Resettably-Sound Resettable Zero Knowledge Arguments for NP ⋆

Yi Deng

State Key Laboratory of Information Security,Institute of Software,
Chinese Academy of Sciences, Beijing, 100190, China
ydeng@is.iscas.ac.cn

Abstract. We construct resettably-sound resettable zero knowledge arguments for NP based on standard hardness assumption (the existence of claw-free permutations) in the plain model. This proves the simultaneous resettability conjecture posed by Barak et al. in [FOCS 2001].

Our construction, inspired by the paradigm for designing concurrent zero knowledge protocols, makes crucial use of a tool called instance-dependent resettably-sound resettable WI argument of knowledge (IDWIAOK (and a special-purpose variant), introduced recently by Deng and Lin in [Eurocrypt 2007]). Roughly speaking, for a NP statement of the form $x_0 \vee x_1$, IDWIAOK is an argument for which resettable WI property holds when both x_0 and x_1 are YES instances, and resettably-sound argument of knowledge property holds when x_0 is a NO instance.

The heart of the simulator for our protocol is a new technique that allows us to embed the (non-black-box) straight-line simulation strategy in the (black-box) recursive rewinding simulation strategy.

1 The problem and our result

It is well known that randomness is essential to zero knowledge proofs/arguments[17]. Moreover, in the multi-executions of a zero knowledge protocol, we often require that all parties use *independent* randomness in each execution for security purpose. This gives rise to natural questions: Is it possible to achieve zero knowledge when the prover uses the *same* randomness in multi-executions? Is it possible to achieve soundness when the verifier uses the *same* randomness in multi-executions? Both questions were resolved in the positive. Canetti et al. [6] put forward and realized the concept of resettable zero knowledge (stronger than the concept of *concurrent* zero knowledge[12]) argument, which allows an honest prover to use the same random tape in polynomially many executions without sacrificing the zero knowledge property; Barak et al. [3] put forward and realized the concept of resettably-sound zero knowledge argument, which allows an honest verifier to use the *same* random tape in polynomially many executions without sacrificing the soundness property.

It should be noted that the above two questions were answered *separately*: the proof system presented by Canetti et al. is resettable zero knowledge but not resettably-sound, whereas the argument system presented by Barak et al. is resettably-sound but not resettable zero knowledge. This leaves a challenge in this line of research: Can we construct a single argument system for some nontrivial language that remains resettable zero knowledge and resettably-soundness *simultaneously*? Indeed, Barak et al. conjectured the following [3]:

Simultaneous resettability conjecture: there exist resettably-sound resettable ZK arguments for NP.

We stress that this conjecture is for the case of argument system (rather than proof system) and non-black box zero knowledge. Previous work [3] showed that, for non-trivial

⋆ submitted on Nov. 18.

language, neither resettably-sound zero knowledge proof system nor resettably-sound black-box zero knowledge argument system exists.

Related work. Recently, Deng and Lin introduced a series of instance-dependent primitive/protocols (we will give a detailed description later), and made the first attempt to tackle this problem [9,10]. In particular, they constructed resettably-sound *class-bounded* resettable zero knowledge arguments for NP [9] and proved the above conjecture in the BPK model [10], which assumes each verifier deposits a public key in a public file before any interaction with the prover begins.

Our result. In this paper we prove the simultaneous resettability conjecture, i.e., we construct resettably-sound resettable zero knowledge arguments for NP based on standard hardness assumption (the existence of claw-free permutations) in the plain model. Our protocol takes $k = n^{\epsilon}$ round for an arbitrary constant ϵ, i.e., the same (up to a constant factor) round complexity as the protocol suggested by Richardson and Kilian [22], where n is the security parameter.

Theorem 1. If there exist claw-free trapdoor permutations, then there exist resettably-sound resettable ZK arguments for NP in the plain model.

Techniques. Our construction can be viewed as an extension of the one given in the BPK model [10] by increasing many "rewinding slot" for simulator, which is similar in spirit to the typical method for achieving concurrent zero knowledge [22,21]. Our protocol makes crucial use of a tool called instance-dependent resettably-sound resettable WI argument of knowledge (IDWIAOK, introduced recently by Deng and Lin in [9]). Roughly speaking, for a NP statement of the form $x_0 \vee x_1$, IDWIAOK is an argument for which resettable WI property holds when both x_0 and x_1 are YES instances, and resettably-sound argument of knowledge property holds when x_0 is a NO instance.

The most important feature of our construction is a simple but useful variant of IDWIAOK which we call special-purpose IDWIAOK. This argument plays a pivotal role in the security analysis of our non-black-box simulator.

The heart of the simulator for our protocol is a new technique that allows us to embed the (non-black-box) straight-line simulation strategy in the (black-box) recursive rewinding simulation strategy. Here the recursive rewinding strategy we use is the original one suggested by Richardson and Kilian [22]. The analysis of our simulator heavily relies on the fact that the *recursive depth* of the RK rewinding strategy is upperbounded by a super-constant. The reason why we don't adopt the more advanced PRS strategy [21] is that the analysis of the PRS simulation strategy requires us to deal with the *rewind interval*, which, given the current state of non-black-box techniques, can hardly be done in our setting where non-black-box simulation is needed.

Rest of the paper. Due to space limitations, we postpone definitions to appendix A. In section 3, we briefly recall several instance-dependent protocols introduced in [9,10] and postpone its detailed description and formal analysis to appendix B. We describe our protocol and a high level description of the simulation strategy in section 4 and 5. The formal description of our simulator and its analysis can be found in appendix C. The proof of soundness of our protocol is postponed to appendix D. The analysis of the running time of our simulator is presented in appendix E.

2 Definitions

Due to space limitations, we defer formal definitions to appendix A. Here we just give informal definitions of class of sessions, and the class-bounded resetting attack.

A class of sessions consists of all sessions between a verifier and a fixed incarnation of prover that share the *same* verifier's first message. We denote a class containing all sessions between $P^{(l,m)} = P_{x_l,w_l,r_m}$ and V^* with the same V^*'s first message $\mathsf{f\text{-}msg}$ with $C^{(l,m)}_{\mathsf{f\text{-}msg}}$, where x_l is a common input, w_l is the corresponding witness, and r_m is the random tape.

Class-bounded resetting attack and Class-bounded resettable ZK. Class-bounded resetting attack is a resetting attack under the restriction that there is an a-priori bound on the total number of resetting malicious party's distinct first messages and the incarnations of honest party with which the malicious resetting party interact. An argument system is said to be Class-bounded resettable ZK if it remains zero knowledge against this kind of resetting attack mounted by malicious verifier.

3 Instance-dependent protocols of [9,10] and their variants for our special purpose

In this section, we briefly recall the instance-dependent protocols introduced in [9,10] and give some variants of these protocols tailored to our special requirement. For completeness, we present the formal description of these variants and their security analysis in appendix B.

The start point of [9] is a simple observation: To prove the simultaneous resettability conjecture, we just need to construct an *instance-dependent* argument such that, given instance x as common input, the resettable zero knowledge property holds when x is a YES instance, and the resettably-soundness holds when x is a NO instance. This inspired the authors of [9] to introduce a series of *instance-dependent* protocols and made partial progress toward this conjecture.

3.1 Instance-dependent verifiable random function.

The most basic notion introduced in [9] is instance-dependent verifiable random functions (InstD-VRFs). Informally, an InstD-VRF is, in some sense, a verifiable random function [18] with a special public key of the form$(y, \cdot)$, where y is an instance with respect to a specific NP language L and may be generated via an (possibly)*interactive* protocol, but the security requirements on such a function are relaxed: we only require the *pseudorandomness* property when $y \in L$ and only require the *uniqueness* property when $y \notin L$, instead of requiring both pseudorandomness and uniqueness to hold *simultaneously*.

The InstD-VRF can be constructed as follows [9]. The querier Alice and the function owner Bob execute a protocol KGProt to produce an key_instance y, and Alice sends a first-round message ρ of the two-round (resettable-soundness and resettable WI) proof system ZAP [11], then Bob selects a pseudorandom function f_s at random and computes a commitment $c = Com(s, r)$ to the description s of f_s useing a *statistical binding* commitment scheme Com. This establishes a key pair $(PK = (y, c, \rho), SK = (s, r))$, and gives rise to the following InstD-VRF function:

- $\mathsf{F}_{(PK,SK)} = (f_s(\cdot), prov(\cdot))$ (where $prov$ is the ZAP prover strategy): On input a string a in the domain of f_s, $\mathsf{F}_{(PK,SK)}$ returns $f_s(a)$ and a ZAP proof π that either this function value is correct or $y \in L$ using the witness SK.

It is easy to verify the *uniqueness* of $\mathsf{F}_{(PK,SK)}$ on NO instance y (i.e., there exist no values $(a, b, b', PK, \pi, \pi')$ such that $\mathsf{Ver}(a, b, PK, \pi) = \mathsf{Ver}(a, b', PK, \pi') = 1$ except with a negligible probability.). If y is a YES instance, The *pseudorandomness* of $\mathsf{F}_{(PK,SK)}$ is demonstrated in the following way: the output of $\mathsf{F}_{(PK,SK)}$ is indistinguishable from the output of the following function:

– $\mathsf{FakeF}_{(PK,w_y)}$: On input a string a in the domain of f_s, $\mathsf{FakeF}_{(PK,SK)}$ returns a truly random string b ($|b| = |f_s(a)|$), and a ZAP proof π that either this function value is correct or $y \in L$ *using the witness* w_y.

We should note that we do not specify how to generate the instance y here, and as we will see, designing the key generation protocol KGProt is a subtle issue and may vary depending on specific applications.

3.2 The key_instance-dependent ZK argument and our *perfect* ZK variant

The illuminative paradigm for the simultaneously resettable argument suggested in [9] is the so-called key_instance-dependent resettably-sound class-bounded resettable zero knowledge argument (denoted $\mathsf{ZK}_{\mathsf{KInstD}}$ argument). In this protocol, the verifier generates an extra key_instance y in its first step, and its security properties depends on y: The class-bounded resettable ZK holds on NO instance y and the resettable-soundness holds on YES instance y.

Here we require class-bounded resettable *perfect* ZK property of the $\mathsf{ZK}_{\mathsf{KInstD}}$ argument to hold under some specific conditions (we denote such an argument $\mathsf{PZK}_{\mathsf{KInstD}}$ argument) for our purpose. To achieve this, we use a simple version of the Pass-Rosen variant of Barak's public-coin protocol[20] as a building block. This simplified Pass-Rosen protocol satisfies both bounded concurrent *perfect* zero knowledge property and argument of knowledge property. See appendix B for the formal description.

We construct $\mathsf{PZK}_{\mathsf{KInstD}}$ argument by applying the same transformation of [9] to the simplified Pass-Rosen protocol, which consists of two steps:

1. From the simplified (*perfect ZK*) Pass-Rosen protocol to a resettably-sound bounded concurrent ZK argument $(P_{\mathrm{R}}, V_{\mathrm{R}})$. This can be done using the same transformation presented in [3], i.e., by having the verifier apply a pseudorandom function to history to generate its messages.
2. From $(P_{\mathrm{R}}, V_{\mathrm{R}})$ to $\mathsf{PZK}_{\mathsf{KInstD}}$ argument. This step is depicted in figure 1[1]

The $\mathsf{PZK}_{\mathsf{KInstD}}$ argument enjoys the following properties (see appendix B for detailed analysis).

1. It is t-class-bounded resettable ZK if all y are NO (key) instances. Note that the uniqueness of the $\mathsf{InstD\text{-}VRF}$ function $\mathsf{F}_{(PK,SK)}$ on NO (key) instance y guarantees that, except with negligible small probability, there exist no (r, π) and (r', π') with $r \neq r'$ such that $(r, \pi) = (r', \pi') = \mathsf{F}_{(PK,SK)}(hist)$. Thus, when y is a NO instance, the class of sessions (having the same verifier first message of form $(y, \cdot)$) with respect to (P, V) contains only a *single* session with respect to $(P_{\mathrm{R}}, V_{\mathrm{R}})$.
2. It is t-bounded-class resettable *perfect* ZK if the following conditions (**Conditions for PZK**) hold:

[1] There is a minor difference between the transformation presented here and the one of [9]: We found that it is not necessary to base the first prover step message ρ on the session history, and this message can be fixed once and for all.

$\mathsf{PZK}_{\mathsf{KInstD}}$

Common input: $x \in L$ ($|x| = n$).
The Prover's private input: the witness w such that $(x, w) \in R_L$.
Prover's randomness: (ρ, r_p).
Verifier's randomness: r_v.

Stage 1: the key generation protocol KGProt

$V \to P$: V sets $(s_0, r_0) = f_{r_v}(x)$, computes $c_0 = Com(s_0, r_0)$ to the pseudorandom function f_{s_0} using the *statistically-binding commitment scheme* Com and generates an instance $y \in L' \cap \{0,1\}^n$, stores $SK = (s_0, r_0)$; V Sends c_0, y;

$P \to V$: P sends the first message ρ for a ZAP.
At the end of this step, the key pair $(PK, SK) = ((y, c_0, \rho), (s_0, r_0))$ *for an* *InstD-VRF* F *is set up for* V.

Stage 2: the *modified* resettably-sound t-bounded concurrent perfect ZK argument $(P_{\mathrm{R}}, V_{\mathrm{R}})$

V: Given the current history *hist* of execution of $(P_{\mathrm{R}}, V_{\mathrm{R}})$ (excluding those ZAP proofs), V sends $(r, \pi) = \mathsf{F}_{(PK,SK)}(hist) = (f_{s_0}(hist), prov(hist))$, where V applies f_{r_v} to *hist* to generate the randomness used by *prov*. V accepts if only if V_{R} accepts the transcript that generated by P_{R} and V_{R}.
Note that r *can be viewed as the output of* $V_{\mathrm{R}}(s_0, hist)$.

P: sets $r'_p = f_{r_p}(x, c_0, y)$ and feeds P_{R} with random tape r'_p. P checks that the current message (r, π) received from V is correct, if so, P runs P_{R} to produce the next prover message; Otherwise, aborts.

Fig. 1. The $\mathsf{PZK}_{\mathsf{KInstD}}$ Argument.

- The honest incarnation of the prover feeds P_{R} with independent and truly random tape in stage 2 under the consistence condition that the random tape for P_{R} is the same in every session having the same verifier first message (y, c_0);
- The verifier never sends two "correct" messages (r, π) and (r', π') with $r \neq r'$ for the same history *hist* in stage 2 of $\mathsf{PZK}_{\mathsf{KInstD}}$.

3. It is resettable-sound *argument of knowledge* if all y are YES (key) instances.

Remarks on Conditions for PZK. We stress that the first condition for PZK mentioned above does not hold for the actual honest provers of our protocol, and hence the subprotocol $\mathsf{PZK}_{\mathsf{KInstD}}$ therein is not (class-bounded resettable) perfect zero knowledge. This perfect ZK property is only used to analyze our simulator (see Lemma 2 in appendix C): we construct a hybrid simulator (without doing any non-black-box simulation) and use the perfect ZK property of the subprotocol $\mathsf{PZK}_{\mathsf{KInstD}}$ to show that this hybrid simulator is identical to the actual simulator. We think of the hybrid simulator as the honest provers (with valid witnesses) of the $\mathsf{PZK}_{\mathsf{KInstD}}$ arguments, and for the hybrid one, the conditions for PZK hold.

3.3 The instance-dependent resettably-sound resettable WI AOK (IDWIAOK)

To make $\mathsf{PZK}_{\mathsf{KInstD}}$ argument secure in the real world (instead of depending on the verifier's behavior), Deng and Lin [9] introduce another instance-dependent protocol — the instance-dependent resettably-sound resettable WI argument of knowledge (IDWIAOK)— to settle the problem of generating the key_instance y *securely* and *fairly*. Informally, IDWIAOK is an argument for NP statement of the form $x_0 \vee x_1$ for which resettable WI property holds when both x_0 and x_1 are YES instances, and resettably-sound argument of knowledge property

holds when x_0 is a NO instance[2]. It is interesting to note that, this IDWIAOK, aiming at settling the key_instance generation issue arising in the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument, uses $\mathsf{PZK}_{\mathsf{KInstD}}$ argument as a building block itself.

