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

January 26, 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 standard assumptions (the existence of one-to-one one-way functions and collision-resistant hash functions), 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 a minor revision of the version that appears in the proceedings of TCC 2015.

1 Introduction

Zero-knowledge proofs and non-black-box simulation. *Zero-knowledge* (ZK) *proofs* [GMR89], with which the prover can convince the verifier of the correctness of a mathematical statement without providing any additional knowledge, have played fundamental roles in cryptography. In particular, ZK protocols[1] 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 in a black-box way. Although such a simulator can get advantage only through the rewinding of the adversary, black-box simulation is known to be powerful enough to construct ZK protocols with a variety of additional properties, security, and efficiency.

Black-box simulation has, however, inherent limitations. For example, let us consider *public-coin* ZK *protocols*—the ZK protocols such that in each round the verifier sends only the outcome of its coin-tossing—and *concurrent* ZK *protocols*—the ZK protocols such that the zero-knowledge property holds even when the adversary concurrently interacts with many provers in an arbitrary schedule. It was shown that public-coin ZK protocols and concurrent ZK protocols can be constructed with black-box simulation techniques [GMW91, RK99, KP01, PRS02]. However, it was also shown that neither of them can be constructed with black-box simulation technique if we additionally require round efficiency. Concretely, it was shown that no *constant-round* public-coin ZK protocol and no $o(\log n/\log\log n)$*-round* concurrent ZK protocol can be proven secure via black-box simulation [GK96, CKPR02]. Furthermore, it was also shown that no public-coin concurrent ZK protocol can be proven secure via black-box simulation irrespective to its round complexity [PTW09].

Because of these impossibility results on black-box simulation, developing *non-black-box simulation* techniques is an important research direction. Developing non-black-box simulation techniques is however considered to be a significantly hard task since non-black-box simulation seems to require the “reverse engineering” of the adversary.

The first non-black-box simulation technique was proposed in a groundbreaking work of Barak [Bar01]. The simulation technique of Barak is completely different from previous ones. In particular, in the simulation technique of Barak, the simulator runs in a “straight-line” manner, i.e., it does not rewind the adversary. With his non-black-box simulation technique, Barak showed that we can go beyond the black-box simulation barrier; in particular, Barak constructed the first constant-round public-coin ZK protocol, which cannot be proven secure via black-box simulation.

Non-black-box simulation in the concurrent setting. Since we can overcome the black-box impossibility result of constant-round public-coin ZK protocols by using Barak’s non-black-box simulation technique, it is natural to expect that we can also overcome other black-box impossibility results by using Barak’s technique. In particular, since Barak’s simulation technique works in a straight-line manner and therefore completely removes the issue of *recursive rewinding* [DNS04], it is natural to think that we can overcome the black-box impossibility results of $o(\log n/\log\log n)$-round concurrent ZK protocols and public-coin concurrent ZK protocols by using Barak’s simulation technique in the concurrent setting.

Unfortunately, Barak’s non-black-box simulation technique does not work in the concurrent setting. Although Barak’s simulation technique can be extended so that it can handle *bounded-concurrent execution* [Bar01] (i.e., a concurrent execution such that there is an a-priori upper-bound on the number of concurrent 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.

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

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. These works then showed that, with their extended simulation techniques, we can 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, Lin, and Paneth [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, Lin, and Pass [CLP13b] constructed a constant-round concurrent ZK protocol by assuming the existence of *$\mathcal{P}$-certificates* (i.e., "succinct" non-interactive proofs/arguments for $\mathcal{P}$), and Pandey, Prabhakaran, and Sahai [PPS13] constructed a constant-round concurrent ZK protocols by assuming the existence of *differing-input indistinguishability obfuscations*.

Additionally, Goyal [Goy13] showed that even in the plain model under standard assumptions, Barak's non-black-box simulation technique can be extended so that it can handle fully concurrent execution. With his simulation technique, then, Goyal constructed 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; thus, in the simulation technique of Goyal the simulator performs straight-line concurrent simulation. Because of this straight-line concurrent simulation property, the simulation technique of Goyal has huge potential. In fact, Goyal notes in [Goy13] that his technique can be used to obtain new results on concurrently secure multi-party computation and blind signatures.

Thus, we currently have several good 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 even in the plain model under standard assumptions [Goy13].[2] However, the state-of-the-art is still not satisfactory and there are 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. 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 an important research direction.

1.1 Our Result

In this paper, we show a new non-black-box simulation technique that works in the fully concurrent setting. With this simulation technique, then, we give a new proof of the following theorem, which was originally proven by Goyal [Goy13].

Theorem. *Assume the existence of one-to-one one-way functions and 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 [Goy13], our simulation technique is based on Barak's simulation technique, can handle fully concurrent execution in the plain model under standard assumptions, and has a simulator that runs in a straight-line manner even in the fully concurrent setting. Our non-black-box simulation technique requires the same hardness assumption and the same round complexity as that of Goyal, and therefore it does not lead to immediate improvement over the result of Goyal. Nonetheless, our simulation technique is meaningful since it is different from the technique of Goyal and the analysis of it is (in our opinion) simpler than the analysis of Goyal's technique. Since

[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.

there is only a limited number of non-black-box simulation techniques that can handle fully concurrent execution (and in particular there is only one straight-line concurrent simulation technique in the plain model under standard assumptions), constructing a new non-black-box simulation technique in the concurrent setting is meaningful even when there is no improvement. 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, assuming familiarity 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). In any of these slots, the simulator can generate a PCP-proof as a trapdoor witness for the universal argument (UA). Thus, with multiple slots, the simulator can choose which slot to use in the generation of the PCP-proof, and therefore by using a good "proving strategy" that determines which slot to use, the simulator can avoid the blow-up of its running time and can generate PCP-proofs in all sessions in polynomial time even in the concurrent setting. The proving strategy that CLP uses 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 transcript is recursively divided into blocks and then PCP-proofs are generated only at the end of the blocks.

A problem that CLP encountered is that there is only one opportunity for the simulator to give a UA-proof in each session and therefore the simulator need to remember all previously generated PCP-proofs during the simulation. 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 its running time. (Our proving strategy works so that a PCP-proof that is generated at the end of a block is used only until the end of the "parent block" of this block; thus, the simulator need to remember PCP-proofs 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, when we show that every session is successfully simulated, we need to use only a simple counting argument. Because of this, the analysis of our simulation technique is quite simple.

