\kappa+1}$ is expanded by a pseudorandom generator.) Let τ_s be the first-round message of Com of the s-th session, which can be found from trans.

- If msg is the prover message in a Π-slot, compute $\mathsf{msg} = (C_1, \dots, C_{N_{\mathrm{col}}})$ by $C_j \leftarrow \mathsf{Com}_{\tau_s}(v_{s,j})$ for every $j \in [N_{\mathrm{col}}]$, where $v_{s,j}$ is read from T_Π.
- If msg is the prover message in a UA-slot, compute $\mathsf{msg} = (D_1, \dots, D_{N_{\mathrm{col}}})$ by $D_j \leftarrow \mathsf{Com}_{\tau_s}(\mathsf{ua}_{s,j})$ for every $j \in [N_{\mathrm{col}}]$, where $\mathsf{ua}_{s,j}$ is read from T_{UA}.
- If msg is the first prover message of WIPOK, compute msg by using w_s as a witness, where w_s is read from T_{W}; if w_s is not a valid witness, abort with output stuck.
- If msg is the second prover message of WIPOK, reconstruct the prover state of WIPOK from w_s and $\rho_1, \dots, \rho_\kappa$ and then honestly compute msg by using this prover state, where w_s is read from T_{W}.[9]

2. Output msg.

Step 2b: Simulation (recursive case). If $\ell > 0$, do the following.

1. `/* Storing it own code in` T_Π `*/`
   Let $\mathsf{T}'_\Pi, \mathsf{T}'_{\mathsf{UA}}, \mathsf{T}'_{\mathsf{W}}$ be the tables that are obtained by copying the current entries of $\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}$. Let

   $$\Pi_{\mathrm{myself}}(\cdot) \stackrel{\mathrm{def}}{=} \Pi^{\mathsf{T}'_\Pi, \mathsf{T}'_{\mathsf{UA}}, \mathsf{T}'_{\mathsf{W}}}(x, z, \ell, \Pi, \mathsf{trans}, \cdot) .$$

   Then, for each $s \in [m]$, if the s-th session has already started in trans, do the following: Let h_s be the hash function chosen by V^* in the s-th session in trans; then, update T_Π by setting $v_{s,\ell} := h_s(\Pi_{\mathrm{myself}})$.

2. Set temporary variables $\mathsf{ctr}_s := 0$ and $\mathsf{tmp}_s := \perp$ for every $s \in [m]$, and a temporary variable $\mathsf{new\text{-}trans} := \varepsilon$. Define a function $\mathsf{head}(\cdot)$ as $\mathsf{head}(k) \stackrel{\mathrm{def}}{=} \kappa + 1 + (k-1)q^{\ell-1}$.

3. For each $k \in [q]$, do the following:

   (a) `/* Storing the machine that executes the` k`-th child-block. */`
   Let $\mathsf{T}^{(k)}_\Pi, \mathsf{T}^{(k)}_{\mathsf{UA}}, \mathsf{T}^{(k)}_{\mathsf{W}}$ be the tables that are obtained by copying the current entries of T_Π, T_{UA}, T_{W}. Then, let

   $$\Pi_k(\cdot) \stackrel{\mathrm{def}}{=} \Pi^{\mathsf{T}^{(k)}_\Pi, \mathsf{T}^{(k)}_{\mathsf{UA}}, \mathsf{T}^{(k)}_{\mathsf{W}}}(x, z, \ell - 1, \Pi, \mathsf{trans} \| \mathsf{new\text{-}trans}, \cdot) .$$

   (b) `/* Executing the` k`-th child-block. */`
   Compute

   $$\mathsf{trans}_k := \Pi^{\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}}(x, z, \ell - 1, \Pi, \mathsf{trans} \| \mathsf{new\text{-}trans}, \sigma_{\mathsf{head}(k+1)})$$

   while reading and updating $\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}$ for Π.

   (c) For each $s \in [m]$, if the s-th session has already started in $\mathsf{trans} \| \mathsf{new\text{-}trans}$, do the following. Let (h_s, τ_s) be the first-round message of the s-th session.

   Case 1. $\mathsf{ctr}_s = 0$, and trans_k contains a Π-slot of the s-th session.
   `/* Computing offline proof */`
   Let sl denote the Π-slot that is contained by trans_k. (If there are more than one such Π-slot, sl denote the first such one.) Let i_1 denote the *slot-index* of sl, i.e., $i_1 \in [N_{\mathrm{slot}}]$ such that sl is the i_1-th Π-slot in the s-th session. Let $(\overline{C}_{i_1}, r_{i_1})$ denote the messages in sl, where $\overline{C}_{i_1} = (C_{i_1,1}, \dots, C_{i_1,N_{\mathrm{col}}})$.

[9] From the construction of $\mathcal{S}$ and $\mathsf{aux}\text{-}\mathcal{S}$, the first prover message of WIPOK in trans must have been computed with witness w_s and randomness in $\rho_1, \dots, \rho_\kappa$.

i. From $\rho_{\mathsf{head}(k)}, \ldots, \rho_{\mathsf{head}(k+1)-1}$ (which are computed in Step 1), find the randomness R_1 that was used for generating $C_{i_1,\ell-1}$.[10] Then, compute a PCP proof π_s for statement $(h_s, \tau_s, C_{i_1,\ell-1}, r_{i_1}) \in \Lambda_2$ using $(\Pi_k, R_1, \sigma_{\mathsf{head}(k+1)})$ as a witness (see Section 4.1.2 for details). Then, compute $\mathsf{UA}_2 := h_s(\pi_s)$.

ii. Update T_{UA} by setting $\mathsf{ua}_{s,\ell-1} := \mathsf{UA}_2$.

iii. Update $\mathsf{tmp}_s := (i_1, \pi_s, \mathsf{UA}_2)$ and $\mathsf{ctr}_s := \mathsf{ctr}_s + 1$.

Case 2. $\mathsf{ctr}_s = 1$, and trans_k contains a UA-slot of the s-th session.

`/* Computing WIPOK witness */`

Let sl denote the UA-slot that is contained by trans_k. (If there are more than one such UA-slot, sl denote the first such one.) Let i_2 be the slot-index of sl. Let $(\overline{D}_{i_2}, \omega_{i_2})$ denote the messages in sl, where $\overline{D}_{i_2} = (D_{i_2,1}, \ldots, D_{i_2,N_{\mathrm{col}}})$.

i. Parse $(i_1, \pi_s, \mathsf{UA}_2) := \mathsf{tmp}_s$. Then, compute a fourth-round UA message UA_4 from the offline proof $(h_s, \pi_s, \mathsf{UA}_2)$ and the third-round UA message ω_{i_2}.

ii. From $\rho_{\mathsf{head}(k)}, \ldots, \rho_{\mathsf{head}(k+1)-1}$ (which are computed in Step 1), find the randomness R_2 that was used for generating $D_{i_2,\ell-1}$. Then, if $w_s = \perp$, update T_{W} by setting $w_s := (i_1, i_2, \ell - 1, \mathsf{UA}_2, \mathsf{UA}_4, R_2)$.

iii. Update $\mathsf{ctr}_s := \mathsf{ctr}_s + 1$.

Case 3. All the other cases.

Do nothing.

(d) Update $\mathsf{new\text{-}trans} := \mathsf{new\text{-}trans} \| \mathsf{trans}_k$.

4. Output $\mathsf{new\text{-}trans}$.

4.1.2 Correctness of $\mathcal{S}$.

In this section, we observe the correctness of $\mathcal{S}$. Specifically, we observe that $\mathsf{aux}\text{-}\mathcal{S}$ can indeed compute a valid PCP proof π_s and a valid WIPOK witness w_s in Step 2b.

