Public-Coin Concurrent Zero-Knowledge in Logarithmic Rounds ⋆

Yi Deng

SKLOIS, Institute of Information Engineering,
Chinese Academy of Sciences
ydeng.cas@gmail.com

Abstract. We construct $O(\log^{1+\epsilon} n)$-round *public-coin* concurrent zero knowledge arguments for NP from standard (against any polynomial-time adversary) collision-resilient hash functions for arbitrarily small constant ϵ. Our construction is *straight-line simulatable*. This is the first public-coin concurrent zero knowledge protocol based on standard/long-studied assumption that (almost) achieves the best known round-complexity of its private-coin counterpart [Prabhakaran et al., FOCS 02]. Previously, such public-coin constructions require either polynomial number of rounds [Goyal, STOC 13], newly-introduced assumptions [Chung et al. FOCS 13], or stronger model [Canetti et al., TCC 13].

This result has strong consequences: it yields the first (almost) logarithmic round simultaneously resettable arguments for NP and the first (almost) logarithmic round concurrent multi-party computation in the single input setting. These results significantly improve over the polynomial round-complexity of the best known protocols based on standard assumptions in both cases.

Our technical contribution is two-fold. First, we introduce a simulation strategy called *clearance* that yields a simulation tree of very *special combinatorial structure* and enables us to instantiate Barak's protocol [Barak, FOCS 01] using the recent Ben-Sasson et al.'s quasi-linear construction of PCP system [Ben-Sasson et al, STOC 13] to obtain logarithmic round-complexity; Secondly, we show how to modify Barak's protocol such that the soundness of overall construction does not rely on the (implicit/explicit) proof of knowledge property of the underlying universal argument/PCP system, which in turn allows us to benefit from progress on short PCP system of more general types *without assuming stronger/superpolynomial hardness*.

1 Introduction

Zero knowledge proof, introduced by Goldwasser et al. in [GMR89], has played a pivotal role in modern cryptography. It offers a magical method of proving theorems in which the prover reveals nothing to the verifier but their validity. Shortly afterwards, Brassard *et al.* [BCC88] defined the notion of zero knowledge *argument* whose soundness is only required to hold against polynomial-time cheating provers; Goldreich et al.[GMW91] showed that every NP language admits a zero knowledge protocol under standard hardness assumption. These generalizations enable a wide range of its applications in cryptography.

The original security notions are defined in the *standalone* setting, where only a single execution of the protocol is considered. To deal with security concerns arising from in asynchronous networks like the Internet, Dwork et al.[DNS98] put forward the notion of concurrent zero knowledge. They consider a setting where multiple executions of the same protocol take place, and a malicious adversary may control the message scheduling and corrupt honest parties. A protocol is called concurrent zero knowledge if it preserves zero knowledge even in this concurrent setting.

Over the last few decades, concurrent zero knowledge has attracted considerable attention and stimulated a few ingenious innovations of simulation techniques, such as Richardson and Kilian's recursive simulation technique [RK99] and its oblivious version introduced by Kilian and Petrank

⋆ This work was supported by the National Natural Science Foundation of China Under Grant NO.61379141.

[KP01]. Prabhakaran et al. [PRS02] refined the analysis of the simulator in [KP01] and finally proved (almost) logarithmic ($\widetilde{O}(\log n)$) round-complexity is sufficient for concurrent zero knowledge protocol, which almost matches the black-box lower bound of [CKPR01]. To this date, Prabhakaran et al.'s upper bound on round-complexity still remains the lowest among all known constructions of concurrent zero knowledge based on standard/long-studied assumptions in the plain model.

In his breakthrough paper, Barak [Bar01] showed how to set up a trapdoor using the code of the (malicious) verifier and then construct zero knowledge protocol in the "FLS framework" [FLS99]. This non-black-box simulation technique was used to break a number of known lower bounds for black-box zero knowledge. For instance, it gives rise to constant-round public-coin zero knowledge argument, and the associated simulator proceeds in a *straight-line* manner and runs in strict polynomial time. These features have been proved impossible to achieve when using black-box simulation [GK96, BL04]. Since its introduction, Barak's technique actually found many applications beyond zero knowledge.

However, for concurrent zero knowledge, Barak's non-black-box technique just gave us a bounded-concurrent zero knowledge argument, here "bounded concurrence" refers to the setting where the number of concurrent sessions initiated by the malicious verifier is known prior to designing of the protocol. Though a huge body of work was devoted to study Barak's non-black-box technique, no public-coin fully concurrent zero knowledge protocol was known until very recently. In 2013, Goyal [Goy13] developed an oblivious straight-line simulation and constructed the first public-coin concurrent zero knowledge protocol from the existence of collision-resistant hash functions, which runs in polynomial rounds. Chung et al. [CLP13b] introduced a new non-interactive argument system called "P-certificate", and presented the first public-coin constant-round concurrent zero knowledge argument with uniform soundness based on the existence of P-certificate. Meanwhile, in the global hash model, Canetti et al. [CLP13a] showed that public-coin concurrent zero knowledge can be achieved with $O(\log^{1+\epsilon} n)$ round-complexity.

Why public-coin and straight-line simulator? We would like to stress that public-coin and straight-line simulatable protocols, in which the verifier simply sends independently random coins at each of his steps, have been found to be more broadly applicable and versatile than "private-coin" protcols. For example, the public-coin proof systems for IP can be transformed into zero knowledge proofs [BOGG$^+$88]; The transformation in [DL07] of public-coin concurrent zero knowledge into simultaneously resettable arguments is much simpler than the transformation of [DGS09], and more importantly, the transformation of [DL07] (together with a technique from [DGS09, DFG$^+$11]) introduces only *constant* overhead[1] in the number of rounds complexity, whereas the transformation of [DGS09] introduces a *polynomial* round-complexity overhead. As already mentioned in [Goy13], the public-coin and straight-line simulatable concurrent protocol can be used to bypass the impossibility results of [Lin08, PTW11] and obtain concurrent secure multiparty computation for some functionalities beyond zero knowledge.

1.1 Our results

Main result. Assuming that standard (against any polynomial-time adversary) collision-resilient hash functions exist, we construct $O(\log^{1+\epsilon} n)$-round *public-coin* and *straight-line simulatable* concurrent zero knowledge arguments for NP for arbitrarily small constant ϵ. This is the first public-coin

[1] A constant-round instant-dependent resettably-sound resettable WI argument– the main building block in the transformation of [DL07] – were implicit in the work [DGS09, DFG$^+$11].

concurrent zero knowledge protocol based on standard assumption that (almost) achieves the best known round-complexity of the private-coin protocol presented in [PRS02], and significantly improves on the construction of [Goy13] which requires *polynomial* round-complexity. Our construction achieves the same round-complexity of the protocol in [CLP13a] but does not rely on either global hash or super-polynomial hardness. This result is incomparable to the one of [CLP13b]: the protocol of [CLP13b] obtains constant round-complexity but relies on a newly-introduced assumption and achieves only uniform soundness.

Applications. Our result has strong consequences. Following the transformation from [DL07], it yields a logarithmic-round simultaneously resettable argument for NP.

Our protocol also has applications beyond zero knowledge. As pointed out in [Goy13, CLP13b], the public-coin and straight-line simulatable concurrent zero knowledge protocol can be applied to achieve fully concurrent secure computation in the single input setting (or, more generally, for a family of functionalities satisfying some certain property) with essentially the same round-complexity as the underlying concurrent zero knowledge protocol[2]. Thus, when our protocol is applied, we obtain an (almost) logarithmic round protocol for concurrent secure computation in the same setting.

These two results significantly improve over the *polynomial round-complexity* of the best known protocols based on standard/long-studied assumptions in both cases.

1.2 Technique overview I: Clearance and the combinatorial structure of the simulation tree

Our initial construction, which requires super-polynomial hardness assumption, is basically a multi-slot version of Barak's protocol[Bar01], with the following two modifications. First, we replace the universal argument (UA for short) of [BG08] with Ben-Sasson et al's construction of almost UA [BSCGT13]. As we will see, the "short" (quasi-linear) length of the underlying PCP proof of [BSCGT13] is one of critical properties for the analysis of our simulator to go through. Secondly, we have the verifier send the hash function α for the universal argument in its first message of the global protocol. This allows us to borrow an idea from [CLP13a] that enables the simulator to prepare offline UA.

Roughly, this construction proceeds in three stages. In the first stage the verifier sends two random hash functions h, α (α will be used for UA) first, and then the prover and the verifier run k iterations (we call them *slots*) of the preamble of Barak's protocol: for each $1 \leq i \leq k$, the prover sends a dummy commitment c_i, to which the verifier responds with a random string r_i; the second stage[3] is an encrypted UA of [BSCGT13] in which the prover proves that one of slots, say (c_i, r_i), satisfies the following condition: c_i is a commitment to a hash value of some code Π, and, given input c, Π outputs r_i in some super-polynomial time; in the final stage, the prover gives 3-round WI proof that second stage, either $x \in L$, i.e., the statement in question is true, or the encrypted UA is acceptable.

The soundness of this protocol follows from previous standard analysis, except that here we need to assume the underlying hash functions h, α is collision-resistant against *super-polynomial* time adversary, due to that the standard analysis of soundness for Barak's protocol relies on the

[2] Actually, Goyal mentioned in [Goy13] that they achieve this result and concurrent blind signature in an unpublished work. We believe that our protocol can also be applied to obtain concurrent blind signature with significant improvement on the round complexity.

[3] Note that this stage slightly differs from the first stage of witness indistinguishable UA of [BG08] where the prover proves to the verifier via encrypted UA that either the preamble generates an YES instance or $x \in L$. Our construction of the last two stages follows from the work of [PR05].

(weak) proof of knowledge property of the underlying UA, and that such property of almost UA of [BSCGT13] was only established from super-polynomial collision-resistant hash functions. In next subsection, we will show how to remove the requirement of super-polynomial hardness.

The focus of this subsection is to present a simulation technique for the above protocol. At a very level, for each session, our simulator chooses a slot (c_i, r_i) to prepare a PCP proof (together with its Merkle hash treee, we ignore the latter for now for simplicity) with respect to this slot, which will enable it to go through the last two stages in a straight line way. This can be done by having the simulator commit to hash value of the joint code of *itself* and the verifier. Observe that such a joint code will generates the same transcript of concurrent sessions between c_i and r_i internally as in the simulation and thus the slot (c_i, r_i), together with the hash function h in this session, actually forms an YES instance. A major issue with this approach is, in concurrent setting, the slot (c_i, r_i) may cover many UAs of other sessions (By " slot (c_i, r_i) covers a message" we mean this message appears within this slot), which in turn requires our simulator to construct PCP proofs for (the correctness of) constructions of other PCP proofs, and furthermore, with the quasi-linear $n \log^{O(1)} n$ length PCP in use, a polynomial time simulator could only reconstruct these PCP proofs up to recursive depth $O(\frac{\log n}{\log \log n})$. Now, one problem is left:

> *How to choose a slot of a session for which we construct a PCP proof when we limit recursive depth to $O(\frac{\log n}{\log \log n})$ and k to logarithm?*

Before describing challenges to answer this question, we fix some terminology (some of which are subject to change as we present our new technique.). We will focus on the first prover step of UA in which the simulator needs to prepare a PCP proof, and for now we call this step a PCP step. We call a session *solved* if the simulator already obtained a PCP proof for some slot which enables it to complete this session. We say that a slot (c_j, r_j) is level-0 slot if it does not cover any PCP step, and that a PCP proof is level-0 PCP proof if it is constructed for a level-0 slot; A slot is called level-i slot if the highest level PCP proof(s) covered by it is at level $i-1$; Similarly, A level-i PCP proof refers to the one that is constructed for a level-i slot.

