An Alternative Approach to Non-black-box Simulation in Fully Concurrent Setting

Susumu Kiyoshima

NTT Secure Platform Laboratories, Japan.
kiyoshima.susumu@lab.ntt.co.jp

December 18, 2015

Abstract

We give a new proof of the existence of public-coin concurrent zero-knowledge arguments for $\mathcal{NP}$ in the plain model under the existence of collision-resistant hash function families, which was originally proven by Goyal (STOC'13). 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 a variety of fields of cryptography.

Traditionally, the security of all ZK protocols was proven via *black-box simulation*. That is, their zero-knowledge property was proven by showing a simulator that uses the adversary only as an oracle. Since black-box simulators use the adversary as an oracle, their only advantage is the ability to rewind the adversary. Nevertheless, it has been shown that black-box simulation is sufficient to construct ZK protocols with a variety of additional properties, security, and efficiency.

Black-box simulation is however known to have inherent limitations. For example, let us consider *public-coin* ZK *protocols* and *concurrent* ZK *protocols*, i.e., ZK protocols such that the verifier sends only the outcome of its coin-tossing during the protocols and ZK protocols such that their zero-knowledge property holds even in the concurrent setting. It is known that both of them can be constructed with black-box simulation techniques [GMW91, RK99, KP01, PRS02]. However, it is also known that neither of them can be constructed with black-box simulation techniques if we additionally require round efficiency, i.e., 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 is also known that no public-coin concurrent ZK protocol can be proven secure via black-box simulation irrespective to its round complexity [PTW09].

Because of these limitations, it is important to develop *non-black-box simulation* techniques. However, non-black-box simulation techniques is significantly hard to develop. In fact, in order to develop non-black-box simulation techniques, it seems that we need to find a way to "reverse engineer" adversaries, which is believed to be very difficult.

A breakthrough in non-black-box simulation techniques was made in a groundbreaking work of Barak [Bar01], who presented the first non-black-box simulation technique. The simulation technique by Barak is completely different from previous ones. In particular, in the simulation technique of Barak, the simulator does not rewind the adversary; instead, it runs in a "straight-line" manner and simulates the adversary's view by using the code of the adversary. With his non-black-box simulation technique, Barak showed that it is possible to go beyond the black-box simulation barrier. In particular, Barak showed that it is possible to construct a constant-round public-coin ZK protocol, which cannot be proven secure via black-box simulation as noted above.

Non-black-box simulation in the concurrent setting. Since Barak's technique allows us to overcome the black-box impossibility result of constant-round public-coin ZK protocol, it is natural to ask whether we can overcome other black-box impossibility results as well by using Barak's non-black-box simulation 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 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-*

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

concurrent execution [Bar01] (i.e., a concurrent execution with a bounded number of sessions) and *parallel execution* [PRT13], it had been open for years to extend Barak's simulation technique so that it can handle fully concurrent execution.

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. Furthermore, they showed that with their extended simulation techniques of Barak, 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. [PPS13] constructed a constant-round concurrent ZK protocols by assuming the existence of *differing-input indistinguishability obfuscations*, and Chung et al. [CLP15] constructed a constant-round concurrent ZK protocols by assuming the existence of indistinguishability obfuscations.

Very recently, Goyal [Goy13] proposed an extended version of Barak's non-black-box simulation technique that can handle fully concurrent execution *even in the plain model under standard assumptions*. In particular, Goyal extended Barak's simulation technique and used it to construct the first public-coin concurrent ZK protocol in the plain model under standard assumptions (the existence of a family of collision-resistant hash functions). 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 standard assumptions [Goy13].[2] 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, where $\epsilon > 0$ is an arbitrary constant, and thus 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 show a new variant of Barak's non-black-box simulation technique that can handle fully concurrent execution. We then use our variant 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 standard assumptions, and it has a simulator that runs in a straight-line

[2] Also, in their groundbreaking works [BP12, BP13], Bitansky and Paneth showed a non-black-box simulation technique that is *not* based on Barak's simulation technique.

manner in the fully concurrent setting. However, since it requires the same hardness assumption and the same round complexity as that of Goyal, it does not immediately lead to improvement over the result of Goyal. Nevertheless, we believe that our simulation technique is meaningful because it is different from the technique 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 the techniques of CLP. 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 with multiple slots, it is possible to avoid the blow-up of the simulator's running time, which is the main obstacle we encounter when using Barak's simulation technique in the concurrent setting. More precisely, CLP's observation is that with multiple slots, the simulator can generate a PCP proof w.r.t. any of these slots when giving universal argument (UA), and therefore by using a good "proving strategy" that determines which slot to use in the generation of PCP proofs, the simulator can avoid the blow-up of its running time in the concurrent setting. The proving strategy that CLP use is similar in spirit to the *oblivious rewinding strategy* of [KP01, PRS02] (in which black-box concurrent ZK protocols are constructed). In particular, in the proving strategy of CLP, the simulator recursively divides the transcript into blocks and then generates PCP proofs only at the end of the blocks.

A problem that CLP encountered is that since there is only one opportunity for the simulator to give a UA proof in each session, the simulator need to remember all previously generated PCP proofs until it finally reaches the UA proof. Because of this problem, the length of the PCP proofs can be rapidly blowing up in the concurrent setting and therefore 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 also has multiple opportunities to give UA proofs. We then show that by using a good proving strategy that also determines which opportunity the simulator takes to give UA proofs, the simulator can avoid the blow-up of its size as well as that of its running time. Our proving strategy works so that a PCP proof generated at the end of a block is used only in the "parent block" of that block; because of this property, 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 UA proofs, and 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 protocol BarakZK proceeds as follows.

Protocol BarakZK

1. The verifier V chooses a hash function $h \in \mathcal{H}_n$ and sends h 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 random string r 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 $(h, c, r) \in \Lambda$ holds if and only if there exists a machine Π such that c is a commitment to $h(\Pi)$ and Π outputs r in $n^{\log\log n}$ steps.[3]

Note that the statement proven in Step 3 is not in $\mathcal{NP}$. Thus, P proves this statement by a witness-indistinguishable *universal argument* (WIUA), with which P can prove any statement in $\mathcal{NEXP}$.

Intuitively, the security of BarakZK is proven as follows. The soundness is proven by showing that even when a cheating prover P^* commits to $h(\Pi)$ for a machine Π, it holds that $\Pi(c) \neq r$ with overwhelming probability. 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. This 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 $\Pi(c)$.

Barak's protocol in the concurrent setting. The proof of the ZK property of BarakZK does not work in the concurrent setting. In particular, the above simulator does not work in the concurrent setting since we have $V^*(c) \neq r$ when V^* receives messages during a slot (i.e., when V^* receives messages in other sessions before sending r).

A potential approach for achieving concurrent ZK property with 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 that although V^* receives unbounded number of messages during a slot, all of these messages are generated by $\mathcal{S}$; hence, if the committed machine Π can emulate $\mathcal{S}$ from the point that V^* receives c to the point that V^* sends r, Π can output r even when V^* receives many messages during a slot.

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

- The i-th session is executed so that the $(i+1)$-th session is completely contained in the slot of the i-th session. That is, the $(i+1)$-th session starts after V^* receives c in the i-th session, and the $(i+1)$-th session ends before V^* sends r in the i-th session.

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

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 $m = \omega(\log n)$.

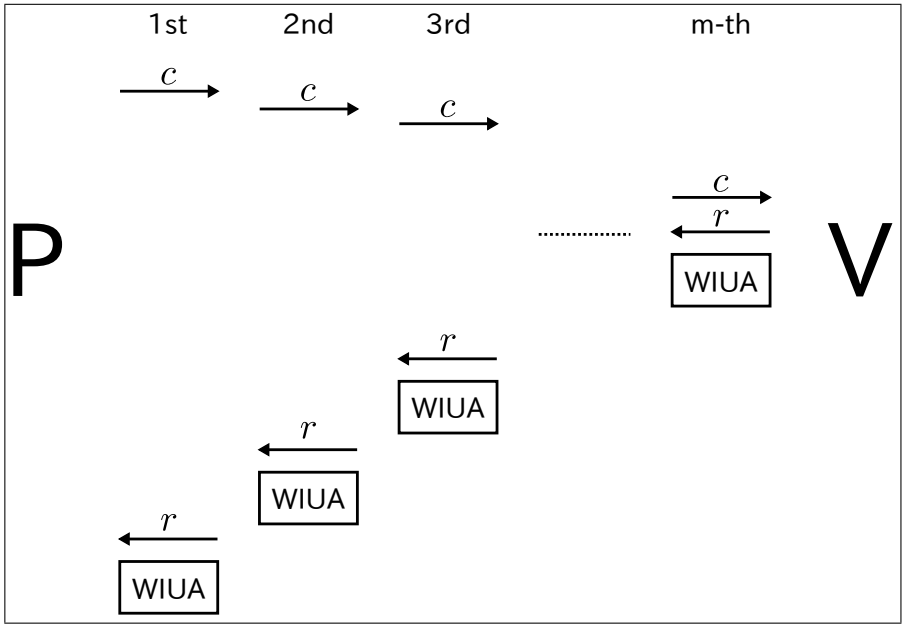

Figure 1: The "nested concurrent sessions" schedule.