First, we see that $\mathsf{aux}\text{-}\mathcal{S}$ can indeed compute a PCP proof π_s in Step 2b. Specifically, we see that when $\mathsf{aux}\text{-}\mathcal{S}$ computes π_s in Step 2b, $(\Pi_k, R_1, \sigma_{\mathsf{head}(k+1)})$ is indeed a witness for $(h_s, \tau_s, C_{i_1,\ell-1}, r_{i_1}) \in \Lambda_2$ (that is, $C_{i_1,\ell-1}$ is a commitment to $h_s(\Pi_k)$ and Π_k outputs a string that has r_{i_1} as a substring).

1. First, we observe that $C_{i_1,\ell-1}$ is a commitment to $h_s(\Pi_k)$. Recall that when $\mathsf{aux}\text{-}\mathcal{S}$ computes π_s in Step 2b, the Π-slot $(C_{i_1,\ell-1}, r_{i_1})$ is contained by trans_k. Then, since trans_k is generated by a recursive execution

$$\Pi^{\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}}(x, z, \ell - 1, \Pi, \mathsf{trans}, \sigma_{\mathsf{head}(k+1)}) \enspace ,$$

and this recursive execution updates $v_{s,\ell-1} \in \mathsf{T}_\Pi$ to be the hash of its own code at the beginning, $C_{i_1,\ell-1}$ is a commitment to the hash of this code, which is identical with $h_s(\Pi_k)$.

2. Next, we observe that Π_k outputs trans_k on input $\sigma_{\mathsf{head}(k+1)}$. This is because from its definition $\Pi_k(\sigma_{\mathsf{head}(k+1)})$ is identical with

$$\Pi^{\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}}(x, z, \ell - 1, \Pi, \mathsf{trans}, \sigma_{\mathsf{head}(k+1)})$$

(including the entries in the tables), which outputs trans_k.

[10]From the construction of $\mathsf{aux}\text{-}\mathcal{S}$, every message in the k-th child-block is computed by using randomness in $\rho_{\mathsf{head}(k)}, \ldots, \rho_{\mathsf{head}(k+1)-1}$.

Since r_{i_1} is contained by trans_k, we conclude that $(\Pi_k, R_1, \sigma_{\mathsf{head}(k+1)})$ is indeed a witness for $(h_s, \tau_s, C_{i_1,\ell-1}, r_{i_1}) \in \Lambda_2$.

Next, we see that $\mathsf{aux}\text{-}\mathcal{S}$ can compute a WIPOK proof in Step 2a as long as $w_s \neq \perp$. In other words, we see that if $\mathsf{aux}\text{-}\mathcal{S}$ updates w_s to $(i_1, i_2, \ell - 1, \mathsf{UA}_2, \mathsf{UA}_4, R_2)$ in Step 2b, it is indeed a valid WIPOK witness (that is, $D_{i_2,\ell-1}$ is a commitment to UA_2 and $(h_s, \mathsf{UA}_2, \omega_{i_2}, \mathsf{UA}_4)$ is an accepting UA proof for $(h_s, \tau_s, C_{i_1,\ell-1}, r_{i_1}) \in \Lambda_2$).

1. First, UA_2 is the hash value of a PCP proof π_s that is computed at the end of a previous recursive execution, and from what is observed above, π_s is a valid PCP proof for statement $(h_s, \tau_s, C_{i_1,\ell-1}, r_{i_1}) \in \Lambda_2$. Hence, $(h_s, \mathsf{UA}_2, \omega_{i_2}, \mathsf{UA}_4)$ is an accepting proof for $(h_s, \tau_s, C_{i_1,\ell-1}, r_{i_1}) \in \Lambda_2$

2. Next, since $\mathsf{ua}_{s,\ell-1} \in \mathsf{T}_{\mathsf{UA}}$ is not updated during level-$(\ell - 1)$ recursive executions, $D_{i_2,\ell-1}$ is a commitment to UA_2.

Thus, $(i_1, i_2, \ell - 1, \mathsf{UA}_2, \mathsf{UA}_4, R_2)$ is a valid WIPOK witness.

Finally, we see that we have $w_s \neq \perp$ when $\mathsf{aux}\text{-}\mathcal{S}$ computes a WIPOK proof in Step 2a. This follows from the following claim.

Claim 1. *During the execution of $\mathcal{S}$, any execution of* $\mathsf{aux}\text{-}\mathcal{S}$ *does not output* stuck.

Proof. We first introduce notations. Recall that an execution of $\mathcal{S}$ involves recursive executions of $\mathsf{aux}\text{-}\mathcal{S}$. We use *block* to denote each execution of $\mathsf{aux}\text{-}\mathcal{S}$. Notice that each block can be identified by the value of ℓ and $\kappa = |\mathsf{trans}|$. A block is in *level* ℓ if the corresponding $\mathsf{aux}\text{-}\mathcal{S}$ is executed with input ℓ. The *child-blocks* of a block are the blocks that are recursively executed by this block; thus, each block has q child-blocks. A block *contains* a slot of a session if the execution of $\mathsf{aux}\text{-}\mathcal{S}$ that corresponds to this block outputs a transcript that includes this slot (i.e., includes both the prover and the verifier message of this slot), where we use *slots* to refer to both Π-slots and UA-slots. A block is *good* w.r.t. a session if this block contains at least two slots of this session but contain neither the first verifier message of this session nor the first prover message of WIPOK of this session.[11]

Given these notations, we prove the claim as follows. From the constructions of $\mathcal{S}$ and $\mathsf{aux}\text{-}\mathcal{S}$, none of $\mathsf{aux}\text{-}\mathcal{S}$ outputs stuck if for every session that reaches Stage 3, there exists a block such that two of its child-blocks are good. (If there exists such a block for a session, the first good child-block contains a Π-slot of that session and the second one contains a UA-slot of that session; thus, a WIPOK witness for that session is computed at the end of the second good child-block.) Thus, it suffices to show that if a session reaches Stage 3, there exists a block such that at least two of its child-blocks are good w.r.t. that session. To show this, it suffices to show that if a session reaches Stage 3, there exists a block such that at least four of its child-blocks contain two or more slots of that session. (If four child-blocks contain two or more slots, two of them are good since at most one child-block contains the first verifier message and at most one child-block contains the first prover message of WIPOK.) Assume for contradiction that there exists a session s^* such that s^* reaches Stage 3 but every block has at most three child-blocks that contain two or more slots of session s^*. For $\ell \in \{0, \dots, d\}$ and $\kappa \in [q^d]$, let $\Gamma_\kappa(\ell)$ be the number of the slots that belong to session s^* and are contained by the block that is identified by ℓ and κ, and let $\Gamma(\ell) \stackrel{\text{def}}{=} \max_\kappa(\Gamma_\kappa(\ell))$. Then, since for each block b,

- at most three child-blocks of b contain two or more slots of s^*, and the other child-blocks contain at most a single slot of s^*, and

- s^* has at most $q - 1$ slots that are contained by block b but are not contained by its child-blocks,

[11]The definition of good blocks here is slightly different from that in the technical overview in Section 2.2.

we have

$$\Gamma(\ell) \leq 3 \cdot \Gamma(\ell - 1) + (q - 2) \cdot 1 + q - 1 = 3\Gamma(\ell - 1) + 2q - 3 \ .$$

Thus, we have

$$\begin{aligned}\Gamma(d) &\leq 3\Gamma(d-1) + 2q - 3 \\ &\leq 3^2\Gamma(d-2) + 3(2q-3) + 2q - 3 \\ &\leq \cdots \leq 3^d\Gamma(0) + \sum_{i=0}^{d-1} 3^i(2q-3) \\ &= 3^d\Gamma(0) + \frac{1}{2}(3^d - 1)(2q - 3) \ .\end{aligned}$$

From $d = O(1)$ and $\Gamma(0) = 0$, we have $\Gamma(d) = O(q)$. Then, since $\mathcal{S}$ outputs the view of V^* that is generated by the level-d block, there are at most $O(q) = O(n^{\epsilon/2})$ slots of s^* in the simulated transcript. Then, since we have $N_{\mathrm{slot}} = O(n^\epsilon)$, this contradicts to the assumption that s^* reaches Stage 3. □

From the above three observations, we conclude that aux-$\mathcal{S}$ can indeed compute a valid PCP proof π_s and a WIPOK valid witness w_s in Step 2b.