2 Overview of Our Technique

As briefly described 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 $\mathsf{BarakZK}$ proceeds as follows.

Protocol $\mathsf{BarakZK}$

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 non-interactive commitment scheme. 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 $\mathsf{BarakZK}$ is proven as follows. The soundness is proven by showing that $\Pi(c) \neq r$ holds with overwhelming probability when a cheating prover P^* commits to $h(\Pi)$ for a machine Π. The zero-knowledge property is proven by using a simulator that commits to $h(\Pi)$ such that Π is a machine that emulates the cheating verifier V^*; since $\Pi(c) = V^*(c) = r$ holds 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 $\mathsf{BarakZK}$ 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 $\mathsf{BarakZK}$ is to use a simulator $\mathcal{S}$ that commits to a machine that emulates $\mathcal{S}$ itself. A key observation behind this approach is that although V^* can receive unbounded number of messages during a slot, all of these messages are generated by $\mathcal{S}$. Thus, if the committed machine Π emulates $\mathcal{S}$ 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 (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.

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

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

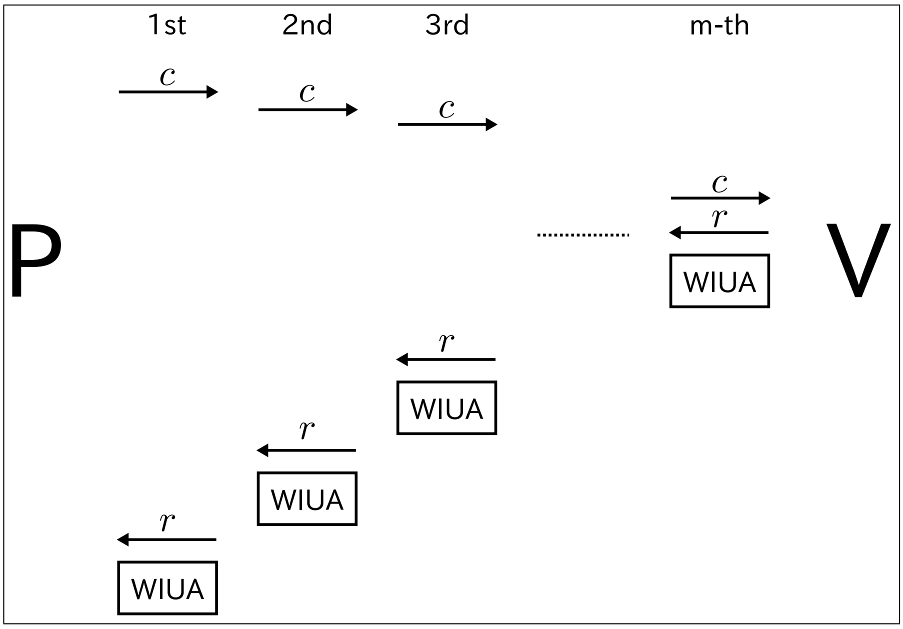

Figure 1: The "nested concurrent sessions" schedule.

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. As explained later, this online/offline property enables $\mathcal{S}$ to perform all heavy computations only at specific points during the simulation, which is crucial to avoid the nested computation in WIUA. 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 queries Q and sends them to V. We denote these replies by UA_4.

[4] Recall that the PCP-verifier performs the verification by making a few queries to the PCP-proof.

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. 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 [CLP13a] 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 $i \in [N_{\text{slot}}]$ in sequence, P and V do the following.

- P sends $c_i \leftarrow \mathsf{Com}(0^n)$ to V. Then, V sends a random string r_i 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 $i \in [N_{\text{slot}}]$ 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 of 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 proving strategy that determines which slot to use as a witness. Thus, roughly speaking, the simulation proceeds 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 a 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. As a proving strategy that determines which slot to use as a witness, CLP considered 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.
- 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.

[5] Here, P is assumed to have random access to π.

- 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 since the size of $\mathcal{S}$'s state can become super-polynomial. Recall that in the simulation, $\mathcal{S}$ generates an offline proof in Stage 2 and uses it in Stage 3. Then, since V^* can choose any concurrent schedule (and therefore can delay the execution of Stage 3), $\mathcal{S}$ need to remember all previously generated offline proofs during its execution. Thus, each committed machine 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 [CLP13a] 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 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 sessions.

In this way, CLP [CLP13a] 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 machines so 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.

(Recall that the maximum level of the blocks is $d < n$.) 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 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.

(A machine that emulates $\mathcal{S}$ from the beginning of a block is already considered in [CLP13a] for a different purpose. In [CLP13a], such a machine is used to reduce the round complexity. Here, we use such a machine to avoid the blow-up of the simulator's size.)

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, let t_i be the maximum time spent by the simulation of a level-i block. (Thus, 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

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

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 may 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 this block starts. (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 of WIPOK before 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 witness of WIPOK 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 $i \in [N_{\mathrm{slot}}]$ 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 $i_1, i_2 \in [N_{\mathrm{slot}}]$, $j \in [n]$, 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.

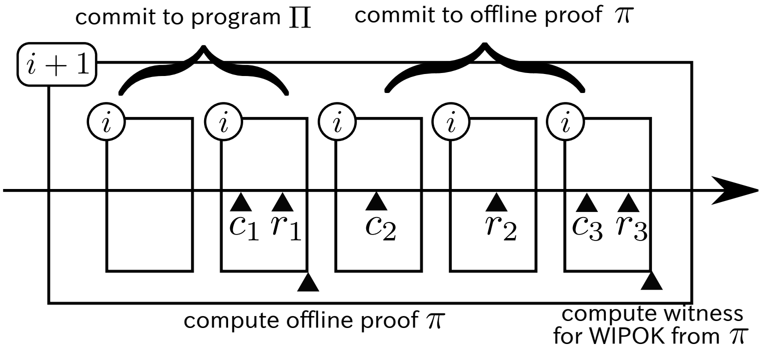
commit to program Π
commit to offline proof π
$i+1$
i
c_1 r_1
c_2
r_2
c_3 r_3
compute offline proof π
compute witness for WIPOK from π

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

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.