The construction of IDWIAOK[9,10] employs the classic 3-round WI argument of knowledge [5] as a building block. Let (a, e, z) be the three messages exchanged in an execution of this argument. Note that the argument of [5] has the following property: we can efficiently extract the witness for the statement from two different transcripts (a, e, z) and (a, e', z') when $e \neq e'$. Given $x_0 \in L_0$ *or* $x_1 \in L_1$ as the common input, the original IDWIAOK in [9] proceeds as follows. The verifier first sends a commitment to a pseudorandom function; Upon receiving the first message a of the underlying 3-round WI argument from the prover, it generates the query e by applying the committed function to a and executes a $\mathsf{PZK}_{\mathsf{KInstD}}$ argument to prove the query e is correct. In the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument, the instance x_0, serves as the key_instance for an InstD-VRF used by the verifier (the prover in the global system) in the stage 2. We note that the original construction of IDWIAOK [9] satisfies only a *weak* type of resettable-soundness on NO instance x_0, and fortunately, this was improved and in a follow-up [10], in which the full-fledged IDWIAOK (as defined in definition 5) was finally constructed. This is done by plugging an extra InstD-VRF-like component into the original IDWIAOK. In particular, in the improved IDWIAOK the prover sends a commitment c^a to a pseudorandom function as initial message. Upon receiving the verifier's message c^e (the first message sent in the original IDWIAOK), the prover produces the message a using the randomness generated by applying the pseudorandom function committed in c^a to the history so far and uses a ZAP to prove that $x_0 \in L_0$ or a is computed correctly[3]. The rest part of this improved IDWIAOK proceeds as the original one. The key idea of this modification allowing us to achieve *fully* resettable soundness on NO instance x_0 is that, when $x_0 \notin L_0$, for each verifier's first message c^e, any resetting prover cannot produce two different messages a and a' (if V produces c^e depending on c^a).

Note also that with this improvement, the verifier can commit to the challenge e directly in its first step, not necessarily commit to a pseudorandom function.

We present the construction of [10] in figure 2, and postpone the detailed security analysis to appendix B. For simplification, we view the 2-round perfect-hiding commitment scheme Com_v, which can be based on claw-free trapdoor permutations, as a non-interactive scheme.

We would like to emphasize the following features of the IDWIAOK.

- The key_instance of the underlying $\mathsf{PZK}_{\mathsf{KInstD}}$ argument is the first instance x_0 of the common input for the IDWIAOK. This leads the IDWIAOK to be *instance-dependent*.
- The commitment scheme Com_v is *perfect hiding*. (whereas The commitment scheme Com_p is *statistically-binding*). This typically requires the underlying argument $\mathsf{PZK}_{\mathsf{KInstD}}$ to be *argument of knowledge* (on YES key_instance) for the purpose of security analysis, since for any challenge e, the statement "e is correct" is true. Actually, to show "e is correct", the verifier proves its knowledge of s such that $c^e = Com_v(e, s)$ via argument $\mathsf{PZK}_{\mathsf{KInstD}}$.
- Note that 1-class-bounded resettable ZK property of the underlying $\mathsf{PZK}_{\mathsf{KInstD}}$ argument on NO instance x_0 is sufficient to achieve full-fledged resettable-soundness (due to the

[2] It should be noted that the resettably-sound resettable WI proof system ZAPs do not statisfy proof of knowledge property.

[3] This requires the first message a to be *uniquely* determined by the randomness used in this step and the common input. We note that the classic parallelized version of Blum's (WI) proof of knowledge for Hamiltonian Cycle satisfies this property.

IDWIAOK

Common input: two instances $x_0 \in L_0$ *or* $x_1 \in L_1$, a security parameter n.
The Prover's private input: the witness w such that $(x_0, w) \in R_{L_0}$ or $(x_1, w) \in R_{L_1}$.
Prover's randomness: r_p.
Verifier's randomness: r_v.

$P \to V$ P sets $(r_p^1, r_p^2) = f_{r_p}(x_0, x_1)$ and $r_p^1 = (s', r')$, where f_{r_p} is the pseudorandom function specified by r_p, and computes $c^a = Com_p(s', r')$ using a statistically binding commitment scheme Com_p.
Using the randomness r_p^2, P invokes the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument (it is sufficient to use a $\mathsf{PZK}_{\mathsf{KInstD}}$ that is only *1-class-bounded resettable ZK*) in which P plays the role of verifier, produces the first message c_0 of this $\mathsf{PZK}_{\mathsf{KInstD}}$ argument.
Sends c^a,c_0;

$V \to P$ V sets $(r_v^1, \rho') = f_{r_v}(x_0, x_1, c^a, c_0)$ and $r_v^1 = (e, s)$, where f_{r_v} is the pseudorandom function specified by r_v, and computes $c^e = Com_v(e, s)$ using a perfect-hiding commitment scheme Com_v. ρ' will serve as the first message of a ZAP used in next P's step.
Sends c^e, ρ';

$P \to V$ P reduces the instance (x_0, x_1) to a Hamiltonian graph G, and sets $(r_\pi, r_M) = f_{s'}(x_0, x_1, c^a, c_0, c^e, \rho')$, here r_π represents a permutation over the vertices of G, and r_M is a random string matrix as defined before.
P invokes the 3 round WI argument for Hamiltonian Cycle in which it proves G (or, equivalently, $x_0 \in L$ or $x_1 \in L$) has a Hamiltonian cycle, produces the first message $a = Com(M_{r_\pi(G)}, r_M)$ (where Com is also a statistically-binding commitment scheme), and uses the witness s' and r' to prove that $x_0 \in L_0$ or there exist s' and r' such that $c^a = Com_p(s', r') \wedge (r_\pi, r_M) = f_{s'}(x_0, x_1, c^a, c_0, c, \rho') \wedge a = Com(M_{r_\pi(G)}, r_M)$ via a ZAP (with the first message ρ'). Let the second message (the proof) be τ.
Sends a, τ;

$V \to P$ Sends e (the string committed in c^e);

$V \Rightarrow P$ V sets $r_v^2 = f_{r_v}(x_0, x_1, c^a, c_0, a, \tau, e)$. Using r_v^2 as *random tape*, V runs the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument in which he plays the role of prover, and proves that there exists s such that $c^e = Com_v(e, s)$. In this $\mathsf{PZK}_{\mathsf{KInstD}}$ argument, the public key for P's InstD-VRF function (P plays the role of the verifier in this subprotocol) consists of (x_0, c_0) and the first V's (prover's) message ρ sent in this execution of the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument, and the corresponding secret key is the decommitment to c_0.

***Comment**: The key_instance of the underlying* $\mathsf{PZK}_{\mathsf{KInstD}}$ *argument is* x_0*, the first instance of the common input for the* IDWIAOK*. Note also that the argument* $\mathsf{PZK}_{\mathsf{KInstD}}$ *satisfies argument of knowledge property on YES instance* x_0*.*

$P \to V$ Sends the answer z to the query e according to the 3 round WI argument for Hamiltonian Cycle if the above transcript is accepting.

V**'s Decision** V accepts if only if the transcript (a, e, z) is accepting.

Fig. 2. The instance-dependent resettably-sound resettable WI argument of knowledge.

fact that we just need to focus on a single class of sessions to justify soundness). However, for some IDWIAOK used as building block in our global protocol presented in next section, we need $\log n$-class-bounded resettable ZK property of the underlying $\mathsf{PZK}_{\mathsf{KInstD}}$ argument.

For common input of the form $x_0 \in L_0$ *or* $x_1 \in L_1$, the IDWIAOK satisfies (see appendix B.3 for detailed proof):

1. Resettable WI (with respect to the witness for x_0 and the witness for x_1), as in definition 3.
2. Resettably-sound argument of knowledge property when $x_0 \notin L_0$.

Remark on the extraction strategy. We present an extraction strategy E in appendix B.3, which will be used in the proof of resettable-soundness for our main protocol. However,

we adopt a different and simpler extraction strategy in the simulation (for establishing resettable ZK) of the main protocol.

3.4 The overall paradigm for simultaneously resettable argument of [9]

The paradigm for simultaneously resettable argument in the plain model in [9] is to modify the stage 1 of the argument $\mathsf{PZK}_{\mathsf{KInstD}}$ in the following way: instead of letting verifier choose the key_instance y, we have the honest prover generate an NO instance y for the verifier and carry out the IDWIAOK to prove that the statement $x \in L$ (the common input) is true or y is a YES instance. The stage 2 of the argument $\mathsf{PZK}_{\mathsf{KInstD}}$ remains unchanged. As showed in [9], the resulting protocol satisfies some restricted version of simultaneous resettability.

4 The Resettably-sound Resettable Zero Knowledge Argument.

We extend the simultaneously resettable argument in the BPK model recently proposed in [10] to the plain model by adding many *rewinding slots* (with some *crucial* modifications). Increasing the rewinding opportunities is a typical method used to achieve concurrent zero knowledge. Our argument takes $k = n^{\epsilon}$ rounds for an arbitrary constant ϵ, i.e., the same (up to a constant factor) round complexity as the protocol suggested by Richardson and Kilian [22], where n is the security parameter.

Let G be a pseudorandom generator and f be an one-way function. Roughly, the protocol proceeds as follows. The prover first chooses a random string γ and proves that $x \in L$ (the common input) or there exists δ such that $\gamma = G(\delta)$ via an IDWIAOK (denoted $\mathsf{IDWIAOK}_p^{\mathsf{S}}$), then the verifier chooses a random string α, computes $\beta = f(\alpha)$, and repeats an special-purpose IDWIAOK (defined below) k times (which gives the simulator many opportunities to rewind), in each iteration (denoted $\mathsf{IDWIAOK}_v^i$, $i = 1, ..., k$) he proves that there exist δ such that $\gamma = G(\delta)$ or he knows a preimage of β; At the last stage, the prover proves that $x \in L$ or he knows a preimage of β via an IDWIAOK (denoted $\mathsf{IDWIAOK}_p^{\mathsf{M}}$).

Keep in mind that the IDWIAOK is sensitive to the order of the OR statements (given as its common input), and that the first instance (statement) of its common input serves as the key_instance of the underlying argument $\mathsf{PZK}_{\mathsf{KInstD}}$.

The formal description of our construction is depicted in figure 3. We point out here the most distinguishing features of our protocol.

- All $\mathsf{IDWIAOK}_v^i$ are special-purpose IDWIAOK. The only difference between (ordinary) IDWIAOK and special-purpose $\mathsf{IDWIAOK}_v^i$ is the following:
  - In the special-purpose $\mathsf{IDWIAOK}_v^i$, to show the correctness of the challenge e, the verifier (the prover of the global protocol) proves *an OR statement* that it knows s such that $c^e = Com_v(e, s)$ *or* $x \in L$;
  - In the (ordinary) $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ and $\mathsf{IDWIAOK}_p^{\mathsf{M}}$, to show the correctness of the challenge e, the verifier proves *a sole statement* that it knows r such that $c^e = Com_v(e, s)$.
- The $\mathsf{PZK}_{\mathsf{KInstD}}$ argument used in those $\mathsf{IDWIAOK}_v^i$ satisfies $\log n$-class-bounded resettable zero knowledge on NO key_instance (i.e., the random string γ), whereas the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument used in $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ and $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ satisfies only 1-class-bounded resettable zero knowledge on NO key_instance (i.e., the common input x). Note that the former can be constructed from the simplified Pass-Rosen $\log n$-bounded concurrent ZK protocol.

Note that the common input for every special-purpose $\mathsf{IDWIAOK}_v^i$ consists of two instances γ and β, and that γ serves as the key_instance of the underlying argument $\mathsf{PZK}_{\mathsf{KInstD}}$. With the above modification, the special-purpose $\mathsf{IDWIAOK}_v^i$ has the following properties, which play a pivotal role in the security analysis of the main protocol.

1. The special-purpose $\mathsf{IDWIAOK}_v^i$ remains the resettably-sound argument of knowledge property on NO instance γ (i.e., γ is truly random). This is because that the extraction strategy for (ordinary) IDWIAOK applies to the special-purpose $\mathsf{IDWIAOK}_v^i$ by simply ignoring the second statement $x \in L$ of the OR statements being proven via the underlying argument $\mathsf{PZK}_{\mathsf{KInstD}}$.
2. When $x \in L$, the knowledge of the witness for $x \in L$ enables us to do extraction from the special-purpose $\mathsf{IDWIAOK}_v^i$ *without using any non-black-box simulation.* Observe that in execution of the special-purpose $\mathsf{IDWIAOK}_v^i$, to show the challenge e is valid, we can use the witness for $x \in L$ to carry out the underlying argument $\mathsf{PZK}_{\mathsf{KInstD}}$ (to prove the knowledge of s such that $c^e = Com_v(e, s)$ or $x \in L$).
   Note that we only need the resettably-sound argument of knowledge property of the special-purpose $\mathsf{IDWIAOK}_v^i$ to establish the resettable ZK for our main protocol (see appendix C for the detailed proof), and that the zero knowledge condition refers only to YES instance x.
3. When $x \notin L$, the special-purpose $\mathsf{IDWIAOK}_v^i$ remains the resettable WI property. The proof of resettable WI property of the special-purpose IDWIAOK is exactly the same as the one for ordinary IDWIAOK in case $x \notin L$.
   Note that we only need the resettable WI property of the special-purpose $\mathsf{IDWIAOK}_v^i$ to establish the resettable-soundness for our main protocol (see appendix D for the detailed proof), and that the soundness condition refers only to NO instance x.

The above second observation allows us to construct a hybrid simulator for our main protocol (given all witnesses for the common inputs) that does not use any non-black-box technique, which plays a crucial role in analysis of the actual non-black-box simulator (see proof of Lemma 2 in appendix C.2). This is the motivation for introducing the special-purpose IDWIAOK.

Intuition behind our construction. Consider the following simulator. In every session, it generates a YES instance γ (i.e., $\exists \delta$ s.t. $\gamma = G(\delta)$) and uses δ as witness to execute the first $\mathsf{IDWIAOK}_p^{\mathsf{S}}$, and then it takes a recursive rewinding strategy, as the simulator in the concurrent model, to extract a preimage to β from executions of those special-purpose $\mathsf{IDWIAOK}_v^i$; Once a preimage is obtained, it can complete the last stage $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ successfully. Note that $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ and $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ are resettable WI. Thus, to prove zero knowledge property, we need to make sure: 1) The extraction can be done by using the extractor associated with special-purpose $\mathsf{IDWIAOK}_v^i$ *even though γ is a YES instance in this setting* (Note that special-purpose $\mathsf{IDWIAOK}_v^i$ is an *argument of knowledge* when γ, the first instance of the common input to this argument, is NO instance.[4]); and 2) Develop an recursive rewinding strategy that works with non-black-box extraction.

To prove *resettable-soundness*, we construct an algorithm B that break the one-wayness of f in the following way: B first extracts the solution to the puzzle γ from the malicious resetting prover P^* in execution of $\mathsf{IDWIAOK}_p^{\mathsf{S}}$, and uses this preimage to complete all those special-purpose $\mathsf{IDWIAOK}_v^i$, then it extracts a preimage to β in execution of $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ (i.e., finds the preimage of β). B works since $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ and $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ are resettably-sound argument of knowledge when $x \notin L$, and all special-purpose $\mathsf{IDWIAOK}_v^i$ are resettable WI in case of $x \notin L$.

[4] It seems that the extractor for IDWIAOK (presented in appendix B.3, as mentioned before, it works also for special-purpose IDWIAOK) works only in case the first instance γ is NO instance. However, as we will see, assuming the language defined by G is a hard-to-decide, we can also do extraction even when γ is a YES instance in our setting where the YES instance γ is generated by the simulator. This is implied by lemma 2 and lemma 3 presented in appendix C.