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 that was originally used in the black-box concurrent zero-knowledge protocols of [RK99, KP01, PRS02]. The idea is that with many sequential slots, $\mathcal{S}$ can choose any of them as a witness in WIUA, and therefore with a good *proving strategy* that determines which slot to use as a witness, $\mathcal{S}$ can avoid the nested computation in WIUA. To implement this approach, CLP first observed that the four-round public-coin UA of [BG08], from which WIUA can be constructed, can be divided into the *offline phase* and the *online phase* such that all heavy computation is done in the offline phase. Concretely, the UA of [BG08] is divided 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. Then, P computes $\mathsf{UA}_2 := h(\pi)$. In the following, (h, π, UA_2) is called the *offline proof*.
- 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 queries Q by executing the PCP-verifier with statement $x \in L$ and randomness ρ.[4] Then, P computes the replies for the queries Q and sends them to V. We denote these replies by UA_4.
4. V verifies the correctness of the replies by executing the PCP-verifier.

Note that the only heavy computation—the generation of π and the computation of $h(\pi)$—is performed in the offline phase; the other computations can be performed in a fixed polynomial time. Thus, in the online phase, the running time of P can be bounded by a fixed polynomial in n.[5] 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 h to P.

Stage 2. For each $i \in [N_{\mathrm{slot}}]$ 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 to P.

Stage 3. P and V execute the special-purpose WIUA of [PR05] with the UA system of [BG08] being used as the underlying UA system.

1. P sends $c_{\mathsf{UA}} \leftarrow \mathsf{Com}(0^n)$ to V.
2. V sends the third 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 exists $i \in [N_{\mathrm{slot}}]$ and the second and the fourth UA messages $\mathsf{UA}_2, \mathsf{UA}_4$ such that UA_2 is the committed value of c_{UA} 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 the nested computation in WIUA by using a good proving strategy that determines which slot to use as a witness. Based on this idea, the simulation by CLP proceeds roughly as follows: First, $\mathcal{S}$ commits to a machine Π in each slot and then computes an offline proof (in particular, a PCP proof) w.r.t. a slot chosen according to the proving strategy; then, $\mathcal{S}$ commits to the second UA message (i.e., the hash of the PCP proof) in Stage 3-1 and gives a valid WIPOK proof in Stage 3-3. The proving strategy that CLP considered is a strategy that is similar in spirit to the oblivious rewinding strategy of [KP01, PRS02]. In this strategy, the entire transcript of all sessions is recursively divided into blocks. Let M be the total number of messages across the sessions, and let q be a parameter called the *splitting factor*. Assume for simplicity that M is a power of q, i.e., $M = q^d$ for $d \in \mathbb{N}$. Then, the entire transcript is divided into blocks as follows.

- The level-d block is the entire transcript of all sessions. Thus, the level-d block contains $M = q^d$ messages.

[4] Recall that the PCP-verifier performs the verification by making queries to the PCP proof.
[5] Here, P is assumed to have random access to π.

- Then, the level-d block is divided into q sequential blocks, where each block 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 block 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 level-0 blocks are obtained. A level-0 block contains only a single message.

Then, at the end of each block of each level, $\mathcal{S}$ computes offline proofs w.r.t. all slots that are contained in this block. Note that when $q = n^\epsilon$ for a constant ϵ, the maximum level of blocks (i.e., d) is constant. Thus we have at most constant level of nesting in the execution 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., at least one offline proof is computed before Stage 3 begins in every session except with negligible probability.

The protocol 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 for the following reasons. Recall that in the simulation, $\mathcal{S}$ 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 therefore can delay the execution of Stage 3 arbitrarily), in general, $\mathcal{S}$ need to remember all previously generated offline proofs during its execution. This means that each committed machine also need to contain all previously generated offline proofs, and therefore an offline proof w.r.t. a slot (which is generated by using a machine committed in this slot as a witness) is as long as the total length of all offline proofs that are generated before this slot. Thus, the length of offline proofs can be rapidly blowing up and therefore 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 protocol executions. Roughly speaking, CLP avoids the blow-up of the simulator's size by considering machines that contain only the hash value of the offline proofs; then, to guarantee that the simulation works with such machines, they modified BasicCLP so that P proves in WIUA that $x \in L$ or the committed machine outputs r given an access to the *hash-inversion oracle*; in the simulation, $\mathcal{S}$ commits to a machine that emulates $\mathcal{S}$ by recovering offline proofs from the hash value with 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 an $O(n^\epsilon)$-round protocol by removing the use of a global hash function from the protocol of CLP [CLP13a]. Recall that in the protocol of CLP, a global hash function is used 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 previously computed offline proofs. Below, to obtain our protocol, we first modify the machines to be committed by the simulator (and slightly modify BasicCLP and the simulator accordingly). The modified machines do not contain previously generated offline proofs and therefore their sizes are bounded by a fixed polynomial. We then modify BasicCLP and the simulator so that the simulation works even when the simulator commits to the modified machines. In the following, we set $q := n^\epsilon$ and $N_{\mathrm{slot}} := \omega(q)$.

Modification on the machines to be committed. We modify the machines in a way that they emulate the simulator *not from the start of a slot but from a more prior point of the simulation*; thus, the modified machines emulate more part of the simulation than before. Intuitively, if a machine emulates more part of the simulation, it potentially generates more offline proofs by itself, and therefore more likely to be able to output r even when it contains no offline proof. For example, let us consider an extreme case that each committed machine emulates the simulator from the beginning of the simulation. In this case, each committed machine generates every offline proofs by itself, and therefore it can output r even when it contains no offline proof. Unfortunately, in this case the running time of the simulator becomes super-polynomial since the running time of each committed machine is too long. Thus, we need to consider machines that do not emulate too much of the simulation.

Concretely, we consider a machine that emulates the simulator *from the beginning of a block*. In particular, for each $i \in [n]$, we consider the following machine Π_i.

- Π_i emulates the simulator from the beginning of the level-i block that contains the commitment in which Π_i is committed. Π_i does not contain any previously generated offline proofs, and if the emulation fails due to the lack of the offline proofs, Π_i terminates and outputs fail.

Then, we modify $\mathsf{BasicCLP}$ so that P gives n commitments in parallel in each slot, and let the simulator commit to Π_i in the i-th commitment. More precisely, the simulator does the following. In the interaction with V^*, for each $i \in [d]$, we say that a level-i block is the *current level-i block* if it will contain the next-scheduled message. In each slot, we call the i-th commitment the i-th *column*.

- In each slot, in the i-th column for each $i \in [n]$, the simulator commits to the machine Π_i, which emulates the simulator from the beginning of the current level-i block.
- At the end of each block in each level, for each slot that is contained in this block, the simulator generates the offline proof w.r.t. this slot by using a machine that emulates the simulator from the beginning of this block. Note that such a machine must have been committed in each slot.

When the simulator commits to these machines, the running time of the simulator can be bounded by a polynomial in n as follows. First, since each committed machine contains no offline proofs, the size of each committed machine is bounded by a fixed polynomial. Then, we bound the maximum time t_i spent by the simulation of a level-i block in the following way. Notice that at the end of a level-i block, each offline proof can be computed in time $\mathsf{poly}(t_i)$. Then, since a level-i block contains q level-$(i-1)$ blocks, and since at most $m := \mathsf{poly}(n)$ offline proofs are generated at the end of each level-$(i-1)$ block, we have

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

Then, since the maximum level $d = \log_q M$ is constant and since we have $t_0 = \mathsf{poly}(n)$, we have $t_d = \mathsf{poly}(n)$. Thus, 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 proofs, they do contain all previously generated witnesses of WIPOK (i.e., UA_2 and UA_4).[6] As explained below, allowing the machines to contain all previously generated witnesses is crucial to obtain a protocol and a simulator with which the simulation works even when the modified machines are committed.

[6] Since the length of the witnesses of WIPOK is bounded by a fixed polynomial, the size of the machines does not blow up even when they contain all previously generated witnesses of WIPOK.

Modifications on the protocol and the simulator. When the above machines are committed, the simulation can fail since the committed machines can 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 this block. (If such a block exists, the machines that are committed in this block output fail since they do not contain the necessary 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 this 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 computes a valid witness before each WIPOK starts.

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, the simulator uses only the offline proofs that are generated in this block. In BasicCLP, it is hard to construct such a simulator since an offline proof may be used long after it is generated. (Recall that during the simulation, offline proofs are generated in Stage 2 and they are used in Stage 3 to compute witnesses of WIPOK.) Thus, we modify BasicCLP so that the simulator can use offline proofs soon after generating them. In particular, we modify BasicCLP so that the simulator can use the offline proofs in Stage 2. Toward this end, we first observe the following.

- The simulator can compute a WIPOK witness from the offline proof anytime after Stage 3-2.
- The pair of Stage 3-1 and Stage 3-2 is syntactically the same as a slot: P sends a commitment in Stage 3-1 and V sends a random string in Stage 3-2. Thus, we can merge Stage 3-1 and Stage 3-2 into sequential slots.[7] [8]

Following these observations, we modify BasicCLP and obtain the following protocol. (As stated before, we also modify BasicCLP so that P gives parallel commitments in each slot.)