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

Toward the final protocol. To obtain a formal proof of security, we need to add a slight modification to the above protocol. In particular, as pointed out in previous 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. If we use the technique of [CLP13a, CLP13b], which requires only one-way functions, we can remove the requirement of one-to-one one-way functions from our result. However, to simplify the analysis, in this paper we use the technique of [Goy13], which requires one-to-one 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 interactive proofs of knowledge, witness-indistinguishable proofs, and commitment schemes.

3.1 Notations

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

3.2 Assumptions

We assume the existence of a family of collision-resistant hash functions $\mathcal{H} = \{h_\alpha\}_{\alpha \in \{0,1\}^*}$. Let $\mathcal{H}_n \overset{\text{def}}{=} \{h_\alpha \in \mathcal{H} : \alpha \in \{0,1\}^n\}$. Then, we require that $\mathcal{H}_n$ satisfies the following properties.

- For any $h \in \mathcal{H}_n$, the domain of h is $\{0,1\}^*$ and the range of h is $\{0,1\}^n$.
- For any $h \in \mathcal{H}_n$, $x \in \{0,1\}^{\mathsf{poly}(n)}$, and $i \in \{1, \ldots, |x|\}$, after computing $h(x)$, we can compute a short certificate $\sigma_i \in \{0,1\}^{n^2}$ for the fact that the i-th bit of x is x_i.

We can achieve these properties by using the Merkle hash tree.

We also assume the existence of one-to-one one-way function f. Recall that from one-to-one one-way function f, we can construct perfectly binding non-interactive commitment scheme Com, where $\mathsf{Com}(b; r) = (f(r), \mathsf{hc}(r) \oplus b)$ for $b \in \{0,1\}$ and the hard-core bit hc of f. (See, e.g., [Gol01].) Note that each valid commitment of Com has a unique decommitment.

3.3 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 $(1^n, x, z)$, V^* concurrently interacts with $m(n)$ 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 $\mathsf{view}_{V^*}\langle P(w), V^*(z)\rangle(1^n, x)$ be the view of V^* in the above concurrent execution, where 1^n and $x \in L$ are 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 following are computationally indistinguishable.

- $\{\mathsf{view}_{V^*}\langle P(w), V^*(z)\rangle(1^n, x)\}_{n\in\mathbb{N}, x\in L\cap\{0,1\}^{\mathsf{poly}(n)}, w\in\mathbf{R}_L(x), z\in\{0,1\}^*}$
- $\{\mathcal{S}(1^n, x, z)\}_{n\in\mathbb{N}, x\in L\cap\{0,1\}^{\mathsf{poly}(n)}, w\in\mathbf{R}_L(x), z\in\{0,1\}^*}$

◇

3.4 PCP and Universal Argument

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

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

For simplicity, we show the definitions of PCPs and universal arguments that prove only the membership of a single "universal" language $L_{\mathcal{U}}$. For triplet $y = \langle M, x, t\rangle$, 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 $y = \langle M, x, t\rangle \in L_{\mathcal{U}}$ 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 $L_{\mathcal{U}}$ via mapping $x \mapsto \langle M_L, x, 2^{|x|}\rangle$, where M_L is any fixed non-deterministic polynomial-time machine that decides L. Thus, a proof system for $L_{\mathcal{U}}$ allows us to handle all $\mathcal{NP}$ statements.[11]

3.4.2 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 $n \in \mathbb{N}$ and every $y \in L_{\mathcal{U}} \cap \{0,1\}^{\mathsf{poly}(n)}$, there exists an oracle π such that
$$\Pr\left[V^{\pi}(1^n, y) = 1\right] = 1 \ .$$
- **Soundness:** For every $n \in \mathbb{N}$, every $y \in \{0,1\}^{\mathsf{poly}(n)} \setminus L_{\mathcal{U}}$, and every oracle π, there exists a negligible function $\mathsf{negl}(\cdot)$ such that
$$\Pr\left[V^{\pi}(1^n, y) = 1\right] < \mathsf{negl}(n) \ .$$

◇

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.

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

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

$\diamond$

3.4.3 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 $y = \langle M, x, t\rangle \in L_{\mathcal{U}}$, 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 $y = \langle M, x, t\rangle$, the total time spent by (probabilistic) verifier strategy V on inputs 1^n and y is at most $p(n + |y|)$.
- **Completeness by a relatively efficient prover:** For every $n \in \mathbb{N}$, $y = \langle M, x, t\rangle \in L_{\mathcal{U}} \cap \{0, 1\}^{\mathsf{poly}(n)}$, and $w \in \mathbf{R}_{\mathcal{U}}(y)$, it holds that

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

  Furthermore, there exists a polynomial q such that the total time spent by P, on input $(1^n, y, w)$, is at most $q(n + |y| + T_M(x, w)) \leq q(n + |y| + t)$.

- **Computational Soundness:** For every PPT Turing machine P^*, there is a negligible function $\mathsf{negl}(\cdot)$ such that for every $n \in \mathbb{N}$, $y = \langle M, x, t\rangle \in \{0, 1\}^{\mathsf{poly}(n)} \setminus L_{\mathcal{U}}$, and $z \in \{0, 1\}^*$, it holds that

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

- **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 large $n \in \mathbb{N}$, every $y = \langle M, x, t\rangle \in \{0, 1\}^{\mathsf{poly}(n)}$, and every $z \in \{0, 1\}^*$, if $\Pr\left[\langle P^*(z), V\rangle(1^n, y) = 1\right] > 1/p(n)$, then

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

  where $E_r^{P^*(1^n, y, z)}(\cdot, \cdot, \cdot)$ denotes the function defined by fixing the randomness of E to equal r, and providing the resulting E_r with oracle access to $P^*(1^n, y, z)$. $\diamond$

The weak proof-of-knowledge property of universal arguments only guarantees that each individual bit w_i of some witness w can be extracted in probabilistic polynomial time. Given an input 1^n and $y = \langle M, x, t\rangle \in L_{\mathcal{U}} \cap \{0, 1\}^{\mathsf{poly}(n)}$, since the witness $w \in \mathbf{R}_{\mathcal{U}}(y)$ is of length at most t, it follows that there exists an extractor running in time polynomial in $\mathsf{poly}(n) \cdot t$ 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, and in the offline stage, the running time of the prover is $\mathsf{poly}(n + |y| + T_M(x, w))$. In this paper, the third message UA_3 satisfies $|\mathsf{UA}_3| = n \cdot \mathsf{poly}(\log |y|) \leq n^2$ and the fourth message UA_4 satisfies $|\mathsf{UA}_4| = \mathsf{poly}(n)$.