The Resettably-Sound Resttable ZK Argument (P, V)

Common input: $x \in L$ ($|x| = n$).
P**'s private input**: the witness w such that $(x, w) \in R_L$.
P**'s randomness**: (γ, r_p), where γ is chosen uniformly at random and $|\gamma| = 2n$.
V**'s randomness**: r_v.

/***Important note:** *Keep in mind that for every* IDWIAOK *(including those* special-purpose *ones) used here, the key_instance of the underlying argument* $\mathsf{PZK}_{\mathsf{KInstD}}$ *is the first instance of the common input (consisting of an OR statement) to the corresponding* IDWIAOK*, and thus the order of these OR statements can not be changed.*

Common input to $\mathsf{IDWIAOK}_p^{\mathsf{S}}$*:* (x, γ)*; key_instance of the underlying* $\mathsf{PZK}_{\mathsf{KInstD}}$*:* x
Common input to special-purpose $\mathsf{IDWIAOK}_v^i$*:* (γ, β)*; key_instance of the underlying* $\mathsf{PZK}_{\mathsf{KInstD}}$*:* γ
Common input to $\mathsf{IDWIAOK}_p^{\mathsf{M}}$*:* (x, β)*; key_instance of the underlying* $\mathsf{PZK}_{\mathsf{KInstD}}$*:* x */*

Stage Initiation

$P \longrightarrow V$**:** P sends γ;

$V \longrightarrow P$**:** Set $(\alpha, r_v^1, ..., r_v^k) = f_{r_v}(x, \gamma)$, computes $\beta = f(\alpha)$, where f is an one-way function; V invokes k special-purpose $\mathsf{IDWIAOK}_v^1, ..., \mathsf{IDWIAOK}_v^k$, in each $\mathsf{IDWIAOK}_v^i$, V uses r_v^i as the prover's randomness and proves to P that there exists δ such that $\gamma = G(\delta)$ or there exists α such that $\beta = f(\alpha)$, where G is a pseudorandom generator specified by P; V computes the first messages $(c_1^a, c_0^1), ..., (c_k^a, c_0^k)$ of these $\mathsf{IDWIAOK}_v^i$.
V sends β and $(c_1^a, c_0^1), ..., (c_k^a, c_0^k)$;

$P \longrightarrow V$**:** Set $(r_p^1, ..., r_p^k) = f_{r_p}(x, \gamma, \beta, c_1^a, c_0^1, ..., c_k^a, c_0^k)$. For each i, P computes the second message c_i^e, ρ^i of the special-purpose $\mathsf{IDWIAOK}_v^i$, in which P plays the role of verifier and uses r_p^i as verifier's random tape.
P sends $(c_1^e, \rho_1), ..., (c_k^e, \rho_k)$;

$V \longrightarrow P$**:** For each i, V computes the third message a_i, τ_i of the special-purpose $\mathsf{IDWIAOK}_v^i$. V sends $(a_1, \tau_1), ..., (a_k, \tau_k)$;

/**Comment: note that the verifier's action in all these special-purpose* $\mathsf{IDWIAOK}_v^i$ *is essentially determined by its first message./**

Stage Setup

$P \Longrightarrow V$**:** If all a_i are correct (otherwise, P aborts), P and V execute an $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ in which P proves that $x \in L$ or there exists δ such that $\gamma = G(\delta)$ by using w as witness; At the beginning of $\mathsf{IDWIAOK}_p^{\mathsf{S}}$, P (V) applies f_{r_p} (f_{r_v}, resp.) to the history sofar (the statement and the above four messages) to generates its random tape for the prover (verifier) in this subprotocol.

Stage Iteration

For $i = 1$ to k do

$P \longrightarrow V$**:** P sends e_i (committed in c_i^e) according to the special-purpose $\mathsf{IDWIAOK}_v^i$.

$P \Longrightarrow V$**:** P and V execute the subprotocol $\mathsf{PZK}_{\mathsf{KInstD}}$ (which is $\log n$-class-bounded resettable ZK on NO (key) instance γ) in the special-purpose $\mathsf{IDWIAOK}_v^i$, in which P uses s_i as witness to proves the following *OR statements*: there exist s_i such that (e_i, s_i) is a valid decommitment to c_i^e or $x \in L$.
/**Note that proving an OR statements to justify the challenge is the only speciality of the special-purpose* $\mathsf{IDWIAOK}_v^i$*./**

$V \longrightarrow P$**:** If the above subproof is accepting, V sends z_i, the last message of the subprotocol 3-round WI argument in the special-purpose $\mathsf{IDWIAOK}_v^i$; Otherwise, V aborts.

Stage Mainproof

$P \Longrightarrow V$**:** P and V execute an $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ in which P proves that $x \in L$ or there exists α such that $\beta = f(\alpha)$ by using w as witness; Again, at the beginning of $\mathsf{IDWIAOK}_p^{\mathsf{M}}$, P (V) applies f_{r_p} (f_{r_v}, resp.) to the history sofar to generates its random tape for the prover (verifier) in this subprotocol.

Fig. 3. The resettably-sound resettable ZK argument for a NP language L.

Hardness assumption. Note that the 2-round perfect-hiding commitment scheme can be based on claw-free trapdoor permutations, which already implies existence of the instance-dependent VRF and statistically-binding commitment scheme. Note also that claw-free trapdoor permutations implies collision-free hash functions, thus we can base Barak's public-coin zero knowledge argument on the sole assumption of existence of claw-free trapdoor permutations.

Observation on the structure of a class of sessions. We categorize the real interaction into different classes of sessions. Recall that a class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$ contains all sessions between $P^{(l,m)} = P_{x_l,w_l,r_m}$ and V^* with the same V^*'s first message f-msg.

We observe the following: 1) In the real interaction between honest provers and a malicious resetting verifier V^*, except with negligible probability, all sessions in a single class share the *same* sequence of main messages (except for *those ZAP proofs for validating the corresponding verifier's messages in stage 2 of the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument*[5].) in the stage Iteration, due to that γ, which serves as the key_instance of all arguments $\mathsf{PZK}_{\mathsf{KInstD}}$ in stage Iteration, is NO instance; 2) Different classes are (almost) independent due to the fact that the prover refreshes its randomness after the verifier's first message.

These observations indicate that V^* acts as a malicious *concurrent* verifier in some sense. This enable us to adopt the recursive rewinding strategy as a high level framework for our simulator, though it is not enough.

With the above observations, We will abuse the term *class* a little bit for the sake of simplifying presentation. For example, when we say a class reaches the i^{th} iteration (i.e., the i^{th} phase of stage Iteration), we mean the challenge message of the i^{th} iteration is *first* reached by a session in this class; If a session belonging to a specific class reaches the end of its stage Iteration, we say that this class completes its stage Iteration.

$\cdot\cdot$	$tran_{\mathsf{S},1}$	$(\cdot\cdot,\cdot\cdot,\cdot\cdot)$	$\dots$	$(\cdot\cdot,\cdot\cdot,\cdot\cdot)$	$\dots$	$(\cdot\cdot,\cdot\cdot,\cdot\cdot)$	$tran_{\mathsf{M},1}$	session 1
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$
$(\gamma, \mathsf{f\text{-}msg}, \cdot, \cdot)$	$tran_{\mathsf{S},s}$	$(\mathbf{e}_1, \mathbf{tr}_{1,s}, \mathbf{z}_1)$	$\dots$	$(\mathbf{e}_i, \mathbf{tr}_{i,s}, \mathbf{z}_i)$	$\dots$	$(\mathbf{e}_k, \mathbf{tr}_{k,s}, \mathbf{z}_k)$	$tran_{\mathsf{M},s}$	session s
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$
$\underbrace{\cdot\cdot}_{stage\ \text{Initialization}}$	$\underbrace{tran_{\mathsf{S},p}}_{stage\ \text{Setup}}$	$(\cdot\cdot,\cdot\cdot,\cdot\cdot)$	$\dots$	$(\cdot\cdot,\cdot\cdot,\cdot\cdot)$	$\dots$	$\underbrace{(\cdot\cdot,\cdot\cdot,\cdot\cdot)}_{stage\ \text{Iteration}}$	$\underbrace{tran_{\mathsf{M},p}}_{stage\ \text{Mainproof}}$	session p

Fig. 4. The class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$ (all ZAP proofs for validating the corresponding verifier's messages are ignored).

Here $tran_{\mathsf{M},j}$ and $tran_{\mathsf{S},j}$ represent the transcripts of $\mathsf{IDWIAOK}^{\mathsf{M}}_p$ and $\mathsf{IDWIAOK}^{\mathsf{S}}_p$ in session j respectively; $\mathbf{tr}_{i,j}$ is the transcript of the subprotocol $\mathsf{PZK}_{\mathsf{KInstD}}$ in the $\mathsf{IDWIAOK}^i_v$ in session j; By "$\cdot\cdot$" we mean this message is exactly the same one in the same position in session s. From above figure, we see that all sessions in a single class share the *same* messages $\mathbf{e}_1, \mathbf{tr}_{1,s}, \mathbf{z}_1, \cdots, \mathbf{e}_k, \mathbf{tr}_{k,s}, \mathbf{z}_k$ in stage Iteration.

[5] Note that, even in a single class of sessions, these ZAP proofs may differ from one session to another (V^* might use different randomness to generate these proofs). However, the honest prover (in the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument) just check these proofs to decide whether or not to continue, but does not base its next message on these proofs (see the description of $\mathsf{PZK}_{\mathsf{KInstD}}$ argument.). In our analysis, we will (w.l.o.g) assume V^* always sends valid messages, and therefore we can assume they do not exist at all.

5 A high level description of the simulator

In this section, we give a high level description of our simulator. Due to space limitation, we postpone the detailed description and its analysis to appendix C.

As mentioned in introduction, we employ a *mix* of recursive rewinding simulation strategy (as used for black-box concurrent zero knowledge proofs in [22,6]) and straight-line simulation strategy (as used for non-black-box zero knowledge argument in [1]). We briefly describe the two strategies below and show how our non-black-box simulator works in this complex setting.

A high level description of the recursive rewinding strategy. We adopt the original RK strategy of [22] (a detailed description of this strategy appeared in [6]). The main reason for not using the more advanced PRS strategy of [21] is that we don't know how to deal with the *rewind interval* in the analysis of the simulator. For the RK strategy, we just need to deal with *recursive depth*, which is upperbounded by a superconstant.

Assume that a malicious resetting verifier V^* initiates at most K class of sessions, where K is some polynomial. We set the recursive depth to be $T = 2\log^K_{k/128}$. Our simulator assumes that K is fixed in advance. However, it can be extended to deal with the situation where K is not known in advance in standard way[6].

We say a session *solved* if for some i, a transcript (a_i, e'_i, z'_i) of the underlying 3-round WI argument in $\mathsf{IDWIAOK}^i_v$ was obtained, where e'_i is a challenge different from the honest one e_i, and say a class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$ *solved* if some session in this class is *solved*. Note that, for a solved class, the simulator can carry out any session belonging to this class: It executes the honest prover strategy to obtain the transcript (a_i, e_i, z_i) of the underlying 3-round WI argument in $\mathsf{IDWIAOK}^i_v$ (where the challenge message e_i is an honest one); When a session in this class reaches stage Mainproof, it can extract an valid witness (that should be the preimage to β with overwhelming probability[7]) from these two transcripts and then uses it to complete the stage Mainproof of this session.

Given a malicious verifier V^* as input, our simulator $\mathbf{S}$ makes recursive calls to a procedure Simulate to simulate the view of V^*.

The procedure Simulate maintains a table $\mathcal{Q}$, a waiting (to be solved) list consisting of tuples of the form $(C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, (a_i, e'_i), t)$ which indicates the executions of i^{th} iteration of class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$ (in which the simulator proves its knowledge of s'_i such that $c^e_i = Com_v(e'_i, s'_i)$ or $x \in L$ via the argument $\mathsf{PZK}_{\mathsf{KInstD}}$) is currently being simulated, where t is the recursion level where this tuple is created. Initially, $\mathcal{Q} = \emptyset$.

At the level t ($1 \le t \le T$) of the recursion, the procedure Simulate proceeds as follows. For any *solved* session, Simulate acts as honest prover except that it generates a YES instance γ, and uses the corresponding witnesses δ (such that $\gamma = G(\delta)$) and α (such that $\beta = f(\alpha)$) to execute the $\mathsf{IDWIAOK}^{\mathsf{S}}_p$ and $\mathsf{IDWIAOK}^{\mathsf{M}}_p$ respectively; For every session belonging to a class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$ listed in the table $\mathcal{Q}$ (let the corresponding entry be $(C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, (a_i, e'_i), t')$, $t' \ge t$), Simulate forwards the V^*'s message in the i^{th} iteration of this session to a non-black-box simulator $\mathsf{Sim}^t_{\mathsf{KID}}$, and replies with what $\mathsf{Sim}^t_{\mathsf{KID}}$ replies; When a challenge message in i^{th}

[6] As pointed out in [6], we can keep running the simulator with exponentially increasing values of K until a successful simulation is generated. This applies to our non-black-box simulator, and the key point is that for any K, the recursive depth to be $T = 2\log^K_{k/128} < \log n$. (Recall that the the subprotocol $\mathsf{PZK}_{\mathsf{KInstD}}$ in the special-purpose $\mathsf{IDWIAOK}^i_v$ is $\log n$-class-bounded resettable ZK on NO (key) instance γ).

[7] In case the witness extracted from these two transcripts is the seed (δ) of γ, we still cannot complete Mainproof. Fortunately, the Lemma 3 (presented in appendix C) says this occurs only with negligible probability.

iteration of a *unsolved* class (say $C^{(l,m)}_{\mathsf{f\text{-}msg}}$) is reached for the first time, Simulate invokes a procedure called Solve at level $t-1$ to look ahead, aiming at solving this class or one of the classes on the current waiting list $\mathcal{Q}$; Once the procedure Solve returns, Simulate acts as honest prover in the i^{th} iteration of the current session, regardless of whether this session was solved or not; When an unsolved session reaches its stage Mainproof, or there are too many (set to be $(k/128)^t/16+1^{st}$) new class of sessions appearing within this invocation of Simulate, Simulate aborts and returns.

At the bottom level $t=0$, Simulate acts as the one at higher level except that it does not invoke Solve any more.

The procedure Solve at level $t-1$, invoked at the beginning of the i^{th} iteration of class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$, makes many independent attempts to solve one of classes in the current table $\mathcal{Q}$ or the class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$, where we refer to a single execution of step 1 within a call of Solve (see 10) as an *attempt*. At the beginning of an attempt, Solve chooses a random challenge e'_i, extends $\mathcal{Q}$ to include the entry $(C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, (a_i, e'_i), t-1)$, and then run the procedure Simulate at level $t-1$ on input of the new table $\mathcal{Q}$.

THE OUTPUT OF **S**. We call a simulation path in which the states of V^* evolve consecutively a *thread*, and call the top level simulation path *main thread*. Note that in the main thread, **S** executes the stage Iteration of any session *honestly*. At the end of the simulation, **S** outputs the *main thread*.

KEY OBSERVATION. We observe that, in a single thread, as the recursion goes down by one level (say from level t to level $t-1$), only one new entry of the form $(C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, (a_i, e'_i), t-1)$ was appended to the table $\mathcal{Q}$, and a new class of sessions *with respect to the argument* $\mathsf{PZK}_{\mathsf{KInstD}}$ [8](i.e., the executions of the $\mathsf{PZK}_{\mathsf{KInstD}}$ in the i^{th} iteration in class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$) needs to be simulated. Note also that for each level $0 \le t < T-1$, the waiting list $\mathcal{Q}$ contains only one entry of the form $(\cdot,\cdot,\cdot,t)$. This means in a single thread there are at most $T < \log(n)$ classes of subsessions *with respect to the argument* $\mathsf{PZK}_{\mathsf{KInstD}}$ being simulated.