4.1.3 Running Time of $\mathcal{S}$

Lemma 3. *$\mathcal{S}(x, z)$ runs in polynomial time.*

Proof. We bound the running time of $\mathcal{S}$ as follows. Recall that an execution of $\mathcal{S}$ involves recursive executions of aux-$\mathcal{S}$. We identify each execution of aux-$\mathcal{S}$ by the value of ℓ and $\kappa = |\mathsf{trans}|$. In the following, we use aux-$\mathcal{S}_{\ell,\kappa}$ to denote the execution of aux-$\mathcal{S}$ with ℓ and κ. Let $t_{\ell,\kappa}$ be the running time of aux-$\mathcal{S}_{\ell,\kappa}$, and let $t_\ell \overset{\mathrm{def}}{=} \max_\kappa(t_{\ell,\kappa})$. Then, observe that in the execution of aux-$\mathcal{S}_{\ell,\kappa}$, every computation is performed in fixed polynomial time in n except for the following computations.

1. The recursive executions of aux-$\mathcal{S}$ (i.e., the executions of aux-$\mathcal{S}_{\ell-1,\kappa}$, aux-$\mathcal{S}_{\ell-1,\kappa+q^{\ell-1}}, \ldots$).
2. The generations of the offline proofs (i.e., PCP proofs and their hash values) and the fourth-round UA messages.

Each recursive execution takes at most $t_{\ell-1}$ steps. Furthermore, from the relatively efficient oracle construction property of PCP systems, each offline proof can be generated in $\mathsf{poly}(t_{\ell-1})$ steps. Then, since for each $k \in [q]$ there are a single recursive execution and at most m computations of offline proofs and fourth-round UA proofs, we have

$$t_\ell \leq q \cdot (t_{\ell-1} + m \cdot \mathsf{poly}(t_{\ell-1}) + \mathsf{poly}(n)) + \mathsf{poly}(n) \leq \mathsf{poly}(t_{\ell-1})$$

for any $\ell \in [d]$. Then, since we have $d = O(1)$ and $t_0 = \mathsf{poly}(n)$, we have $t_d = \mathsf{poly}(n)$. Thus, $\mathcal{S}$ runs in polynomial time. □

4.1.4 Indistinguishability of Views

Lemma 4. *The output of $\mathcal{S}(x, z)$ is computationally indistinguishable from the view of V^*.*

Proof. We prove this lemma by considering a sequence of hybrid experiments, $H_0, \ldots, H_{q^d+1}$. Hybrid H_0 is identical with the real interaction between V^* and honest provers, and H_{q^d+1} is identical with the execution of $\mathcal{S}$. Hybrid H_i ($i \in [q^d]$) is identical with H_{q^d+1} except that, roughly speaking, the simulation stops after simulating the i-th message, and later on the prover messages are generated honestly as in H_0. Formally, we define H_i ($i \in [q^d]$) by using the following hybrid auxiliary simulator $\mathsf{aux}\text{-}\tilde{\mathcal{S}}_i$, which differs from $\mathsf{aux}\text{-}\mathcal{S}$ in that it simulates the transcript only until the i-th message and that it simulates the i-th message using true randomness ρ (rather than pseudorandomness ρ_i derived by $\mathsf{f\text{-}PRG}$). Though $\mathsf{aux}\text{-}\tilde{\mathcal{S}}_i$ is very similar to $\mathsf{aux}\text{-}\mathcal{S}$, we give a complete description of $\mathsf{aux}\text{-}\tilde{\mathcal{S}}_i$ below. The differences from $\mathsf{aux}\text{-}\mathcal{S}$ are highlighted by blue color and underline.

Hybrid Auxiliary Simulator $\mathsf{aux}\text{-}\tilde{\mathcal{S}}_i^{\mathsf{T}_\Pi,\mathsf{T}_{\mathsf{UA}},\mathsf{T}_{\mathsf{W}}}(x, z, \ell, \Pi, \mathsf{trans}, \sigma, \underline{\tilde{\Pi}_i, \rho})$:

Step 1.
`/* Preparing randomness for simulating the next` q^ℓ `messages */`
Let $\kappa := |\mathsf{trans}|$ (i.e., κ be the number of the messages that are included in trans). Then, compute

$$\underline{(\sigma_{i-1}, \ldots, \sigma_1, \rho_{i-1}, \ldots, \rho_1) := \mathsf{f\text{-}PRG}(\sigma, i-1)}$$

<u>and let $\rho_i := \rho$.</u>

Step 2a: Simulation (base case). If $\ell = 0$, do the following.

1. If the next-scheduled message msg is a verifier message, feed trans to $V^*(x, z)$ and receive msg from V^*.

   If the next-scheduled message msg is a prover message of the s-th session ($s \in [m]$), do the following with randomness $\rho_{\kappa+1}$. (If necessary, $\rho_{\kappa+1}$ is expanded by a pseudorandom generator.) Let τ_s be the first-round message of Com of the s-th session, which can be found from trans.

   - If msg is the prover message in a Π-slot, compute $\mathsf{msg} = (C_1, \ldots, C_{N_{\mathrm{col}}})$ by $C_j \leftarrow \mathsf{Com}_{\tau_s}(v_{s,j})$ for every $j \in [N_{\mathrm{col}}]$, where $v_{s,j}$ is read from T_Π.
   - If msg is the prover message in a UA-slot, compute $\mathsf{msg} = (D_1, \ldots, D_{N_{\mathrm{col}}})$ by $D_j \leftarrow \mathsf{Com}_{\tau_s}(\mathsf{ua}_{s,j})$ for every $j \in [N_{\mathrm{col}}]$, where $\mathsf{ua}_{s,j}$ is read from T_{UA}.
   - If msg is the first prover message of WIPOK, compute msg by using w_s as a witness, where w_s is read from T_{W}; if w_s is not a valid witness, abort with output stuck.
   - If msg is the second prover message of WIPOK, reconstruct the prover state of WIPOK from w_s and $\rho_1, \ldots, \rho_\kappa$ and then honestly compute msg by using this prover state, where w_s is read from T_{W}.[12]

2. Output msg.

Step 2b: Simulation (recursive case). If $\ell > 0$, do the following.

1. `/* Storing it own code in` T_Π `*/`
   Let T'_Π, $\mathsf{T}'_{\mathsf{UA}}$, T'_{W} be the tables that are obtained by copying the current entries of T_Π, T_{UA}, T_{W}. Let

$$\Pi_{\mathrm{myself}}(\cdot) \overset{\mathrm{def}}{=} \Pi^{\mathsf{T}'_\Pi,\mathsf{T}'_{\mathsf{UA}},\mathsf{T}'_{\mathsf{W}}}(x, z, \ell, \Pi, \mathsf{trans}, \cdot) \ .$$

[12]From the construction of $\mathcal{S}$ and $\mathsf{aux}\text{-}\mathcal{S}$, the first prover message of WIPOK in trans must have been computed with witness w_s and randomness in $\rho_1, \ldots, \rho_\kappa$.