The known strategies seem not to be able to solve the above question. Goyal [Goy13] developed a random marking strategy to select a slot for each session, but it seems that the analysis of this technique relies intrinsically on the fact there are polynomial number of slots for each session to bound the failure probability, which is independent of length/efficiency of the underlying PCP proof system.

One may also wonder if the following naive strategy works: The simulator follows the above high level idea, and whenever entering the first prover step of UA of a session, it chooses a slot at lowest level to prepare PCP proof for this session. Again, this one is not as good as it looks like. Consider the following verifier scheduling (depicted in figure 1):

> *An example of verifier scheduling V^*:* Suppose that a malicious verifier V^* initiates $n \cdot k$ sessions and uses the following layered scheduling: at the bottom layer, it executes the first k sessions sequentially; at the second layer, it executes another k sessions as follows. For each session i on the second layer, $k+1 \leq i \leq 2k$, and for all $1 \leq j \leq k$, V^* has its j-th slot, denoted by (c_j^i, r_j^i), cover the last two stages (called WIUA) of the session j on the bottom layer, and then completes these k sessions sequentially; Similarly, the verifier arranges the k sessions at the third layer in the same way except that it has the j-th slot of each of these sessions cover WIUA of the $(k+j)$-th session at the second layer, and so on.

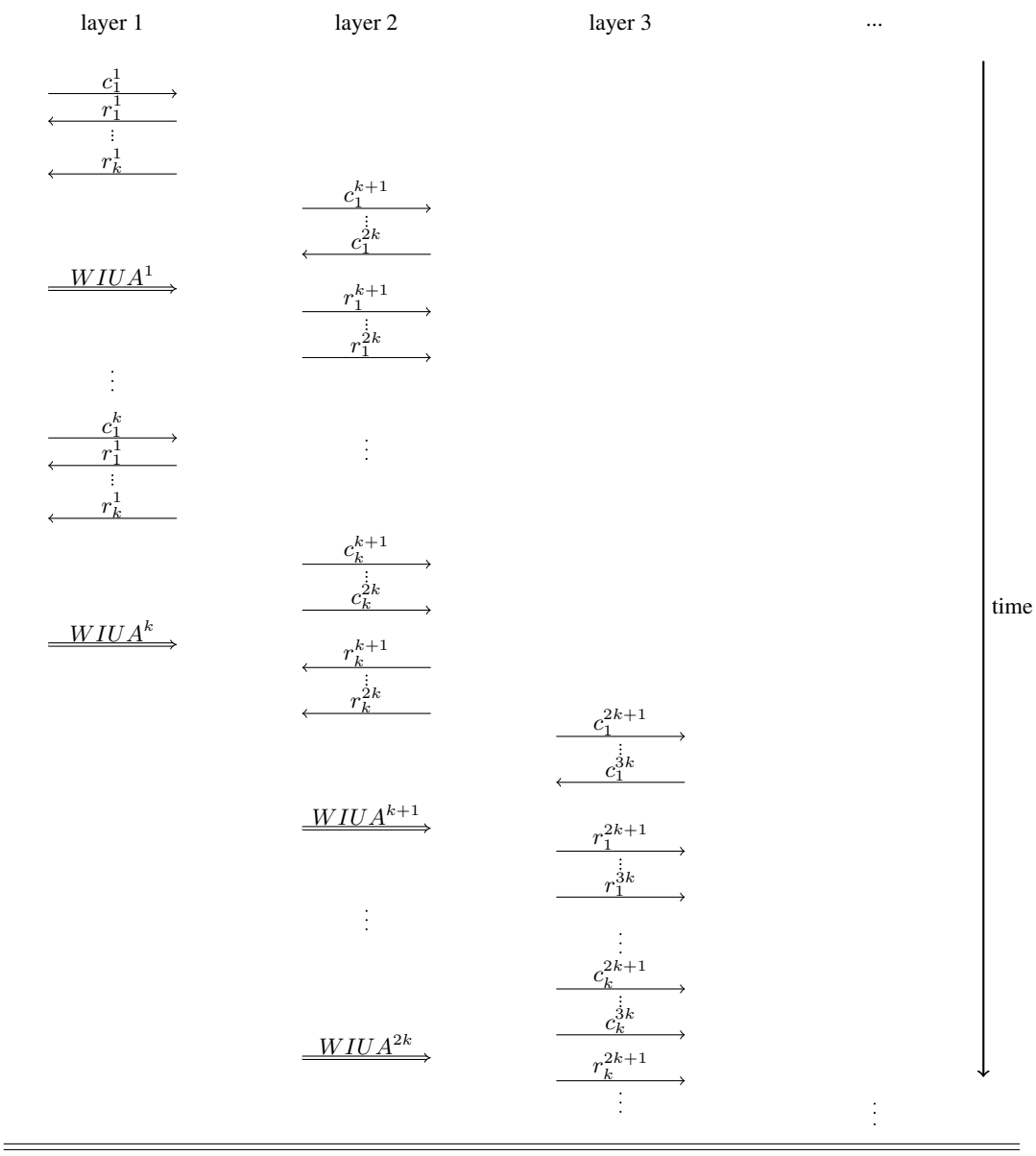

Fig. 1. The scheduling V^*

It is easy to see that the above simulation strategy does not work for this scheduling: there are n layers, and for each session at layer l, the lowest level of its slots is level $l-1$. This results in exponentially blow-up of the running time of the simulator.

Our solution is to introduce a technique we call "clearance" into the above simple simulation strategy. Our simulator proceeds in a straight-line way, whenever it enters the first prover step of UA of an *unsolved* session s, it does the following at this single step:

1. *Solve this session*: choose a lowest level slot (say, at level i) of this session (If there are several slots at the same lowest level, it chooses the first one.), construct a PCP proof for this slot, and store it in a table.
   Now this PCP proof is at level-i.
2. *Clearance*: For every session t for which at least one slot appears in the current history but has not been solved yet, if the lowest slot of this session so far is at level $\leq i$, solve it by using the above strategy, and store the corresponding PCP proofs in a table.
   Note that, after this clearance step, the simulator can simply retrieve the corresponding PCP proof (and its Merkle hash tree) to complete session t.

We will call the session s (which remains unsolved until the simulator enters its second stage) *normal*, and all those sessions being *cleared*/*solved* in clearance step *lucky*.

At first glance, our strategy seems even worse than the above naive one since for a lucky session the simulator may not use its lowest level slot to construct PCP proof. However, we observe that:

- In the analysis of the simulator's running time, we just need to care about the time spent on the prover steps in which the simulator actually constructs PCP proofs. We will call such prover steps *PCP construction step*. Observe that PCP construction steps are actually those prover first steps of UA of *normal* sessions, and that *no lucky* session contains PCP construction step (thus all prover steps of a lucky session are "light"). In a PCP construction step, the highest level PCP proof is the one for the corresponding normal session.
- Although it may be the case that in a single PCP construction step the simulator needs to construct arbitrarily polynomial number of PCP proofs, as we will see in section 5, the major contributor to the blow-up of the simulator's running time is the (highest) level of PCP proof, rather than the number of PCP proofs constructed in this step.
- Furthermore, perhaps surprisingly, using clearance will keep every PCP proof at desirable low level, and thus even for those lucky sessions the slot we choose for simulation is also at low level (though this choice may not be optimal). We will discuss this property shortly.

From now on, we call a PCP construction step level-i PCP construction step if the highest level of PCP proof constructed at this step is i, and a slot level-i slot if the highest level of PCP construction step covered by it is $i-1$.

To take a closer look at our simulation strategy, we now form a simulation tree and examine its combinatorial structure.

The simulation tree and its structure. Think of a PCP construction step as a node, which (possibly) contains many PCP proofs. Our simulator proceeds from one node to another in a straight line manner. We create the parent-child relationship between these nodes as follows. Consider a node u on level i at which an unsolved normal session, say session s, just arrives its second stage. Denote by S_u the set of sessions that will be solved at node u (thus $s \in S_u$), and for each session $t \in S_u$, suppose that the slot (c_{j_t}, r_{j_t}) is the one for which our simulator will construct a PCP proof for session t, and that c_{j_l} is the first one appearing in the simulation path among all those prover messages in the slots $\{(c_{j_t}, r_{j_t}) : t \in S_u\}$. Then, when the simulator arrives at u on level i, we set the node u to be the parent node of all nodes v that:

1. v is covered within c_{j_l} and the time when the simulator arrives at the node u; or,
2. v is on level $\leq i-1$ and doesn't have a parent node yet.

Note that, in extreme cases, to construct PCP proofs at node u may require the simulator to reconstruct all PCP proofs within the entire subtree rooted at u.

A general simulation path consists of several consecutive simulation trees. The most compact simulation tree with $k = 3$ is depicted in figure 2, here by "most compact" we mean that, among all possible simulation paths of the same height, the most compact one achieves the minimum number of nodes on every level (see section 5 for the analysis).

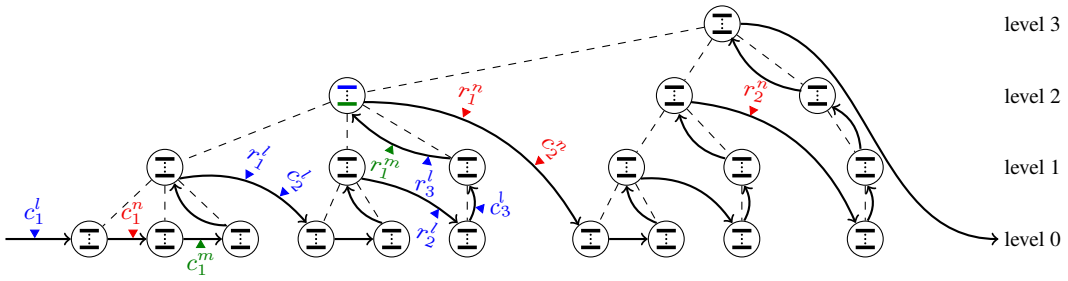

Fig. 2. The most compact simulation tree with $k = 3$. Here, the arrows refer to the real simulation path, the dashed lines represent parent-child relationship, and each short bold line that lies in a node represents a PCP proof. In this tree, the session l is *normal* and gets solved when the simulator arrives at the first node at level 2, while the session m is *lucky* and is solved by clearance at the same node with session l. Until the verifier message r_2^n, the session n still remains unsolved, but it still has chance to get solved at lower (than 2) level node.

One striking property of our simulation tree is the following. For a node u, and for a session i remaining unsolved at u, we denote by n_u^i the number of slots of session i that do not appear before the node u yet. Then, if the minimum number $n_u^i = c$ among all those session i's, our simulator will go through (at least) c nodes at level 0, and build a (sub)tree from this bottom level again.