- **Input:** The common input of the prover P and the verifier V is $y = \langle M, x, t\rangle \in L_{\mathcal{U}}$. The private input of P is $w \in \mathbf{R}_{\mathcal{U}}(y)$.
- **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 $\mathsf{UA}_4 := \{(i, \pi_i, \sigma_i)\}_{i\in Q}$ to V, where π_i is the i-th bit of π and σ_i 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].

4 Our Public-Coin Concurrent Zero-Knowledge Argument

Theorem 1. *Assume the existence of one-to-one one-way functions and 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 the following building blocks are used in cZKAOK.

- Perfectly binding non-interactive commitment scheme Com such that each valid commitment of Com has a unique decommitment. (As noted in Section 3.2, such a commitment scheme can be constructed from one-to-one one-way functions.)
- Constant-round public-coin witness-indistinguishable proof of knowledge WIPOK.
- Four-round public-coin universal argument UA of [BG08] (Figure 3 in Section 3.4.3).

Clearly, cZKAOK is public-coin and its round complexity is $O(n^\epsilon)$. 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.

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.

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.

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

1. P computes $C_{i,j} \leftarrow \mathsf{Com}(0^n)$ for each $j \in [n]$. Then, P sends $\vec{C_i} = (C_{i,1}, \ldots, C_{i,n})$ 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
- $\langle h, \vec{C}_1, r_1, \ldots, \vec{C}_{N_{\text{slot}}}, r_{N_{\text{slot}}} \rangle \in \Lambda_1$, where language Λ_1 is shown in Figure 5.

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

□

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 $\mathsf{poly}(T(n))$-time adversaries. To make it work assuming collision resistance against polynomial-time adversaries, one should use a "good" error-correcting code ECC (i.e., with constant relative distance and with polynomial-time encoding and decoding), and replacing the condition $C = \mathsf{Com}(h(\Pi); R)$ with $C = \mathsf{Com}((|\mathsf{ECC}(\Pi)|, h(\mathsf{ECC}(\Pi))); R)$.

4.1 Concurrent Zero-knowledge Property

Proof of Lemma 1. Let V^* be a cheating verifier. Without loss of generality, we assume that V^* is deterministic. Let $m(\cdot)$ be a polynomial such that V^* invokes $m(n)$ concurrent sessions. Let $q \overset{\text{def}}{=} n^{\epsilon/2}$. We assume without loss of generality that in the interaction between V^* and provers, the total number of messages across all sessions is always the power of q (i.e., it is q^d for an integer d). Note that since the number of messages is polynomially bounded, we have $d = \log_q(\mathsf{poly}(n)) = O(1)$.

4.1.1 Simulator $\mathcal{S}$

Before describing our simulator $\mathcal{S}$, we first introduce *blocks*. The level-d block is defined to be the entire transcript of all sessions. Then, for $\ell \in \{0, \ldots, d-1\}$, level-$\ell$ blocks are defined by dividing each level-$(\ell+1)$ block into q sequential blocks of equal length. Thus, for every $\ell \in \{0, \ldots, d\}$, a level-$\ell$ block contains q^{ℓ} messages.

We next introduce a subroutine SOLVE, which generates a simulated transcript by recursively executing itself. (On input $\ell \in [d]$, SOLVE generates a simulated transcript of a level-ℓ block by executing itself q times on input $\ell - 1$.) The details of SOLVE is described below.

We give SOLVE an oracle access to O, where O is the oracle that is defined in the definition of Λ_2 (Figure 5). Roughly speaking, we give SOLVE an access to O to avoid the issue of randomness

Language Λ_1: (statement for WIPOK)

$\langle h, \vec{C}_1, r_1, \ldots, \vec{C}_{N_{\text{slot}}}, r_{N_{\text{slot}}} \rangle \in \Lambda_1$ if and only if there exist

- $i_1, i_2 \in [N_{\text{slot}}]$ 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

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

Language Λ_2:

Let $T(\cdot)$ be a slightly super-polynomial function (say, $T(n) = n^{\log\log n}$). Then, $\langle h, C, r \rangle \in \Lambda_2$ if and only if there exist

- an oracle machine Π such that $|\Pi| \leq T(n)$
- a string τ such that $|\tau| \leq T(n)$
- randomness $R \in \{0,1\}^{\mathsf{poly}(n)}$ for Com
- a string y such that $|y| \leq T(n)$

such that

- $C = \mathsf{Com}(h(\Pi); R)$, and
- r is a substring of τ, and
- Π^O outputs τ within $T(n)$ steps, where O is an oracle that receives a commitment of Com and returns the (unique) decommitment of this commitment, and
- In the execution of Π^O, for every query $\tilde{\rho}$ from Π to O, there exists $(\tilde{\rho}, \rho, r) \in y$ such that $\tilde{\rho} = \mathsf{Com}(\rho; r)$ (i.e., (ρ, r) is the decommitment of $\tilde{\rho}$).

Figure 5: Languages used in cZKAOK.

sketched in Section 2.2.[12] Specifically, we give SOLVE only "encrypted randomness" $\tilde{\rho}$, which is a Com commitment to which true randomness ρ is committed. SOLVE computes the committed value ρ from $\tilde{\rho}$ by using $\mathcal{O}$ and then uses ρ as randomness in the simulation of Com and WIPOK. (When SOLVE is used in $\mathcal{S}$, oracle $\mathcal{O}$ is emulated by $\mathcal{S}$ in polynomial time.)

We also give SOLVE the following input $(x, z, \ell, \mathsf{trans}, \mathsf{V}, \mathsf{W}, \tilde{\rho})$:

- x and z are the input of V^*.
- $\ell \in \{0, \dots, d\}$ is an execution level.
- $\mathsf{trans} \in \{0,1\}^{\mathsf{poly}(n)}$ is a partial transcript that was simulated so far. The goal of SOLVE is to simulate subsequent q^ℓ messages after trans.
- $\mathsf{V} = \{v_{s,j}\}_{s\in[m],j\in[n]}$ is the values to be committed in Com in the simulation. (In a slot of the s-th session, $v_{s,1}, \dots, v_{s,n}$ are committed.) Each $v_{s,j}$ is $\bot$ if the value to be committed has not been determined yet.
- $\mathsf{W} = \{w_s\}_{s\in[m]}$ is the WIPOK witnesses that are computed so far. Each w_s is $\bot$ if the witness of the s-th session has not been computed yet.
- $\tilde{\rho} = (\tilde{\rho}_1, \dots, \tilde{\rho}_{q^d})$ is a vector of "encrypted" randomness (i.e., commitments of Com), where the randomness decrypted from $\tilde{\rho}_i$ is used to simulate the i-th messages.