A high level description of our non-black-box simulation strategy.

A SLIGHTLY SIMPLIFIED TREATMENT. With the above observation and the observation on the structure of a single class of sessions[9], we simply think of a class of subsessions *with respect to the argument* $\mathsf{PZK}_{\mathsf{KInstD}}$ mentioned above a single session, and show how to simulate these T subsessions (with respect to the relevant subprotocol $\mathsf{PZK}_{\mathsf{KInstD}}$.) in concurrent model in a thread here. (The actual non-black-box simulator is a little more involved.)

Since $T < \log n$, one may think of using Barak's simulator for $\log n$-bounded concurrent ZK protocol: This non-black-box simulator stands at level T, handles all messages belonging to these T subsessions in a thread.

However, this strategy does not work. Note that once Barak's simulator is invoked at the beginning of the i^{th} iteration of class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$, it is active as long as this the i^{th} iteration of class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$ is still being simulated, and note also that this iteration may go through many threads. Thus, Barak's simulator needs to handle all subsessions (we cannot bounded the number of these subsessions by any *polynomial*) that needs to be simulated in *all threads*

[8] Observe that all subexecutions of this $\mathsf{PZK}_{\mathsf{KInstD}}$ in a specific iteration of a single class (with respect to the global protocol) form one class of sessions with respect to the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument, i.e., the class specified by the same incarnation of prover in the global protocol and the verifier's first message with respect to this $\mathsf{PZK}_{\mathsf{KInstD}}$ that appeared in the verifier's first message with respect to the global protocol.

[9] Though this observation holds on NO instance γ, as we will see in appendix C, it holds in the simulation where γ is a YES instance.

that appear during the simulation of the i^{th} iteration of class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$. This is impossible because Barak's simulator works only in bounded concurrent model.

We use a novel technique to handle this situation. Instead of having a single Barak's simulator handle all these T subsessions in a thread, We have the $\mathsf{Simulate}$ at each level t in a thread use a Barak's simulator (for $\log n$-bounded concurrent ZK protocol) $\mathsf{Sim}^t_{\mathsf{KID}}$ to handle the current subsession for which this invocation of $\mathsf{Simulate}$ was initiated. In particular, when $\mathsf{Simulate}$ at each level t is invoked at the beginning of the i^{th} iteration of class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$ (letting $(C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, (a_i, e'_i), t)$ be the corresponding entry in $\mathcal{Q}$), it runs a Barak's simulator (for $\log n$-bounded concurrent ZK protocol) $\mathsf{Sim}^t_{\mathsf{KID}}$, which *mainly* handle the current subsession. This simulator $\mathsf{Sim}^t_{\mathsf{KID}}$ acts as follows.

- (Main task) For every query from V^* regarding the subsession with respect to the i^{th} iteration of the current class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$, $\mathsf{Sim}^t_{\mathsf{KID}}$ performs in the following way:
  - At the first prover step of this session (where the Barak's simulator needs to commit a code), $\mathsf{Sim}^t_{\mathsf{KID}}$ commits to the hash value of the *joint* code of $\mathsf{Simulate}$ at level t (except the current subroutine $\mathsf{Sim}^t_{\mathsf{KID}}$, of course) and V^*.
    We stress that the code this $\mathsf{Simulate}$ at level t includes all codes of the subprocedures Solve at level $t-1$ invoked by it, which in turn includes all codes of $\mathsf{Simulate}$ at level t' and $\mathsf{Sim}^{t'}_{\mathsf{KID}}$[10] for $t' < t$ that are invoked by the above Solve at level $t-1$.
  - After the first prover step, $\mathsf{Sim}^t_{\mathsf{KID}}$ acts as the same as Barak's simulator by treating the messages output by $\mathsf{Sim}^{t'}_{\mathsf{KID}}$ at higher level $t' > t$ as *external* message (which will be used as witness to carry out the current session).
- (Minor task) For any query from V^* regarding a subsession belonging to class $C^{(l',m')}_{\mathsf{f\text{-}msg'}}$ with an entry of the form $(C^{(l',m')}_{\mathsf{f\text{-}msg'}}, \cdot, \cdot, t')$ in $\mathcal{Q}$, $t' > t$, $\mathsf{Sim}^t_{\mathsf{KID}}$ forwards this query to $\mathsf{Sim}^{t'}_{\mathsf{KID}}$, stores its response and forwards it to V^*.

Note that, in a single thread, for each $0 \le t \le T$, there is only one invocation of $\mathsf{Simulate}$ at level t, in which $\mathsf{Sim}^t_{\mathsf{KID}}$ is invoked only once. Also, observe that, for each $t \le t_i \le T-1$, there is only one subsession belonging to the class with an entry of the form $(\cdot, \cdot, \cdot, t_i)$ in $\mathcal{Q}$ (handled by $\mathsf{Sim}^{t_i}_{\mathsf{KID}}$) being simulated during a specific execution of $\mathsf{Simulate}$ at level t, but for $t_j < t$, this $\mathsf{Simulate}$ at level t may see *many* simulated subsessions (each of them requires an independent execution of $\mathsf{Sim}^{t_j}_{\mathsf{KID}}$) that appear in many executions of $\mathsf{Simulate}$ at level t_j, due to that it may go through many threads.

The crux of this non-black-box simulation strategy is that, during a specific execution of $\mathsf{Sim}^t_{\mathsf{KID}}$, $\mathsf{Sim}^t_{\mathsf{KID}}$ does not handle these simulated subsessions that appear in execution of $\mathsf{Simulate}$ at lower level $t_j < t$, which are handled by some $\mathsf{Sim}^{t_j}_{\mathsf{KID}}$ *internally*. Note that the code that $\mathsf{Sim}^t_{\mathsf{KID}}$ commits to in its first step already includes these subroutines $\mathsf{Sim}^{t_j}_{\mathsf{KID}}$. In the execution of $\mathsf{Sim}^t_{\mathsf{KID}}$, we view the joint code $\mathsf{Simulate}$ at level t (includes all subroutines at lower level therein but except the current $\mathsf{Sim}^t_{\mathsf{KID}}$) and V^* as a malicious verifier. It is easy to verify that $\mathsf{Sim}^t_{\mathsf{KID}}$ succeeds as long as the total length of the *external* messages is "short". This is the case because there are only $T - t < \log n$ subsessions (that simulated by some $\mathsf{Sim}^{t'}_{\mathsf{KID}}$ at higher level $t' > t$.) for which $\mathsf{Sim}^t_{\mathsf{KID}}$ treats the prover's messages therein as *external* messages.

[10] It should be noted that when the simulator commits to a code, this code needs to be well-defined, thus the simulator $\mathsf{Sim}^{t''}_{\mathsf{KID}}$ at lower level is not able to commit to the simulator $\mathsf{Sim}^t_{\mathsf{KID}}$ at higher level; In our setting, at the bottom level, all $\mathsf{Sim}^0_{\mathsf{KID}}$ is well defined; Note that $\mathsf{Sim}^1_{\mathsf{KID}}$ is well defined as long as all $\mathsf{Sim}^0_{\mathsf{KID}}$ is well defined; Repeating this, it is easy to see that when $\mathsf{Sim}^t_{\mathsf{KID}}$ commits to the code of $\mathsf{Simulate}$ at level t (except the $\mathsf{Sim}^t_{\mathsf{KID}}$), all codes of $\mathsf{Sim}^{t'}_{\mathsf{KID}}$ at lower level $t' < t$ therein are well defined.

Extension to the resettable model. As observed in last section, in the resettable model, all these T class of subsessions mentioned above are actually T subsessions: A class of subsessions is just a class of copies of the same session. Thus, the above non-black-box simulator $\mathsf{Sim}^t_{\mathsf{KID}}$ can be easily extended to the resettable model by simply replying with the same answer to the same query. In the actual procedure $\mathsf{Sim}^t_{\mathsf{KID}}$, this is done by an intermediator $\mathsf{Intermed}^t$, which is a part of $\mathsf{Sim}^t_{\mathsf{KID}}$ (See figure 11 in appendix C for its actual description).

Some subtle problems. We point out some subtle technical problem in the actual analysis of our simulator.

1. In our simulation, the first prover message γ is a YES instance. Thus the first thing we needs to do is to prove that the structure of a class of sessions in the simulation remains the same as in the real interaction (depicted in last section), where γ is a NO instance.
2. The malleability issue. The malicious resetting verifier may learn how to simulate a proof for the correctness of its challenge.
3. Handling external messages in analysis of security. Typically, we prove the output of a simulator is indistinguishable from the real interaction by setup a series of hybrid simulators, each of which is slightly different from its neighbors. However, we must be careful with even simple claims in the non-black-box simulation. For example, when we claim S_1 and S_2 are indistinguishable unless we can break the pseudorandomness of a pseudorandom function, we need to have S_1 (or S_2) take this function as an oracle (thus its code is unavailable to S_1 and S_2), in this case, S_1 needs to handle the answers from this oracle as *external* messages. The problem is that, in some cases, the non-black-box simulator S_1 (S_2) is not able to handle many external messages.

As we will see in appendix C (Lemma 2 and lemma 3), our solution to these problems relies on two facts: All $\mathsf{IDWIAOK}^i_v$ are special-purpose IDWIAOK and the $\mathsf{PZK}_{\mathsf{KInstD}}$ arguments in $\mathsf{IDWIAOK}^i_v$ are perfect zero knowledge under some conditions, as stated in section 3.2. These two facts enable us to construct a non-uniform simulator that performs as the same as our non-black-box simulator *without doing any non-black-box simulation*!

Acknowledge. We thank Vipul Goyal[11], Pino Persiano, Amit Sahai, Ivan Visconti and Moti Yung for helpful discussions on an earlier version of [10].

References

[1] B. Barak. How to go beyond the black-box simulation barrier. In Proc. of IEEE FOCS 2001, pp.106-115.
[3] B. Barak, O. Goldreich, S. Goldwasser, Y. Lindell. Resettably sound Zero Knowledge and its Applications. In Proc. of IEEE FOCS 2001, pp. 116-125.
[4] B. Barak, O. Goldreich. Universal Arguments and Their Applications. In Proc. of IEEE CCC 2002, pp. 194-203.
[5] M. Blum. How to Prove a Theorem so No One Else can Claim It. In Proc. of ICM'86, pp. 1444-1451, 1986.
[6] R. Canetti, O. Goldreich, S. Goldwasser, S. Micali. Resettable Zero Knowledge. In Proc. of ACM STOC 2000, pp.235-244
[7] R. Canetti, J. Kilian, E. Petrank and A. Rosen. Concurrent Zero-Knowledge requires $\Omega($ $logn)$ rounds. In Proc. of ACM STOC 2001, pp.570-579.
[8] D. Dolev, C. Dwork and M. Naor. Non-malleable Cryptography. SIAM J. on Computing 30(2):391-437, 2000.
[9] Yi Deng, Dongdai Lin. Instance-Dependent Verifiable Random Functions and Their Application to Simultaneous Resettability. In Advances in Cryptology-Eurocrypt'07, LNCS4515, pp.148-168, 2007.

[11] When we were working on this problem , we were told that Vipul Goyal and Amit Sahai also had a similar result. Their independent work is available at http://eprint.iacr.org/2008/545.

[10] Yi Deng, Dongdai Lin. Simultaneously Resettable Argument with Public Keys. Manuscript.
[11] C. Dwork, M. Naor. Zaps and Their Applications. In Proc. of IEEE FOCS 2000, pp.283-293
[12] C. Dwork, M. Naor and A. Sahai. Concurrent Zero-Knowledge. In Proc. of ACM STOC 1998, pp.409-418.
[13] U.Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In Proc. of ACM STOC 1990, pp.416-426.
[14] O. Goldreich. Foundation of Cryptography-Basic Tools. Cambridge University Press, 2001.
[15] O. Goldreich, A. Kahan: How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. J. Cryptology 9(3): 167-190,,1996.
[16] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. SIAM. J. Computing, 18(1):186-208, 1989.
[17] O. Goldreich, Y. Oren: Definitions and Properties of Zero-Knowledge Proof Systems. J. Cryptology 7(1): 1-32, 1994.
[18] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In Proc. of IEEE FOCS'99, pp. 120-130, 1999.
[19] R. Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In Proc. of ACM STOC 2004: 232-241.
[20] R. Pass, A. Rosen. Concurrent Non-Malleable Commitments. In Proc. of IEEE FOCS 2005, pp. 563-572, 2005.
[21] M. Prabhakaran, A. Rosen and A. Sahai. Concurrent Zero-Knowledge with Logarithmic Round Complexity. Manoj Prabhakaran, Alon Rosen and Amit Sahai. Concurrent Zero-Knowledge with Logarithmic Round Complexity. In Proc. of IEEE FOCS'02, pp.366-375, 2002.
[22] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In Advances in Cryptology-Eurocrypt'99, LNCS1592, pp.4158-431, 1999.

A Definitions

In this section we give formal definitions of zero knowledge and soundness in the resettable setting.

Resetting and class-bounded resetting attacks. We first recall resetting attack introduced in [6].

Resetting attack [6]. Let *poly* be an arbitrary polynomial, and Let $\overline{x} = x_1, \cdots, x_{poly} \in L \cap \{0,1\}^n$ be a sequence of distinct common inputs and $\overline{w} = w_1, \cdots, w_{poly}$ be a corresponding witness sequence for $\overline{x}$. The verifier V^*'s resetting attack is defined by the following random process depending on P and V. 1) Randomly pick and fix *poly* random tapes, $r_1, \cdots, r_{poly}$, resulting in $poly^2$ deterministic incarnations $P^{(i,j)} = P_{x_i,w_i,r_j}$ defined by $P_{x_i,w_i,r_j}(\alpha) = P(x_i, w_i, r_j, \alpha)$, for $(i,j) \in \{1, \cdots, p(n)\} \times \{1, \cdots, poly\}$; 2) V^* is allowed to run polynomial many sessions with the $P^{(i,j)}$'s. Throughout those sessions, V^* is allowed to schedule all sessions in interleaving way [12]: V^* can send arbitrary messages to each of the $P^{i,j}$, and obtain the responses of $P^{(i,j)}$ to such messages immediately; 3) Once V^* decides it is done interacting with the $P^{(i,j)}$'s, it produces an output based on its view of the whole interaction. We denote this output by $(P(\overline{w}), V^*)(\overline{x})$.

A restricted version of the resetting attack—the so-called *class-bounded* resetting attack—was introduced in [9], where a class (of sessions) contains all sessions between verifier and a fixed incarnation of prover into a *class* that share the *same* verifier's first message msg. For instance, all sessions associated with the same incarnation $P^{(i,j)}$ and sharing the same verifier's first message $\mathsf{f\text{-}msg}$ belong to the same class (denoted by $C^{(l,m)}_{\mathsf{f\text{-}msg}}$).

Class-bounded resetting attack.[9] This is a resetting attack under the following restrictions: 1) the malicious verifier V^* is only allowed to interact with an *a-priori bounded*

[12] Actually, in the resettable setting, V^* cannot gain more power from concurrent scheduling than from sequential scheduling (cf. [6]).

number of incarnations; 2) the number of different V^*'s first messages to each incarnation is also a-priori bounded. Formally, a t^3-bounded-class (t be an a-priori fixed polynomial) resetting attack is executed by V^* in the way as defined above, but in which V^* is only allowed to interact with t^2 incarnations of P_{x_i,w_i,r_j}'s, $(i,j) \in \{1,\cdots,t\} \times \{1,\cdots,t\}$, and the number of different V^*'s first messages to each incarnation P_{x_i,w_i,r_j} is a priori bounded by t, where $\overline{x} = x_1,\cdots,x_t \in L \cap \{0,1\}^n$ is a sequence of distinct common inputs and $\overline{w} = w_1,\cdots,w_t$ is a correspondingly witness sequence for $\overline{x}$ as defined in resetting attack, $r_1,\cdots,r_t$ are those provers' random tapes. Note that this results in at most t^3 classes of sessions during the whole execution of this attack.