This property corresponds to an interesting combinatorial structure of the most compact simulation tree, as depicted in figure 3. Actually, for the most compact tree with height $l \geq 2$, as we will show, the number of nodes at level 0 is exactly equal to

$$\frac{(k+l-2)(k+l-1)\binom{k+l-3}{l-2}}{(l-1)l}$$

When the slot number of each session $k \in O(\log^{1+\epsilon} n)$, then the height l of the simulation tree is bounded by $O(\frac{\log n}{\log \log n})$ to make sure that the above number is still a polynomial. This is the key to show that the simulator runs in polynomial time.

For the example of verifier scheduling V^* described before, the simulation path is depicted in figure 4. This path contains only one tree of height 1. When entering the first prover step of UA in session $k+1$, the simulator constructs a PCP proof for this session, and by clearance strategy, it also constructs PCP proofs for all other sessions on the second layer (i.e., session $k+2$ to session $2k$). This forms the only node/PCP construction step at level 1. For session $2k+1$ to session $3k$ on layer 3, each of them has the second slot cover the WIUA of session $k+2$, in which the first prover step of UA is a non-PCP-construction-step (since session $k+2$ has already been solved), thus by clearance

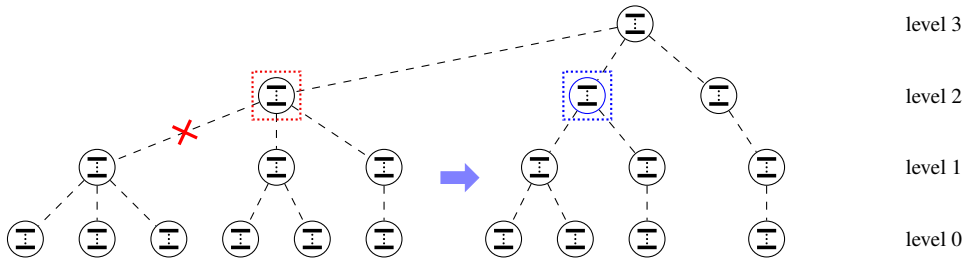

Fig. 3. The combinatorial structure of the most compact simulation tree with height $l \geq 2$ ($l = 3$ in this example): (1) For every $i > 0$, the left-most node on level i has k (=3, in this example) children; and (2) for every node u at level $i \geq 2$, and any u's two consecutive (according to the order of appearance) child nodes, the subtree T_2 rooted at the second child node can be obtained from the subtree T_1 rooted at the first child node by deleting the first branch of T_1.

strategy again, at the first prover step of UA in the session $2k + 1$, the simulator will construct PCP proofs for all of them with respect to their second slots, which forms a new node at level 0. This observation applies to higher layer sessions, and thus all subsequent nodes are at level 0.

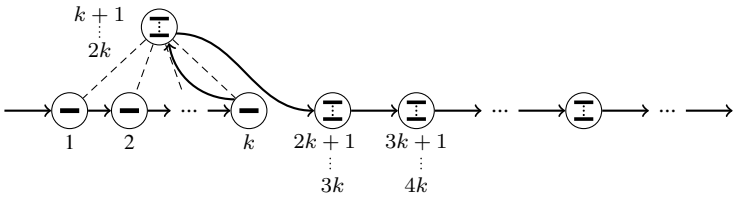

Fig. 4. The simulation path for the example of verifier's scheduling V^*. In this path, there is only one tree of height $l = 1$.

1.3 Technique overview II: Removing the requirement of PoK property from UA/PCP

One slightly annoying problem with the above protocol is that we have to assume super-polynomial collision-resistant hash functions for the soundness of the protocol to go through. This, as explained in [BSCGT13], is mainly due to that the PCP proof developed [BSCGT13] does not enjoy implicit proof of knowledge property.

However, a closer look at the overall zero knowledge construction and its soundness analysis of [BG08] actually inspires us to modify our initial construction in such a way that the soundness analysis for the overall protocol does not rely on the knowledge of proof property of the underlying UA. Recall that in the preamble of zero knowledge protocol in [BG08], the simulator is supposed to commit to a pair $(|ECC(\Pi)|, MH_h(ECC(\Pi))$, where ECC is some error correcting code of constant relative distance and $MH_h(ECC(\Pi))$ is the Merkle hash tree value of $ECC(\Pi)$ via hash function h, and then proves in UA that the transcript of the preamble stage is an YES instance.

The basic strategy for soundness analysis for the construction of [BG08] is as follows. Suppose that there is a malicious prover that can cheat with non-negligible probability, then when running

it twice on the same first two messages (h, c) but difference challenges r, r', the probability that the prover gives accepting proofs for both sessions is still high. Note that the (part of) witnesses Π and Π' used by the prover in these two sessions must be different. Now due to the error correcting property of ECC, the codes $ECC(\Pi)$ and $ECC(\Pi')$ differ on a constant fraction. Thus, when we choose a random location and apply the extractor of UA to extract a *single* bit at this location (and their corresponding authenticators) for each of $ECC(\Pi)$ and $ECC(\Pi')$, these two bits will differ still with high probability, and thus we derive a collision for h.

Our key observation is that the length of the target collision, i.e., a bit of $ECC(\Pi)$ at a random location together with its authenticator, can be bound[4] by n^2. This enables us to prove the knowledge of the bit and its authenticator via a standard 3-round WI argument of knowledge. For this reason, we modified our initial protocol in the following way:

1. Add an extra stage between the slots stage and encrypted UA stage of our initial protocol, in which we have the verifier send a random location j for the string $ECC(\Pi)$, and then have the prover response with a commitment c_{ecc} to 0^{n^2}.
   In simulation, the simulator in this stage is supposed to commit to $(|ECC(\Pi)|, b||\sigma_b)$, where b and σ_b are the j-th bit of $ECC(\Pi)$ and its corresponding authenticator.
2. Have the prover prove in UA that there is some slot (c_i, r_i) such that, (1) (h, c_i, r_i) is an YES instance, and (2) $c_{ecc} = Com(|ECC(\Pi)|, b||\sigma_b)$, and $c_i = Com(|ECCC(\Pi)|, MH_h(ECCC(\Pi)))$.
3. In addition to the second part of the OR statement that $x \in L$ or the encrypted transcript of UA is acceptable, we also have the prover prove of knowledge of the content of c_{ecc} in the last 3-round WI argument of knowledge.

Suppose now that a cheating prover gives an accepting proof on a false statement $x \notin L$. The soundness of the UA guarantees, when we apply the extractor to the last 3-round WI argument of knowledge, the extracted content of c_{ecc} must be the required information about the program defined in the relevant slot (c_i, r_i), i.e., the program committed in c_i that actually outputs r_i. This, by standard rewinding technique, finally enables us to derive collisions for the hash function h without requiring further proof of knowledge property from the underlying UA. As we will see in section 3, if not required to satisfy (weak) proof of knowledge property, the construction of almost UA in [BSCGT13] is actually built on standard (against polynomial time adversary) collision-resistant hash functions. This follows directly from the first two step of the proof for establishing weak proof of knowledge property presented in [BG08].

1.4 Related work

Concurrent zero knowledge in stronger model/under stronger assumptions. Despite great effort made in the last decades, the round complexity of concurrent zero knowledge is still far from being well understood. To circumvent the black-box lower bound of [CKPR01], researchers resort to several stronger models/set-up assumptions, such as the timing model [DNS98], common reference string model [Dam00], bounded player model [GJO+13], or relax the prover's security to allow super-polynomial time simulation [Pas03, Pas03, BS05, PV08]. In the plain model, several new non-black-box techniques for obtaining constant-round complexity were developed under stronger (non-standard) assumptions in very recent years. Besides the work of [CLP13b] mentioned before,

[4] since the length of Π can be bound by some super-polynomial.

two other (private-coin) constant-round concurrent zero knowledge arguments, based on knowledge assumption and differing-input obfuscation respectively, were recently presented in [GS12, PPS13].

Resettability. Another notable security notion stronger than concurrrent zero knowledge is resettable zero knowledge[CGGM00], for which the malicious verifier is allowed to execute polynomial number of concurrent sessions with a prover with a fixed random tape. Barak et al. [BGGL01] put forward an analogue notion of resettable-soundness and presented a transformation of public-coin zero knowledge protocol into a resettably-sound zero knowledge protocol. After the first simultaneously resettable (resettably-sound resettable zero knowledge) argument was constructed[DGS09], several new non-black-box techniques were developed and the assumption needed for simultaneously resettable arguments was reduced to the mere existence of one-way function [COPV13, BP13][5], but all these protocols were constructed via the general transformation of [DGS09] and hence require polynomial number of rounds.

[CGGM00] also introduced the Bare public-key model, aiming at obtaining constant-round resettable zero knowledge. As shown in recent work of [DFG+11, COSV12], constant-round simultaneously resettable argument can also be achieved in this model. We refer readers to references therein for many other studies on resettability in the Bare public-key model.

Concurrently self-composable secure computations. The concurrent self-composition[6] was later extended to general two/multi-party computations [Lin03, PR03, Pas04]. These works handle only bounded concurrency and was recently subsumed by [Goy12], which constructed fully concurrent secure protocols in the single input setting[7]. Note that it was showed to be impossible to achieve fully concurrent security for a large family of functionalities, regardless of the type of simulation technique used [Lin08]. Due to its public-coin and straight-line (non-black-box) simulation properties, the very recent construction for concurrent zero knowledge [Goy13] can be applied to achieve fully concurrently secure protocols with improved (but still *polynomial*) round complexity over [Goy12]. Under the newly-introduced assumption (namely the existence of P-certificate), this result can be achieved with significant improvement on round complexity [CLP13b].

1.5 Rest of the paper

We provide the definition of concurrent zero knowledge in section 2. In section 3, we recall universal argument and point out that Ben-Sasson et al.'s construction of UA can be based on standard collision-resistant hash functions if we do not require explicit proof of knowledge property from it. We present our initial construction based on super-polynomial time collision-resistant hash function in section 4, and the simulator for this construction in section 5. Our final construction is presented in section 6.

2 Preliminaries

In this section, we mainly present definition of concurrent zero knowledge argument, and refer readers to the textbook [Gol01] for some basic definitions of building blocks used in this paper.

[5] It is worth noting that [BP12, BP13] developed a non-black-box technique based on the impossibility of general program obfuscation [BGI+01], instead of the PCP mechanism.

[6] This refers to the setting where the same protocol is run many times concurrently. A more general setting of composition call universal composition, where a protocol is executed concurrently with arbitrary other protocols, was introduced by Canetti [Can01].

[7] This can be generalized to provide fully concurrently secure protocols for a family of functionalities that enjoy some certain property (close to bounded pseudoentroy conjecture) [Goy12].

A function $negl(n)$ is called to be *negligible* if for every polynomial $q(n)$ there exists a positive integer N such that for all $n \geq N$, it holds that $negl(n) \leq 1/q(n)$.

For a $\mathcal{NP}$ language L, the associated relation R_L is defined to be

$$R_L = \{(x, w) | \, x \in L; w \text{ is a witness for } \ x \in L\}$$

.