The output of SOLVE is $(\mathsf{trans}', \mathsf{W}', \Pi^{\langle\cdot\rangle})$, where trans' is the simulated messages that are generated by this execution of SOLVE, W' is the updated table of the WIPOK witnesses, and $\Pi^{\langle\cdot\rangle}$ is a machine that emulates this execution of SOLVE and outputs trans'.

Given the above input, SOLVE does the following. Below, we use a function car that takes a triple (a, b, c) as input and outputs a.

$\underline{\mathsf{SOLVE}^{\mathcal{O}}(x, z, \ell, \mathsf{trans}, \mathsf{V}, \mathsf{W}, \tilde{\rho})}$

- When $\ell = 0$ (the base case), do the following.
  - If the next-scheduled message msg is a verifier message, feed (x, z, trans) to V^* and receive msg from V^*.
  - If the next-scheduled message msg is a prover message, do the following. Let $\mathsf{init} \in [q^d]$ be the index of msg across all sessions (thus, trans contains $\mathsf{init} - 1$ messages). Parse $(\tilde{\rho}_1, \dots, \tilde{\rho}_{q^d}) \leftarrow \tilde{\rho}$ and compute the committed value ρ_{init} of $\tilde{\rho}_{\mathsf{init}}$ by using $\mathcal{O}$. Then, compute msg as follows with randomness ρ_{init}.
    - $*$ If msg is a message of Com in the s-th session for $s \in [m]$, compute the following for each $j \in [n]$.

$$C_j \leftarrow \begin{cases} \mathsf{Com}(v_{s,j}) & \text{if } v_{s,j} \neq \bot \\ \mathsf{Com}(0^n) & \text{otherwise} \end{cases}$$

      Then, set $\mathsf{msg} := (C_1, \dots, C_n)$.
    - $*$ If msg is the first message of WIPOK in the s-th session for $s \in [m]$, honestly compute msg by using witness w_s. (If w_s is not a valid witness, aborts with output stuck.) If msg is another message of WIPOK, honestly compute msg by reconstructing the prover state of WIPOK from trans, W, and $\tilde{\rho}$.

[12] This technique is borrowed from [Goy13].

- Output $(\mathsf{msg}, \mathsf{W}, \Pi^{\langle\cdot\rangle})$, where $\Pi^{\langle\cdot\rangle}$ is a machine that computes $\mathsf{car}(\mathsf{SOLVE}^{\langle\cdot\rangle}(x, z, \ell, \mathsf{trans}, \mathsf{V}, \mathsf{W}, \tilde{\rho}))$.

- When $\ell > 0$, do the following.

  Step 1: Updating values to be committed. Let $\Pi^{\langle\cdot\rangle}$ be a machine that computes $\mathsf{car}(\mathsf{SOLVE}^{\langle\cdot\rangle}(x, z, \ell, \mathsf{trans}, \mathsf{V}, \mathsf{W}, \tilde{\rho}))$. Then, for every $s \in [m]$ such that $v_{s,\ell} = \perp$ holds and the s-th session has already started in trans, update V by setting $v_{s,\ell} := h_s(\Pi)$, where h_s is the hash function used in the s-th session.

  Step 2: Initializing temporary variables. Let $\mathsf{ctr}_s := 0$ and $\mathsf{tmp}_s := \perp$ for every $s \in [m]$.

  Step 3 to Step $2q + 2$: For each $k \in [q]$, do the following:

  Step $2k + 1$: Executing the k-th child-block. Compute

  $$(\mathsf{trans}_k, \mathsf{W}_k, \Pi_k) \leftarrow \mathsf{SOLVE}^{O}(x, z, \ell - 1, \mathsf{trans}, \mathsf{V}, \mathsf{W}, \tilde{\rho}) \ .$$

  Then, update $\mathsf{trans} := \mathsf{trans} \,\|\, \mathsf{trans}_k$ and $\mathsf{W} := \mathsf{W}_k$. Let y_k be the set of all query-answer pairs with O during this recursive execution.

  Step $2k + 2$: For each $s \in [m]$ such that (i) trans_k includes a slot sl of the s-th session and (ii) the s-th session has started before trans_k , do the following.[13]

  Case 1. When $\mathsf{ctr}_s = 0$, do the following.

  1. Let i_1 be the *slot-index* of slot sl (i.e., $i_1 \in [N_{\mathrm{slot}}]$ s.t. slot sl is the i_1-th slot of the s-th session). Let $(\vec{C}_{i_1}, r_{i_1})$ denote slot sl, where $\vec{C}_{i_1} = (C_{i_1,1}, \ldots, C_{i_1,n})$.
  2. **Computing offline proof.** By using $\tilde{\rho}$ and O, compute the randomness R_1 used for $C_{i_1,\ell-1}$. Then, compute PCP-proof π_s of statement $\langle h_s, C_{i_1,\ell-1}, r_{i_1}\rangle \in \Lambda_2$ with witness $\langle \Pi_k, \mathsf{trans}_k, R_1, y_k\rangle$ and compute $\mathsf{UA}_2 := h_s(\pi_s)$.
     COMMENT: SOLVE *and* $\mathcal{S}$ *are designed so that the committed value of* $C_{i_1,\ell-1}$ *is* Π_k*, which outputs* trans_k *by emulating the recursive execution of* SOLVE *of Step* $2k + 1$.
  3. **Updating values to be committed.** Update V by setting $v_{s,\ell-1} := h_s(\pi_s)$.
  4. Update $\mathsf{tmp}_s := (i_1, \pi_s, \mathsf{UA}_2)$.
  5. Update $\mathsf{ctr}_s := \mathsf{ctr}_s + 1$.

  Case 2: When $\mathsf{ctr}_s = 1$, do the following.