Definition 1. *[resettable ZK argument] Let poly be a polynomial, $\overline{x} = x_1,\cdots,x_{poly} \in L \cap \{0,1\}^n$ is a sequence of distinct common inputs and $\overline{w} = w_1,\cdots,w_{poly}$ is a correspondingly witness sequence for $\overline{x}$. An interactive argument (P,V) for a language L is said to be resettable ZK if for every PPT adversary V^* mounting resetting attack, there exists a PPT M so that $(P(\overline{w}),V^*)(\overline{x})$ and $M(\overline{x})$ are computational indistinguishable.*

Definition 2. *[class-bounded resettable ZK argument] Let t be a polynomial, $\overline{x} = x_1,\cdots,x_t \in L \cap \{0,1\}^n$ is a sequence of distinct common inputs and $\overline{w} = w_1,\cdots,w_t$ is a correspondingly witness sequence for $\overline{x}$. An interactive argument (P,V) for a language L is said to be t^3-class-bounded resettable ZK if for every every PPT adversary V^* mounting t^3-class-bounded resetting attack, there exists a PPT M so that $(P(\overline{w}),V^*)(\overline{x})$ and $M(\overline{x})$ are computational indistinguishable.*

Definition 3. *(Resettable WI) Let p be an arbitrary polynomial, L_0 and L_1 be two (possibly the same) NP languages. Let $L = L_0 \vee L_1 = \{(x_0,x_1) : x_0 \in L_0 \text{ or } x_1 \in L_1\}$[13]. An interactive argument (P,V) for language L is said to be resettably witness indistinguishable if for any PPT V^* mounting (*unbounded*) resetting attack, the distribution $(P(\overline{w_0}),V^*)(\overline{x})$ is computationally indistinguishable from $(P(\overline{w_1}),V^*)(\overline{x})$, where $\overline{x} = x^1,\cdots,x^{poly}$, $x^i = (x_0^i,x_1^i) \in L$, $\overline{w_b} = w_b^1,\cdots,w_b^{poly}$ such that $(x_b^i,w_b^i) \in R_{L_b}$ for $1 \le i \le poly$, $b \in \{0,1\}$.*

Resettably-sound argument of knowledge. Resetting attack also gives rise to the notion of resettably-sound argument of knowledge introduced in [3].

Definition 4. *[Resettably-sound argument of knowledge.] A resetting attack of a malicious prover P^* on a resettable verifier V is defined by the following random process, indexed by a security parameter n: 1) Uniformly pick and fix poly random-tapes, denoted $r_1,\cdots,r_{poly}$, for V, resulting in deterministic incarnations $V^{(j)}(x) = V_{x,r_j}$, $x \in \{0,1\}^n$ and $j \in \{1,\cdots,poly\}$, defined by $V_{x,r_j}(\alpha) = V(x,r_j,\alpha)$; 2) Taking as input 1^n, P^* is allowed to initiate poly number sessions with the $V^{(j)}(x)$'s, and is allowed to schedule all sessions in interleaving way as usual: P^* can send arbitrary messages to each of the $V^{(j)}(x)$, and obtain the responses of $V^{(j)}(x)$ to such messages immediately.*

We say an argument system (P,V) is a resettably-sound argument of knowledge system if it satisfies:

1. *Resttable-completeness: Considering an arbitrary resetting attack of a PPT P^*. If P^* follows the strategy of P in some sessions after selecting an incarnation $V^{(j)}(x)$ and $x \in L$, then $V^{(j)}(x)$ rejects with negligible probability.*
2. *Resettably-soundness: For every weak resetting attack of a PPT P^*, the probability that in some sessions the corresponding $V^{(j)}(x)$ has accepted a false statement ($x \notin L$) is negligible.*

[13] In our applications, it is sufficient to consider only the "OR" statements.

3. Argument of knowledge: For every PPT P^*, there exists a PPT machine E such that for every resetting attack of P^*, the probability that E, upon input the description of P^*, outputs a witness for the statement in a session is negligibly close to the probability that P^* convinces V in a session.

Definition 5. *[Instance-dependent resettably-sound resettable WI (IDWIAOK)] Let L_0 and L_1 be two (possibly the same) NP languages. Let $L = L_0 \vee L_1 = \{(x_0, x_1) : x_0 \in L_0 \text{ or } x_1 \in L_1\}$. An interactive argument (P, V) for language L is said to be instance-dependent resettably-sound resettable WI if it satisfies:*

1. *Resettable WI, as in definition 3.*
2. *For any statement $(x_0, x_1) \in L$, the resettably-sound argument of knowledge property holds when $x_0 \notin L_0$.*

B The simplified Pass-Rosen protocol and the security proofs of $\mathsf{PZK}_{\mathsf{KInstD}}$ and IDWIAOK

B.1 The simplified Pass-Rosen protocol

The Pass-Rosen variant of Barak's protocol [20] (the protocol ZK_{tag} in[20], an earlier version of this protocol appeared in [19]) was originally tailored to their specific application in non-malleable commitment. There are two notable features of this variant: it is (bounded concurrent) *perfect* ZK and satisfies the argument of knowledge property (rather than the weak argument of knowledge property of Barak's protocol).

Before giving a formal presentation of the simplified Pass-Rosen protocol, we recall the Barak's constant round public-coin t-bounded concurrent zero knowledge argument [1]. Informally, Barak's protocol for a $\mathcal{NP}$ language L consists of two subprotocol: a generation protocol and a WI universal argument. An execution of the generation protocol will generate an instance with respect to the following language Λ. Let n be security parameter and $\{\mathcal{H}_n\}_{n\in\mathbb{N}}$ be a collection of hash functions where a hash function $h \in \mathcal{H}_n$ maps $\{0,1\}^*$ to $\{0,1\}^n$, and let Com be a statistically binding commitment scheme. We say a triplet $(h, c, r) \in \mathcal{H}_n \times \{0,1\}^n \times \{0,1\}^{tn^3}$ is in Λ, if there exist a program Π, string $s \in \{0,1\}^{poly(n)}$ and a string ω such that $\omega \leq |r|/2 = tn^3/2$, such that $z = Com(h(\Pi), s)$ and $\Pi(z, \omega) = r$ within superpolynomial time (i.e., $n^{\omega(1)}$).

The simplified Pass-Rosen protocol used in this paper can be viewed as a hybrid of Pass-Rosen protocol and Barak's protocol: it uses *perfect hiding* commitment scheme Com and the special-purpose universal argument like Pass-Rosen protocol, but has only one slot like Barak's protocol. See figure 5 for a detailed description.

As the Pass-Rosen protocol, the above simplified version is also bounded concurrent *perfect* zero knowledge and satisfies the argument of knowledge property.

B.2 Security proof of $\mathsf{PZK}_{\mathsf{KInstD}}$

When applying the same transformation of [3](i.e., having the verifier generate its coins by applying a pseudorandom function to the history so far) to the above simplified Pass-Rosen (t-bounded concurrent ZK) protocol, we obtain an argument $(P_{\mathrm{R}}, V_{\mathrm{R}})$ that satisfies:

1. t-bounded concurrent *perfect* ZK property. This is due to that the transformation preserves perfect zero knowledge.
2. Resettable-sound *argument of knowledge*. This is guaranteed by proposition 2.5 of [3].

The simplified Pass-Rosen protocol (P, V)

Common input: $x \in L$ ($|x| = n$).

Stage 1 (Generation)

$V \longrightarrow P$**:** send a random hash function h.
$P \longrightarrow V$**:** send $c = Com(0^n)$. (all commitment schemesCom are *perfect hiding*.)
$V \longrightarrow P$**:** send $r \leftarrow_R \{0,1\}^{tn^3}$.

Stage 2 (Encrypted UARG)

$V \longrightarrow P$**:** send $\alpha \leftarrow_R \{0,1\}^n$.
$P \longrightarrow V$**:** send $\beta' = Com(0^n)$.
$V \longrightarrow P$**:** send $\gamma \leftarrow_R \{0,1\}^n$
$P \longrightarrow V$**:** send $\delta' = Com(0^n)$.

Stage 3 (Main body of the proof)

$P \Longrightarrow V$**:** P and V execute a witness independent argument of knowledge in which P proves the OR of the following statements:

1. $\exists w \in \{0,1\}^{poly(n)}$ so that $(x, w) \in R_L$.
2. $\exists(\beta, \delta, s_1, s_2)$ 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 UARG proving the statement $(h, c, r) \in \Lambda$.

Fig. 5. The simplified Pass-Rosen (t-bounded concurrent ZK) protocol for language L.

Given a resettably-sound t-bounded concurrent *perfect* ZK argument $(P_{\mathrm{R}}, V_{\mathrm{R}})$, we prove that the transformation presented in section 3.2 leads to an argument $\mathsf{PZK}_{\mathsf{KInstD}}$ that enjoys the following properties:

1. It is t-class-bounded resettable ZK if all y are NO (key) instances.
2. It is t-bounded-class resettable *perfect* ZK if the following conditions (**Conditions for PZK**) hold:
   - The honest incarnation of the prover feeds P_{R} with independent and truly random tape in stage 2 under the consistence condition that the random tape for P_{R} is the same in every session having the same verifier first message (y, c_0);
   - The verifier never sends two "correct" messages (r, π) and (r', π') with $r \neq r'$ for the same history *hist* in stage 2 of $\mathsf{PZK}_{\mathsf{KInstD}}$.
3. It is resettable-sound *argument of knowledge* if all y are YES (key) instances.

proof of item 1 and item 2. We first note that the simulators for both the simplified Pass-Rosen protocol and the argument $(P_{\mathrm{R}}, V_{\mathrm{R}})$ proceed exactly the same as Barak's simulator.

We construct a simulator $\mathsf{Sim}_{\mathsf{KID}}$ for the t-class-bounded resettable ZK argument $\mathsf{PZK}_{\mathsf{KInstD}}$. The main part of $\mathsf{Sim}_{\mathsf{KID}}$ is the (slightly modified) Barak's simulator $\mathsf{Sim}_{\mathsf{B}}$ for t-session-bounded concurrent ZK argument. For our simulator to work in resettable model, we have $\mathsf{Sim}_{\mathsf{KID}}$ use an intermediator Intermed to store the output of $\mathsf{Sim}_{\mathsf{B}}$ and reply with the same answer for the same query from V^*. The simulator $\mathsf{Sim}_{\mathsf{KID}}$ is depicted in figure 6.

It is easy to check that the underlying procedure $\mathsf{Sim}_{\mathsf{B}}$ performs well: It takes only t sessions (in concurrent model), and whenever a session reaches the stage 2 (Encrypted

$\mathsf{Sim}_{\mathsf{KID}}$:

Run the joint code of the intermediator Intermed and the (slightly modified) Barak's simulator $\mathsf{Sim}_{\mathsf{B}}$ on the common inputs and the code of V^*, output what Intermed outputs.

Intermed: Create an interface with V^* and an interface with $\mathsf{Sim}_{\mathsf{B}}$. Upon receiving a session history h from V^*, act as follows.

1. If the current verifier message (i.e., the last message in h) is not correct, abort this session; Otherwise, do the following.
2. If the next scheduled prover message is the first prover message of Stage 1 a and V^* does not query on h before, Intermed sends an random string ρ to V^* as the honest prover, and stores it on a table R; If V^* queried on h before, Intermed retrieves the same answer from R and sends it to V^*.
3. If the next scheduled prover message is a prover message of Stage 2, Intermed retrieves the subtranscript tr of the underlying argument $(P_{\mathrm{R}}, V_{\mathrm{R}})$. Let $tr = (tr'||r)$. (r is the current verifier message.)
   (a) If a session history (tr', r') with $r \neq r'$ is stored in R (This breaks the uniqueness of the InstD-VRF function $\mathsf{F}_{(PK,SK)}$,), output $\perp$;
   (b) If V^* does not query on h before, send it to $\mathsf{Sim}_{\mathsf{B}}$. Once the next message is obtained, store it in R and forward it to V^*; If V^* queried on h before, retrieve the same answer from R and send it to V^*.
   (c) If V^* terminates the interaction, output the whole V^*'s view.

$\mathsf{Sim}_{\mathsf{B}}$: Act as exactly the same as Barak's simulator for the (t-session-bounded concurrent ZK) simplified Pass-Rosen protocol except that, when the Barak's simulator is instructed to commit to hash value of the *joint* code of the residual procedures Intermed and V^*, i.e., $\mathsf{Sim}_{\mathsf{B}}$ treats the joint code of Intermed and V^* as the *concurrent* adversary. (Note that both Intermed and V^*, i.e., $\mathsf{Sim}_{\mathsf{B}}$ are well-defined.)

Fig. 6. The procedure $\mathsf{Sim}_{\mathsf{KID}}$.

UARG), it already knew the corresponding "valid" witness (i.e., the joint code of the residual procedures Intermed and V^*, and some relevant prover messages sent in other sessions) to carry out the stage 2 and 3 of this session.

We denote the output of $\mathsf{Sim}_{\mathsf{KID}}$ with $\mathrm{Out}_{\mathsf{Sim}_{\mathsf{KID}}}$, and denote the view of V^* in the real world with $\mathrm{View}^{real}_{V^*}$. The proof of item 1 and item 2 can be done by the following steps:

1. We first prove item 2. Let $\mathrm{View}^{ideal}_{V^*}$ denote the V^*'s view in an interaction with honest provers in the *ideal world* where the **Conditions for PZK** hold. We prove that $\mathrm{Out}_{\mathsf{Sim}_{\mathsf{KID}}}$ and $\mathrm{View}^{ideal}_{V^*}$ are *identical*.
2. We then consider a *semi-ideal world* where *only the first* **Condition for PZK** is assumed to hold. Let $\mathrm{View}^{semi}_{V^*}$ denote the view of V^* in an interaction with honest prover in this semi-ideal world. It is easy to show that $\mathrm{View}^{semi}_{V^*}$ is computational indistinguishable from $\mathrm{View}^{ideal}_{V^*}$ due to the fact that the *second* **Condition for PZK** holds in any world except with negligible probability (because of the uniqueness of the InstD-VRF function $\mathsf{F}_{(PK,SK)}$ on NO (key) instance y).
3. Finally, observe that if a PPT algorithm can distinguish between $\mathrm{View}^{semi}_{V^*}$ and $\mathrm{View}^{real}_{V^*}$, then we can use this algorithm to distinguish output of a pseudorandom function (which the honest prover uses to generate the random tape for the underlying P_{R} in the real world) and the truly random string. Thus, we conclude that $\mathrm{View}^{semi}_{V^*}$ and $\mathrm{View}^{real}_{V^*}$ are computational indistinguishable.

   This implies that $\mathrm{Out}_{\mathsf{Sim}_{\mathsf{KID}}}$ is computational indistinguishable from $\mathrm{View}^{real}_{V^*}$, which establishes item 1.

Now we only need to establish step 1 to complete the proof of item 1 and item 2. Consider a *concurrent* adversary $V^{**} = (\mathsf{Intermed}, V^*)$ (i.e., V^{**} is the joint code of $\mathsf{Intermed}$ and V^*) that interacts with t incarnations of the prover P_{R} of the simplified Pass-Rosen protocol in the concurrent model and outputs what $\mathsf{Intermed}$ outputs. We use the following procedure $\mathsf{Sim'_B}$ to simulate the output[14] of V^{**}: $\mathsf{Sim'_B}$ is exactly the same as $\mathsf{Sim_B}$ except that it outputs what V^{**} outputs when V^{**} terminates the whole interaction.