Then, for each $s \in [m]$, if the s-th session has already started in trans, do the following: Let h_s be the hash function chosen by V^* in the s-th session in trans; then, update T_Π by setting $v_{s,\ell} := h_s(\Pi_{\mathrm{myself}})$.

2. Set temporary variables $\mathsf{ctr}_s := 0$ and $\mathsf{tmp}_s := \perp$ for every $s \in [m]$, and a temporary variable $\mathsf{new\text{-}trans} := \varepsilon$. Define a function $\mathsf{head}(\cdot)$ as $\mathsf{head}(k) \overset{\mathrm{def}}{=} \kappa + 1 + (k-1)q^{\ell-1}$.

3. For each $k \in [q]$ such that $\mathsf{head}(k) + q^{\ell-1} < i$, do the following:

   (a) `/* Storing the machine that executes the` k`-th child-block. */`
   Let $\mathsf{T}_\Pi^{(k)}$, $\mathsf{T}_{\mathsf{UA}}^{(k)}$, $\mathsf{T}_{\mathsf{W}}^{(k)}$ be the tables that are obtained by copying the current entries of T_Π, T_{UA}, T_{W}. Then, let

   $$\Pi_k(\cdot) \overset{\mathrm{def}}{=} \Pi^{\mathsf{T}_\Pi^{(k)},\mathsf{T}_{\mathsf{UA}}^{(k)},\mathsf{T}_{\mathsf{W}}^{(k)}}(x, z, \ell-1, \Pi, \mathsf{trans} \| \mathsf{new\text{-}trans}, \cdot) \ .$$

   (b) `/* Executing the` k`-th child-block. */`
   Compute

   $$\mathsf{trans}_k := \Pi^{\mathsf{T}_\Pi,\mathsf{T}_{\mathsf{UA}},\mathsf{T}_{\mathsf{W}}}(x, z, \ell-1, \Pi, \mathsf{trans} \| \mathsf{new\text{-}trans}, \sigma_{\mathsf{head}(k+1)})$$

   while reading and updating $\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}$ for Π.

   (c) For each $s \in [m]$, if the s-th session has already started in $\mathsf{trans} \| \mathsf{new\text{-}trans}$, do the following. Let (h_s, τ_s) be the first-round message of the s-th session.

   Case 1. $\mathsf{ctr}_s = 0$, and trans_k contains a Π-slot of the s-th session.
   `/* Computing offline proof */`
   Let sl denote the Π-slot that is contained by trans_k. (If there are more than one such Π-slot, sl denote the first such one.) Let i_1 denote the *slot-index* of sl, i.e., $i_1 \in [N_{\mathrm{slot}}]$ such that sl is the i_1-th Π-slot in the s-th session. Let $(\overline{C}_{i_1}, r_{i_1})$ denote the messages in sl, where $\overline{C}_{i_1} = (C_{i_1,1}, \ldots, C_{i_1,N_{\mathrm{col}}})$.

      i. From $\rho_{\mathsf{head}(k)}, \ldots, \rho_{\mathsf{head}(k+1)-1}$ (which are computed in Step 1), find the randomness R_1 that was used for generating $C_{i_1,\ell-1}$.[13] Then, compute a PCP proof π_s for statement $(h_s, \tau_s, C_{i_1,\ell-1}, r_{i_1}) \in \Lambda_2$ using $(\Pi_k, R_1, \sigma_{\mathsf{head}(k+1)})$ as a witness (see Section 4.1.2 for details). Then, compute $\mathsf{UA}_2 := h_s(\pi_s)$.

      ii. Update T_{UA} by setting $\mathsf{ua}_{s,\ell-1} := \mathsf{UA}_2$.

      iii. Update $\mathsf{tmp}_s := (i_1, \pi_s, \mathsf{UA}_2)$ and $\mathsf{ctr}_s := \mathsf{ctr}_s + 1$.

   Case 2. $\mathsf{ctr}_s = 1$, and trans_k contains a UA-slot of the s-th session.
   `/* Computing WIPOK witness */`
   Let sl denote the UA-slot that is contained by trans_k. (If there are more than one such UA-slot, sl denote the first such one.) Let i_2 be the slot-index of sl. Let $(\overline{D}_{i_2}, \omega_{i_2})$ denote the messages in sl, where $\overline{D}_{i_2} = (D_{i_2,1}, \ldots, D_{i_2,N_{\mathrm{col}}})$.

      i. Parse $(i_1, \pi_s, \mathsf{UA}_2) := \mathsf{tmp}_s$. Then, compute a fourth-round UA message UA_4 from the offline proof $(h_s, \pi_s, \mathsf{UA}_2)$ and the third-round UA message ω_{i_2}.

      ii. From $\rho_{\mathsf{head}(k)}, \ldots, \rho_{\mathsf{head}(k+1)-1}$ (which are computed in Step 1), find the randomness R_2 that was used for generating $D_{i_2,\ell-1}$. Then, if $w_s = \perp$, update T_{W} by setting $w_s := (i_1, i_2, \ell-1, \mathsf{UA}_2, \mathsf{UA}_4, R_2)$.

      iii. Update $\mathsf{ctr}_s := \mathsf{ctr}_s + 1$.

[13]From the construction of $\mathsf{aux}\text{-}\mathcal{S}$, every message in the k-th child-block is computed by using randomness in $\rho_{\mathsf{head}(k)}, \ldots, \rho_{\mathsf{head}(k+1)-1}$.

Case 3. All the other cases.
Do nothing.

(d) Update $\mathsf{new\text{-}trans} := \mathsf{new\text{-}trans} \| \mathsf{trans}_k$.

4. For $k \in [q]$ such that $\mathsf{head}(k) \le i < \mathsf{head}(k+1)$, do the following.

   (a) Compute

$$\mathsf{trans}_k := \tilde{\Pi}_i^{\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}}(x, z, \ell - 1, \Pi, \mathsf{trans} \| \mathsf{new\text{-}trans}, \sigma_{\mathsf{head}(k+1)}, \tilde{\Pi}_i, \rho)$$

   while reading and updating $\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}$ for $\tilde{\Pi}_i$.

   (b) Update $\mathsf{new\text{-}trans} := \mathsf{new\text{-}trans} \| \mathsf{trans}_k$.

5. Output $\mathsf{new\text{-}trans}$.

Now, we formally define the hybrids as follows.

Hybrids $H_0, \ldots, H_{q^d+1}$:

Hybrid H_0 is the same as the real interaction between V^* and honest provers.

Hybrid H_i ($i \in [q^d]$) is the same as the real execution of $\mathcal{S}$ except for the following.

1. $\mathcal{S}$ obtains trans by executing

$$\mathsf{aux}\text{-}\tilde{\mathcal{S}}_i^{\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}}(x, z, d, \mathsf{aux}\text{-}\mathcal{S}, \varepsilon, \sigma_i, \mathsf{aux}\text{-}\tilde{\mathcal{S}}_i, \rho_i)$$

   rather than

$$\mathsf{aux}\text{-}\mathcal{S}^{\mathsf{T}_\Pi, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}}(x, z, d, \mathsf{aux}\text{-}\mathcal{S}, \varepsilon, \sigma_{q^d+1}) \ ,$$

   where $(\sigma_i, \rho_i) := \mathsf{f\text{-}PRG}(\sigma_{i+1}, 1)$ for a randomly chosen seed σ_{i+1} of $\mathsf{f\text{-}PRG}$. We remark that in trans, the view of V^* is simulated up until the i-th message (inclusive) in a way that the i-th message is simulated by using ρ_i as randomness.