  1. Let i_2 be the slot-index of slot sl and let $(\vec{C}_{i_2}, r_{i_2})$ denote slot sl, where $\vec{C}_{i_2} = (C_{i_2,1}, \ldots, C_{i_2,n})$.
  2. **Completing UA.** Parse $(i_1, \pi_s, \mathsf{UA}_2) \leftarrow \mathsf{tmp}_s$. Then, compute the fourth UA message UA_4 from offline proof $(h_s, \pi_s, \mathsf{UA}_2)$ and the third UA message r_{i_2}.
  3. Let R_2 be the randomness used for $C_{i_2,\ell-1}$. Then, update W by setting $w_s := \langle i_1, i_2, \ell - 1, \mathsf{UA}_2, \mathsf{UA}_4, R_2\rangle$.
     COMMENT: SOLVE *and* $\mathcal{S}$ *are designed so that the committed value of* $C_{i_2,\ell-1}$ *is* $\mathsf{UA}_2 = h_s(\pi_s)$.
  4. Update $\mathsf{ctr}_s := \mathsf{ctr}_s + 1$.

  Step $2q + 3$: Output $(\mathsf{trans}_1 \,\|\, \mathsf{trans}_2 \,\|\cdots\|\, \mathsf{trans}_q, \mathsf{W}, \Pi^{\langle\cdot\rangle})$.

With SOLVE, the simulator $\mathcal{S}$ is defined as follows.

[13] We note that if trans_k includes a slot, we have $|\mathsf{trans}_k| \geq 1$ and thus we have $\ell \geq 2$.

$\underline{\mathcal{S}(1^n, x, z)}$

1. For every $i \in [q^d]$, choose random $\rho_i \in \{0,1\}^n$ and $r \in \{0,1\}^{\mathsf{poly}(n)}$, and compute $\tilde{\rho}_i := \mathsf{Com}(\rho_i; r)$. Let $\tilde{\rho} := (\tilde{\rho}_1, \ldots, \tilde{\rho}_{q^d})$.
2. Let $\mathsf{V} := \{v_{s,j}\}_{s\in[m],j\in[n]}$ and $\mathsf{W} := \{w_s\}_{s\in[m]}$, where $v_{s,j} = w_s = \perp$ for every $s \in [m]$ and $j \in [n]$.
3. Compute $(\mathsf{trans}', \mathsf{W}', \Pi^{\langle\cdot\rangle}) \leftarrow \mathsf{SOLVE}^{\langle\cdot\rangle}(x, z, d, \varepsilon, \mathsf{V}, \mathsf{W}, \tilde{\rho})$, where ε is an empty string. When SOLVE queries $\tilde{\rho}$ for $\mathcal{O}$, find i such that $\tilde{\rho} = \mathsf{Com}(\rho_i; r_i)$ and return (ρ_i, r_i) to SOLVE.[14]
4. Output trans'.

4.1.2 Running Time of $\mathcal{S}$

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

Proof. We first show that in each execution of SOLVE, for any $k \in [q]$, the size of Π_k in Step $2k + 1$ is bounded by a fixed polynomial in n. From the construction of SOLVE, Π_k is a machine that computes $\mathsf{car}(\mathsf{SOLVE}^{\langle\cdot\rangle}(x, z, \ell-1, \mathsf{trans}, \mathsf{V}, \mathsf{W}, \tilde{\rho}))$. Then, since the length of $(x, z, \ell-1, \mathsf{V}, \mathsf{W}, \mathsf{trans}, \tilde{\rho})$ is bounded by a fixed polynomial in m, the size of Π_k is bounded by a fixed polynomial in m. Thus, the size of Π_k is bounded by a fixed polynomial in n.

We then bound the running time of $\mathcal{S}$ as follows. Note that from the constructions of $\mathcal{S}$ and SOLVE, each execution of SOLVE can be uniquely identified by the value of ℓ and $\mathsf{init} \overset{\text{def}}{=} |\mathsf{trans}| + 1$. For $\ell \in \{0, \ldots, d\}$ and $\mathsf{init} \in [q^d]$, we use $\mathsf{SOLVE}_{\ell,\mathsf{init}}$ to denote the execution of SOLVE with input ℓ and init. Let $t_{\ell,\mathsf{init}}$ be the running time of $\mathsf{SOLVE}_{\ell,\mathsf{init}}$, and let $t_\ell \overset{\text{def}}{=} \max_{\mathsf{init}}(t_{\ell,\mathsf{init}})$. Note that in every $\mathsf{SOLVE}_{\ell,\mathsf{init}}$, the running time of the recursive execution of SOLVE in Step $2k + 1$ is at most $t_{\ell-1}$; thus, in Step $2k + 2$, PCP-proof π_s can be computed in time $\mathsf{poly}(t_{\ell-1})$ and the length of π_s is at most $\mathsf{poly}(t_{\ell-1})$ for every $s \in [m]$. Note also that every computation in $\mathsf{SOLVE}_{t,\mathsf{init}}$ can be performed in fixed polynomial time in n except for the following computations:

Type-1 computation. The recursive executions of SOLVE.

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 $t_{\ell-1}$, and each type-2 computation can be performed in time $\mathsf{poly}(t_{\ell-1})$. Then, since for each $k \in [q]$ there are a single type-1 computation and m type-2 computations, we have

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

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

4.1.3 Indistinguishability of Views

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

Proof. We prove this lemma by considering a sequence of hybrid experiments. Let H_0 be the real execution of V^* and honest provers. Then, for each $i \in [q^d]$, we consider the following hybrids. In what follows, we use $\mathsf{SOLVE}_{\ell,\mathsf{init}}$ to denote the execution of SOLVE with input ℓ and init (see the proof of Lemma 3).

[14] From the construction of SOLVE, there must exist such i.

Hybrid H_i proceeds identically to the execution of $\mathcal{S}$ until the end of the execution of $\mathsf{SOLVE}_{0,i}$. At this point, the view of V^* is simulated up until the i-th message across all sessions (inclusive). Let $(\mathsf{msg}, \mathsf{W}, \Pi^{\langle\cdot\rangle})$ be the output of $\mathsf{SOLVE}_{0,i}$, and let trans be the simulated view of V^* at this point (including msg). Then, after $\mathsf{SOLVE}_{0,i}$ is executed, the view of V^* is simulated from trans as follows: Every message is computed with true randomness (instead of "decrypted" randomness), every commitment is generated by committing to 0^n, every WIPOK that starts after trans is executed with witness for $x \in L$, and every WIPOK that starts in trans is executed as in SOLVE (i.e., by reconstructing the prover state). The output of H_i is the simulated view of V^*.