Protocol OurZK

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

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

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

Stage 3. P proves the following statement with WIPOK.

- $x \in L$, or
- there exist $i_1, i_2 \in [N_{\text{slot}}]$, $j \in [n]$, and the second and the fourth UA message UA_2 and UA_4 such that UA_2 is the committed value of $c_{i_2,j}$ and $(h, \mathsf{UA}_2, r_{i_2}, \mathsf{UA}_4)$ is an accepting proof of the statement $(h, c_{i_1,j}, r_{i_1}) \in \Lambda$.

In OurZK, a witness of WIPOK can be computed in a session if there are two slots such that (i) a machine is committed in a slot and (ii) an offline proof w.r.t. this slot is committed in the other slot. The computation of the WIPOK witness can be done anytime after such two slots, and after that, the offline proof will never be used.

We next modify the simulator as follows. Recall that, as noted above, we need the simulator such that (i) each committed machine does not output fail due to the lack of offline proofs, and (ii) the simulator does not get stuck.

[7] The idea of merging a part of special purpose WIUA into slots is also used in [COP+14] for different purpose. In [COP+14], this idea is used to reduce the round complexity.

[8] Alternatively, we can also think of executing the pair of Stage 3-1 and Stage 3-2 in parallel with each slot.

Roughly speaking, our simulator does the following (see Figure 2). Recall that for each $i \in \{0, \ldots, d-1\}$, a level-$(i+1)$ block is divided into q level-i blocks. Then, in each level-$(i+1)$ block, for each session, our simulator first tries to obtain a level-i block that contains a slot in which a machine is committed in the i-th column. If it succeeds in obtaining such a level-i block, our simulator computes an offline proof w.r.t. this slot. Next, our simulator tries to obtain a level-i block that contains a slot in which this offline proof is committed in the i-th column. If it succeeds in obtaining such a level-i block, our simulator computes a witness of WIPOK from this offline proof.

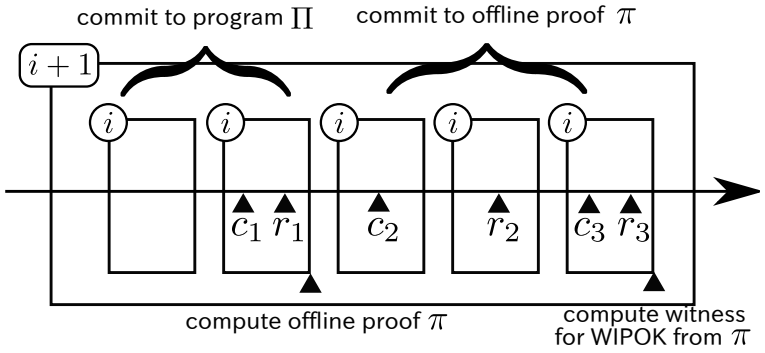

Figure 2: Our simulator's strategy, when splitting factor is $q = 5$.

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 each slot of each session, in the i-th column for each $i \in [n]$, the simulator commits to a machine that emulates the simulator from the beginning of the current level-i block *if no sibling of the current level-i block contains a slot of this session*; if there exists a sibling that contains a slot of this session, an offline proof must have been computed at the end of this sibling (see below), and the simulator commits to this offline proof.

- At the end of each level-i block for each $i \in \{0, \ldots, d-1\}$, the simulator does the following for every slot that is contained in this block: if a machine is committed in the i-th column of this slot, the simulator computes an offline proof by using the committed machine; if an offline proof is committed in the i-th column of this slot, the simulator computes a witness of WIPOK from this offline proof.

- When WIPOK starts, the simulator does the following: if the simulator have a valid witness, the simulator give a valid proof with this witness; if the simulator does not have a valid witness, the simulator aborts with output stuck.

Note that the simulator can compute a witness of WIPOK if there exists a block in which the simulator obtains two lower-level blocks such that each of them contains a slot.

We first note that each committed machine does not fail due to the lack of offline proofs. This follows immediately from the fact that in each block, the simulator uses only offline proofs that are generated in this block.

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

- A block is *good* w.r.t. a session if it contains a slot of this session and does not contain the first prover message of WIPOK of this session.

- 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, the simulator does not get stuck if for each session that reaches WIPOK in the simulation, there exists a block that has at least two children that are good w.r.t. this session. Thus, we show that for each session that reaches WIPOK in the simulation, there exists a block that has at least two children that are good w.r.t. this session. To prove this, it suffices to show that for each session that reaches WIPOK in the simulation, there exists a block such that it has at least three children that contain a slot of this session. (This is because at most one child contains the first message of WIPOK.) Assume for contradiction that there exists a session such that it reaches WIPOK and every block has at most two children that contain a slot of this session. Let $C(i)$ be the maximum number of slots that are contained by a level-i block. Then, since in each block there are at most $q-1$ slots that are contained by the block but not contained by its children, we have

$$C(i) \leq 2C(i-1) + q - 1 \ .$$

Then, since $C(0) = 0$ and since the maximum level d is constant, we have

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

This means that in the entire transcript there are at most $O(q)$ slots of this session. Since $N_{\mathrm{slot}} = \omega(q)$, this contradicts the assumption that this session reaches WIPOK. Thus, for each session that reaches WIPOK, there exists a block that has at least two children that are good w.r.t. this session. Thus, the simulator does not get stuck.

Since $q = O(n^{\epsilon})$ and $N_{\mathrm{slot}} = \omega(q)$, 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 works [Goy13, CLP13a, CLP13b, PPS13], when the code of the simulator is committed in the simulation, we have to take special care to the randomness of the simulator.[9] Fortunately, the techniques used in the previous works 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.

Since both the simulation technique of ours and that of Goyal are based on Barak's non-black-box simulation technique, there are many similarities between them: For example, the simulator commits to a machine that emulates itself; the protocol is modified so that it has multiple slots; the simulator is given multiple opportunities to give UA proofs[10]; the blocks are used to determine which opportunity the simulator takes to give UA proofs.

[9] 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.

[10] In our work, the simulator is given multiple opportunities to give UA proofs by modifying the protocol so that the encrypted UA is merged into the sequential execution of slots. In [Goy13], the simulator is given multiple opportunities to give UA proofs by modifying the protocol so that the encrypted UA is explicitly executed multiple times.

However, there are also differences between them. A notable difference is how the simulator determines which opportunity 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 sibling of the current block contains a slot; see Figure 2 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 protocols, such as commitment schemes, interactive proofs of knowledge, and witness-indistinguishable proofs.

3.1 Notations

We use n to denote the security parameter. For any $k \in \mathbb{N}$, let $[k] \stackrel{\text{def}}{=} \{1, \ldots, k\}$. For any randomized algorithm Algo, we use $\mathsf{Algo}(x; r)$ to denote the execution of Algo with input x and randomness r. We use $\mathsf{Algo}(x)$ to denote the execution of Algo with input x and uniformly chosen randomness.

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 $h \in \mathcal{H}_n \stackrel{\text{def}}{=} \{h_\alpha \in \mathcal{H} : \alpha \in \{0,1\}^n\}$, 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 $i \in \{1, \ldots, |x|\}$, we can compute a short certificate $\mathsf{auth}_i(x) \in \{0,1\}^{n^2}$ such that given $h(x)$, x_i, and $\mathsf{auth}_i(x)$, one can verify that the i-th bit of x is indeed x_i.

We obtain such a collision-resistant hash function family 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 Schemes

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 the binding and hiding property hold 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 Concurrent Zero-Knowledge

We recall the definition of *concurrent zero-knowledge*. 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 inputs, $w \in \mathbf{R}_L(x)$ is the private input of P, and z is the auxiliary input of V^*.

Definition 1 (Concurrent Zero-Knowledge)**.** *An interactive proof or argument $\langle P, V\rangle$ for 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 are computationally indistinguishable.*

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

3.5 PCP and Universal Argument

We recall the definitions of *probabilistically checkable proof* (PCP) systems and *universal argument* system.

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

For simplicity, we show the definitions of PCPs and universal arguments that prove 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. 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 $y = (M, x, t) \in L_{\mathcal{U}}$ 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.[11]

3.5.2 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 2 (PCP system—basic definition)**.** *A **probabilistically checkable proof** (PCP) system (with a negligible soundness error) is a* PPT *oracle machine V (called 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*

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

◇

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

In this paper, we use PCP systems as a building block in the universal argument UA of [BG08]. 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. To understand this paper, the definitions of the first two properties are required; for the definitions of other properties, see [BG08].

Definition 3 (PCP system—auxiliary properties)**.** *Let V be a PCP-verifier.*

- ***Relatively efficient oracle construction:*** *There exists an algorithm P (called prover) such that, given any $(y, w) \in \boldsymbol{R}_{\mathcal{U}}$, algorithm P outputs an oracle π_y that makes V always accepts (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.*

◇

3.5.3 Universal Argument.

Universal arguments [BG08], 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 $y = (M, x, t) \in L_{\mathcal{U}}$, let $T_M(x, w)$ be the running time of M on input x with witness w, and let $\mathbf{R}_{\mathcal{U}}(y) \stackrel{\text{def}}{=} \{w : (y, w) \in \mathbf{R}_{\mathcal{U}}\}$.