Let $\mathrm{Out}_{V^{**}}^{real}$ denote the output of V^{**} at the end of the real interaction with t honest provers P_{R} of the simplified Pass-Rosen protocol, and let $\mathrm{Out}_{\mathsf{Sim'_B}}$ denote the output of $\mathsf{Sim'_B}$. We have:

- $\mathrm{Out}_{\mathsf{Sim_{KID}}}$ is identical to $\mathrm{Out}_{\mathsf{Sim'_B}}$. This is simply due to that $\mathsf{Sim'_B}$ outputs what $\mathsf{Sim_{KID}}$ outputs;
- $\mathrm{Out}_{\mathsf{Sim'_B}}$ is identical to $\mathrm{Out}_{V^{**}}^{real}$. This is simply due to the t-bounded concurrent *perfect zero knowledge* property of the simplified Pass-Rosen protocol.

The remaining task is to show that $\mathrm{Out}_{V^{**}}^{real}$ is identical to $\mathrm{View}_{V^*}^{ideal}$. Recall that V^* is a t-class-bounded resetting adversary. Hence the whole interaction between V^* and honest incarnations of prover in the ideal world consists of at most t class of sessions. Denote these classes with $C_{\mathsf{f\text{-}msg}_1}^{(l_1,m_1)}$,..., $C_{\mathsf{f\text{-}msg}_k}^{(l_i,m_j)}$,..., where the number of distinct tuple $(l_i, m_j, \mathsf{f\text{-}msg}_k)$ is at most t. (hence the number of honest incarnations $P^{(l_i,m_j)}$ of the prover of the argument $\mathsf{PZK_{KInstD}}$ is at most t.)

Thinking of the $\mathsf{Intermed}$ (incorporating with the honest provers P_{R} of the underlying argument $(P_{\mathrm{R}}, V_{\mathrm{R}})$) as those provers of the argument $\mathsf{PZK_{KInstD}}$ in the ideal model. Note that the interaction between V^* and the above provers in the ideal model is identical to $\mathrm{Out}_{V^{**}}^{real}$ (recall that $V^{**} = (\mathsf{Intermed}, V^*)$). We make the following observations.

- The first prover message ρ sent by $\mathsf{Intermed}$ (incorporating with the honest provers P_{R}) is identical to the one sent by the the honest prover of the argument $\mathsf{PZK_{KInstD}}$ in every session, and in both $\mathrm{Out}_{V^{**}}^{real}$ and $\mathrm{View}_{V^*}^{ideal}$, different sessions either share the same first prover message or have independent ones. *Furthermore, the first prover message is independent of all prover messages in stage 2 of $\mathsf{PZK_{KInstD}}$ in a session.*
- By the first **Condition for PZK**, in a single class of sessions $C_{\mathsf{f\text{-}msg}_k}^{(l_i,m_j)}$, the honest incarnation $P^{(l_i,m_j)}$ of the prover of the argument $\mathsf{PZK_{KInstD}}$ runs the underlying P_{R} *only once* (we ignore those repeated invocations of P_{R} on the same session history), and for different classes, the incarnations therein of the prover runs *independent copies* P_{R} (each with independent random tape). The same happens to $\mathsf{Intermed}$.
  This guarantees that the stage 2 prover messages of $\mathsf{PZK_{KInstD}}$ in each class of sessions in $\mathrm{Out}_{V^{**}}^{real}$[15] is identical to the one in $\mathrm{View}_{V^*}^{ideal}$, and that in both $\mathrm{Out}_{V^{**}}^{real}$ and $\mathrm{View}_{V^*}^{ideal}$, the stage 2 prover messages in one class of session is *independent* of the ones in another class of sessions.
- By the second **Condition for PZK**, $\mathsf{Intermed}$ never outputs $\perp$. Note that the honest prover of the argument $\mathsf{PZK_{KInstD}}$ never aborts or output $\perp$ even in the case that V^* sends two "correct" messages (r, π) and (r', π') with $r \neq r'$ for the same history *hist*

[14] Here we adopt the formulation of zero knowledge that requires the simulator to simulate the output of the (malicious) verifier just for simplifying the presentation. Note that this is equivalent to the original formulation of zero knowledge that requires the simulator to simulate the view of the (malicious) verifier (cf. [14]).

[15] Since $\mathrm{Out}_{V^{**}}^{real}$ is the output of $\mathsf{Intermed}$ that interacts with P_{R}'s, we can define class of session in the same way as in the resettable model.

in stage 2 of $\mathsf{PZK}_{\mathsf{KInstD}}$. (Recall that the honest prover checks only whether the current verifier's message is correct or not.)

It follows that $\mathrm{Out}_{V^{**}}^{real}$ and $\mathrm{View}_{V^*}^{ideal}$ are identical, and hence $\mathrm{Out}_{\mathsf{Sim}_{\mathsf{KID}}}$ and $\mathrm{View}_{V^*}^{ideal}$ are identical. This complete the proof of item 2. □

proof of item 3. The pseudorandomness of the InstD-VRF function $\mathsf{F}_{(PK,SK)}$ on YES (key) instance y enables us to construct a malicious resetting prover P_{R}^* of the underlying argument $(P_{\mathrm{R}}, V_{\mathrm{R}})$ from a malicious resetting prover P^* of the argument $\mathsf{PZK}_{\mathsf{KInstD}}$ with negligibly close cheating probability. This can be done by using the witness of y to generate a valid proof for each message from external honest verifier V_{R} as showed in [9]. Note that $(P_{\mathrm{R}}, V_{\mathrm{R}})$ satisfies resettable-sound *argument of knowledge*, so does the argument $\mathsf{PZK}_{\mathsf{KInsD}}$ (on YES (key) instance y). □

B.3 Security proof of IDWIAOK

For completeness, we present the security proof of IDWIAOK of [10] here.

Proof of resettably-sound argument of knowledge when $x_0 \notin L$. The basic extraction strategy is to obtain two different accepting transcripts of the underlying 3-round WI argument by sending different challenges (an honest one and a random one with simulated correctness proof), and apply the Goldreich and Kahan [15] technique to bound its running time. Assume that a resetting prover P^* can convince an incarnation $V^j(x)$ on statement $x = (x_0, x_1) \in L$ such that $x_0 \notin L_0$ with probability p in the last session[16], the extractor E can be constructed as follows.

Note that in the extraction process, the subroutine $\mathsf{Sim}_{\mathsf{KID}}^{\mathsf{E}}$ needs to take the code of P^* as input, hence E is an non-black-box extractor. It is easy to verify that the underlying Barak's simulator $\mathsf{Sim}_{\mathsf{B}}^{\mathsf{E}}$ handles only a single session, and the *joint* code of the residual procedures $\mathsf{Intermed}^{\mathsf{E}}$ and E(except the current subroutine $\mathsf{Sim}_{\mathsf{KID}}^{\mathsf{E}}$) and P^* is a proper (partial) witness for $\mathsf{Sim}_{\mathsf{B}}^{\mathsf{E}}$ to carry out this session.

We also observe that, for a session with prefix $((c_0^a, c_0), (c_0^e, \rho'), a)$ and a session with prefix $((c_1^a, c_0'), (c_1^e, \rho''), a')$, $((c_0^a, c_0), (c_0^e, \rho'), a) \neq ((c_1^a, c_0'), (c_1^e, \rho''), a')$ in the real interaction, the statements "the challenge is correct" in the above two sessions are (almost) uncorrelated: since $((c_0^a, c_0), (c_0^e, \rho'), a) \neq ((c_1^a, c_0'), (c_1^e, \rho''), a')$, it must be the case $(c_0^a, c_0) \neq (c_1^a, c_0')$[17], and this will cause the verifier to choose different and (almost) independent challenges (and randomness used in the commitment c^e) via a pseudorandom function.

This crucial observation enables us to employ argument $\mathsf{PZK}_{\mathsf{KInstD}}$ satisfying only 1-class-bounded resettable zero knowledge property (for verifier to prove "the challenge is correct") is that all executions of argument $\mathsf{PZK}_{\mathsf{KInstD}}$ in those sessions with the same prefix $((c^a, c_0), (c^e, \rho'), a)$ fall into a *single* class (namely, the class specified by the *same* incarnation of verifier $V^j(x)$ – the prover in argument $\mathsf{PZK}_{\mathsf{KInstD}}$, and the *same* P^*'s (the verifier in $\mathsf{PZK}_{\mathsf{KInstD}}$ argument) first message c_0 of argument $\mathsf{PZK}_{\mathsf{KInstD}}$), and E needs only to simulate this class of sessions and plays as honest verifier in any other sessions (that are uncorrelated to the simulated class of sessions) in straight line way.

Assume P^* convinces an incarnation $V^j(x)$ on statement $x = (x_0, x_1) \in L$ but $x_0 \notin L_0$ with probability p in the last session.

[16] This is just for simplicity, see definition 4.

[17] Note that for the deterministic $V^j(x)$, if $(c_0^a, c_0) = (c_1^a, c_0')$, then we have $(c_0^e, \rho') = (c_1^e, \rho'')$. By the fact that message a is uniquely determined by the first two round messages (for simplicity we consider the same common input $x = (x_0, x_1) \in L$) when $x_0 \notin L_0$, we have $((c_0^a, c_0), (c_0^e, \rho'), a) = ((c_1^a, c_0'), (c_1^e, \rho''), a')$, which contradicts the condition $((c_0^a, c_0), (c_0^e, \rho'), a) \neq ((c_1^a, c_0'), (c_1^e, \rho''), a')$.

The extractor E:

1. E selects a random tape for P^*.
2. E interacts with P^*. In this step, E executes the honest verifier's strategy except that when the honest verifier is instructed to applying the pseudorandom function specified by its random tape to generate randomness, it uses truly random coins, making sure that on the same session prefix the same coins are used in computing the next message (We make this exception just for simplifying the analysis.). In the last session, if E obtains an accepting transcript (a, e, z) of the underlying 3 round WI argument in the last session, E goes to the next step; Otherwise, outputs "$\perp$".
   Suppose the first three messages exchanged in the last session are $(c^a, c_0), (c^e, \rho')$ *and* (a, τ)*, and suppose* τ^* *was the valid correctness proof for* a *that appeared for the* first *time in all sessions with the prefix* $((c^a, c_0), (c^e, \rho'), a)$ *(note that* τ^* *doesn't necessarily equal* τ*, and this possibly happens due to the resetting attack from* P^**).*
3. (**Estimation**) E rewinds P^* to the point (we call it **rewinding point**) where the prefix $((c^a, c_0), (c^e, \rho'), (a, \tau^*))$ was first appeared, and repeats the following until it receives the accepting transcript (a, e, z) of the underlying 3 round WI argument n^2 times: choose *independent randomness* to complete the experiment described as in step 2.
   We denote by X *the total number of iterations of step 2.*
4. E rewinds P^* to the same point as in step 3, and repeats the following until it obtains another accepting transcript (a, e', z') with $e \neq e'$ or the $X+1^{st}$ trial is reached. If all trials fails, output "$\perp$".
   - For those sessions having the same prefix $(c^a, c_0), (c^e, \rho')$ and a valid a (we call them **target sessions**), E chooses another query $e' \neq e$ randomly, and sends e' to P^*, then runs the non-black-box simulator $\mathsf{Sim}^{\mathsf{E}}_{\mathsf{KID}} = (\mathsf{Intermed}^{\mathsf{E}}, \mathsf{Sim}^{\mathsf{E}}_{\mathsf{B}})$ for the underlying $\mathsf{PZK}_{\mathsf{KInstD}}$ to prove knowledge of s such that $e' = Com_v(e', s)$, where $\mathsf{Sim}^{\mathsf{E}}_{\mathsf{KID}}$ proceeds exactly as the same as $\mathsf{Sim}_{\mathsf{KID}}$ (presented in figure 6) except that:
     - $\mathsf{Sim}^{\mathsf{E}}_{\mathsf{KID}}$ handles only a *single* class of sessions with respect to the underlying argument $\mathsf{PZK}_{\mathsf{KInstD}}$. Note that all executions of $\mathsf{PZK}_{\mathsf{KInstD}}$ argument in those sessions having the same prefix $((c^a, c_0), (c^e, \rho'), a)$ fall into a *single* class, that is, all these executions are carried out with the *same* incarnation of verifier $V^j(x)$ under the *same* P^*'s (the verifier in $\mathsf{PZK}_{\mathsf{KInstD}}$ argument) first message c_0 of $\mathsf{PZK}_{\mathsf{KInstD}}$.
     - When $\mathsf{Sim}_{\mathsf{B}}$ is instructed to commit to hash value of a code, the corresponding subroutine $\mathsf{Sim}^{\mathsf{E}}_{\mathsf{B}}$ of $\mathsf{Sim}^{\mathsf{E}}_{\mathsf{KID}}$ commits to the *joint* code of the residual procedures $\mathsf{Intermed}^{\mathsf{E}}$ and E(except the current subroutine $\mathsf{Sim}^{\mathsf{E}}_{\mathsf{KID}}$, we view it as an external procedure.) and P^*, where $\mathsf{Intermed}^{\mathsf{E}}$ is exactly the same as $\mathsf{Intermed}$ presented in figure 6.
   - For all other sessions, E executes the strategy described in step 2.
5. E computes the witness for x_1 (note that we assume $x_0 \notin L$) from the two transcripts (a, e, z) and (a, e', z'), outputs it.

Fig. 7. The extractor E.

Recall the extractor E chooses *independent* random coins whenever the honest verifier would have generated these coins by applying a pseudorandom function to the current session history. We first give some notations.

- R: the set of all possible choices of random string that E uses in its step 2.
- R_{good} and R_{pre}. We call a random string used by E in step 2 that causes E to accept *good*, and denote by R_{good} the set of all *good* random strings. Given a good random string $r \in R_{\mathsf{good}}$, we call its prefix r_0 used in E's step 2 before the rewinding point *rewinding-prefix*, and denote by R_{pre} the set of the rewinding-prefixes of all good random strings. *Note that the rewinding-prefix of a single good random string is unique, and different good random strings may have the same rewinding-prefix.*
- r_0-*good* and G_{r_0}. A good random string r is called r_0-*good* if the rewinding-prefix of r is r_0. In other words, if the segment of execution of E's step 2 using r_0 ends with a session prefix $((c^a, c_0), (c^e, \rho'), (a, \tau))$, then a r_0-good random string r causes E to accept in a session with the prefix of form $((c^a, c_0), (c^e, \rho'), (a, \tau^*))$ (τ^* may be different from τ). We denote G_{r_0} the set of all r_0-good random strings. Obviously,

$$R_{\mathsf{good}} = \cup_{r_0 \in R_{\mathsf{pre}}} G_{r_0}, \quad and \quad G_{r_0} \cap G_{r_0'} = \emptyset \quad when \quad r_0 \neq r_0'$$

- R_{bad}. We call a random string r in R *bad* if any prefix of r does not fall into R_{pre}, and denote R_{bad} the set of all bad random strings. **Warning**: $R_{\mathsf{bad}} \neq R \setminus R_{\mathsf{good}}$!

A random experiment equivalent to E's step 2. We can view E's step 2 as the following random process: 1) E picks a string t from the following set uniformly:

$$R' = (R_{\mathsf{bad}}, \underbrace{r_0^1, r_0^1, ...r_0^1}_{|\{r_1^1 : r_0^1 || r_1^1 \in R\}|}, \quad \underbrace{r_0^2, ...r_0^2}_{|\{r_1^2 : r_0^2 || r_1^2 \in R\}|}, ..., \quad \underbrace{r_0^i, ..., r_0^i}_{|\{r_1^i : r_0^i || r_1^i \in R\}|}, ...)$$

where $r_0^i \in R_{\mathsf{pre}}$ ($i \geq 1$) and "$||$" represents concatenation. For instance, E picks r_0^i with probability $|\{r_1^i : r_0^i || r_1^i \in R\}|/|R'|$. 2) If $t \in R_{\mathsf{bad}}$, outputs "$\perp$"; If $t \in R_{\mathsf{pre}}$, chooses a string r_1 from the set $\{r_1 : t || r_1 \in R\}$ at random, then if $t||r_1$ is t-good[18], E goes to step 3, otherwise, outputs "$\perp$".