2. After trans, the simulation of V^*'s view is continued as follows:
   - Every message is computed with true randomness.
   - Every message in Π-slots and UA-slot is generated by committing to 0^n.
   - Every WIPOK that starts after trans is executed with a witness for $x \in L$.
   - Every WIPOK that already started in trans is executed as in $\mathsf{aux}\text{-}\mathcal{S}$ (i.e., by reconstructing the prover state).

Hybrid H_{q^d+1} is the same as the real execution of $\mathcal{S}$.

From a hybrid argument, it suffices to show the indistinguishability between the outputs of each neighboring hybrids. In the following, we show the indistinguishability in the reverse order, i.e., we show that the output of H_i is indistinguishable from that of H_{i-1} for every $i \in [q^d + 1]$.

Claim 2. *The output of H_{q^d+1} and that of H_{q^d} are identically distributed.*

Proof. This claim can be proven by inspection. Notice that the only difference between H_{q^d+1} and H_{q^d} is that in H_{q^d+1}, trans is obtained by executing

$$\text{aux-}\mathcal{S}^{\mathsf{T}_\Pi,\mathsf{T}_{\mathsf{UA}},\mathsf{T}_\mathsf{W}}(x, z, d, \text{aux-}\mathcal{S}, \varepsilon, \sigma_{q^d+1})$$

whereas in H_{q^d}, trans is obtained by executing

$$\text{aux-}\tilde{\mathcal{S}}_{q^d}^{\mathsf{T}_\Pi,\mathsf{T}_{\mathsf{UA}},\mathsf{T}_\mathsf{W}}(x, z, d, \text{aux-}\mathcal{S}, \varepsilon, \sigma_{q^d}, \text{aux-}\tilde{\mathcal{S}}, \rho_{q^d})$$

such that $(\sigma_{q^d}, \rho_{q^d}) := \text{f-PRG}(\sigma_{q^d+1}, 1)$. The former execution simulates the q^d messages in trans using the randomness $\rho_1, \ldots, \rho_{q^d}$ that are obtained by $\text{f-PRG}(\sigma_{q^d+1}, q^d)$, whereas the latter execution simulates the first $q^d - 1$ messages using the randomness $\rho_1, \ldots, \rho_{q^d-1}$ that are obtained by $\text{f-PRG}(\sigma_{q^d}, q^d - 1)$ and then simulates the q^d-th message using the randomness ρ_{q^d}. Then, since $(\sigma_{q^d}, \rho_{q^d}) = \text{f-PRG}(\sigma_{q^d+1}, 1)$ in H_{q^d}, it follows from the consistency of f-PRG that the messages in the latter execution are simulated using the randomness $\rho_1, \ldots, \rho_{q^d}$ that are obtained by $\text{f-PRG}(\sigma_{q^d+1}, q^d)$. Hence, the messages in the latter execution are generated identically with those in the former execution. □

Claim 3. *The output of H_i and that of H_{i-1} are computationally indistinguishable for every $i \in [q^d]$.*

Proof. To prove this claim, we consider a sequence of intermediate hybrids in which H_i is gradually changed to H_{i-1} as follows.

Hybrid $H_{i:1}$ is the same as H_i except that $\mathcal{S}$ obtains trans by executing

$$\text{aux-}\tilde{\mathcal{S}}_i^{\mathsf{T}_\Pi,\mathsf{T}_{\mathsf{UA}},\mathsf{T}_\mathsf{W}}(x, z, d, \text{aux-}\mathcal{S}, \varepsilon, \sigma_i, \text{aux-}\tilde{\mathcal{S}}_i, \rho_i)$$

for random σ_i and ρ_i rather than for $(\sigma_i, \rho_i) := \text{f-PRG}(\sigma_{i+1}, 1)$.

Notice that in $H_{i:1}$, the i-th message is simulated with true randomness rather than pseudorandomness.

Hybrid $H_{i:2}$ is the same as $H_{i:1}$ except that if the i-th message is a prover message of Com (either in a Π-slot or in a UA-slot), then the message is computed as in the honest prover (i.e., $C_j \leftarrow \mathsf{Com}(0^n)$ or $D_j \leftarrow \mathsf{Com}(0^n)$ for every $j \in [N_{\mathrm{col}}]$).

Hybrid $H_{i:3}$ is the same as $H_{i:2}$ except that if the i-th message is the first prover message of WIPOK, then subsequently all messages in this WIPOK are computed by using a witness for $x \in L$.

From a hybrid argument, it suffices to show the indistinguishability between each neighboring intermediate hybrids.

Claim 4. *For every $i \in [q^d]$, the output of $H_{i:1}$ is computationally indistinguishable from that of H_i.*

Proof. The indistinguishability follows immediately from the forward security of f-PRG. Assume for contradiction that the output of H_i and that of $H_{i:1}$ are distinguishable. Then, consider the following adversary $\mathcal{D}$ against the forward security of f-PRG. On input (σ'_i, ρ'_i), adversary $\mathcal{D}$ internally invokes V^* and simulates H_i for V^* honestly except for the following.

- Rather than executing

$$\text{aux-}\tilde{\mathcal{S}}_i^{\mathsf{T}_\Pi,\mathsf{T}_{\mathsf{UA}},\mathsf{T}_\mathsf{W}}(x, z, d, \text{aux-}\mathcal{S}, \varepsilon, \sigma_i, \text{aux-}\tilde{\mathcal{S}}_i, \rho_i)$$

for $(\sigma_i, \rho_i) := \text{f-PRG}(\sigma_{i+1}, 1)$, $\mathcal{D}$ executes it for $\sigma_i := \sigma'_i$ and $\rho_i := \rho'_i$.

When σ'_i and ρ'_i are generated by $(\sigma'_i, \rho'_i) := \mathsf{f\text{-}PRG}(\sigma_{i+1}, 1)$, the output of $\mathcal{D}$ is identically distributed with that of H_i, and when σ'_i and ρ'_i are chosen randomly, the output of $\mathcal{D}$ is identically distributed with that of $H_{i:1}$. Therefore, $\mathcal{D}$ breaks the forward security of f-PRG from the assumption, and thus we reach a contradiction.