Definition 4 (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 $w \in \boldsymbol{R}_{\mathcal{U}}(y)$, it holds that*

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

  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 is a negligible function $\mathsf{negl}(\cdot)$ such that for every $y = (M, x, t) \notin L_{\mathcal{U}}$ and $z \in \{0,1\}^*$, it holds that*

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

- ***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 equal 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. Given an input $y = (M, x, t) \in L_{\mathcal{U}}$, since the witness $w \in \mathbf{R}_{\mathcal{U}}(y)$ is of length at most t, it follows that there exists an extractor (called the *global extractor*) that runs in time polynomial in $\mathsf{poly}(|y|) \cdot t$ that extracts the whole witness; we refer to this as 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 [BG08] (Figure 3). As observed in [CLP13a], the construction of UA can be separated into an expensive *offline stage* and an efficient *online stage*. In the online stage, the running time of the prover is polynomial in $n = |y|$, and in the offline stage, the running time of the prover is $\mathsf{poly}(n + T_M(x, w))$. We notice that the third message UA_3 satisfies $|\mathsf{UA}_3| = \omega(\log n) \cdot \mathsf{poly}(\log n) \leq n^2$ and the fourth message UA_4 satisfies $|\mathsf{UA}_4| = \mathsf{poly}(n)$.

- **Input:** The common input is $y = (M, x, t) \in L_{\mathcal{U}}$, and the private input of 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 $\rho \in \{0,1\}^{n^2}$ for the PCP-verifier and sends $\mathsf{UA}_3 := \rho$ to P.
  3. P computes queries Q by executing the PCP-verifier with statement $y \in L_{\mathcal{U}}$ and randomness ρ. 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 certificates and checks whether the PCP-verifier accepts on input $(y, \{(i, \pi_i)\}_{i \in Q})$ with randomness ρ.

Figure 3: Online/offline UA system of [BG08, CLP13a].

3.6 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 definition of a forward secure pseudorandom generator that is a variant of the one used in [CLP13b]. We notice that in the following definition, the indices of the seeds and pseudorandomness are written in reverse order because we use them in reverse order in the analysis of our concurrent zero-knowledge protocol.

Definition 5 (Forward-secure PRG)**.** *We say that a polynomial-time computable function* f-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 $(\rho_\ell, \ldots, \rho_1)$ such that the following properties hold.*

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

  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

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* cZKAOK*.*

Proof . cZKAOK is shown in Figure 4, where we use the following building blocks.

- Naor's two-round statistically binding commitment scheme Com, which can be constructed from one-way functions.
- A four-round public-coin witness-indistinguishable proof of knowledge WIPOK, which can be constructed from one-way functions.
- Four-round public-coin universal argument UA of [BG08] (Figure 3 in Section 3.5.3), which can be constructed from collision-resistant hash functions.

Clearly, cZKAOK is public-coin and its round complexity is $2N_{\mathrm{slot}} + 5 = O(n^\epsilon)$. Thus, Theorem 1 follows from the following lemmas.

Lemma 1. cZKAOK *is concurrently 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. □

Remark 1. The languages Λ_2 in Figure 5 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 given by [BG08], 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.

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

Parameter: An integer $N_{\text{slot}} = O(n^\epsilon)$.

Stage 1: The verifier V chooses a random hash function $h \in \mathcal{H}_n$ and sends h to the prover P. Additionally, V sends the 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.

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

Stage 3: P proves the following by using WIPOK.

- $x \in L$, or
- $(h, \tau, \overline{C}_1, r_1, \ldots, \overline{C}_{N_{\text{slot}}}, r_{N_{\text{slot}}}) \in \Lambda_1$, where language Λ_1 is shown in Figure 5.

Figure 4: Public-coin concurrent zero-knowledge argument cZKAOK.

4.1 Concurrent Zero-knowledge Property

Proof of Lemma 1. Let V^* be any cheating verifier. 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. 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). Note that since the total number of messages is at most $M \stackrel{\text{def}}{=} (2N_{\text{slot}} + 5) \cdot m$, we have $d = \log_q M = \log_q(\mathsf{poly}(n)) = O(1)$.

4.1.1 Simulator $\mathcal{S}$

In this section, we describe the simulator $\mathcal{S}$. We recommend that the readers browse the overview of our simulation technique in Section 2.2 before reading this section.

The simulator $\mathcal{S}$ simulates the view of V^* by using an auxiliary simulator algorithm $\widetilde{\mathcal{S}}$. Roughly speaking, $\widetilde{\mathcal{S}}$ is an algorithm that simulates a part of the view by recursively executing itself. The input of $\widetilde{\mathcal{S}}$ is the execution level ℓ, a partial transcript trans that is simulated so far, and tables T_{UA} and T_{W} of the second-round UA messages and the WIPOK witnesses that are computed so far. The goal of $\mathcal{S}(\ell, \mathsf{trans}, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}})$ is to simulate the next q^ℓ messages from trans; in particular, the goal is to output newly simulated messages $\mathsf{new\text{-}trans}$ and updated table T_{W} of the witnesses. The simulator $\mathcal{S}$ simulates the entire view of V^* by executing $\widetilde{\mathcal{S}}$ with level d.

We remark that for technical reasons, the simulation by $\mathcal{S}$ is a bit more complex than the one described in Section 2.2.

- To avoid the circularity that arises when $\widetilde{\mathcal{S}}$ uses it own code, we use a technique of [CLP13b]. Roughly speaking, $\widetilde{\mathcal{S}}$ takes the code of a machine Π as input and uses this code instead of its own code; we then design $\mathcal{S}$ and $\widetilde{\mathcal{S}}$ in a way that when $\widetilde{\mathcal{S}}$ is invoked, it always holds $\Pi = \widetilde{\mathcal{S}}$.

- To avoid the issue of randomness that we sketched in Section 2.2, we use a technique of [CLP13a, CLP13b] that uses a forward-secure PRG $\mathsf{f\text{-}PRG}$. Specifically, $\widetilde{\mathcal{S}}$ takes a seed σ

Language Λ_1: (statement for WIPOK)

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

- $i_1, i_2 \in [N_{\mathrm{slot}}]$ and $j \in [n]$ such that $i_1 < i_2$
- the second and fourth UA messages $\mathsf{UA}_2 \in \{0,1\}^n$ and $\mathsf{UA}_4 \in \{0,1\}^{\mathsf{poly}(n)}$
- randomness $R \in \{0,1\}^{\mathsf{poly}(n)}$ for Com

such that

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

Language Λ_2: (statement for UA)

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

- a machine Π (with some inputs being hardwired) such that $|\Pi| \leq n^{\log\log n}$
- randomness $R \in \{0,1\}^{\mathsf{poly}(n)}$ for Com
- a string y such that $|y| \leq 3n$

such that

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

Figure 5: Languages used in cZKAOK.

of $\mathsf{f\text{-}PRG}$ as input, computes a sequence of pseudorandomness $\rho_1, \rho_2, \ldots$, and then simulates messages in a way that the i-th message is computed with randomness ρ_i.

- To simplify the analysis, we design $\widetilde{\mathcal{S}}$ in a way that there are several "modes" of the execution, which is specified by input mode. When $\mathsf{mode} = \mathsf{main}$, $\widetilde{\mathcal{S}}$ outputs $\mathsf{new\text{-}trans}$ and $\mathsf{T_W}$ as noted above. When $\mathsf{mode} = \mathsf{aux}_s$ for $s \in [m]$, $\widetilde{\mathcal{S}}$ outputs the verifier message r of a slot of session s that is contained by $\mathsf{new\text{-}trans}$. We notice that $\widetilde{\mathcal{S}}$ is always executed with $\mathsf{mode} = \mathsf{main}$ and it commits the code of itself with $\mathsf{mode} \in \{\mathsf{aux}_1, \ldots, \mathsf{aux}_m\}$.

- To simplify the analysis, we give two additional input ξ and ρ that are used only in a special mode hyb. On the first look, the readers can ignore input ξ, ρ and mode hyb.

  (Jumping ahead, when $\mathsf{mode} = \mathsf{hyb}$, $\widetilde{\mathcal{S}}$ simulates the transcript up until the ξ-th message in a way that the ξ-th message is simulated with randomness ρ. This mode is used to argue the indistinguishability of two hybrid experiments in which pseudorandomness is replaced with true randomness.)

The formal descriptions of $\mathcal{S}$ and $\widetilde{\mathcal{S}}$ are given below. The input of $\mathcal{S}$ is statement x and auxiliary input z, and the input of $\widetilde{\mathcal{S}}$ is $(x, z, \ell, \Pi, \mathsf{trans}, \mathsf{T_{UA}}, \mathsf{T_W}, \mathsf{mode}, \langle\sigma, \xi, \rho\rangle)$ such that:

- $\ell \in \{0, \ldots, d\}$ is the execution level.