We will abbreviate probabilistic polynomial-time with PPT. An interactive proof system (P, V) for a language L is a pair of interactive Turing machines, for which two conditions, called completeness and soundness, are required to hold. The completeness says honest prover can convince the verifier of the truth of the truth of the statement being proven, while the soundness prevents the honest verifier from cheating, that is, it is infeasible for a (even unbounded) prover to make the verifier accept a false statement.

In this paper, we consider a variant of proof system called *argument* system, whose soundness condition is only required to hold against PPT prover strategy. We denote by $(P, V)(x)$ the output of V at the end of interaction on common input x, and denote by $\text{View}_V^P(x)$ a random variable that describes the view of the verifier consisting of the common input x, the random tape of V and all the prover messages it received.

Definition 1 (Interactive Argument). *A pair of PPT interactive Turing machines (P, V) is called an* interactive argument system for language L *if the following conditions hold:*

- *Completeness: For every $x \in L$, $Pr[(P, V)(x) = 1] = 1$.*
- *Soundness: For every $x \notin L$, and every non-uniform PPT prover P^*,*

$$Pr[(P^*, V)(x) = 1] < neg(|x|).$$

A zero knowledge argument system is an interactive argument for which the view of the (even malicious) verifier in an interaction can be efficiently reconstructed. In this paper, following the work of [DNS98], we consider a powerful *concurrent* (malicious) verifier which interact with a polynomial number of independent provers over an asynchronous network. The concurrent verifier is allowed to fully control over the scheduling of all messages in these interactions. We abuse the notation and denote by $\text{View}_V^P(x)$ also the random variable of the verifier's view in this concurrent setting.

Definition 2 (Concurrent zero knowledge). *We say that an interactive argument (P, V) for language L is concurrent zero-knowledge if for every polynomial $q(\cdot)$, and every concurrent verifier V^* that opens at most $q(n)$ sessions, there exists a PPT S_q running in time polynomial in $q(|x|)$ and $|x|$ such that for any $x \in L$, the random ensemble $\{\text{View}_{V^*}^P(x)\}_{x \in L}$ is computationally indistinguishable from the random ensemble $\{\mathsf{S}_q(x)\}_{x \in L}$.*

3 Almost Universal Argument in Use

In this section we first recall universal argument of [BG08] and the recent efficient construction of *almost* universal argument by Ben-Sasson et al.[BSCGT13]. We will use the latter construction in our protocol. As mentioned, we will not require (explicit) proof of knowledge property from the construction of [BSCGT13], which will enable us to base our concurrent zero knowledge argument on standard (polynomial time) assumption.

Recall that the universal set $\mathcal{S}_{\mathcal{U}}$ is the set of all triples $\{(M,x,t) : \exists \omega \; s.t. \; ((M,x,t),\omega) \in \mathcal{R}_{\mathcal{U}}\}$, where $((M,x,t),\omega) \in \mathcal{R}_{\mathcal{U}}$ if M accepts (x,ω) within t steps. We denote by $T_M(x,\omega)$ the running time of M on input (x,ω). Notice that $((M,x,t),\omega) \in \mathcal{R}_{\mathcal{U}}$ means $T_M(x,\omega) \leq t$.

One virtue of the universal set $\mathcal{S}_{\mathcal{U}}$ is that, a single argument system for $\mathcal{S}_{\mathcal{U}}$ can handle *any* NP language (which is actually needed in our application), since every NP language L is linear-time reducible to $\mathcal{S}_{\mathcal{U}}$ by the mapping $x \longmapsto (M_L, x, 2^{|x|})$, where M_L is the corresponding deterministic polynomial-time machine that decides L.

Definition 3. *[Universal argument [BG08]] A universal argument system is a pair of machines (P,V) that satisfies the following conditions:*

1. *Efficient Verification: There exists a polynomial p such that for any $y = (M,x,t)$, the total time spent by the probabilistic verifier V, on common input y, is at most $p(|y|) = p(|M|+|x|+\log t)$. In particular, all messages exchanged in the protocol have length smaller than $p(|y|)$.*
2. *Completeness via a relatively-efficient prover: For every $((M,x,t),\omega) \in \mathcal{R}_{\mathcal{U}}$,*

$$\Pr[(P(\omega),V)(M,x,t) = 1] = 1$$

Furthermore, there exists a polynomial p such that for every $((M,x,t),\omega) \in \mathcal{R}_{\mathcal{U}}$ the total time spent by $P(\omega)$ on common input (M,x,t) is at most $p(|M| + T_M(x,\omega)) \leq p(|M| + t)$.
3. *Computational soundness: For every polynomial-size circuit family $\{P_n\}_{n\in N}$, and every $(M,x,t) \in \{0,1\}^n \setminus \mathcal{S}_{\mathcal{U}}$,*

$$\Pr[(\widetilde{P}_n,V)(M,x,t) = 1] < negl(n).$$

4. *A weak proof of knowledge property: For every positive polynomial p there exists a positive polynomial p' and a probabilistic polynomial-time oracle machine E such that the following holds: for every polynomial-size circuit family $\{\widetilde{P}_n\}_{n\in N}$, and every sufficiently long $y = (M,x,t) \in \{0,1\}^*$ if $\Pr[(\widetilde{P}_n,V)(M,x,t) = 1] > 1/p(n)$, then, taken over the randomness r for E,*

$$\Pr_r[\exists \omega = \omega_1 \cdots \omega_t \in \mathcal{R}_{\mathcal{U}}(y), \forall i \in [t], E^{\widetilde{P}_n}(y,i,r) = \omega_i] > \frac{1}{p'(n)},$$

where $\mathcal{R}_{\mathcal{U}}(y) \stackrel{def}{=} \{\omega : (y,\omega) \in \mathcal{R}_{\mathcal{U}}\}$.

Recently, Ben-Sasson et al. developed a PCP proof system for NEXT that are both efficient and amenable to universal arguments, and following the construction of [BG08], they realized the following *almost* universal argument.

Definition 4 (Almost universal argument [BSCGT13]). *A universal argument system is a pair of machines (P,V) that satisfies the following conditions:*

1. *Efficient Verification: As defined in the item 1 of definition 3.*
2. *Completeness via a relatively-efficient prover: For every $((M,x,t),\omega) \in \mathcal{R}_{\mathcal{U}}$,*

$$\Pr[(P(\omega),V)(M,x,t) = 1] = 1$$

Furthermore, there exists a polynomial p such that for every $((M,x,t),\omega) \in \mathcal{R}_{\mathcal{U}}$ the total time spent by $P(\omega)$ on common input (M,x,t) is at most $(|M| + T_M(x,\omega)) \cdot polylog(T_M(x,\omega))$, i.e., $(|M| + T_M(x,\omega)) \cdot \log^{O(1)}(T_M(x,\omega))$.

3. *Computational soundness: As defined in the item 3 of definition 3.*
4. *Explicit weak proof of knowledge property: For every two positive polynomials s and p there exists a positive polynomial p' and a probabilistic polynomial-time oracle machine E such that the following holds: for every s-size circuit family $\{\widetilde{P}_n\}_{n\in N}$, and every sufficiently long $y = (M, x, t) \in \{0,1\}^*$ if $\Pr[(\widetilde{P}_n, V)(M, x, t) = 1] > 1/p(n)$, then the oracle machine E, on input $(y, 1^t)$ and with oracle access to $\widetilde{P}_n$, outputs a valid witness ω for y with probability greater than $1/p'(n)$, i.e.,*

$$\Pr_r[E^{\widetilde{P}_n}(y, 1^t, r) = \omega \wedge \omega \in \mathcal{R}_{\mathcal{U}}(y)] > \frac{1}{p'(n)},$$

where $\mathcal{R}_{\mathcal{U}}(y) \stackrel{def}{=} \{\omega : (y, \omega) \in \mathcal{R}_{\mathcal{U}}\}$.

We will use Ben-Sasson et al.'s construction of *almost* universal argument in our main construction. The simulator for our overall protocol heavily exploits its super efficiency: to prove the correctness of a T-step computation, the prover runs only in time quasi-linear in T (roughly, $T \cdot \log^{O(1)}(T)$).

The time efficiency of this construction comes at a price. As mentioned in [BSCGT13], due to the fact that the PCP proof system of [BSCGT13] is based on techniques that do not enjoy good local decoding properties, the resulting almost universal argument does not have knowledge extractor as defined in universal argument of [BG08], where the extractor is required to be an implicit representation of a valid witness. Instead, in an almost universal argument, the extractor E is an explicit representation of a valid witness, and needs to take the running time T of the machine M as input, and runs in polynomial time in T.

Note that in our case, T is a super-polynomial. Thus making use of the proof of knowledge property from UA of [BSCGT13] will require to assume super-polynomial time hardness for proving the soundness of the overall protocol. Fortunately, we find a way to avoid using the proof of knowledge property of UA in establishing the soundness of our protocol (see section 6), and furthermore, the soundness of the UA construction of [BSCGT13] can be proven under standard collision-resistant hash functions against polynomial time adversary.

Theorem 1 ([BG08, BSCGT13]). *If there exist standard collision-resistant hash functions against polynomial time adversary, then the almost UA constructed in [BSCGT13] satisfies condition 1,2,3 of definition 3.*

Proof. Note that the almost UA of [BSCGT13] is constructed by plugging the short PCP system of [BSCGT13] in the UA of Barak and Goldreich. It satisfies the first two conditions unconditionally.

As already mentioned in [BG08], the proof of soundness of UA essentially follows from the basic soundness condition of the underlying PCP system (rather its (weak) proof of knowledge property) and standard collision-resistant hash functions. The intuition behind this proof is the following: given a false statement, the PCP system will spread the errors on a large fraction of its "proof", and then by picking random leaves in the Merkle hash tree of this "proof" we will obtain collisions for the underlying hash function from a successful cheating prover with high probability.

The actually proof of soundness based on standard collision-resistant hash functions follows immediately from the claim 3.5.1 and claim 3.5.2 of [BG08], which are used as the first step in establishing the explicit proof of knowledge property of UA in [BSCGT13]. In essence, these two claims say that, given a prover that makes the verifier accept with non-negligible probability, if the probability (η_α in claim 3.5.2) that the prover gives conflicting answers is noticeable, i.e., larger

than inverse polynomial, then we can find collisions for the hash function α in use in polynomial time with non-negligible probability. To establish proof of knowledge property, we need further to prove that if η_α is smaller than inverse of some polynomial, then we are able to extract (in some super-polynomial time for the case of [BSCGT13]) a witness for the statement with high probability. Logically, since in proof of soundness, we start with a false statement, for which no witness exists, the hypothesis of claim 3.5.2 that η_α is noticeable is true. Thus if there is polynomial time prover that can break the soundness of UA of [BSCGT13], we can use it to find collisions for α in polynomial time, which breaks the assumption that α is (standard) collision-resistant. ■

4 An Initial Construction of Our Protocol

In this section we present an initial construction of concurrent zero knowledge argument based on collision-resistant hash function against super-polynomial adversary. Since we will improve this construction and give analysis of soundness for the improved construction later, we will omit the soundness analysis of this initial construction. The straight-line (non-black-box) simulator and its analysis, which apply to the later improved construction directly, will be given in next section.