□

Claim 5. *For every $i \in [q^d]$, the output of $H_{i:2}$ is computationally indistinguishable from that of $H_{i:1}$.*

Proof. It suffices to consider the case that the i-th message msg is a prover message of Com. Note that both in $H_{i:1}$ and $H_{i:2}$, the Com commitments in msg are generated by using true randomness; furthermore, the information about their committed values and randomness are not used in the other messages (e.g., the randomness is not hardwired in the committed machines, and the committed value and the randomness are not used as a witness in the generations of PCP and WIPOK). Thus, the indistinguishability follows from the hiding property of Com. □

Claim 6. *For every $i \in [q^d]$, the output of $H_{i:3}$ is computationally indistinguishable from that of $H_{i:2}$.*

Proof. It suffices to consider the case that the i-th message msg is the first prover message of WIPOK. Note that both in $H_{i:2}$ and in $H_{i:3}$, this WIPOK proof is generated with true randomness that is not used anywhere else; furthermore, from Claim 1, a valid witness is used both in $H_{i:2}$ and in $H_{i:3}$. Thus, the indistinguishability follows from the witness indistinguishability of WIPOK. □

Claim 7. *For every $i \in [q^d]$, the output of H_{i-1} is identically distributed with that of $H_{i:3}$.*

Proof. From the consistency of f-PRG, the first $(i-1)$ messages are computed both in $H_{i:3}$ and in H_{i-1} by using the pseudorandomness that is generated by $\mathsf{f\text{-}PRG}(\sigma_i, i-1)$ for random σ_i. In addition, the i-th message msg is computed in exactly the same way in $H_{i:3}$ and H_{i-1}. Thus, the claim follows. □

From Claims 4, 5, 6, and 7, the output of H_i and that of H_{i-1} are computationally indistinguishable. This completes the proof of Claim 3. □

From Claims 2 and 3, the output of H_0 and that of H_{q^d+1} are indistinguishable. This completes the proof of Lemma 4. □

This completes the proof of Lemma 1. □

4.2 Argument of Knowledge Property

As noted in Remark 2, the language Λ_2 shown in Figure 8 is slightly over-simplified, and we can prove the argument-of-knowledge property of cZKAOK only when $\mathcal{H}$ is collision resistant against $\mathsf{poly}(n^{\log\log n})$-time adversaries.

Below, we prove the argument-of-knowledge property assuming that $\mathcal{H}$ is collision resistant against $\mathsf{poly}(n^{\log\log n})$-time adversaries. By using a trick shown in [BG09], we can extend this proof so that it works even under the assumption that $\mathcal{H}$ is collision resistant only against polynomial-time adversaries. The details are given at the end of this section.

Proof of Lemma 2 (when $\mathcal{H}$ is collision resistant against $\mathsf{poly}(n^{\log\log n})$-time adversaries). For any cheating prover P^*, let us consider the following extractor E.

- Given oracle access to P^*, the extractor E emulates a verifier of cZKAOK for P^* honestly until the beginning of Stage 3. E then extract a witness from WIPOK using its extractor.

To show that E outputs a witness for $x \in L$, it suffices to show that the extracted witness is a witness for $(h, \tau, \overline{C}_1, r_1, \overline{D}_1, \omega_1, \ldots, \overline{C}_{N_{\mathrm{slot}}}, r_{N_{\mathrm{slot}}}, \overline{D}_{N_{\mathrm{slot}}}, \omega_{N_{\mathrm{slot}}}) \in \Lambda_1$ only with negligible probability. In the following, we call a witness for $(h, \tau, \overline{C}_1, r_1, \overline{D}_1, \omega_1, \ldots, \overline{C}_{N_{\mathrm{slot}}}, r_{N_{\mathrm{slot}}}, \overline{D}_{N_{\mathrm{slot}}}, \omega_{N_{\mathrm{slot}}}) \in \Lambda_1$ a *fake witness*, and we say that P^* is *bad* if E outputs a fake witness with non-negligible probability. Below, we show that if there exists a bad cheating prover, we can break the collision resistance of $\mathcal{H}$.

We first show the following claim, which roughly states that if there exists a bad P^*, there also exists a prover P^{**} that can prove a statement in Λ_2 with non-negligible probability.

Claim 8. *For any ITM P, let us consider an experiment* $\mathsf{Exp}_1(n, P)$ *in which P interacts with a verifier V as follows.*

1. ***Interactively generating statement.*** *V sends a random $h \in \mathcal{H}_n$ and $\tau \in \{0,1\}^{3n}$ to P. Then, P sends a commitment C of* Com *to V, and V sends a random $r \in \{0,1\}^{n^2}$ to P.*

2. ***Generating*** UA ***proof.*** *P sends a second-round* UA *message* UA_2 *of statement $(h, \tau, C, r) \in \Lambda_2$ to V. Then, V sends a third-round* UA *message ω to P, and P sends a fourth-round* UA *message* UA_4 *to V.*

3. *We say that P wins in the experiment if $(h, \mathsf{UA}_2, \omega, \mathsf{UA}_4)$ is an accepting* UA *proof for $(h, \tau, C, r) \in \Lambda_2$.*

Then, if there exists a bad P^*, there exists a PPT *ITM P^{**} that wins in* $\mathsf{Exp}_1(n, P^{**})$ *with non-negligible probability.*

Proof. From the assumption that P^* is bad, for infinitely many n we can extract a fake witness from P^* with probability at least $\delta(n) \stackrel{\text{def}}{=} 1/\mathsf{poly}(n)$. In the following, we fix any such n. From an average argument, there exist $i_1^*, i_2^* \in [N_{\mathrm{slot}}]$ and $j^* \in [N_{\mathrm{col}}]$ such that with probability at least $\delta'(n) \stackrel{\text{def}}{=} \delta(n)/N_{\mathrm{col}} N_{\mathrm{slot}}^2 > \delta(n)/n^3$, we can extract a fake witness $(i_1, i_2, j, \ldots)$ such that $(i_1, i_2, j) = (i_1^*, i_2^*, j^*)$. Then, we consider the following cheating prover P^{**} against Exp_1.

1. P^{**} internally invokes P^* and emulates a verifier of cZKAOK for P^* honestly with the following differences:
   - In Stage 1, P^{**} forwards h and τ from the external V to the internal P^*.
   - In the i_1^*-th Π-slot of Stage 2, P^{**} forwards $C_{i_1^*, j^*}$ from the internal P^* to the external V and forwards r from V to P^*.
   - In Stage 3, P^{**} extracts a witness w from P^* by using the extractor of WIPOK.

2. If w is not a fake witness of the form $(i_1^*, i_2^*, j^*, \ldots)$, P^{**} aborts with output fail. Otherwise, parse $(i_1^*, i_2^*, j^*, \mathsf{UA}_2, \mathsf{UA}_4, R) := w$. Then, P^{**} sends UA_2 to the external V and receives ω.

3. P^{**} rewinds the internal P^* to the point just after P^* sent the Com commitments in the i_2^*-th UA-slot. Then, P^{**} sends ω to P^* as the verifier message of the i_2^*-th UA-slot, interacts with P^* again as an honest verifier, and then extracts a witness w' in Stage 3.

4. If w' is not a fake witness of the form $(i_1^*, i_2^*, j^*, \ldots)$, P^{**} aborts with output fail. Otherwise, parse $(i_1^*, i_2^*, j^*, \mathsf{UA}_2', \mathsf{UA}_4', R') := w'$. Then, P^{**} sends UA_4' to the external V.

To analyze the probability that P^{**} wins in $\mathsf{Exp}_1(n, P^{**})$, we first observe that the transcript of cZKAOK that is internally emulated by P^{**} is "good" with probability at least $\delta'/2$. Formally, let trans be the prefix of a transcript of cZKAOK up until the prover message of the i_2^*-th UA-slot (inclusive). Then, we say that trans is *good* if under the condition that trans is a prefix of the transcript, a fake witness of the form $(i_1^*, i_2^*, j^*, \ldots)$ is extracted from P^* with probability at least $\delta'/2$. From an average argument, the prefix of the transcript is good with probability at least $\delta'/2$ when P^* interacts with an honest verifier of cZKAOK. Then, since a transcript of cZKAOK is perfectly emulated in Step 1 of P^{**}, the prefix of the internally emulated transcript is good with probability at least $\delta'/2$.