- Π is a code of a machine. (Below, $\widetilde{\mathcal{S}}$ is used in a way that $\Pi = \widetilde{\mathcal{S}}$.)
- $\mathsf{trans} \in \{0,1\}^{\mathsf{poly}(n)}$ is the partial transcript that is simulated so far.
- $\mathsf{T}_{\mathsf{UA}} = \{\mathsf{ua}_{s,j}, \mathsf{stamp}_{s,j}\}_{s\in[m],j\in[n]}$ is the table of the second-round UA messages that are computed so far. Each $\mathsf{stamp}_{s,j}$ is the timestamp that indicates when $\mathsf{ua}_{s,j}$ is updated. (If $\mathsf{ua}_{s,j}$ is updated just after the k-th message is simulated, $\mathsf{stamp}_{s,j}$ is set to be k.)
- $\mathsf{T}_{\mathsf{W}} = \{w_s, \mathsf{stamp}_s\}_{s\in[m]}$ is the table of the WIPOK witnesses that are computed so far.
- $\mathsf{mode} \in \{\mathsf{main}, \mathsf{aux}_1, \ldots, \mathsf{aux}_m, \mathsf{hyb}\}$.
- σ is a seed of $\mathsf{f\text{-}PRG}$, $\xi \in [q^d]$, and $\rho \in \{0,1\}^n$.

For $a, b \in \mathbb{N}$, let $\lfloor a \rfloor_b \stackrel{\text{def}}{=} a - (a \bmod b)$. (Notice that if the next scheduled message is the $(\kappa+1)$-th message, the beginning of the current level-ℓ block is the $(\lfloor \kappa \rfloor_\ell + 1)$-th message.) Let Filter be an algorithm such that for any $k \in \mathbb{N}$ and any set $A = \{a_i, \mathsf{stamp}_i\}_i$, $\mathsf{Filter}(A, k)$ outputs a set $A' = \{a'_i, \mathsf{stamp}'_i\}_i$ such that $a'_i = a_i$ and $\mathsf{stamp}'_i = \mathsf{stamp}_i$ if $\mathsf{stamp}_i \le k$ and $a'_i = \perp$ and $\mathsf{stamp}' = \infty$ if $\mathsf{stamp}_i > k$.

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

1. Choose random seed $\sigma_{q^d+1} \in \{0,1\}^n$ of $\mathsf{f\text{-}PRG}$ and random $\rho_{q^d+1} \in \{0,1\}^n$. Initialize $\mathsf{T}_{\mathsf{UA}} = \{\mathsf{ua}_{s,j} = \perp, \mathsf{stamp}_{s,j} = \infty\}_{s\in[m],j\in[n]}$ and $\mathsf{T}_{\mathsf{W}} = \{w_s = \perp, \mathsf{stamp}_s = \infty\}_{s\in[m]}$.
2. Compute $(\mathsf{trans}, \mathsf{T}'_{\mathsf{W}}) := \widetilde{\mathcal{S}}(x, z, \widetilde{\mathcal{S}}, d, \varepsilon, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}, \mathsf{main}, \langle \sigma_{q^d+1}, q^d + 1, \rho_{q^d+1} \rangle)$, where ε is the empty string.
3. Output (x, z, trans).

Auxiliary Simulator $\widetilde{\mathcal{S}}(x, z, \ell, \Pi, \mathsf{trans}, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}, \mathsf{mode}, \langle \sigma, \xi, \rho \rangle)$:

Step 1: Initialization.
Let $\kappa := |\mathsf{trans}|$ (i.e., κ is the number of messages that are included in trans).
If $\mathsf{mode} \in \{\mathsf{main}, \mathsf{aux}_1, \ldots, \mathsf{aux}_m\}$:

- Let $\sigma_{\kappa+q^\ell+1} := \sigma$. Then, compute

$$(\sigma_{\kappa+q^\ell}, \ldots, \sigma_1, \rho_{\kappa+q^\ell}, \ldots, \rho_1) := \mathsf{f\text{-}PRG}(\sigma_{\kappa+q^\ell+1}, \kappa + q^\ell) \ .$$

If $\mathsf{mode} = \mathsf{hyb}$:

- Let $\sigma_\xi := \sigma$. Then, compute

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

and let $\rho_\xi := \rho$

Step 2a: Simulation (base case). When $\ell = 0$, 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, do the following with randomness $\rho_{\kappa+1}$. (If necessary, $\rho_{\kappa+1}$ is expanded by a pseudorandom generator.)

- If msg is a message of Com in the s-th session ($s \in [m]$), find the first-round message τ_s of Com of the s-th session from trans and then compute $\mathsf{msg} = (C_1, \ldots, C_n)$ as follows.
  For every $j \in [d]$ such that $\mathsf{ua}_{s,j} = \bot$:
  (a) **Obtaining a machine that computes the current level-j block.** Let $\mathsf{trans}_j := \mathsf{trans}[1, \ldots, \lfloor \kappa \rfloor_{q^j}]$ be the prefix of trans up until the $\lfloor \kappa \rfloor_{q^j}$-th message. Also, let $\mathsf{T}_{\mathsf{UA}}^{(j)} := \mathsf{Filter}(\mathsf{T}_{\mathsf{UA}}, \lfloor \kappa \rfloor_{q^j})$ and $\mathsf{T}_{\mathsf{W}}^{(j)} := \mathsf{Filter}(\mathsf{T}_{\mathsf{W}}, \lfloor \kappa \rfloor_{q^j})$. Then, let
  $$\widetilde{\Pi}_{s,j}(\cdot) := \Pi(x, z, j, \Pi, \mathsf{trans}_j, \mathsf{T}_{\mathsf{UA}}^{(j)}, \mathsf{T}_{\mathsf{W}}^{(j)}, \mathsf{aux}_s, \cdot) \ .$$
  (b) From trans, find the hash function h_s of the s-th session.
  (c) Compute $C_j \leftarrow \mathsf{Com}_{\tau_s}(h_s(\widetilde{\Pi}_{s,j}))$.
  For every $j \in [d]$ such that $\mathsf{ua}_{s,j} \neq \bot$: Computes $C_j \leftarrow \mathsf{Com}_{\tau_s}(\mathsf{ua}_{s,j})$.
  For every $j \in \{d+1, \ldots, n\}$: Computes $C_j \leftarrow \mathsf{Com}_{\tau_s}(0^n)$.
- If msg is the first prover message of WIPOK in the s-th session ($s \in [m]$), honestly compute msg by using witness w_s. If w_s is not a valid witness, aborts with output stuck.
- If msg is the second prover message of WIPOK, honestly compute msg by reconstructing the prover state of WIPOK from w_s and $\rho_1, \ldots, \rho_\kappa$.[12]

2. Output $(\mathsf{msg}, \mathsf{T}_{\mathsf{W}})$.

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

1. Let $\mathsf{ctr}_s := 0$ and $\mathsf{tmp}_s := \bot$ for every $s \in [m]$. Let $\mathsf{head}(k) \stackrel{\text{def}}{=} \kappa + (k-1)q^{\ell-1} + 1$ (i.e., $\mathsf{head}(k)$ is the index of the beginning of the k-th child-block).
   If $\mathsf{mode} \in \{\mathsf{main}, \mathsf{aux}_1, \ldots, \mathsf{aux}_m\}$:
   - Let $\tilde{q} := q$.

   If $\mathsf{mode} = \mathsf{hyb}$:
   - Let $\tilde{q}$ be the integer that satisfies $\mathsf{head}(\tilde{q}) \leq \xi < \mathsf{head}(\tilde{q}+1)$ (i.e., the integer such that the $\tilde{q}$-th child-block will contain the ξ-th message).
2. For each $k \in [\tilde{q}]$, do the following:
   (a) **Executing the k-th child-block.** Let
   $$\Pi_{\ell-1}(\cdot, \cdot) \stackrel{\text{def}}{=} \Pi(x, z, \ell-1, \Pi, \mathsf{trans}, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}, \cdot, \cdot) \ .$$
   If $\mathsf{mode} \in \{\mathsf{main}, \mathsf{aux}_1, \ldots, \mathsf{aux}_s\}$ or $\mathsf{mode} = \mathsf{hyb} \wedge k < \tilde{q}$, compute
   $$(\mathsf{trans}_k, \mathsf{T}'_{\mathsf{W}}) := \Pi_{\ell-1}(\mathsf{main}, \langle \sigma_{\mathsf{head}(k+1)}, \mathsf{head}(k+1), \rho \rangle) \ .$$
   If $\mathsf{mode} = \mathsf{hyb} \wedge k = \tilde{q}$, compute
   $$(\mathsf{trans}_k, \mathsf{T}'_{\mathsf{W}}) := \Pi_{\ell-1}(\mathsf{hyb}, \langle \sigma, \xi, \rho \rangle) \ .$$
   Comment: *Recall that $\mathcal{S}$ executes $\widetilde{\mathcal{S}}$ in a way that Π is instantiated with $\widetilde{\mathcal{S}}$.*
   (b) Update $\mathsf{trans} := \mathsf{trans} \| \mathsf{trans}_k$ and $\mathsf{T}_{\mathsf{W}} := \mathsf{T}'_{\mathsf{W}}$.