As mentioned in the introduction, this construction is basically a multiple-slot version of Barak's protocol [Bar01], with the following two modifications: (1) as in [CLP13a], we replace the UA of [BG08] with almost UA of [BSCGT13], and (2) we have the verifier send the hash function α for the universal argument in its first message. The protocol is depicted in Fig 11.

Definition of the language $\widehat{\Lambda}$ [Bar01]: Let n be security parameter and $h \in \{0,1\}^n$ be a hash function mapping $\{0,1\}^*$ to $\{0,1\}^n$, and let Com be a statistically binding commitment scheme. We say a triplet $(h,c,r) \in \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^{n^2}$ is in $\widehat{\Lambda}$, if there exist a program Π, string $s \in \{0,1\}^{poly(n)}$ and a string y such that $|y| \leq |r|/2 = n^2/2$, such that $z = Com(h(\Pi), s)$ and $\Pi(z,y) = r$ within superpolynomial time (i.e., $n^{\omega(1)}$).

Note that the statistically-binding commitment scheme can be based on one-way functions [Nao91][8], whose existence is implied by the existence of hash functions. We prove the following theorem.

Theorem 2. *If collision-resistant hash functions against super-polynomial time adversary exist, then, for an arbitrarily small constant ε, the protocol presented in Fig 5 is a $O(\log^{1+\varepsilon} n)$-round concurrent zero knowledge argument.*

The completeness of this protocol is obvious. The simulator for our protocol and its analysis will be presented in the next session.

5 Clearance: A New Simulation Technique in the Concurrent Setting

5.1 The simulator

Before describing the simulator we first recall some terminology introduced in the introduction. We call a session *solved* if the simulator already obtained a PCP proof and its corresponding Merkle hash tree for some slot of this session (which enable it to complete this session). In the course of simulation, a prover step is called *PCP construction step* if the simulator computes PCP proof(s)

[8] The commitment proposed in [Nao91] is actually a two-round scheme, but here we view it as a non-interactive one, as it simplifies the presentation.

Common input: $x \in L$ ($|x| = n$).
Auxiliary input to P: w, $w \in R_L(x)$.

Stage 1

$\widehat{V} \longrightarrow \widehat{P}$**:** send two random hash functions $h \leftarrow_R \{0,1\}^n$ and $\alpha \leftarrow_R \{0,1\}^n$.

For $i \in [k]$, repeat the following two steps sequentially:

$\widehat{P} \longrightarrow \widehat{V}$**:** send $c_i = Com(0^n)$. (all commitment schemes Com are *statistically-binding*.)

$\widehat{V} \longrightarrow \widehat{P}$**:** send $r_i \leftarrow_R \{0,1\}^{n^2}$.

Stage 2 (Encrypted Almost UA, α serves as the first message of this UA)

$\widehat{P} \longrightarrow \widehat{V}$**:** send $\beta' = Com(0^{\text{poly}(n)})$.

$\widehat{V} \longrightarrow \widehat{P}$**:** send $\gamma \leftarrow_R \{0,1\}^n$

$\widehat{P} \longrightarrow \widehat{V}$**:** send $\delta' = Com(0^{\text{poly}(n)})$.

Stage 3

$\widehat{P} \Longrightarrow \widehat{V}$**:** $\widehat{P}$ and $\widehat{V}$ execute the 3-round witness indistinguishable argument of knowledge [Blu86] in which P proves the OR of the following statements:

1. $\exists\, w$ so that $(x, w) \in R_L$.
2. $\exists\, i$ and $(\beta, \delta, s_1, s_2)$ so that $\beta' = Com(\beta, s_1)$, $\delta' = Com(\gamma, s_2)$, and $(\alpha, \beta, \gamma, \delta)$ is an accepting transcript of the universal argument proving the statement $(h, c_i, r_i) \in \widehat{\Lambda}$.

Fig. 5. The initial construction of concurrent zero knowledge protocol $(\widehat{P}, \widehat{V})$

(and the corresponding Merkle hash tree(s)) at this step. In concurrent executions, if a PCP construction step appears within a slot (c_j, r_j) of a session then we say the slot (c_j, r_j) *covers* this PCP construction step. We define the level of PCP proofs and of PCP construciton steps as follows.

- Level-0 PCP proof: A PCP proof with respect to a slot (c_j, r_j) of some session that does not cover any PCP construction step in the simulation.
- Level-i ($i \geq 1$) PCP proof: A PCP proof with respect to a slot (c_j, r_j) that covers PCP construction steps in which the highest level of PCP proof(s) constructed is level-$(i-1)$.
- Level-i ($i \geq 0$) PCP construction step: A PCP construction step in which the highest level PCP proof(s) constructed is at level-i.
- Level-i ($i \geq 1$) slot (c_j, r_j): the slot (c_j, r_j) is at level i if the highest level PCP construction step(s) covered by it is at level-$(i-1)$.

We would like to stress that there is a huge difference between the PCP construction step and the first prover step in UA. As mentioned before, our simulator construct PCP proof(s) only at the first prover step of UA of *normal* sessions (i.e., those sessions remain unsolved until the simulator enters their second stage), and, due to the clearance technique we introduce, all *lucky* sessions do not contain any PCP construction step in the simulation (since their corresponding PCP proofs are constructed *offline* at the PCP construction steps of some normal sessions). Thus, the simulation of lucky sessions are easy and light, and the major contributor to the simulator's running time is computation of those heavy PCP construction steps.

The Next-Message Algorithm NEXT$(s, \mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}})$:

1. If the last verifier message in $\mathbb{H}$ is a terminate message, simply output $(\mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}})$;
   Otherwise, order all message in $\mathbb{H}$ according to their appearance, NEXT$(s, \mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}})$ extends $\mathbb{H}$ by a new prover/verifier message pair (p, v) and updates the quadruple $(\mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}})$ as follows.
2. If the next prover message p belongs to the Stage 1 of a session, compute $p \leftarrow Com(h(\Pi(\mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}}, \cdot, \cdot)), s)$ (where h is the hash function sent by the verifier in this session, $\Pi(\mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}}, \cdot, \cdot)$ will be defined soon), $v \leftarrow V^*_{\mathbb{H}}(p)$, $\mathbb{H} \leftarrow (\mathbb{H}, (p, v))$ and $\mathbb{M} \leftarrow (\mathbb{M}, \Pi(\mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}}, \cdot, \cdot))$. Output the new quadruple $(\mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}})$.
3. If the next prover message p belongs to the stage 2 or stage 3 of a session, and this session was solved already, compute p using the corresponding PCP proof and its Merkle hash tree values as witness and randomness s (if needed), $v \leftarrow V^*_{\mathbb{H}}(p)$ and $\mathbb{H} \leftarrow (\mathbb{H}, (p, v))$. Output $(\mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}})$.
4. If the next prover message p is the first message of the stage 2 of a session, say session t, and this session has not yet been solved, do the following:
   (a) Look at the k slots $(c_1, r_1), (c_2, r_2), ..., (c_k, r_k)$ of session t in the history $\mathbb{H}$ and all PCP proofs and their Merkle hash trees stored in $\mathbb{P}$. Denote by l_i the level of slot (c_i, r_i) (i.e. highest level of PCP construction step(s) covered by (c_i, r_i) is at level $l_i - 1$) for each $1 \leq i \leq k$. Suppose that the j-th slot (c_j, r_j) is the first one that achieves $l_j = min\{l_1, l_2, ..., l_k\}$, and that c_j is the j_u-th prover message and the last prover message before r_j is the j_v-the prover message according to the order of their appearance in history.
   Then, recover the randomness s_{j_u} for computing c_j and s_{j_v} for computing the j_v-th prover message from the randomness s, retrieve the corresponding program $\Pi(\mathbb{H}', \mathbb{M}', \mathbb{P}', V'^*_{\mathbb{H}'}, \cdot, \cdot)$ whose hash value has been committed in c_j from $\mathbb{M}$, construct the level l_j PCP proof π_t for session t that $(h, c_j, r_j) \in \Lambda$ using witness $(s_{j_u}, \Pi(\mathbb{H}', \mathbb{M}', \mathbb{P}', V'^*_{\mathbb{H}'}, \cdot, \cdot), y = (s_{j_v}, j_v))$, and compute the corresponding Merkle hash tree $MH(\pi_t)$.
   (b) **CLEARANCE**: For every session t' that has at least one slot appear in $\mathbb{H}$ but has not been solved, do the following: Let $(c_1, r_1), (c_2, r_2), ..., (c_e, r_e)$ be the slots of session t' appeared in $\mathbb{H}$ so far. Find the first lowest level slot, say slot (c_d, r_d), as above. *If the slot (c_d, r_d) at a level $\leq l_j$ defined above*, then construct a PCP proof $\pi_{t'}$ and it Merkle hash tree $MH(\pi_{t'})$ for session t' with respect to the slot (c_d, r_d) as in step $3(a)$; Otherwise, do nothing. (This step ensures that for every session t' that is solved in this step, the level of its PCP proof is $\leq l_j$.)

   and then compute the prover message p using $(\pi_t, MH(\pi_t))$ and randomness s, $v \leftarrow V^*_{\mathbb{H}}(p)$, $\mathbb{H} \leftarrow (\mathbb{H}, (p, v))$, and add $(l_j, \pi_t, MH(\pi_t))$ and all those PCP proofs and their Merkle hash trees computed in step $3(b)$ to $\mathbb{P}$. Output $(\mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}})$.

Fig. 6.

The Algorithm SIMULATE$(s_j, j, \mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}})$:

Extend the current history $\mathbb{H}$ to include the prover/verifier message pair (p_j, v), where p_j is the j-th prover message in the history:

1. Let the i-th prover message be the last prover message in $\mathbb{H}$. Compute the randomness $s_{i+1}, s_{i+2}, ..., s_j$ (used for computing the $(i+1)$-th to j-th prover messages): $s_m \leftarrow \text{PRF}_{s_{m+1}}(1^n)$, $i + 1 \leq m \leq j - 1$;
2. Compute $(\mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}}) \leftarrow \text{NEXT}(s_j, \text{NEXT}(s_{j-1}, \cdots \text{NEXT}(s_{i+1}, \mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}}) \cdots))$;
3. Output $\mathbb{H}$.

Fig. 7.

We now turn to the simulator. Assume there are total n^c prover messages in the concurrent executions. We order all prover messages according to their order of appearance. We denote by s_i the randomness used for computing the i-th prover message. Borrowing an idea from [PRT13, CLP13a], all these s_i, $1 \leq i \leq n^c$, are generated by applying a pseudorandom function PRF in a reverse chain manner: $s_i \leftarrow \text{PRF}_{s_{i+1}}(1^n)$. Note that, given s_i, we can recover all randomness used for computing prover messages prior to the i-th one. As we will see, this allows the simulator to commit to the code of itself without input of its randomness, which is essential for the proof of zero knowledge to go through.