The equivalence between the above random experiment and E's step 2 is implied by the condition 2 of fact B1 (below).

We now define the following probabilities:

- p': the probability that E goes to step 3 in the above random experiment. This probability takes over R.
- p_{r_0}: the probability that, given a random string $r_0 \in R_{\mathsf{pre}}$, E picks r_1 such that $r_0||r_1$ is r_0-good. This probability takes over $\{r_1 : r_0||r_1 \in R\}$.
- p'_{r_0}: the probability that, given a random string $r_0 \in R_{\mathsf{pre}}$, P^* convinces E in a single execution of step 4. This probability takes over the randomness used by E in an single execution of its step 4.

[18] Note that the random string $r = t||r_1$ may be good but not t-good, for example, it may be t'-good, and t' is a prefix of t with $t \neq t'$, and then it seems that E should go to step 3 because the random string $t||r_1$ caused it to accept in step 2. However, in this case, we have E in this experiment output $\perp$, because this good random string $r = t||r_1$ was counted into $G_{t'}$ with the form $r = t'||r_1'$. We will see the probability that E goes to its step 3 in this random experiment is exactly the same as in its step 2 in the proof of fact 1.

We now prove the following fact and claims to complete the analysis of resettably-sound argument of knowledge property. In what follows, we denote by $neg(n)$ be an negligible function.

Fact B1. The following conditions hold:

1. $p - p' < neg(n)$;
2. $p' = \sum_{r_0 \in R_{\mathsf{pre}}} Pr[t = r_0] p_{r_0}$;
3. For any $r_0 \in R_{\mathsf{pre}}$, $p_{r_0} - p'_{r_0} < neg(n)$.

proof. The first condition follows from pseudorandomness of the pseudorandom function that the honest verifier uses to generate randomness in its first step.

For the second condition, we observe that $p' = |R_{\mathsf{good}}|/|R|$ and $|R| = |R'|$ (R' defined in the above random experiment), then we have

$$\begin{aligned}
&\sum_{r_0 \in R_{\mathsf{pre}}} Pr[t = r_0] p_{r_0} \\
=& \sum_{r_0^i \in R_{\mathsf{pre}}} |\{r_1^i : r_0^i || r_1^i \in R\}|/|R'| \times |G_{r_0^i}|/|\{r_1^i : r_0^i || r_1^i \in R\}| \\
=& \sum_{r_0 \in R_{\mathsf{pre}}} |G_{r_0}|/|R'| \\
=& |R_{\mathsf{good}}|/|R| \\
=& p'
\end{aligned}$$

Note also that the probability p that E goes to step 3 in the actual extraction presented in figure 7 equals $\sum_{r_0 \in R_{\mathsf{pre}}} Pr[t = r_0] p_{r_0} = p'$. This implies that the equivalence between the aforementioned random experiment and E's step 2.

The condition 3 is valid due to the 1-class-bounded resettable zero knowledge property on NO key_instance x_0 of the underlying argument $\mathsf{PZK}_{\mathsf{KInstD}}$ and the *perfect-hiding* property of scheme Com_v. We can use a simple hybrid argument to establish this. □

Claim B2 E runs in expected polynomial time.

proof. Note that E enters step 3 with probability $p' = \sum_{r_0 \in R_{\mathsf{pre}}} Pr[t = r_0] p_{r_0}$. Given $r_0 \in R_{\mathsf{pre}}$, the expected number of trials (i.e., X) in step 3 is n^2/p_{r_0}. Let $poly_0$ be the running time of a single run of step 3 and $poly_1$ be the running time of a single run of step 4. So the expected running time of E is $(poly_0 + poly_1)(\sum_{r_0 \in R_{\mathsf{pre}}} Pr[t = r_0] p_{r_0} \times n^2/p_{r_0}) < 2n^2$. (Note that $\sum_{r_0 \in R_{\mathsf{pre}}} Pr[t = r_0] < 1$.) □

Claim B3. E extracts a witness of x_1 with probability negligibly close to p.

proof. Note that the Goldreich-Kahan technique [15] guarantees that, for every $r_0 \in R_{\mathsf{pre}}$, the estimation n^2/X of p_{r_0} in step 3 is within a constant factor of p_{r_0} except with exponentially small probability, thus, we conclude that $X > n^2/(c \cdot p_{r_0})$ holds for some constant c except with exponentially small probability.

Conditioned on $t = r_0 \in R_{\mathsf{pre}}$, the probability that E enters step 4 without extracting a witness is $p_{r_0}(1 - p'_{r_0})^X$, and thus the probability that E enters step 4 without extracting a witness (Observe that a single successful run of step 4 will enable E to extract a witness) is

$\sum_{r_0 \in R_{\mathsf{pre}}} Pr[t = r_0] p_{r_0} (1 - p'_{r_0})^X$. By fact 1, we have

$$\begin{aligned} & \sum_{r_0 \in R_{\mathsf{pre}}} Pr[t = r_0] p_{r_0} (1 - p'_{r_0})^X \\ \leq & \sum_{r_0 \in R_{\mathsf{pre}}} Pr[t = r_0] p_{r_0} (1 - p_{r_0} + neg(n))^{n^2/(c \cdot p_{r_0})} \end{aligned}$$

If p_{r_0} is negligible, then $\sum_{r_0 \in R_{\mathsf{pre}}} Pr[t = r_0] p_{r_0} (1 - p'_{r_0})^X$ is negligible; If $p_{r_0} > 1/poly$ for some polynomial $poly$, then $(1 - p_{r_0} + neg(n))^{n^2/(c \cdot p_{r_0})}$ is negligible, thus $\sum_{r_0 \in R_{\mathsf{pre}}} Pr[t = r_0] p_{r_0} (1 - p'_{r_0})^X$ is also negligible.

We now conclude that the probability that E extracts a witness of x_1 with probability at least $p'(1 - neg(n)) = p - neg(n)$. □

■

Remark On choosing commitment scheme Com_v. It seems really hard to do a probability analysis for our extraction strategy when we replace the perfect-hiding commitment scheme Com_v with a statistically-binding one. The difficulty lies in that this time we cannot claim $p_{r_0} - p'_{r_0} < neg(n)$ for any $r_0 \in_{\mathsf{pre}}$ as in fact 1 because the computational-hiding property of Com_v guarantees only the expected values of p_{r_0} and p'_{r_0} (over choices of r_0) are negligible close.

Proof of resettable witness indistinguishability. Let $L = L_0 \vee L_1 = \{(x_0, x_1) : x_0 \in L_0 \text{ or } x_1 \in L_1\}$, $poly(\cdot)$ be an arbitrary polynomial and V^* be an arbitrary malicious PPT verifier strategy mounting resetting attack. Let $\overline{x} = x^1, \cdots, x^{poly(n)}$, $x^i = (x_0^i, x_1^i) \in L$, $\overline{w_b} = w_b^1, \cdots, w_b^{poly(n)}$ such that $(x_b^i, w_b^i) \in R_{L_b}$ for $i = 1, \cdots, poly(n)$, $b = 0, 1$. We set up hybrid experiments, in which the distribution in one of the hybrids is indistinguishable from that in preceding one, to prove the *Resettable witness indistinguishability.*

Hybrid 0 The distribution $(P(\overline{w_0}), V^*)(\overline{x})$ (V^*'s view in interaction with honest prover using $\overline{w_0}$ as witnesses).

Hybrid 1 The distribution $(P_{1,\overline{w_0}}(\overline{w_0}), V^*)(\overline{x})$, where $P_{1,\overline{w_0}}$ follows the P's strategy except that for every i, $1 \leq i \leq poly(n)$, $P_{1,\overline{w_0}}$ uses the witness w_0^i for x_0^i to execute the ZAP in P's second step in which it proves $x_0^i \in L_0$ or the a is computed correctly (in Hybrid 1, Hybrid 2 and Hybrid 3, we use the subscript $\overline{w_0}$ to indicate that the prover will use the witness in the sequence $\overline{w_0}$ to execute the ZAP in its second step). Note that in Hybrid 0 the honest prover always uses the witness for the statement that a is computed correctly to prove that $x_0^i \in L_0$ or the a is computed correctly (see the protocol in figure 1). So, we can claim that this hybrid is indistinguishable from the Hybrid 0 due to the witness indistinguishability of the ZAP.

Hybrid 2 The distribution $(P_{2,\overline{w_0}}(\overline{w_0}), V^*)(\overline{x})$, where $P_{2,\overline{w_0}}$ follows $P_{1,\overline{w_0}}$'s strategy, except that it selects a pseudorandom function $f_{s''}$ at random (independent of $f_{s'}$ that committed in its first message c^a) and produces a in the P's second step using randomness generated by applying this function to the history so far, and for all session shared the same c^a, P_2 always uses the same function $f_{s''}$. This hybrid is indistinguishable from the Hybrid 1 due to computationally hiding of the commitment scheme Com_p.

Hybrid 3 The distribution $(P_{2,\overline{w_0}}(\overline{w_1}), V^*)(\overline{x})$, where $P_{2,\overline{w_0}}$, given both $\overline{w_0}$ (used to execute the ZAP in its second step) and $\overline{w_1}$, follows prover's strategy in Hybrid 2, except that for all i, $1 \leq i \leq poly(n)$, it uses w_1^i for x_1^i to execute the underlying 3-round WI

argument for Hamiltonian Cycle . We will show that if there is a distinguisher can tell this hybrid and Hybrid 2, then there exists a PPT verifier strategy in concurrent model that violates the witness indistinguishability of the underlying 3-round WI argument. Note that witness indistinguishability is preserved in concurrent model, thus we conclude that this hybrid is indistinguishable from the Hybrid 2. Detailed proof follows shortly.

Hybrid 4 The distribution $(P_{3,\overline{w_0}}(\overline{w_1}), V^*)(\overline{x})$, where $P_{3,\overline{w_0}}$ follows $P_{2,\overline{w_0}}$'s strategy, except that it produces a in the same way that the honest prover does, that is, in the P's second step it uses randomness generated by applying $f_{s'}$ that committed in its first message c^a to the history so far to produce a (still, $P_{3,\overline{w_0}}$ uses the witness in the sequence $\overline{w_0}$ to execute the ZAP in its second step). Again, the indistinguishability between this hybrid and Hybrid 3 is due to computationally hiding of the commitment scheme Com_p.

Hybrid 5 $(P(\overline{w_1}), V^*)(\overline{x})$. Note that the prover P in this hybrid follows the honest prover's strategy with the witness sequence $\overline{w_1}$, and therefore the only difference between the P's strategy and the prover's strategy $P_{3,\overline{w_0}}$ in Hybrid 4 is that P uses the witness s' and r' that is used in its first step to compute c^a ($c^a = Com_p(s', r')$) to prove $x_0 \in L_0$ or a is computed correctly via a ZAP in its second step, while the prover's strategy $P_{3,\overline{w_0}}$ in Hybrid 4 uses the witness w_0^i for x_0^i. The indistinguishability between this hybrid and Hybrid 4 is due to the same reason for the indistinguishability between Hybrid 0 and Hybrid 1, i.e., the witness indistinguishability of the ZAP.

Let $(P_{\mathtt{W}}, V_{\mathtt{W}})$ be the underlying 3-round WI argument for Hamiltonian Cycle. Now we show that Hybrid 2 is indistinguishable from Hybrid 3. Assume otherwise, there exists an algorithm D distinguishes the two distributions $((P_{2,\overline{w_0}}(\overline{w_0}), V^*)(\overline{x})$ and $((P_{2,\overline{w_0}}(\overline{w_1}), V^*)(\overline{x})$, then we can construct a PPT $V_{\mathtt{W}}^*$ in concurrent model, such that two distributions $(P_{\mathtt{W}}(\overline{w_0}), V_{\mathtt{W}}^*)(\overline{x})$ and $(P_{\mathtt{W}}(\overline{w_1}), V_{\mathtt{W}}^*)(\overline{x})$ are distinguishable. This contradicts the witness indistinguishability of $(P_{\mathtt{W}}, V_{\mathtt{W}})$ (note that WI holds even in concurrent model, cf [13]).

$V_{\mathtt{W}}^*$, given $(\overline{w_0})$ as input[19], incorporates V^* and handles V^*'s messages as follows. 1) When V^* initiates a session with incarnation $P_{2,\overline{w_0}}^{i,j}$ ($1 \leq i, j \leq poly(n)$), $V_{\mathtt{W}}^*$ computes c^a and c_0 as the honest prover does, and replies with c^a and c_0 internally. 2) When V^* sends a kth new first message (i.e., c^e and ρ') to incarnation $P_{2,\overline{w_0}}^{i,j}$ ($1 \leq i, j \leq poly(n)$), $V_{\mathtt{W}}^*$ initiates a session with an new incarnation $P_{\mathtt{W}}^{i,j_k}$ (defined by $P_{\mathtt{W}}^{i,j_k}(\alpha) = P_{\mathtt{W}}(x^i, w^i, r_{j_k}, \alpha)$, where $x^i = (x_0^i, x_1^i)$, $w^i = (w_0^i, w_1^i)$, r_{j_k}'s are selected independently), obtains the $P_{\mathtt{W}}^{i,j_k}$'s first message a, uses the witness w_0^i to produce the proof τ that the first message a of the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument is computed correctly, stores a and τ and then forwards them to V^*; 3) when V^* sends a query e to $P_{2,\overline{w_0}}^{i,j}$ for its kth first message a, $V_{\mathtt{W}}^*$ holds on and continues the execution of the argument $\mathsf{PZK}_{\mathsf{KInstD}}$ with V^*, and once $V_{\mathtt{W}}^*$ accepts this proof from V^*, forwards the query e to $P_{\mathtt{W}}^{i,j_k}$, stores its response z and forwards it to V^*. All V^*'s repeated messages are replied with the same answer.

Noth that if V^* does not send different challenges to $P_{2,\overline{w_0}}^{i,j}$ regarding the same $P_{2,\overline{w_0}}^{i,j}$'s (actually produced by $P_{\mathtt{W}}^{i,j_k}$) first message a, then $V_{\mathtt{W}}^*$ works in concurrent model (i.e., it holds at most one session with each incarnation $P_{\mathtt{W}}^{i,j_k}$). Furthermore, if all incarnations of $P_{\mathtt{W}}$ use the witness sequence $\overline{w_b} = w_b^1, \cdots, w_b^{poly(n)}$ in above interaction, the V^*'s view in the above experiment is identical to $((P_{2,\overline{w_0}}(\overline{w_b}), V^*)(\overline{x})$ (note the fact that both $P_{2,\overline{w_0}}$ and $V_{\mathtt{W}}^*$ use the same witness in the sequence $\overline{w_0}$ to produce the proof τ), and furthermore, notice that the $V_{\mathtt{W}}^*$'s view (i.e., $(P_{\mathtt{W}}(\overline{w_b}), V_{\mathtt{W}}^*)(\overline{x})$) is just the copy of V^*'s view, Thus, we conclude

[19] Note that WI is required to hold against malicious verifiers that take both $(\overline{w_0})$ and $(\overline{w_1})$ as the auxiliary input

if D can distinguish $((P_{2,\overline{w_0}}(\overline{w_0}), V^*)(\overline{x})$ and $((P_{2,\overline{w_0}}(\overline{w_1}), V^*)(\overline{x})$, it also can distinguish $(P_{\mathtt{W}}(\overline{w_0}), V^*_{\mathtt{W}})(\overline{x})$ and $(P_{\mathtt{W}}(\overline{w_1}), V^*_{\mathtt{W}})(\overline{x})$.

We claim the probability that $V^*_{\mathtt{W}}$ sends different challenges to $P^{i,j_k}_{\mathtt{W}}$'s first message a is negligible. Otherwise, we can use resettably-sound argument of knowledge property on YES instance x_0 of the underlying $\mathsf{PZK}_{\mathsf{KInstD}}$ argument to break the computational binding property of Com_v (See proof of lemma 2 in appendix C for details). This completes the proof that Hybrid 2 is indistinguishable from Hybrid 3.