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

(c) **Computing offline proof or witness using slots in the k-th child-block.** If $\mathsf{mode} \in \{\mathsf{main}, \mathsf{aux}_1, \ldots, \mathsf{aux}_s\}$ or $\mathsf{mode} = \mathsf{hyb} \wedge k < \tilde{q}$, do the following for each $s \in [m]$ such that trans_k includes a slot sl of the s-th session. (If trans_k includes more than one slot of the s-th session, sl is the first such slot.)

When $\mathsf{ctr}_s = 0$:

i. Let i_1 be the *slot-index* of sl (i.e., $i_1 \in [N_{\mathrm{slot}}]$ s.t. sl is the i_1-th slot of the s-th session). Let $(\overline{C}_{i_1}, r_{i_1})$ denote sl, where $\overline{C}_{i_1} = (C_{i_1,1}, \ldots, C_{i_1,n})$.

ii. **Computing offline proof.** From $\rho_{\mathsf{head}(k)}, \ldots, \rho_{\mathsf{head}(k+1)-1}$, find the randomness R_1 that was used for $C_{i_1,\ell-1}$.[13] Then, compute a PCP proof π_s for statement $(h_s, \tau_s, C_{i_1,\ell-1}, r_{i_1}) \in \Lambda_2$ with witness $(\Pi_{s,\ell-1}, R_1, \langle \sigma_{\mathsf{head}(k+1)}, \mathsf{head}(k+1), \rho \rangle)$, where $\Pi_{s,\ell-1}(\cdot) := \Pi_{\ell-1}(\mathsf{aux}_s, \cdot)$. Then, compute $\mathsf{UA}_2 := h_s(\pi_s)$.

iii. Update T_{UA} by setting $\mathsf{ua}_{s,\ell-1} := \mathsf{UA}_2$ and $\mathsf{stamp}_{s,\ell-1} := \mathsf{head}(k+1) - 1$.

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

When $\mathsf{ctr}_s = 1$:

i. Let i_2 be the slot-index of sl. Let $(\overline{C}_{i_2}, r_{i_2})$ denote sl, where $\overline{C}_{i_2} = (C_{i_2,1}, \ldots, C_{i_2,n})$.

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

iii. From $\rho_{\mathsf{head}(k)}, \ldots, \rho_{\mathsf{head}(k+1)-1}$, find the randomness R_2 that is used for $C_{i_2,\ell-1}$. Then, if $w_s = \bot$, update T_{W} by setting $w_s := (i_1, i_2, \ell - 1, \mathsf{UA}_2, \mathsf{UA}_4, R_2)$ and $\mathsf{stamp}_s := \mathsf{head}(k+1) - 1$.

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

3. Let $\mathsf{new\text{-}trans} := \mathsf{trans}_1 \| \mathsf{trans}_2 \| \cdots \| \mathsf{trans}_{\tilde{q}}$.
If $\mathsf{mode} \in \{\mathsf{main}, \mathsf{hyb}\}$:

- Output $(\mathsf{new\text{-}trans}, \mathsf{T}_{\mathsf{W}})$.

If $\mathsf{mode} = \mathsf{aux}_s$ $(s \in [m])$:

- If there is a slot of the s-th session such that both the prover message and the verifier message of the slot is contained by $\mathsf{new\text{-}trans}$, output the verifier message r of the first such slot. Otherwise, output $\bot$.

4.1.2 Correctness of $\mathcal{S}$.

We observe the correctness of $\mathcal{S}$; namely, we observe that $\widetilde{\mathcal{S}}$ can indeed compute a valid PCP proof π_s and a valid witness w_s of WIPOK in Step 2b.

First, we see that when $\widetilde{\mathcal{S}}$ tries to compute a PCP proof π_s in Step 2b, it can indeed compute it. In other words, we see that

$$(\Pi_{s,\ell-1}, R_1, \langle \sigma_{\mathsf{head}(k+1)}, \mathsf{head}(k+1), \rho \rangle)$$

is indeed a valid witness for $(h_s, \tau_s, C_{i_1,\ell-1}, r_{i_1}) \in \Lambda_2$. To see this, we observe the following.

1. First, we observe that $C_{i_1,\ell-1}$ is a commitment to a machine.

   - $C_{i_1,\ell-1}$ is a commitment to a machine if $\mathsf{ua}_{s,\ell-1} = \bot$ when $C_{i_1,\ell-1}$ was computed.

[13] From the construction of $\widetilde{\mathcal{S}}$, 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 $\mathsf{ctr}_s = 0$, we have $\mathsf{ua}_{s,\ell-1} = \bot$ when trans_k was computed, and hence we indeed have $\mathsf{ua}_{s,\ell-1} = \bot$ when $C_{i_1,\ell-1}$ was computed.

2. Next, we observe that the machine committed in $C_{i_1,\ell-1}$ is $\Pi_{s,\ell-1}$.
   - Since $C_{i_1,\ell-1}$ is a message in trans_k, it is the $(\mathsf{head}(k) + \Delta)$-th message across all the sessions, where $0 \le \Delta \le q^{\ell-1} - 1$.
   - Since we have $\lfloor \mathsf{head}(k) + \Delta - 1 \rfloor_{q^{\ell-1}} = \mathsf{head}(k)$, the machine $\widetilde{\Pi}_{s,\ell-1}$ committed in $C_{i_1,\ell-1}$ was obtained by restoring the tables $\mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}$ to the point just before the $(\mathsf{head}(k) + 1)$-th message was computed.
   - Then, since $\Pi_{s,\ell-1}$ is a machine that was obtained by fixing inputs of Π just before $(\mathsf{head}(k) + 1)$-th message was computed, it follows that the committed machine $\widetilde{\Pi}_{s,\ell-1}$ is equal to $\Pi_{s,\ell-1}$.

3. Finally, we observe that $\Pi_{s,\ell-1}$ outputs r_{i_1} on input $\langle \sigma_{\mathsf{head}(k+1)}, \mathsf{head}(k+1), \rho \rangle$.
   - Since $\Pi_{s,\ell-1}(\cdot) = \Pi_{\ell-1}(\mathsf{aux}_s, \cdot)$, it follows that $\Pi_{s,\ell-1}(\cdot)$ internally generates $\mathsf{new\text{-}trans}$ in the same way as $\Pi_{\ell-1}(\mathsf{main}, \cdot)$ and then outputs the verifier message of the first slot of the s-th session that is contained by $\mathsf{new\text{-}trans}$.
   - From the definition of r_{i_1}, this means that $\Pi_{s,\ell-1}$ outputs r_{i_1} on input $\langle \sigma_{\mathsf{head}(k+1)}, \mathsf{head}(k+1), \rho \rangle$.

From the above, we conclude that when $\widetilde{\mathcal{S}}$ tries to compute a PCP proof π_s in Step 2b, it can indeed compute it.

Next, we see that when $\widetilde{\mathcal{S}}$ tries to compute a WIPOK proof in Step 2a, w_s is a valid witness as long as $w_s \neq \bot$. In other words, we see that if $\widetilde{\mathcal{S}}$ updates w_s to $(i_1, i_2, \ell - 1, \mathsf{UA}_2, \mathsf{UA}_4, R_2)$ in Step 2b, $(i_1, i_2, \ell - 1, \mathsf{UA}_2, \mathsf{UA}_4, R_2)$ is indeed a valid WIPOK witness.

1. First, from the definition of UA_2 and what is observed above, UA_2 is the hash value of a valid PCP proof π_s for statement $(h_s, \tau_s, C_{i_1,\ell-1}, r_{i_1}) \in \Lambda_2$. From the construction of UA, this means that $(h, \mathsf{UA}_2, r_{i_2}, \mathsf{UA}_4)$ is an accepting proof for $(h_s, \tau_s, C_{i_1,\ell-1}, r_{i_1}) \in \Lambda_2$

2. Next, since $\mathsf{ua}_{s,\ell-1}$ was updated to UA_2 after UA_2 was computed, it follows that $C_{i_2,\ell-1}$ is a commitment to UA_2 with randomness R_2.

From the above and the definition of the language Λ_1, we conclude that $(i_1, i_2, \ell - 1, \mathsf{UA}_2, \mathsf{UA}_4, R_2)$ is a valid WIPOK witness.

Finally, we see that when $\widetilde{\mathcal{S}}$ tries to compute a WIPOK proof in Step 2a, we have $w_s \neq \bot$. This follows from the following claim.

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

Proof. We first introduce notations. Recall that $\mathcal{S}$ recursively executes $\widetilde{\mathcal{S}}$ many times. We use *block* to denote an execution of $\widetilde{\mathcal{S}}$. Notice that from the constructions of $\mathcal{S}$ and $\widetilde{\mathcal{S}}$, each block can be uniquely identified by the value of ℓ and $\kappa = |\mathsf{trans}|$. A block is in level ℓ if the corresponding $\widetilde{\mathcal{S}}$ is executed with input ℓ. For each block, the *child-blocks* of this block are the blocks that are recursively executed by this block; thus, each block has q child-blocks. For any slot of any session, we say that a block *contains* this slot if the corresponding execution of $\widetilde{\mathcal{S}}$ outputs a transcript that includes this slot (both the prover and the verifier message). A block is *good* w.r.t. a session if this block contains a slot of this session but does not contain the first prover message of WIPOK of this session.