The Algorithm $\Pi(\mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}}, \cdot, \cdot)$:

Input: $(c_j, y = (s_t, t))$[a]

Compute $\mathbb{H} \leftarrow (\mathbb{H}, c_j)$, $\mathbb{H} \leftarrow \text{SIMULATE}(s_t, t, \mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}})$, and output the last verifier message r_j in $\mathbb{H}$.

[a] Note that given the history $\mathbb{H}$ as input to Π, c_j is the next scheduled prover message, and that the input $(c_j, (s_t, t))$ instructs $\Pi(\mathbb{H}, \mathbb{M}, \mathbb{P}, V^*_{\mathbb{H}}, c_j, y)$ to generate internally the transcript within a slot (c_j, r_j) of some other sessions, for which the last prover message is the t-th prover message in the history, and output r_j.

Fig. 8.

At a high level, the simulator proceeds in a straight-line manner as follows. It runs the next-message algorithm NEXT (depicted in Fig 6) to simulate the prover message one by one. On input a randomness, a triple of tables $(\mathbb{H}, \mathbb{M}, \mathbb{P})$ and the verifier $V^*_{\mathbb{H}}$ at the current point, NEXT produces the next prover message p in the following way: If p is a prover message of stage 1 of a session, NEXT commits to a hash value of a procedure Π (depicted in Fig 8), which basically a joint code of itself (without input of its randomness) and the verifier (with some semantic modifications), and stores the new Π in $\mathbb{M}$; If p belongs to stage 2 or 3 of a session, and this session has already been solved, it fetches the corresponding PCP proof and its Merkle hash tree values from $\mathbb{P}$ and computes this prover message; and if p is first prover message of stage 2 of a session that is not solved yet, it adopts the solving with clearance strategy presented in the introduction to construct PCP proofs for this session and some other relevant lucky sessions, and then computes p. See Fig 6 for details.

With the procedure NEXT, we define SIMULATE (depicted in Fig 7) that generates a long transcript by simply run NEXT in chain, which we use to give a precise definition of the algorithm Π, as depicted in Fig 8.

Now, the simulator S formally proceeds as follows.

The simulator S:
Compute $\mathbb{H} \leftarrow \text{SIMULATE}(s_{n^c}, n^c, x, \emptyset, \emptyset, V^*_x)$, and output $\mathbb{H}$.

5.2 Analysis of the simulator

Next we prove that the simulator S presented in the preceding section runs in polynomial time and its output is indistinguishable from real concurrent executions between honest provers and V^*. This concludes that the protocol $(\widehat{P}, \widehat{V})$ is concurrent zero knowledge.

A general simulation path consists of a number of consecutive trees (defined in section 1.2), as in Fig 9 (in Fig 9 we drop out those bold lines, each representing a PCP proof, in the circle.). Recall that, in extreme cases, when the simulator constructs PCP proofs at a node v (a PCP construction step), the simulator may need to regenerate the subtree rooted at v. Note also that at a single node, the simulator may need to construct many PCP proofs.

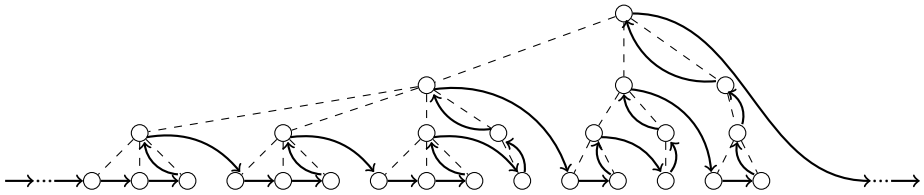

Fig. 9. A general simulation path.

The key to analyze the running time of S is to bound the height of these trees in the entire simulation. We do this by giving lowerbound on the number of nodes at each level of the first simulation tree of maximum height in the simulation path. Observe that the most compact tree (of the same height) presented in the introduction achieves such lowerbounds for every level. This is why we call such a tree "most compact" tree.

Lemma 1. *For any tree appearing in the simulation path, its height is bounded by* $O(\frac{\log n}{\log\log n})$. *That is, the highest level of PCP proofs constructed during the entire simulation is bounded by* $O(\frac{\log n}{\log\log n})$.

Proof. Consider the first simulation tree that achieves the maximum height l in the simulation path.

Let node u on the level l be the root of this tree. We count the number of its children. We first verify that node u has at least k child nodes at level $(l-1)$. This can be deduced from the following reasoning. By the construction of the procedure NEXT, when the simulator arrives at the node u, there is a *normal* session t that arrives its stage 2 but has not been solved yet. We claim that each slot of session t covers at least one node on level $(l-1)$ (i.e., level-$(l-1)$ PCP construction step). Suppose otherwise, if one slot, say slot i, of the session t for which the highest level of nodes covered by it is $\leq (l-2)$, we distinguish the following two cases and show for both cases we will arrive at contradiction:

- If for every $j \geq i$, slot j of session t, the highest level of nodes covered by it is also $\leq (l-2)$, then session t should have been solved at some node at level $\leq l-1$, by our simulation strategy described in step 4(a) of the procedure NEXT;
- If there is $j > i$, slot j of session t covers a node v at level $= (l-1)$, then session t should have been solved at this node by our simulation strategy *clearance* in step 4(b) of the procedure NEXT. This means session t is a *lucky* session, not a *normal* one.

We now lowerbound the number of nodes at each level in this tree. We use a $(l-j+1)$-tuple $(i_l = u, i_{l-1}, i_{l-2}, i_{l-3}, ..., i_j)$ to describe the location of a node v on level j in the following way[9]: the first element i_l specifies the root node u; suppose the first $l-e$ elements $(i_l, i_{l-1}, i_{l-2}, ...i_e)$ (for

[9] Actually, these tuples only describe nodes appearing before u, and this strengthens our result.

$e > j$) specifies a node v' at level e, the $(l-e+1)$-th element i_{e-1} specifies the i_{e-1}-th child node on level $e-1$ of v'.

Note that for the node v on level j at location $(i_l = u, i_{l-1}, i_{l-2}, i_{l-3}, ..., i_j)$, if $k - \sum_{e=j}^{l-1}(i_e - 1) > 0$, then before arriving v the the simulator goes through $(i_{l-1} - 1)$ subtrees rooted at nodes on level $(l-1)$, then $(i_{l-2} - 1)$ subtrees rooted at nodes on level $(l-2)$,..., and then $(i_j - 1)$ subtrees rooted at nodes on level j (see Fig 2 or 9). For example, in the tree in Fig 10, before arriving the node in the red box, the simulator goes through those subtrees rooted at those node in dotted boxes.

By our simulation strategy there is a *normal* session, say session t, that just arrives at its stage 2 but has not been solved yet when the simulator arrives at node v. Note first that if a slot of session t covers a non-root node in one of above $\sum_{e=j}^{l-1}(i_e - 1)$ subtrees, it must cover (at least) one root of these subtrees[10], since otherwise it should have been solved due to the clearance step of our simulator.

Note that even if every root of the above $\sum_{e=j}^{l-1}(i_e - 1)$ number of subtrees is covered by a slot of session t, the session t still has $k - \sum_{e=j}^{l-1}(i_e - 1)$ slots left[11], and again, we can distinguish two cases as above and claim that each of these slots of session t must cover at least one node on level $(j-1)$.

We conclude that, for every $1 \le j \le l-1$, every node v on level j at location $(i_l = u, i_{l-1}, i_{l-2}, ..., i_j)$, as long as $k - \sum_{e=j}^{l-1}(i_e - 1) > 0$, the node v has at least $k - \sum_{e=j}^{l-1}(i_e - 1)$ child nodes on level $(j-1)$. This gives us that the total number of nodes on level $(j-1)$ is at least

$$
\begin{aligned}
&\sum_{\substack{\{i_{l-1},...,i_j\}: \sum_{e=j}^{l-1}(i_e-1)<k \\ 1\le i_e \le k}} \left(k - \sum_{e=j}^{l-1}(i_e - 1)\right) \\
&= \sum_{s=1}^{k} s \cdot \left(\sum_{\substack{\{i_{l-1},...,i_j\}: \sum_{e=j}^{l-1}(i_e-1)=k-s \\ 1\le i_e \le k}} 1 \right) \\
&= \sum_{s=1}^{k} s \cdot \binom{k-s+l-j-1}{l-j-1} \\
&= \frac{(k+l-j-1)(k+l-j)\binom{k+l-j-2}{l-j-1}}{(l-j)(l-j+1)}
\end{aligned}
$$

When $j = 1$, then the number of nodes at the bottom level (i.e., level 0) is

[10] Note that the height of each of these $\sum_{e=j}^{l-1}(i_e - 1)$ subtrees is smaller than or equal to the height of the preceding (according to their order of appearance in the simulation) one, and thus if a slot of session t, say slot s, covers a non-root node of a subtree rooted at node v', then there are only two cases in which the session t was not solved at node v': 1) this slot also covers v', i.e., the verifier's message of this slot does not yet appear when the simulator arrives at node v', and, 2) the slot covers (at least) the root of the preceding subtree. In the latter case, even the verifier's message of slot s appeared before the root v', session t still would not get solved (with respect to slot s) at v' because of the height condition for the clearance step.

[11] Note that, by the second condition for parent-child relationship, the first simulation tree of maximum height we consider here is actually the first tree that appeared in the simulation path. Thus, all slots of session t only cover nodes in this tree.

$$\geq \frac{(k+l-2)(k+l-1)\binom{k+l-3}{l-2}}{(l-1)l}$$
$$> \binom{k+l-3}{l-2}$$

Note that there are only polynomial number of nodes at the level 0. When taking $k = O(\log^{1+\varepsilon}(n))$, in order to make sure $\binom{k+l-3}{l-2}$ is polynomial, it must be the case that $l \in O(\frac{\log n}{\log\log n})$, otherwise, if there is some super-constant function $\alpha(n)$ such that for infinite n's, $l > \alpha(n) \cdot \frac{\log n}{\log\log n}$, then

$$\begin{aligned}
&\binom{k+l-3}{l-2} \\
&\geq \left(\frac{k+l-3}{l-2}\right)^{l-2} \\
&\geq \left(\frac{c \cdot \log^{\varepsilon} n \cdot \log\log n}{\alpha(n)}\right)^{\alpha(n)\frac{\log n}{\log\log n}-2} \\
&\geq (\log^{\varepsilon} n)^{\alpha(n)\frac{\log n}{\log\log n}-2} \quad (\text{when } \alpha(n) < \log\log n) \\
&= \left(2^{\varepsilon \log\log n}\right)^{\alpha(n)\frac{\log n}{\log\log n}-2} \\
&= \frac{n^{\varepsilon\alpha(n)}}{\log^{2\varepsilon} n}
\end{aligned}$$

which is a super-polynomial. (Note that the $\binom{k+l-3}{l-2}$ of l is monotonically increasing, so it is without loss of generality to make the above requirement $\alpha(n) < \log\log n$.)