We next observe that under the condition that the prefix of the internally emulated transcript is good in Step 1 of P^{**}, P^{**} wins in $\mathsf{Exp}_1(n, P^{**})$ with probability at least $(\delta'/2)^2 - \mathsf{negl}(n)$. First, from the definition of a good prefix, it follows that under the condition that the prefix of the internally emulated transcript is good in Step 1 of P^{**}, the probability that both w and w' are fake witnesses of the form $(i_1^*, i_2^*, j^*, \ldots)$ is at least $(\delta'/2)^2$. Next, if w and w' are fake witnesses, both UA_2 and UA_2' are the committed values of $D_{i_2^*, j^*}$, so $\mathsf{UA}_2 = \mathsf{UA}_2'$ holds except with negligible probability; this means that if w and w' are fake witnesses, $(h, \mathsf{UA}_2, \omega, \mathsf{UA}_4')$ is an accepting UA proof except with negligible probability. Hence, under the aforementioned condition, P^{**} wins in $\mathsf{Exp}_1(n, P^{**})$ with probability at least $(\delta'/2)^2 - \mathsf{negl}(n)$.

By combining the above two observations, we conclude that the probability that P^{**} wins in $\mathsf{Exp}_1(n, P^{**})$ is at least

$$\frac{\delta'}{2}\left(\left(\frac{\delta'}{2}\right)^2 - \mathsf{negl}(n)\right) \geq \frac{1}{\mathsf{poly}(n)} \ .$$

□

Next, we show the following claim, which roughly states that if there exists P^{**} that proves a statement in Λ_2 with non-negligible probability, then we can extract a valid witness from P^{**} with non-negligible probability.

Claim 9. *For any ITM E^*, let us consider an experiment $\mathsf{Exp}_2(n, E^*)$ in which E^* interacts with a verifier V as follows.*

1. ***Interactively generating statement.*** *This step is the same as the one in Exp_1, where E^* plays as P. Let (h, τ, C, r) be the interactively generated statement.*

2. ***Outputting witness.*** *E outputs $w = (\Pi, R, y)$. We say that E wins in the experiment if w is a valid witness for $(h, \tau, C, r) \in \Lambda_2$.*

Then, if there exists a PPT *ITM P^{**} that wins in $\mathsf{Exp}_1(n, P^{**})$ with non-negligible probability, there exists a $\mathsf{poly}(n^{\log\log n})$-time ITM E^* that wins in $\mathsf{Exp}_2(n, E^*)$ with non-negligible probability.*

Proof. We first observe that UA satisfies weak/global proof-of-knowledge property even when the statement is generated after the hash function h is chosen, i.e., even when the first-round message of UA is sent before the statement is generated. Roughly speaking, the UA extractor by [BG09] extracts a witness by combining the extractor of the underlying PCP system with an oracle-recovery procedure that (implicitly) recovers a PCP proof for the extractor of the PCP system. A nice property of the UA extractor by [BG09] is that it invokes the oracle-recovery procedure on input a random hash function h that is chosen independently of the statement. Because of this property, the UA extractor can be modified straightforwardly so that it works even when h is chosen before the statement.

We then obtain E^* by simply using the global UA extractor for P^{**}. Since the running time of the global UA extractor is $\mathsf{poly}(n^{\log\log n})$, the running time of E^* is also $\mathsf{poly}(n^{\log\log n})$. □

Finally, we reach a contradiction by showing that given E^* described in Claim 9, we can break the collision resistance of $\mathcal{H}$.

Claim 10. *If there exists a $\mathsf{poly}(n^{\log\log n})$-time ITM E^* that wins in $\mathsf{Exp}_2(n, E^*)$ with non-negligible probability, there exists a $\mathsf{poly}(n^{\log\log n})$-time machine $\mathcal{A}$ that breaks the collision resistance of $\mathcal{H}$.*

Proof. We consider the following $\mathcal{A}$.

1. Given $h \in \mathcal{H}$, $\mathcal{A}$ internally invokes E^* and emulates $\mathsf{Exp}_2(n, E^*)$ for E^* perfectly except that $\mathcal{A}$ forwards h to E^* in Step 1. Let (h, τ, C, r) and w be the statement and the output of E^* in this emulated experiment.

2. If w is not a valid witness for $(h, \tau, C, r) \in \Lambda_2$, $\mathcal{A}$ aborts with output fail. Otherwise, let $(\Pi, R, y) := w$.

3. $\mathcal{A}$ rewinds E^* to the point just after E^* sent C, and from this point $\mathcal{A}$ emulates $\mathsf{Exp}_2(n, E^*)$ again with fresh randomness. Let (h, τ, C, r') be the statement and w' be the witness in this emulated experiment.

4. If w' is not a valid witness for $(h, C, r') \in \Lambda_2$, $\mathcal{A}$ aborts with output fail. Otherwise, let $(\Pi', R', y') := w'$.

5. $\mathcal{A}$ outputs (Π, Π') if $\Pi \neq \Pi'$ and $h(\Pi) = h(\Pi')$. Otherwise, $\mathcal{A}$ outputs fail.

First, we show that both w and w' are valid witnesses with non-negligible probability. From the assumption that E^* wins in $\mathsf{Exp}_2(n, E^*)$ with non-negligible probability, E^* outputs a valid witness in $\mathsf{Exp}_2(n, E^*)$ with probability $\epsilon \overset{\text{def}}{=} 1/\mathsf{poly}(n)$ for infinitely many n. In the following, we fix any such n. Let trans be the prefix of a transcript of $\mathsf{Exp}_2(n, E^*)$ up until E^* sends C (inclusive). We say that trans is *good* if under the condition that trans is a prefix of the transcript, E^* outputs a valid witness with probability at least $\epsilon/2$. From an average argument, the prefix of the internally emulated transcript is good with probability at least $\epsilon/2$. Thus, the probability that both w and w' are valid witnesses is at least $(\epsilon/2)(\epsilon/2)^2 = (\epsilon/2)^3$.

Next, we show that when $\mathcal{A}$ obtains two valid witnesses $w = (\Pi, R, y)$ and $w' = (\Pi', R', y')$, we have $\Pi \neq \Pi'$ and $h(\Pi) = h(\Pi')$ except with negligible probability. First, from the binding property of Com, we have $h(\Pi) = h(\Pi')$ except with negligible probability. (Recall that from the condition that w and w' are valid witnesses, we have $\mathsf{Com}_\tau(h(\Pi); R) = \mathsf{Com}_\tau(h(\Pi'); R') = C$.) Next, since r' is chosen randomly after Π is determined, and since we have

$$\begin{aligned} &\left|\left\{r'' \in \{0,1\}^{n^2} \mid \exists y \in \{0,1\}^n \text{ s.t. } r'' \text{ is a substring of the output of } \Pi(y)\right\}\right| \\ &\leq n^{\log\log n} \cdot 2^n \leq 2^{2n} \quad , \end{aligned}$$

the probability that there exists $y'' \in \{0,1\}^n$ such that r' is a substring of the output of $\Pi(y'')$ is at most $2^{2n}/2^{n^2} = \mathsf{negl}(n)$. Then, since r' is a substring of the output of $\Pi'(y')$, we conclude that we have $\Pi \neq \Pi'$ except with negligible probability.

From the above two observations, we conclude that $\mathcal{A}$ finds a collision of $\mathcal{H}$ with non-negligible probability. □

From Claims 8, 9, and 10, it follows that there exists no bad P^*. Thus, the extractor E outputs a witness for $x \in L$ except with negligible probability. This concludes the proof of Lemma 2. □

On the proof of Lemma 2 when $\mathcal{H}$ is secure only against poly-time adversaries.