Note that the output of H_{q^d} is identical to that of $\mathcal{S}$.

To show the indistinguishability between the output of H_i and that of H_{i-1} for each $i \in [q^d]$, we consider a sequence of intermediate hybrid experiments in which H_i is gradually changed to H_{i-1} as follows.

Hybrid $H_{i:1}$ is the same as H_i except that in the execution of $\mathsf{SOLVE}_{0,i}$, the next message msg is computed with true randomness (instead of the one decrypted from $\tilde{\rho}_i$).

Hybrid $H_{i:2}$ is the same as $H_{i:1}$ except that in the execution of $\mathsf{SOLVE}_{0,i}$, 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 $\mathsf{SOLVE}_{0,i}$, if the next message is the first message of WIPOK, then subsequently all messages in this WIPOK are computed with witness for $x \in L$.

Before showing the indistinguishability among these intermediate hybrids, we show the following claim.

Claim 1. *In H_{q^d}, $\mathcal{S}$ does not output* stuck.

Proof. We first introduce notation. Recall that in hybrid H_{q^d}, SOLVE is recursively executed many times. We use *block* to denote an execution of SOLVE. A block is in level ℓ if the corresponding SOLVE is executed with input ℓ. For each block, the *child-blocks* of this block are 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 SOLVE 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 (i) contains a slot of this session, (ii) begins after this session begins, and (iii) does not contain the first message of WIPOK of this session.[15]

From the construction, $\mathcal{S}$ does not output 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, Case 2 of Step $2k + 2$ is executed in this block for this session and therefore a witness of WIPOK is computed.)

Thus, it remains to show that for every session that reaches Stage 3, there exists a block that has two good child-blocks. To show this, it suffices to show that for every session that reaches Stage 3, there exists a block that has four child-blocks that contain slots of this session. (If four child-blocks contain slots, two of them are good since they begin after this session begins and they do not contain the first message of WIPOK.) Assume for contradiction that there exists a session such

[15] To simplify the description of our simulator, we use a definition of the good block that is slightly different from that given in the overview of our technique (Section 2). (Here, we additionally require that the block begins after the session begins.) We can also use the definition given Section 2 if we modify SOLVE so that the machine that emulates itself is committed even in the sessions that start after it begins.

that every block has at most three child-blocks that contain slots of this session. For $\ell \in \{0, \ldots, d\}$ and $\mathsf{init} \in [q^d]$, let $C_{\mathsf{init}}(\ell)$ be the number of slots of this session that are contained by the block corresponding to $\mathsf{SOLVE}_{\ell,\mathsf{init}}$, and let $C(\ell) \overset{\text{def}}{=} \max_{\mathsf{init}}(C_{\mathsf{init}}(\ell))$. Then, since each block contains at most three child-blocks that contain slots, and since in each block there are at most $q-1$ slots that are contained by this block but do not contained by any of its child-block, we have

$$C(\ell) \leq 3C(\ell-1) + q - 1 \ .$$

Thus, we have

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

From $d = O(1)$ and $C(0) = 0$, we have $C(d) = O(q)$. Then, since $\mathcal{S}$ outputs the view of V^* that was 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. □

Now, we are ready to show the indistinguishability among the intermediate hybrids.

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

Proof. Recall that $H_{i:1}$ differs from H_i in that ρ_i is replaced with true randomness that is independent of $\tilde{\rho}$. Note that if we replace ρ_i with true randomness, we can no longer compute PCP-proofs w.r.t. $\mathsf{SOLVE}_{0,i}$ and any execution of SOLVE that contains $\mathsf{SOLVE}_{0,i}$. However, since in both H_i and $H_{i:1}$ no PCP-proof is computed after $\mathsf{SOLVE}_{0,i}$, this causes no problem. Thus, the indistinguishability follows from the hiding property of Com. □

Claim 3. *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 $\mathsf{SOLVE}_{0,i}$ 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 4. *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 $\mathsf{SOLVE}_{0,i}$ is the first message of WIPOK. Note that both in $H_{i:2}$ and $H_{i:3}$, WIPOK are 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 $H_{i:3}$. (Recall that $H_{i:2}$ and $H_{i:3}$ proceed identically with H_{q^d} until $\mathsf{SOLVE}_{0,i}$ starts.) Thus, the indistinguishability follows from the witness indistinguishability of WIPOK. □

Claim 5. *For every $i \in [q^d]$, the output of H_{i-1} is identically distributed to that of $H_{i:3}$.*

Proof. Note that in $H_{i:3}$, the next message msg in $\mathsf{SOLVE}_{0,i}$ is computed in exactly the same way as in H_{i-1}. Thus, the claim follows. □

From Claims 2, 3, 4, and 5, the output of H_0 and that of H_{q^d} are computationally 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; in particular, the argument-of-knowledge property of cZKAOK can be proven only when $\mathcal{H}$ is collision resistant against $\mathsf{poly}(T(n))$-time adversaries.

Below, we prove the argument-of-knowledge property assuming that $\mathcal{H}$ is collision resistant against $\mathsf{poly}(T(n))$-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.

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

- Given oracle access to P^*, E interacts with P^* as a honest verifier until the start of Stage 3. Then, E uses the extractor of WIPOK to extract a witness w.

To show that E outputs a witness of $x \in L$, it suffices to show that w is a witness of $\langle h, \vec{C}_1, r_1, \ldots, \vec{C}_{N_{\mathrm{slot}}}, r_{N_{\mathrm{slot}}} \rangle \in \Lambda_1$ with at most negligible probability. In the following, we use the word "fake witness" to denote a witness for $\langle h, \vec{C}_1, r_1, \ldots, \vec{C}_{N_{\mathrm{slot}}}, r_{N_{\mathrm{slot}}} \rangle \in \Lambda_1$. Then, we say that P^* is *bad* if E outputs a fake witness with non-negligible probability. In the following, 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.