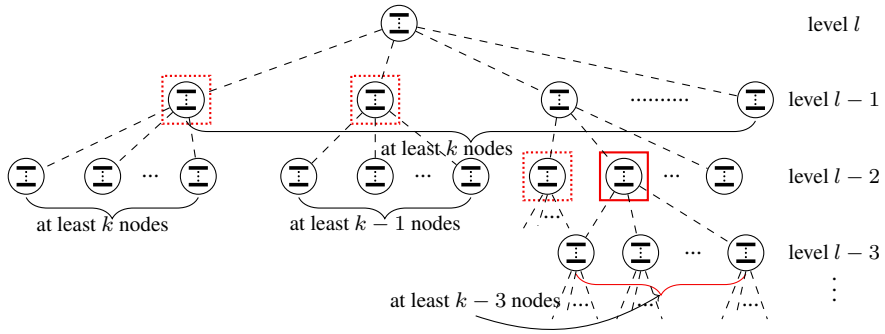

Fig. 10.

■

Given the bound $O(\frac{\log n}{\log\log n})$ on the height of any tree in the simulation path, we are now ready to prove that the simulator runs in polynomial time.

Lemma 2. *The simulator $\mathbb{S}$ runs in polynomial time.*

Proof. Note that, to prove this lemma, it is suffice to prove that the PCP construction step on the highest level l can be done in polynomial time, since there are only polynomial number of PCP construction steps in the entire simulation path.

Note that the total running time of $\mathbb{S}$ except for all PCP construction steps (i.e., the running time of V^* plus the time spent by $\mathbb{S}$ on computing all non-PCP-construction prover steps) can be bounded by a polynomial T. Consider a tree of height l in the simulation path. We let the total number of PCP proofs at level i (counting in all nodes on level i in this tree) be n^{c_i} for some constant c_i. Without loss of generality, we consider the extreme case that to compute a single PCP proof at level i, the simulator needs to reconstruct all those $\sum_{j=0}^{i-1} n^{c_j}$ PCP proofs at level $\leq (i-1)$. Note that if the time spent by the simulator on all PCP proofs at level $(i-1)$ is t_{i-1}, then the time spent on all PCP proofs at level i is bounded by

$$\sum_{j=0}^{i-1} n^{c_i} t_j \log^{O(1)} t_j$$

due to the quasi-linear time of the BSCGT UA/PCP. Note also that t_0 is bounded by $n^{c_0}(T \log^{O(1)} T)$. By a simple calculation, we conclude that the running time for all PCP proofs at level $l \in O(\frac{\log n}{\log \log n})$ is still polynomial in n.

■

The remaining task is then to prove the following lemma.

Lemma 3. *The output of $\mathbb{S}$ is indistinguishable from the real concurrent interaction between V^* and honest provers.*

Proof. We again assume that there are only n^c number of prover messages queried by V^*. Consider the following series of hybrid distributions:

- $\mathrm{H}^{(0,0)}$: $\mathbb{H}_{n^c} \leftarrow \mathrm{SIMULATE}(s_{n^c}, n^c, x, \emptyset, \emptyset, V_x^*)$, i.e., the output of S;
- $\mathrm{H}^{(1,0)}$: $(\mathbb{H}_{n^c-1}, \mathrm{R}_{1,0})$, where $\mathbb{H}_{n^c-1} \leftarrow \mathrm{SIMULATE}(s_{n^c-1}, n^c - 1, x, \emptyset, \emptyset, V_x^*)$, $s_{n^c-1} \leftarrow PRF_{s_{n^c}}(1^n)$ and $\mathrm{R}_{1,0}$ is the last prover message for which we compute using truly randomness independent of s_{n^c}.
- $\mathrm{H}^{(1,1)}(w)$: $(\mathbb{H}_{n^c-1}, \mathrm{R}_{1,1})$, where $(\mathbb{H}_{n^c-1}, \mathrm{R}_{1,1})$ is identical to $(\mathbb{H}_{n^c-1}, \mathrm{R}_{1,0})$ except that in computing $\mathrm{R}_{1,1}$ we use the witness w for x.
- $\cdots$
- $\mathrm{H}^{(i,0)}(w)$: $(\mathbb{H}_{n^c-i}, \mathrm{R}_{i,0})$;
- $\mathrm{H}^{(i,1)}(w)$: $(\mathbb{H}_{n^c-i}, \mathrm{R}_{i,1})$, where $(\mathbb{H}_{n^c-i}, \mathrm{R}_{i,1})$ is identical to $(\mathbb{H}_{n^c-1}, \mathrm{R}_{i,0})$ except that in computing $\mathrm{R}_{i,1}$ we use the witness w for x.
- $\mathrm{H}^{(i+1,0)}(w)$: $(\mathbb{H}_{n^c-i-1}, \mathrm{R}_{i+1,0})$, where $\mathbb{H}_{n^c-i-1} \leftarrow \mathrm{SIMULATE}(s_{n^c-i-1}, n^c - i - 1, x, \emptyset, \emptyset, V_x^*)$, $s_{n^c-i-1} \leftarrow PRF_{s_{n^c-i}}(1^n)$ and $\mathrm{R}_{i+1,0}$ is the remaining transcript for which we compute using truly randomness independent of s_{n^c-i}.
- $\cdots$

Note that $\mathrm{H}^{(0,0)}$ is the simulation, and $\mathrm{H}^{(n^c,n^c)}$ is the real interaction. It follows from the pseudorandomness of the function PRF that for each i, $\mathrm{H}^{(i,0)}(w)$ and $\mathrm{H}^{(i,1)}(w)$ are indistinguishable, and due to the hiding property and the witness indistinguishability of the stage 3 of the protocol, we have $\mathrm{H}^{(i,1)}(w)$ and $\mathrm{H}^{(i+1,0)}(w)$ are indistinguishable.

Observe further that there are only n^{2c+1} hybrids. This concludes the proof of this lemma. ■

6 Removing the Need for Super-Polynomial Hardness: Our Final Construction

In this section, we modify the initial protocol to obtain our final construction for concurrent zero knowledge based on standard/polynomial-time hardness.

As described in the introduction, following [BG08], we modify our initial protocol as follows. We use error correcting code ECC and Merkle hash tree for computing the commitment in the first stage, and add an extra stage (directly after the first stage) to the initial protocol in which we have the verifier send a random location j, and have the prover send back a commitment c_{ecc} to 0^{n^2}; We then let the prover prove in UA that the content of c_{ecc} is actually the j-th bit (and its authenticator) of $ECC(\Pi)$ committed in some c_i, and the corresponding slot (c_i, r_i) (together with the hash function h) forms an YES instance with respect to the language $\hat{\Lambda}$ defined for our initial protocol; In the last stage of 3-round WI argument of knowledge, in addition to the second part of the OR statement that $x \in L$ or the encrypted transcript of UA is acceptable, we also have the prover prove of knowledge of the content of c_{ecc}.

The formal description of the final protocol is depicted in figure 11. We modify the language $\widehat{\Lambda}$ accordingly as follows.

Definition of the language Λ: Let n be security parameter and $h \in \{0,1\}^n$ be a hash function mapping $\{0,1\}^*$ to $\{0,1\}^n$, and let Com be a statistically binding commitment scheme. We say a tuple $(h, c, r, j, c_{ecc}) \in \Lambda$, if there exist $(\eta, \Pi, \sigma, y, b, \sigma_b)$ such that the following conditions hold (here we will ignore the randomness for computing commitments for simplicity):

1. $|y| \leq |r|/2 = n^2/2$, $c = Com(|\eta|, MH_h(\eta))$, $\Pi = ECC^{-1}(\eta)$, σ is the authenticator for η[12], and Π on input (c, y) outputs r within super-polynomial time;
2. $c_{ecc} = Com(|\eta|, b||\sigma_b)$ and that b is the j_m-th bit of $\eta = ECC(\Pi)$ and σ_b is the authenticator for b, where $|\eta| = 2^m$ and j_m is the least significant m bits of j.

Theorem 3. *If standard collision-resistant hash functions (against polynomial time adversary) exist, then, for an arbitrarily small constant ε, the protocol presented in Fig 11 is a $O(\log^{1+\varepsilon} n)$-round concurrent zero knowledge argument.*

Proof. Completeness of this protocol is obvious. The zero knowledge property follows from our simulation strategy with $k \in O(\log^{1+\varepsilon} n)$ presented in previous section with some straightforward modifications.

The proof of soundness follows essentially from the one for the zero knowledge protocol of [BG08], except that here we will apply the extractor for the 3-round WI argument to derive a collision for hash function h. We give details below.

Suppose that there is a prover P^* that cheats on a false statement $x \notin L$ with non-negligible probability p.

Now consider the following collision finding algorithm CollFinder:

1. Invoke $P^*(x)$ on input two random hash functions h and α;

[12] I.e., σ consists of a sequence of authenticator, each for a bit of $\eta = ECC(\Pi)$

Common input: $x \in L$ ($|x| = n$).
Auxiliary input to P: w, $w \in R_L(x)$.

Stage 1

$V \longrightarrow P$: send two random hash functions $h \leftarrow_R \{0,1\}^n$ and $\alpha \leftarrow_R \{0,1\}^n$.

For $i \in [k]$, repeat the following two steps sequentially:

$P \longrightarrow V$: send $c_i = Com(0^n)$. (all commitment schemes Com are *statistically-binding*.)

$V \longrightarrow P$: send $r_i \leftarrow_R \{0,1\}^{n^2}$.

Stage 2

$V \longrightarrow P$: send $j \leftarrow_R \{0,1\}^n$.

$P \longrightarrow V$: send $c_{ecc} = Com(0^{n^2})$.

Stage 3 (Encrypted BSCGT UA, α serves as the first message of this UA)

$P \longrightarrow V$: send $\beta' = Com(0^{\text{poly}(n)})$.

$V \longrightarrow P$: send $\gamma \leftarrow_R \{0,1\}^n$

$P \longrightarrow V$: send $\delta' = Com(0^{\text{poly}(n)})$.

Stage 4

$P \Longrightarrow V$: P and V execute the 3-round witness indistinguishable argument of knowledge of [Blu86] in which P proves the OR of the following statements:

1. $\exists\, w$ so that $(x, w) \in R_L$.
2. Or, the following holds:
   (a) $\exists\, (|\eta|, b||\sigma_b)$ and s so that $c_{ecc} = Com((|\eta|, b||\sigma_b), s)$
   (b) $\exists\ i$ and $(\beta, \delta, s_1, s_2)$ so that $\beta' = Com(\beta, s_1)$, $\delta' = Com(\gamma, s_2)$, and $(\alpha, \beta, \gamma, \delta)$ is an accepting transcript of the universal argument proving the statement $(h, c_i, r_i, j, c_{ecc}) \in \Lambda$.

Fig. 11. The concurrent zero knowledge protocol (P, V)

2. Execute the protocol with $P^*(x)$ using honest verifier strategy, if the transcript is accepting, then invoke the extractor for the 3-round WI argument of knowledge to extract a slot index i and $(|\eta|, b||\sigma_b)$ committed in c_{ecc}; if failure, output " $\perp$";
   Suppose that in this session the random coins that CollFinder chooses in stage 2 is j.
3. Rewind to the point where the commitment c_i in slot i just sent, execute this session again using fresh randomness except for the same j. If the transcript is accepting, then invoke the above extractor again to extract a slot index i' and $(|\eta|, b'||\sigma'_b)$; if failure, output " $\perp$";
4. If $i = i'$, then output $b||\sigma_b$ and $b'||\sigma'_b$; otherwise, output " $\perp$".