Given these notations, we prove the claim as follows. From the construction of $\mathcal{S}$ and $\widetilde{\mathcal{S}}$, none of $\widetilde{\mathcal{S}}$ 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, a witness of WIPOK is computed at the end of the second good child-block.) Thus, it suffices to show that for every session that reaches Stage 3, there exists a block that has two good child-blocks w.r.t. that session. To show this, it suffices to show that for every session that reaches Stage 3, there exists a block that has three child-blocks that contain slots of this session. (If three child-blocks contain slots, two of them are good since at most one child-block can contain the first prover message of WIPOK.) Assume for contradiction that there exists a session such that every block has at most two child-blocks that contain slots of this session. For $\ell \in \{0, \ldots, d\}$ and $\kappa \in [q^d]$, let $C_\kappa(\ell)$ be the number of slots of this session that are contained by the block that is identified by ℓ and κ, and let $C(\ell) \stackrel{\text{def}}{=} \max_\kappa(C_\kappa(\ell))$. Then, since each block has at most two child-blocks that contain slots, and since in each block there are at most $q-1$ slots that are contained by this block but are not contained by any of its child-block, we have

$$C(\ell) \leq 2C(\ell-1) + q - 1 \ .$$

Thus, we have

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

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

From the above three observations, we conclude that $\widetilde{\mathcal{S}}$ can indeed compute a valid PCP proof π_s and a valid witness w_s of WIPOK 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 $\mathcal{S}$ consists of recursive executions of $\widetilde{\mathcal{S}}$. From the constructions of $\mathcal{S}$ and $\widetilde{\mathcal{S}}$, each execution of $\widetilde{\mathcal{S}}$ can be uniquely identified by the value of ℓ and $\kappa = |\mathsf{trans}|$. In the following, we use $\widetilde{\mathcal{S}}_{\ell,\kappa}$ to denote the execution of $\widetilde{\mathcal{S}}$ with ℓ and κ. Let $t_{\ell,\kappa}$ be the running time of $\widetilde{\mathcal{S}}_{\ell,\kappa}$, and let $t_\ell \stackrel{\text{def}}{=} \max_\kappa(t_{\ell,\kappa})$. Then, observe that every computation in $\widetilde{\mathcal{S}}_{\ell,\kappa}$ is performed in fixed polynomial time in n except for the following computations:

Type-1 computation. The recursive executions of $\widetilde{\mathcal{S}}$ (i.e., $\widetilde{\mathcal{S}}_{\ell-1,\kappa}, \widetilde{\mathcal{S}}_{\ell-1,\kappa+q^{\ell-1}}, \ldots$).

Type-2 computation. The generations of the offline proofs (i.e., PCP proofs and their hash values).

Each type-1 computation can be performed in time $t_{\ell-1}$. Furthermore, from the relatively efficient oracle construction property of PCP systems, each type-2 computation can be performed in time $\mathsf{poly}(t_{\ell-1})$.[14] Then, since for each $k \in [q]$ there are a single type-1 computation and at most m type-2 computations, we have

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

[14]Notice that the running time of $\Pi_{s,\ell-1}(\langle \sigma_{\mathsf{head}(k+1)}, \mathsf{head}(k+1), \rho\rangle) = \Pi_{\ell-1}(\mathsf{aux}_s, \langle \sigma_{\mathsf{head}(k+1)}, \mathsf{head}(k+1), \rho\rangle)$ is greater than that of $\Pi_{\ell-1}(\mathsf{main}, \langle \sigma_{\mathsf{head}(k+1)}, \mathsf{head}(k+1), \rho\rangle)$ only in a fixed polynomial amount.

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^*.*

In the proof, we consider hybrid experiments in which $\mathcal{S}$ executes $\widetilde{\mathcal{S}}$ with $\mathsf{mode} = \mathsf{hyb}$. Before giving the proof, we remark that when $\mathcal{S}$ executes $\widetilde{\mathcal{S}}$ with $\mathsf{mode} = \mathsf{hyb}$ and $\langle \sigma, \xi, \rho \rangle$ (instead of $\mathsf{mode} = \mathsf{main}$ and $\langle \sigma_{q^d+1}, q^d + 1, \rho_{q^d+1} \rangle$), the transcript between the provers and V^* is simulated in exactly the same way as before except that the ξ-th message is simulated with randomness ρ and then the simulation terminates after the ξ-th message. To see this, observe that when $\mathcal{S}$ executes $\widetilde{\mathcal{S}}$ with $\mathsf{mode} = \mathsf{hyb}$ and $\langle \sigma, \xi, \rho \rangle$, $\widetilde{\mathcal{S}}$ proceeds identically as before in level d until the beginning of the $\tilde{q}$-th child-block and therefore the transcript are simulated identically as before until the $\lfloor \xi - 1 \rfloor_{q^{d-1}}$-th message; furthermore, since the $\tilde{q}$-th child-block proceeds identically as before until the beginning of its $\tilde{q}$-th child-block, the transcript are also simulated identically as before until the $\lfloor \xi - 1 \rfloor_{q^{d-2}}$-th message. In this way, we can see that the transcript are simulated as before until the $\lfloor \xi - 1 \rfloor_{q^0}$-th (i.e., $(\xi - 1)$-th) message. Then, from the construction of $\widetilde{\mathcal{S}}$ that the ξ-th message is simulated as before except that it is computed with randomness ρ.

Proof of Lemma 4. We prove this lemma by considering a sequence of hybrid experiments. Hybrid H_0 is identical with the real execution of V^* and honest provers, hybrid H_{q^d+1} is identical with the execution of $\mathcal{S}$, and hybrid H_i ($i \in [q^d]$) is identical with the execution of $\mathcal{S}$ except for the following.

- $(\mathsf{trans}, \mathsf{T}'_{\mathsf{W}})$ is obtained by executing

  $$\widetilde{\mathcal{S}}(x, z, \widetilde{\mathcal{S}}, d, \varepsilon, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}, \mathsf{hyb}, \langle \sigma_i, i, \rho_i \rangle)$$

  instead of

  $$\widetilde{\mathcal{S}}(x, z, \widetilde{\mathcal{S}}, d, \varepsilon, \mathsf{T}_{\mathsf{UA}}, \mathsf{T}_{\mathsf{W}}, \mathsf{main}, \langle \sigma_{q^d+1}, q^d + 1, \rho_{q^d+1} \rangle) \ ,$$

  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 computed with randomness ρ_i.

- After trans, the simulation of V^*'s view is continued as follows:
  - Every message is computed with true randomness.
  - Every commitment 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 $\widetilde{\mathcal{S}}$ (i.e., by reconstructing the prover state).

From a hybrid argument, it suffices to show that the outputs of each neighboring hybrids are computationally indistinguishable. 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

$$\widetilde{\mathcal{S}}(x, z, \widetilde{\mathcal{S}}, d, \varepsilon, \mathsf{T_{UA}}, \mathsf{T_W}, \mathsf{main}, \sigma_{q^d+1}, q^d + 1, \rho_{q^d+1})$$

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

$$\widetilde{\mathcal{S}}(x, z, \widetilde{\mathcal{S}}, d, \varepsilon, \mathsf{T_{UA}}, \mathsf{T_W}, \mathsf{hyb}, \sigma_{q^d}, q^d, \rho_{q^d}) \ .$$

From the construction of $\widetilde{\mathcal{S}}$, the former execution simulates the q^d messages using the randomness $\rho_1, \ldots, \rho_{q^d}$ that are obtained by $\mathsf{f\text{-}PRG}(\sigma_{q^d+1}, q^d)$, 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 $\mathsf{f\text{-}PRG}(\sigma_{q^d}, q^d - 1)$ and then simulates the q^d-th message using the randomness ρ_{q^d}. Then, since $(\sigma_{q^d}, \rho_{q^d}) = \mathsf{f\text{-}PRG}(\sigma_{q^d+1}, 1)$ in H_{q^d}, it follows from the consistency of $\mathsf{f\text{-}PRG}$ that the messages in the latter execution is simulated using the randomness $\rho_1, \ldots, \rho_{q^d}$ that are obtained by $\mathsf{f\text{-}PRG}(\sigma_{q^d+1}, q^d)$. 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. (In what follows, we use $\widetilde{\mathcal{S}}_{\ell,\kappa}$ to denote the execution of $\widetilde{\mathcal{S}}$ with ℓ and κ; see the proof of Lemma 3.)

Hybrid $H_{i:1}$ is the same as H_i except that $\mathcal{S}$ obtains trans by executing $\widetilde{\mathcal{S}}(x, z, \widetilde{\mathcal{S}}, d, \varepsilon, \mathsf{T_{UA}}, \mathsf{T_W}, \mathsf{hyb}, \sigma_i, i, \rho_i)$ for random σ_i and ρ_i instead of for $(\sigma_i, \rho_i) := \mathsf{f\text{-}PRG}(\sigma_{i+1}, 1)$.