Claim 6. *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.*** *First, V sends random $h \in \mathcal{H}_n$ to P. Next, 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* UA *message* UA_2 *of statement $\langle h, C, r_1 \rangle \in \Lambda_2$, and V sends to P random $r_2 \in \{0,1\}^{n^2}$. Then, P sends to V the fourth* 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 $\langle h, C, r_1 \rangle \in \Lambda_2$.*

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 $\delta(n) \overset{\text{def}}{=} 1/\mathsf{poly}(n)$. In the following, we fix any such n. Then, from an average argument, there exist $i_1^*, i_2^* \in [N_{\mathrm{slot}}]$ and $j^* \in [n]$ such that with probability at least $\delta'(n) \overset{\text{def}}{=} \delta(n)/n(N_{\mathrm{slot}})^2 > \delta(n)/n^3$, we can extract a fake witness $\langle i_1, i_2, j, \ldots \rangle$ such that $(i_1, i_2, j) = (i_1^*, i_2^*, j^*)$. Then, we consider the following P^{**} that participates in 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 from the external V to P^*.

- In the i_1^*-th slot of Stage 2, P^{**} forwards $C_{i_1^*,j^*}$ from P^* to the external V and forward r_1 from the external 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 $\langle i_1^*, i_2^*, j^*, \ldots \rangle$, P^{**} aborts with output fail. Otherwise, parse $\langle i_1^*, i_2^*, j^*, \mathsf{UA}_2, \mathsf{UA}_4, R \rangle \leftarrow w$. Then, P^{**} sends UA_2 to the external V and receives r_2.

3. P^{**} rewinds the internal P^* to the point that P^* had 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. Then, P^{**} interacts with P^* as a honest verifier and extracts a witness w' in Stage 3.

4. If w' is not a fake witness of the form $\langle i_1^*, i_2^*, j^*, \ldots \rangle$, P^{**} aborts with output fail. Otherwise, parse $\langle i_1^*, i_2^*, j^*, \mathsf{UA}_2', \mathsf{UA}_4', R' \rangle \leftarrow w'$. 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 the following. 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 $\langle i_1^*, i_2^*, j^*, \ldots \rangle$ 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 $(\delta'/2)^2 - \mathsf{negl}(n)$. This follows from the following. First, from the definition of good prefix, both w and w' are fake witnesses of the form $\langle i_1^*, i_2^*, j^*, \ldots \rangle$ with probability at least $(\delta'/2)^2$. Next, when 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; thus, when w and w' are fake witnesses, $(h, \mathsf{UA}_2, r_2, \mathsf{UA}_4')$ is an accepting UA proof except with negligible probability.

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

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

□

Next, we show the following claim.

Claim 7. *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 Exp_1, where E^* plays as P. Let $\langle h, C, r_1 \rangle$ be the interactively generated statement.*

2. ***Outputting witness.*** *E outputs $w = \langle \Pi, \tau, R, y \rangle$. We say that E wins in the experiment if w is a valid witness of $\langle h, C, r_1 \rangle \in \Lambda_2$.*

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

Proof. We first note that the extractor of UA works even when the statement is interactively generated after h is sent. This is because the extractor of UA extracts a witness by first emulating honest execution till the end and then restarting the execution from the second verifier message. Thus, by simply using the extractor of UA for P^{**}, we obtain E^* that outputs a valid witness of $\langle h, C, r\rangle \in \Lambda_2$ with non-negligible probability. Note that since the running time of the global extractor of UA is $\mathsf{poly}(T(n))$, the running time of E^* is also $\mathsf{poly}(T(n))$. □

Finally, we reach a contradiction by showing that given E^* described in Claim 7, we can break the collision-resistance property of $\mathcal{H}$.

Claim 8. *If there exists* $\mathsf{poly}(T)$*-time ITM* E^* *that wins in* $\mathsf{Exp}_2(n, E^*)$ *with non-negligible probability, there exists* $\mathsf{poly}(T)$*-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 $\langle h, C, r\rangle$ and w be the statement and the output of E^* in this emulated experiment.

2. If w is not a valid witness of $\langle h, C, r\rangle \in \Lambda_2$, $\mathcal{A}$ aborts with output fail. Otherwise, parse $\langle \Pi, \tau, R, y\rangle \leftarrow w$.

3. $\mathcal{A}$ rewinds E^* to the point that E^* had sent C, and from this point $\mathcal{A}$ emulates $\mathsf{Exp}_2(n, E^*)$ again with flesh randomness. Let $\langle h, C, r'_1\rangle$ be the statement and w' be the witness in this emulated experiment.

4. If w' is not a valid witness of $\langle h, C, r'_1\rangle \in \Lambda_2$, $\mathcal{A}$ aborts with output fail. Otherwise, let $\langle \Pi', \tau', R', y'\rangle \leftarrow w'$.

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

We first show that both w and w' are valid witnesses with non-negligible probability. Note that for infinitely many n, E^* outputs a valid witness in $\mathsf{Exp}_2(n, E^*)$ with probability $\epsilon \overset{\text{def}}{=} 1/\mathsf{poly}(n)$. 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). Then, 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 $\epsilon/2$. Then, from an average argument, the prefix of the internally emulated transcript is good with probability at least $\epsilon/2$. Thus, w and w' are valid witnesses with probability at least $(\epsilon/2)(\epsilon/2)^2 = (\epsilon/2)^3$.

Next, we show that when $\mathcal{A}$ obtains two valid witnesses $w = \langle \Pi, \tau, R, y\rangle$ and $w' = \langle \Pi', \tau', R', y'\rangle$, 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 a substring of τ and r' is a substring of τ', we have $\tau \neq \tau'$ except with negligible probability. (If $\tau = \tau'$ holds, r' is a substring of τ' with probability at most $T(n)/2^n = \mathsf{negl}(n)$ since r' is chosen at random after τ is determined.) Then, since Π^{O} always outputs τ and Π'^{O} always outputs τ', we conclude that we have $\Pi \neq \Pi'$ except with negligible probability. (Recall that for each query to O, the reply is uniquely determined.)

From the above two observations, we conclude that $\mathcal{A}$ breaks the collision-resistance property of $\mathcal{H}$. □

From Claims 6, 7, and 8, we conclude that there exists no bad P^*. Thus, the extractor E outputs a witness of $x \in L$ except with negligible probability. □

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 help 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.

[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.

[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.

[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.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, August 2001.

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

[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.

[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.