As noted before, we can use a trick by [BG09] to extend the above proof so that it works even when $\mathcal{H}$ is secure only against polynomial-time adversaries. Recall that in the above proof, $\mathcal{H}$ need to be secure against super-polynomial-time adversaries because a collision of $\mathcal{H}$ (i.e., the pair of Π and Π') is found by using the global argument-of-knowledge property of UA. Hence, the overall strategy is to modify the protocol and the proof so that the weak argument-of-knowledge property can be used instead of the global one.

Roughly speaking, the trick by [BG09] works as follows. Recall that, as noted in Section 3.2, $\mathcal{H}$ is a hash function family that is obtained by applying Merkle's tree-hashing technique on any length-halving collision-resistant hash function family. From the properties of the tree-hashing, it follows that for any $h \in \mathcal{H}_n$ and $x = (x_1, \ldots, x_{|x|}) \in \{0,1\}^{\le n^{\log\log n}}$, we can compute short certificates $\mathsf{auth}(x) = \{\mathsf{auth}_i(x)\}_{i \in [|x|]}$ such that given $h(x)$, x_i, and $\mathsf{auth}_i(x)$, one can verify in time polynomial in n that the i-th bit of x is indeed x_i. Furthermore, for any collision (x, x') of $\mathcal{H}$, a collision of the underlying hash function can be found in polynomial time from any pairs of a bit and a certificate $(x_i, \mathsf{auth}_i(x))$ and $(x'_i, \mathsf{auth}_i(x'))$ such that $x_i \neq x'_i$. Then, the idea of the trick by [BG09] is, instead of finding a collision of $\mathcal{H}$ by extracting the whole of Π and Π', finding a collision of the underlying hash function by extracting Π and Π' in a single bit position along with their certificates. Specifically, in the trick by [BG09], the language Λ_2 is first modified in such a way that a witness includes the certificates of the committed machine so that, if we know a bit position in which Π and Π' differ, we can find a collision of the underlying hash function by extracting Π and Π' in that position along with the corresponding certificates. Then, to make sure that we can find a position in which Π and Π' differ with non-negligible probability, the language Λ_2 is further modified in such a way that the cheating prover is required to commit to the hash value of $\mathsf{ECC}(\Pi)$ instead of the hash value of Π, where ECC is an error-correcting code with constant relative distance and with polynomial-time encoding and decoding; since $\mathsf{ECC}(\Pi)$ and $\mathsf{ECC}(\Pi')$ differ in a constant fraction of their indices, they differ in a randomly chosen position with constant probability. Since we can extract $\mathsf{ECC}(\Pi)$ and $\mathsf{ECC}(\Pi')$ in a single position along with their certificates in time polynomial in n using the weak argument-of-knowledge property of UA, the proof now works under collision resistance against polynomial-time adversaries.

More formally, the trick by [BG09] works as follows. First, we replace the language Λ_2 in Figure 8 with the one in Figure 9. Next, we modify Claim 9 in such a way that E^* is required to extract a witness only implicitly (i.e., output the i-th bit of the witness on input any i); the proof of Claim 9 is the same as before except that we use weak argument-of-knowledge property instead of global one. Finally, we modify the proof of Claim 10 in such a way that, instead of extracting the whole of w and w', the adversary $\mathcal{A}$ extracts $\mathsf{ECC}(\Pi)$ and $\mathsf{ECC}(\Pi')$ only in a randomly chosen bit position and then extracts the certificates that correspond to that position; also, at the end $\mathcal{A}$ outputs a collision of the underlying hash function if $\mathcal{A}$ can compute it from the extracted bits and certificates. From essentially the same argument as before, it follows that $\mathcal{A}$ finds a collision of the underlying hash function with non-negligible probability. Since the running time of $\mathcal{A}$ is now bounded by a polynomial, we can derive a contradiction even when the underlying hash function is collision resistant only against polynomial-time adversaries.

5 Acknowledgment

The author greatly thank the anonymous reviewers of Journal of Cryptology and TCC 2015 for their useful comments about the presentation of this paper. I also thank an anonymous reviewer of TCC 2015 for pointing out an error that I made in an earlier version of this paper.

References

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *J. ACM*, 45(1):70–122, January 1998.

Language Λ_2:

Let ECC be an error-correcting code with constant relative distance and with polynomial-time encoding and decoding.
$(h, \tau, C, r) \in \Lambda_2$ if and only if there exist

- a machine Π (with some inputs being hardwired) such that $|\Pi| \leq n^{\log\log n}$
- a set of certificates $\{\mathsf{auth}_i\}_{i \in [|\eta|]}$, where $\eta \overset{\text{def}}{=} \mathsf{ECC}(\Pi)$
- randomness $R \in \{0,1\}^{\mathsf{poly}(n)}$ for Com
- a string y such that $|y| = n$

such that

- $C = \mathsf{Com}_\tau((|\eta|, h(\eta)); R)$, and
- $\mathsf{auth}_i = \mathsf{auth}_i(\eta)$ for every $i \in [|\eta|]$, and
- $\Pi(y)$ outputs a string that has r as a substring, and $\Pi(y)$ outputs it within $n^{\log\log n}$ steps.

Figure 9: A modified version of Λ_2.

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

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *CRYPTO*, pages 390–420, 1992.

[BG09] Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM Journal on Computing*, 38(5):1661–1694, 2009.

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

[Blu86] Manuel Blum. How to prove a theorem so no one else can claim it. In *the International Congress of Mathematicians*, pages 1444–1451, 1986.

[BP12] Nir Bitansky and Omer Paneth. From the impossibility of obfuscation to a new non-black-box simulation technique. In *FOCS*, pages 223–232, 2012.

[BP13] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In *STOC*, pages 241–250, 2013.

[BP15] Nir Bitansky and Omer Paneth. On non-black-box simulation and the impossibility of approximate obfuscation. *SIAM Journal on Computing*, 44(5):1325–1383, 2015.

[BY03] Mihir Bellare and Bennet S. Yee. Forward-security in private-key cryptography. In *CT-RSA*, pages 1–18, 2003.

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

[CKPR02] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires (almost) logarithmically many rounds. *SIAM Journal on Computing*, 32(1):1–47, 2002.

[CLP13a] Ran Canetti, Huijia Lin, and Omer Paneth. Public-coin concurrent zero-knowledge in the global hash model. In *TCC*, pages 80–99, 2013.

[CLP13b] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero knowledge from P-certificates. In *FOCS*, pages 50–59, 2013.

[CLP15] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero-knowledge from indistinguishability obfuscation. In *CRYPTO*, pages 287–307, 2015.

[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, 2009.

[DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC*, pages 416–426, 1990.

[GGS15] Vipul Goyal, Divya Gupta, and Amit Sahai. Concurrent secure computation via non-black box simulation. In *CRYPTO*, pages 23–42, 2015.

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

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 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.

[Goy13] Vipul Goyal. Non-black-box simulation in the fully concurrent setting. In *STOC*, pages 221–230, 2013.

[HILL99] Johan Htad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

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

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[PPS15] Omkant Pandey, Manoj Prabhakaran, and Amit Sahai. Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for NP. In *TCC*, pages 638–667, 2015.

[PR05] Rafael Pass and Alon Rosen. New and improved constructions of non-malleable cryptographic protocols. In *STOC*, pages 533–542, 2005.

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

[PRT13] Rafael Pass, Alon Rosen, and Wei-Lung Dustin Tseng. Public-coin parallel zero-knowledge for NP. *J. Cryptology*, 26(1):1–10, 2013.

[PTW09] Rafael Pass, Wei-Lung Dustin Tseng, and Douglas Wikstrm. On the composition of public-coin zero-knowledge protocols. In *CRYPTO*, pages 160–176, 2009.

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