Denote by (R_{i-1}, j) randomness used by CollFinder in the first $i-1$ slots and the stage 2 respectively. Note that for a $p/2$ fraction of the (R_{i-1}, j), $P^*(x)$ will make the verifier accept with probability at least $\frac{p}{2}$. Thus, in step 3, the probability that $P^*(x)$ gives an accepting proof again with respect to the same slot i is at least $\frac{p}{2k}$.

Recall that conditioned on an accepting transcript, the extractor for the underlying 3-round WI argument of knowledge [Blu86] will extract a witness with probability 1 in expected polynomial time. Thus, the probability that CollFinder outputs $b||\sigma_b$ and $b'||\sigma'_b$ is at least $\frac{p^2}{4k}$.

Note that for slot i, with probability 2^{-n} that the randomness r_i and r'_i used in step 3 and step 3 respectively are equal. and that when $r_i \neq r'_i$, the program Π and Π' used by $P^*(x)$ as part of witness in these two sessions must be different. And, due to the property of error correcting code ECC, the $b||\sigma_b$ and $b'||\sigma'_b$ will differ with constant probability p_c.

Observe that, $b||\sigma_b$ and $b'||\sigma'_b$ output by CollFinder were used as part of witness by $P^*(x)$ to give accepting UA proofs in step 2 and step 3, thus, they are different with probability negligibly close to p_c; otherwise, if they are different with probability at most $p_c - \frac{1}{\text{poly}}$, then $P^*(x)$ can convince an honest verifier of a false statement with probability at least $\frac{1}{2\text{poly}}$, which, by the soundness of UA, is impossible.

In sum, the algorithm CollFinder, running in expected polynomial time, can output a collision with probability at least $\frac{p^2}{4k} \cdot \frac{1}{2^n} \cdot p_c$, which breaks the collision-resistance property of the hash function h.

■

Acknowledgements I am very grateful to Eli Ben-Sasson for helpful clarification on his work of [BSCGT13]. This work evolved over the years, and I am grateful to Boaz Barak for reminding me of some important issues with the approach of "committing to the simulator's code itself" many years ago, and to Ran Canetti for many useful discussions on related topics. I would like to thank YAN Jun and LIU Meicheng for helpful discussions on the combinatorial structure of our simulation tree. Most of figures presented in this paper are drawn by CHEN Yu, many thanks!

Bibliography

[Bar01] B. Barak. How to go beyond the black-box simulation barrier. In *Proceedings of the 42th Annual IEEE Symposium on Foundations of Computer Science - FOCS'01*, pages 106–115. IEEE Computer Society, 2001.

[BCC88] G. Brassard, D. Chaum, and C. Crepeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Science*, 37(2):156C189, 1988.

[BG08] B. Barak and O. Goldreich. Universal arguments and their applications. In *SIAM Journal on Computing*, volume 38(5), pages 1661–1694. Available on line http://www.wisdom.weizmann.ac.il/ oded/R6/ua.pdf, 2008.

[BGGL01] B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell. Resettably sound zero knowledge and its applications. In *Proceedings of the 42th Annual Symposium on Foundations of Computer Science - FOCS'01*, pages 116–125. IEEE Computer Society, 2001.

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yan. On the (im)possibility of obfuscating programs. In *Advances in Cryptology - CRYPTO'01*, LNCS 2139, pages 1–18. Springer, 2001.

[BL04] B. Barak and Y. Lindell. Strict polynomial-time in simulation and extraction. In *SIAM Journal on Computing*, Volume 33(4), pages 738–818, 2004.

[Blu86] M. Blum. How to prove a theorem so no one else can claim it. In *Proceedings of international congress of mathematicians - ICM'86*, 1986.

[BOGG+88] M. Ben-Or, O. Goldreich, S. Goldwasser, J. H?stad, J. Kilian, S. Micali, and P. Rogaway. Everything provable is provable in zero-knowledge. In *Advances in Cryptology - CRYPTO'88*, LNCS 403, pages 37–56. Springer, 1988.

[BP12] N. Bitansky and O. Paneth. From the impossibility of obfuscation to a new non-black-box simulation technique. In *Proceedings of the 53th Annual Symposium on Foundations of Computer Science - FOCS'12*, pages 223–232. IEEE Computer Society, 2012.

[BP13] N. Bitansky and O. Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In *Proceedings of the 45th Annual ACM Symposium on the Theory of Computing - STOC'13*, pages 241–250. ACM Press, 2013.

[BS05] B. Barak and A. Sahai. How to play almost any mental game over the net - concurrent composition via super-polynomial simulation. In *Proceedings of the 46th Annual Symposium on Foundations of Computer Science - FOCS'05*, pages 543–552. IEEE Computer Society, 2005.

[BSCGT13] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *Proceedings of the 45rd Annual ACM Symposium Theory of Computing- STOC'13*, pages 585–594. ACM press, 2013.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42th Annual Symposium on Foundations of Computer Science - FOCS'01*, pages 136–145. IEEE Computer Society, 2001.

[CGGM00] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable zero knowledge. In *Proceedings of the 32rd Annual ACM Symposium Theory of Computing- STOC'00*, pages 235–244. ACM press, 2000.

[CKPR01] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-box concurrent zero-knowledge requires omega(log n) rounds. In *Proceedings of the 33rd Annual ACM Symposium Theory of Computing- STOC'01*, pages 570–579. ACM press, 2001.

[CLP13a] R. Canetti, H. Lin, and O. Paneth. Public-coin concurrent zero-knowledge in the global hash model. In *Theory of Cryptography - TCC'13*, LNCS 7785, pages 80–99. Springer, 2013.

[CLP13b] K. Chung, H. Lin, and R. Pass. Constant-round concurrent zero knowledge from p-certificates. In *Proceedings of the 54th Annual Symposium on Foundations of Computer Science - FOCS'13*, pages 50–59. IEEE Computer Society, 2013.

[COPV13] K. Chung, R. Ostrovsky, R. Pass, and I. Visconti. Simultaneous resettability from one-way functions. In *Proceedings of the 54th Annual Symposium on Foundations of Computer Science - FOCS'13*, pages 60–69. IEEE Computer Society, 2013.

[COSV12] C. Cho, R. Ostrovsky, A. Scafuro, and I. Visconti. Simultaneously resettable arguments of knowledge. In *Theory of Cryptography - TCC'12*, LNCS 7194, page 530C547. Springer, 2012.

[Dam00] I. Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In *Advances in Cryptology - EUROCRYPT'00*, LNCS 1807, pages 418–430. Springer, 2000.

[DFG+11] Y. Deng, D. Feng, V. Goyal, D. Lin, A. Sahai, and M. Yung. Resettable cryptography in constant rounds - the case of zero knowledge. In *Advances in Cryptology - ASIACRYPT'11*, LNCS 7073, pages 390–406. Springer, 2011.

[DGS09] Y. Deng, V. Goyal, and A. Sahai. Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In *Proceedings of the 50th Annual Symposium on Foundations of Computer Science - FOCS'09*, pages 251–260. IEEE Computer Society, 2009.

[DL07] Y. Deng and D. Lin. Instance-dependent verifiable functions and their application to simultaneous resettability. In *Advances in Cryptology - EUROCRYPT'07*, LNCS 4515, pages 148–168. Springer, 2007.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In *Proceedings of the 30rd Annual ACM Symposium Theory of Computing- STOC'98*, pages 409–418. ACM press, 1998.

[FLS99] U. Feige, D. Lapidot, and A. Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM. Journal on Computing*, 29(1):1–28, 1999.

[GJO+13] V. Goyal, A. Jain, R. Ostrovsky, S. Richelson, and I. Visconti. Constant-round concurrent zero knowledge in the bounded player model. In *Advances in Cryptology - Asiacrypt'13*, LNCS 8269, pages 21–40. Springer, 2013.

[GK96] O. Goldreich and A. Kahan. How to construct constant-round zero-knowledge proof systems for np. *Journal of Cryptology*, 9(3):167–190, 1996.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM. Journal on Computing*, 18(1):186–208, 1989.

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

[Gol01] O. Goldreich. *Foundations of Cryptography*, volume Basic Tools. Cambridge University Press, 2001.

[Goy12] V. Goyal. Positive results for concurrently secure computation in the plain model. In *Proceedings of the 53th Annual Symposium on Foundations of Computer Science - FOCS'12*, pages 41–50. IEEE Computer Society, 2012.

[Goy13] V. Goyal. Non-black-box simulation in the fully concurrent setting. In *Proceedings of the 45th Annual ACM Symposium on the Theory of Computing - STOC'13*, pages 221–230. ACM Press, 2013.

[GS12] D. Gupta and A. Sahai. On constant-round concurrent zero-knowledge from a knowledge assumption. In *Cryptology ePrint Archive: Report 2013/754,, http://eprint.iacr.org/2012/572.pdf*, 2012.

[KP01] J. Kilian and E. Petrank. Concurrent and resettable zero-knowledge in poly-loalgorithm rounds. In *Proceedings of the 33rd Annual ACM Symposium Theory of Computing- STOC'01*, pages 560–569. ACM press, 2001.

[Lin03] Y. Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *Proceedings of the 35rd Annual ACM Symposium Theory of Computing-STOC'03*, pages 683–692. ACM press, 2003.

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

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

[Pas03] R. Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In *Advances in Cryptology - EUROCRYPT'03*, LNCS 2656, pages 160–176. Springer, 2003.

[Pas04] R. Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing - STOC'04*, pages 232–241. ACM Press, 2004.

[PPS13] O. Pandey, M. Prabhakaran, and A. Sahai. Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for np. In *Cryptology ePrint Archive: Report 2013/754, http://eprint.iacr.org/2013/754.pdf*, 2013.

[PR03] R. Pass and A. Rosen. Bounded-concurrent secure two-party computation in a constant number of rounds. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science - FOCS'03*, pages 404–413. IEEE Computer Society, 2003.

[PR05] R. Pass and A. Rosen. New and improved constructions of non-malleable cryptographic protocols. In *Proceedings of the 37rd Annual ACM Symposium Theory of Computing- STOC'05*, pages 533–542. ACM press, 2005.

[PRS02] M. Prabhakaran, A. Rosen, and A. Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *Proceedings of the 43th Annual IEEE Symposium on Foundations of Computer Science - FOCS'02*, pages 366–375. IEEE Computer Society, 2002.

[PRT13] R. Pass, A. Rosen, and W. Tseng. Public-coin parallel zero knowledge for np. In *Journal of Cryptology*, Volume 26(1), pages 1–10. Springer, 2013.

[PTW11] R. Pass, W-L. D. Tseng, and D. Wikström. On the composition of public-coin zero-knowledge protocols. *SIAM. Journal on Computing*, 40(6):1529–1553, 2011.

[PV08] R. Pass and M. Venkitasubramaniam. On constant-round concurrent zero-knowledge. In *Theory of Cryptography - TCC'08*, LNCS 4948, pages 553–570. Springer, 2008.

[RK99] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In *Advances in Cryptology - EUROCRYPT'99*, LNCS 1592, pages 415–431. Springer, 1999.