Notice that in $H_{i:1}$, the i-th message is computed in $\widetilde{\mathcal{S}}_{0,i}$ with true randomness instead of pseudorandomness.

Hybrid $H_{i:2}$ is the same as $H_{i:1}$ except that in the execution of $\widetilde{\mathcal{S}}_{0,i}$, if the next message is Com commitments, then the commitments are computed as in the honest prover (i.e., $C_j \leftarrow \mathsf{Com}(0^n)$ for every $j \in [n]$).

Hybrid $H_{i:3}$ is the same as $H_{i:2}$ except that in the execution of $\widetilde{\mathcal{S}}_{0,i}$, if the next message is the first prover message of WIPOK, then subsequently all messages in this WIPOK are computed with a witness for $x \in L$.

From a hybrid argument, it suffices to show the indistinguishability of each neighboring 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 $\mathsf{f\text{-}PRG}$. 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 $\mathsf{f\text{-}PRG}$. On input (σ'_i, ρ'_i), adversary $\mathcal{D}$ internally invokes V^* and simulates H_i for V^* honestly that except for the following.

- Instead of $\widetilde{\mathcal{S}}(x, z, \widetilde{\mathcal{S}}, d, \varepsilon, \mathsf{T_{UA}}, \mathsf{T_W}, \mathsf{hyb}, \sigma_i, i, \rho_i)$ for $(\sigma_i, \rho_i) := \mathsf{f\text{-}PRG}(\sigma_{i+1}, 1)$, $\mathcal{D}$ 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 $\mathsf{f\text{-}PRG}$ 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 next message msg in $\widetilde{\mathcal{S}}_{0,i}$ is Com commitments. Note that both in $H_{i:1}$ and $H_{i:2}$, the Com commitments are generated with true randomness; furthermore, the committed values of Com and the randomness used in Com are not used anywhere else (in particular, not used in the PCP generations 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 next message msg in $\widetilde{\mathcal{S}}_{0,i}$ is the first prover message of WIPOK. Note that both in $H_{i:2}$ and $H_{i:3}$, WIPOK is executed with true randomness that is not used anywhere else; furthermore, from Claim 1, a valid witness is used both in $H_{i:2}$ and $H_{i:3}$. 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 to 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 f-PRG$(\sigma_i, i-1)$ for random σ_i. In addition, the i-th message msg is computed in $\widetilde{\mathcal{S}}_{0,i}$ 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 1, the language Λ_2 shown in Figure 5 is slightly over-simplified and thus we can prove the argument-of-knowledge property of cZKAOK only when $\mathcal{H}$ is collision resistant against poly$(n^{\log\log n})$-time adversaries.

Below, we prove the argument-of-knowledge property assuming that $\mathcal{H}$ is collision resistant against poly$(n^{\log\log n})$-time adversaries. By using a trick shown in [BG08], it is easy to extend this proof so that it works 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 poly$(n^{\log\log n})$*-time adversaries).* For any cheating prover P^*, let us consider the following extractor E.

- Given oracle access to P^*, extractor E interacts with P^* as a honest verifier until the beginning of Stage 3. E then extract a witness w from WIPOK using its extractor.

To show that E outputs a witness for $x \in L$, it suffices to show that w is a witness for $(h, \tau, \overline{C}_1, r_1, \ldots, \overline{C}_{N_{\mathrm{slot}}}, r_{N_{\mathrm{slot}}}) \in \Lambda_1$ only with negligible probability. In the following, we use the word "fake witness" to denote a witness for $(h, \tau, \overline{C}_1, r_1, \ldots, \overline{C}_{N_{\mathrm{slot}}}, r_{N_{\mathrm{slot}}}) \in \Lambda_1$, 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 is a bad P^*, there 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 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 $r_1 \in \{0,1\}^{n^2}$ to P.*

2. ***Generating UA proof.*** *P sends to V the second-round UA message UA_2 of statement $(h, \tau, C, r_1) \in \Lambda_2$. Then, V sends to P random $r_2 \in \{0,1\}^{n^2}$, and P sends to V the fourth-round UA message UA_4.*

3. *We say that P wins in the experiment if $(h, \mathsf{UA}_2, r_2, \mathsf{UA}_4)$ is an accepting UA proof for $(h, \tau, C, r_1) \in \Lambda_2$.*

Then, if there exists a bad P^*, there exists 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 $j^* \in [n]$ such that with probability at least $\delta'(n) \stackrel{\text{def}}{=} \delta(n)/nN_{\mathrm{slot}}^2 > \delta(n)/n^3$, 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 interacts with P^* as a honest verifier of cZKAOK with the following exceptions:
   - In Stage 1, P^{**} forwards h and τ from the external V to the internal P^*.
   - In the i_1^*-th slot of Stage 2, P^{**} forwards $C_{i_1^*, j^*}$ from the internal P^* to the external V and forward r_1 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 r_2.
3. P^{**} rewinds the internal P^* to the point just after P^* sent Com in the i_2^*-th slot. Then, P^{**} sends r_2 to P^* as the verifier message of the i_2^*-th slot, interacts with P^* again as a 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 slot (inclusive). Then, we say that trans is *good* if under the condition that a prefix of the transcript is trans, 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 a honest verifier of cZKAOK. Then, since a transcript of cZKAOK is perfectly emulated in Step 1 of P^{**}, the prefix of the internally emulated transept 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 $C_{i_2^*, j^*}$ and thus we have $\mathsf{UA}_2 = \mathsf{UA}'_2$ except with negligible probability; this means that if w and w' are fake witnesses, $(h, \mathsf{UA}_2, r_2, \mathsf{UA}'_4)$ 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_1) 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 $(h, \tau, C, r_1) \in \Lambda_2$.*

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 sent, i.e., even when the first-round message of UA is sent before the statement is generated. Roughly speaking, the UA extractor by [BG08] 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 [BG08] 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 property of $\mathcal{H}$.

Claim 10. *If there exists $\mathsf{poly}(n^{\log\log n})$-time ITM E^* that wins in $\mathsf{Exp}_2(n, E^*)$ with non-negligible probability, there exists $\mathsf{poly}(n^{\log\log n})$-time machine $\mathcal{A}$ that breaks the collision-resistance property 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 flesh 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 $\epsilon \stackrel{\text{def}}{=} 1/\mathsf{poly}(n)$ 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 a prefix of the transcript is trans, 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 $\mathsf{Com}(h(\Pi); R) = \mathsf{Com}(h(\Pi'); R') = C$.) Next, since r' is chosen randomly after Π is determined, and since we have

$$\left|\left\{r'' \in \{0,1\}^{n^2} \mid \exists y \in \{0,1\}^{\leq 3n} \text{ s.t. } \Pi(y) = r''\right\}\right| \leq 2^{3n+1} \ ,$$

the probability that there exists $y'' \in \{0,1\}^{\leq 3n}$ such that $\Pi(y'') = r'$ is at most $2^{3n+1}/2^{n^2} = \mathsf{negl}(n)$. Then, since we have $\Pi'(y') = r'$, 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 [BG08] 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 [BG08] 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 $x = (x_1, \ldots, x_{|x|}) \in \{0,1\}^{\leq n^{\log\log n}}$, 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 [BG08] 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 [BG08], the language Λ_2 is first modified in 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 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 single position along with their certificate 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 [BG08] works as follows. First, we replace the language Λ_2 in Figure 5 with the one in Figure 6. Next, we modify Claim 9 in 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 a way that, instead of extracting the whole of w and w', the adversary $\mathcal{A}$ extracts Π and Π' 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.

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 $\eta \stackrel{\text{def}}{=} \mathsf{ECC}(\Pi)$
- randomness $R \in \{0,1\}^{\mathsf{poly}(n)}$ for Com
- a string y such that $|y| \leq 3n$

such that

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

Figure 6: A modified version of Λ_2.

5 Acknowledgment

I greatly thank the anonymous reviewers of TCC 2015 for pointing out an error I made in an earlier version of this paper. Their comments also helped me to improve the presentation of this paper.

References

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

[BG08] Boaz Barak and Oded Goldreich. Universal arguments and their applications. *SIAM J. Comput.*, 38(5):1661–1694, 2008.

[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.

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

[CKPR02] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires (almost) logarithmically many rounds. *SIAM J. Comput.*, 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.

[COP+14] Kai-Min Chung, Rafail Ostrovsky, Rafael Pass, Muthuramakrishnan Venkitasubramaniam, and Ivan Visconti. 4-round resettably-sound zero knowledge. In *TCC*, pages 192–216, 2014.

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

[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 J. Comput.*, 25(1):169–192, 1996.

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

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

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

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 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 J. Comput.*, 30(4):1253–1298, 2000.

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

[PPS13] Omkant Pandey, Manoj Prabhakaran, and Amit Sahai. Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for NP. Cryptology ePrint Archive, Report 2013/754, 2013. `http://eprint.iacr.org/`.

[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.