■

C Proof of resettable zero knowledge of our protocol

C.1 Description of the Simulator

In this section, we give detailed description of our simulator.

Convention: In the each prover step, whenever the honest prover is instructed to apply the pseudorandom function *defined by its random tape*[20] to generate random bits, our simulator Simulate tosses coins (i.e., uses truly random bits), following the *consistence* restriction that these random bits are the same in all those sessions for which the honest prover is expected to use the same pseudorandom coins.

For convenience of presentation, we first define some variables and notions.

- t, the recursion depth. We set $T = 2\log^K_{k/128}$ to the initial value of t.
- $\mathcal{Q}$, the waiting (to be solved) list, consisting of tuples of the form $(C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, (a_i, e'_i), t)$ which indicates the executions of i^{th} iteration of class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$ (in which the simulator proves that e'_i is correct via $\mathsf{PZK}_{\mathsf{KInstD}}$ argument) is currently being simulated, where a_i is the first message of underlying 3-round WI argument in the corresponding $\mathsf{IDWIAOK}^i_v$, t is the recursion level where this tuple is created. We initialize $\mathcal{Q} = \emptyset$.
- h, the current execution history of V^*. Initially, $h = (x_1, \cdots, x_{poly})$.
- $\mathcal{C}_t$, a table containing the solved classes. We say a session *solved* if for some i, a transcript (a_i, e'_i, z'_i) of the underlying 3-round WI argument in $\mathsf{IDWIAOK}^i_v$ was obtained, where e'_i is a challenge chosen at random (therefore different from the honest one), and say a class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$ *solved* if some session in this class is *solved*. Note that, for a solved class, the simulator can carry out any session belonging to this class: It executes the honest prover strategy to obtain the transcript (a_i, e_i, z_i) of the underlying 3-round WI argument in $\mathsf{IDWIAOK}^i_v$ (where the challenge message e_i is an honest one); When a session in this class reaches stage Mainproof, it can extract an valid witness from these two transcripts and then uses it to complete the stage Mainproof of this session[21].
- Unfortunate message. We call a message sent by V^* unfortunate messages *with respect to the view of* V^*[22] (*with respect to the view of the simulator* $\mathbf{S}$) if:
  1. It is a second V^*'s message in Stage Initiation or a stage Iteration V^*'s message in which the main part of this message[23] is different from the corresponding one sent

[20] This should not be confused with the case in which the honest party needs to apply some pseudorandom function to which it already committed in a previous step to produce the current message.

[21] In case the witness extracted from these two transcripts is the seed (δ) of γ, we still cannot complete Mainproof. Fortunately, the Lemma 3 (presented in next subsection) says this occurs only with negligible probability.

[22] Note that the view of V^* is the history of the current simulation thread, whereas the view of $\mathbf{S}$ is the whole simulation history so far, which may include many threads.

[23] I.e, the current message excluding those ZAP proofs in this step

in a previous session belonging to the same class that *appears in the current thread* (*appears in the whole simulation history so far*, resp.). Note that the two sessions have the same first three messages, and when the first prover message γ is NO instance, an unfortunate message of this type means that V^* breaks the uniqueness of the relevant InstD-VRF function. Or,

2. It is a V^*'s last message of an accepting correctness proof for a challenge message e of $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ or $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ in the current session, which has the same session-prefix till just before this challenge delivery with a previous session *appearing in the current thread* (*appearing in the whole simulation history so far*, resp.) where the corresponding challenge is *different than* e and V^* already completed the correctness proof successfully for it. This implies that V^* can open a commitment $c^e$to the challenge into two distinct values.

The following table are accessible to all algorithms at all levels of the recursion.

– $\mathcal{S}$, a table containing triplets of the form $(C_{\mathsf{f\text{-}msg}}^{(l,m)}, i, (a_i, e_i', z_i'))$, the information of the *solved* class $C_{\mathsf{f\text{-}msg}}^{(l,m)}$.

Remark on unfortunate message. We stress our simulator checks only the current thread, and outputs $\perp$ when the current message is a unfortunate message with respect to this thread (i.e., *with respect to the view of* V^*). However, in the analysis of our simulator, we need to show that the probability that our simulator receives an unfortunate message *even* with respect to all threads so far (i.e., *with respect to the view of the simulator* **S**) is negligible small (see proofs of lemma 2 and 3 in the next subsection).

The detailed description of our simulator depicted in figure 8, figure 9, and figure 10 and figure 11.

The simulator **S**:

Run $\mathsf{Simulate}(T, (x_1, \cdots, x_{poly}), \emptyset, \emptyset, \emptyset)$. Let h be the output of $\mathsf{Simulate}(T, (x_1, \cdots, x_{poly}), \emptyset, \emptyset, \emptyset)$. If the last message in h is $\perp$, output $\perp$; otherwise, output h.

Fig. 8. The simulator.

Remarks.

1. We remark that the goal of the run of $\mathsf{Solve}(t, h, \mathcal{Q}, \mathcal{C}_{t+1}, C_{\mathsf{f\text{-}msg}}^{(l,m)}, i)$ at level t is to solves one of the classes listed in $\mathcal{Q}' = \mathcal{Q} \bigcup (C_{\mathsf{f\text{-}msg}}^{(l,m)}, i, (a_i, e_i'), t)$, not just the class $C_{\mathsf{f\text{-}msg}}^{(l,m)}$; Furthermore, once a class having an entry $(C_{\mathsf{f\text{-}msg}'}^{(l',m')}, j, (a_j, e_j'), t_{\mathsf{sol}})$ in $\mathcal{Q}'$ (possibly, $t_{\mathsf{sol}} > t$) is solved, the simulator **S** returns *immediately* to the point where this entry was created, i.e., where the first prover step in jth iteration was reached for the first time by one session belonging to class $C_{\mathsf{f\text{-}msg}'}^{(l',m')}$, and all entries with the level index $t \leq t_{\mathsf{sol}}$ are deleted (see step $2(d)\mathrm{i}A$ of Simulate and step $1(f)$ of Solve). Previous rewinding strategies only focus on the current session for which Solve is invoked, and Solve does not return due to some other session is solved.
   Our strategy makes it clear that for each level $0 \leq t < T-1$, the waiting list $\mathcal{Q}$ contains only one entry of the form $(\cdot, \cdot, \cdot, t)$. This means in a single thread there are at most

$\mathsf{Simulate}(t, h, \mathcal{Q}, \mathcal{C}_t, \mathcal{S})$:

1. Extend by one verifier message: Set $\texttt{v-msg} \longleftarrow V^*(h)$, $h \longleftarrow (h, \texttt{v-msg})$.
   (a) If $\texttt{v-msg}$ is $\texttt{halt}$, return h;
   (b) If $\texttt{v-msg}$ is an accepting last message z_i' of i^{th} $\mathsf{Iteration}$ in session $s \in C_{\mathsf{f\text{-}msg}}^{(l,m)}$, and $(C_{\mathsf{f\text{-}msg}}^{(l,m)}, i, (a_i, e_i'), t_{\mathsf{sol}}) \in \mathcal{Q}$, set $\mathcal{S} \longleftarrow (\mathcal{S}, (C_{\mathsf{f\text{-}msg}}^{(l,m)}, i, (a_i, e_i', z_i')))$, $\mathcal{C}_t \longleftarrow (\mathcal{C}_t, C_{\mathsf{f\text{-}msg}}^{(l,m)})$, return h.
   (c) If $\texttt{v-msg}$ is an $\mathsf{unfortunate}$ message with respect to the view of V^* (i.e., with respect to h), set the next prover message $\texttt{p-msg} = \perp$, $h \longleftarrow (h, \texttt{p-msg})$ and return h.
   (d) Otherwise, continue.
2. Extend by one prover message (denote the next scheduled prover message by $\texttt{p-msg}$):
   (a) If $\texttt{p-msg}$ is the prover first message in session s belonging to the $(k/128)^t/16 + 1^{st}$ new class, set $\texttt{p-msg} = \perp$. Otherwise, continue.
   (b) If $\texttt{p-msg}$ is a prover $P^{(l,m)}$'s message in stage $\mathsf{Initialization}$ or stage Setup for session s, produce $\texttt{p-msg}$ by emulating the *honest* incarnation $P^{(l,m)}$ except that $\mathsf{Simulate}$ generates an YES instance γ and uses the corresponding witness for γ to carry out $\mathsf{IDWIAOK}_p^{\mathsf{S}}$.
   (c) If $\texttt{p-msg}$ is an i^{th} $\mathsf{Iteration}$ prover $P^{(l,m)}$'s message in session $s \in C_{\mathsf{f\text{-}msg}}^{(l,m)}$, and $C_{\mathsf{f\text{-}msg}}^{(l,m)} \in \mathcal{C}_t$, then produce $\texttt{p-msg}$ by emulating the *honest* incarnation $P^{(l,m)}$.
   (d) If $\texttt{p-msg}$ is an i^{th} iteration prover $P^{(l,m)}$'s *challenge* message in session $s \in C_{\mathsf{f\text{-}msg}}^{(l,m)}$, and $C_{\mathsf{f\text{-}msg}}^{(l,m)} \notin \mathcal{C}_t$, then
      i. If s is the first session in $C_{\mathsf{f\text{-}msg}}^{(l,m)}$ that reaches this step, produce $\texttt{p-msg}$ by emulating the *honest* incarnation $P^{(l,m)}$. In addition, if $t \geq 1$ (for $t = 0$, this additional step is ignored), do the following:
         A. (Invocation of Solve.) Run $\mathsf{Solve}(t-1, h, \mathcal{Q}, \mathcal{C}_t, C_{\mathsf{f\text{-}msg}}^{(l,m)}, i)$. Set $(\mathcal{C}_t', q_{\mathsf{sol}}) \longleftarrow \mathsf{Solve}(t-1, h, \mathcal{Q}, \mathcal{C}_t, C_{\mathsf{f\text{-}msg}}^{(l,m)}, i)$ and update $C_t \longleftarrow C_t'$. If $q_{\mathsf{sol}} = (C_{\mathsf{f\text{-}msg}'}^{(l',m')}, j, (a_j, e_j'), t_{\mathsf{sol}})$ and $t_{\mathsf{sol}} \geq t$ (this means $q_{\mathsf{sol}} = (C_{\mathsf{f\text{-}msg}'}^{(l',m')}, j, (a_j, e_j'), t_{\mathsf{sol}}) \in \mathcal{Q}$, i.e., the current invocation of $\mathsf{Simulate}$ succeeds in solving one class on the list $\mathcal{Q}$) return h; If $t_{\mathsf{sol}} = t - 1$ (this means the current class $C_{\mathsf{f\text{-}msg}}^{(l,m)}$ is solved, note that this class is not on the list $\mathcal{Q}$) or $q_{\mathsf{sol}} = \emptyset$ (the current invocation of Solve fails), continue.
      ii. If s is *not* the first session in $C_{\mathsf{f\text{-}msg}}^{(l,m)}$ that reaches this step, set $\texttt{p-msg}$ to be the same message sent in a previous session (stored in h) belonging to this class.
   (e) If $\texttt{p-msg}$ is an i^{th} iteration prover $P^{(l,m)}$'s *non-challenge* message (belonging to $\mathsf{PZK}_{\mathsf{KInstD}}$ arugment) in session $s \in C_{\mathsf{f\text{-}msg}}^{(l,m)}$, and $C_{\mathsf{f\text{-}msg}}^{(l,m)} \notin \mathcal{C}_t$, then
      i. If for some e_i', $(C_{\mathsf{f\text{-}msg}}^{(l,m)}, i, (a_i, e_i'), t') \in \mathcal{Q}$ $(t' \geq t)$, then forward $(h, C_{\mathsf{f\text{-}msg}}^{(l,m)}, i, t', t)$ to $\mathsf{Sim}_{\mathsf{KID}}^t$, the simulator for the underlying argument $\mathsf{PZK}_{\mathsf{KInstD}}$, and set $\texttt{p-msg} = \mathsf{Sim}_{\mathsf{KID}}^t(h, C_{\mathsf{f\text{-}msg}}^{(l,m)}, i, t', t)$.
      ii. Otherwise, produce $\texttt{p-msg}$ by emulating the *honest* incarnation $P^{(l,m)}$.
   (f) If $\texttt{p-msg}$ is a prover $P^{(l,m)}$'s message in the stage $\mathsf{Mainproof}$ for session $s \in C_{\mathsf{f\text{-}msg}}^{(l,m)}$, $C_{\mathsf{f\text{-}msg}}^{(l,m)} \in \mathcal{C}_t$, and $(C_{\mathsf{f\text{-}msg}}^{(l,m)}, i, (a_i, e_i', z_i')) \in \mathcal{S}$, retrieve the subtranscript (a_i, e_i, z_i) of the underlying 3-round WI argument of corresponding $\mathsf{IDWIAOK}_v^i$ from h, compute a witness w' from (a_i, e_i', z_i') and (a_i, e_i, z_i), then
      i. If β (sent in the verifier's first step for the current class) equals $f(w')$, then produce $\texttt{p-msg}$ by emulating the *honest* incarnation $P^{(l,m)}$ except that $\mathsf{Simulate}$ uses w' (rather than the witness w_l for $x_l \in L$) as witness to carry out $\mathsf{IDWIAOK}_p^{\mathsf{M}}$.
      ii. Otherwise, set $\texttt{p-msg} = \perp$.
   (g) If $\texttt{p-msg}$ is a prover $P^{(l,m)}$'s message in the stage $\mathsf{Mainproof}$ for session $s \in C_{\mathsf{f\text{-}msg}}^{(l,m)}$, and $C_{\mathsf{f\text{-}msg}}^{(l,m)} \notin \mathcal{C}_t$, then set $\texttt{p-msg} = \perp$.
3. Set $h \longleftarrow (h, \texttt{p-msg})$. If $\texttt{p-msg} = \perp$, return h; Otherwise, go back to step 1.

Fig. 9. The procedure $\mathsf{Simulate}$.

$\mathsf{Solve}(t, h, \mathcal{Q}, \mathcal{C}_{t+1}, C^{(l,m)}_{\mathsf{f\text{-}msg}}, i)$:

1. for $d = 1$ to $24K$, do:
   (a) Set $\mathcal{C}_t \longleftarrow C_{t+1}$ (each attempt begins with the same table $\mathcal{C}_{t+1}$).
   (b) Choose e'_i at random, which serves as the challenge message of the i^{th} Iteration in session $s \in C^{(l,m)}_{\mathsf{f\text{-}msg}}$
   (c) Set $h \longleftarrow (h, e'_i)$, $Q \longleftarrow (Q, (C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, (a_i, e'_i), t))$.
   (d) Run $\mathsf{Simulate}(t, h, \mathcal{Q}, \mathcal{C}_t, \mathcal{S})$.
   (e) Set $\mathcal{C}'_{t+1} \longleftarrow \mathcal{C}_t$ (Store the classes in $\mathcal{C}'_{t+1}$ that solved during the current attempt).
   (f) Check table $\mathcal{S}$ and $\mathcal{Q}$, if for some $C^{(l',m')}_{\mathsf{f\text{-}msg'}}$ and j such that $(C^{(l',m')}_{\mathsf{f\text{-}msg'}}, j, (a_j, e'_j, z'_j)) \in S$ and $(C^{(l',m')}_{\mathsf{f\text{-}msg'}}, j, (a_j, e'_j), t_{\mathsf{sol}}) \in \mathcal{Q}$, then let $q_{\mathsf{sol}} = (C^{(l',m')}_{\mathsf{f\text{-}msg'}}, j, (a_j, e'_j), t_{\mathsf{sol}})$, and
      i. Delete the tuplet $(C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, (a_i, e'_i), t)$ from $\mathcal{Q}$. (Before Solve ends, it delete the tuple created in this run from $\mathcal{Q}$.)
      ii. Return $(\mathcal{C}'_{t+1}, q_{\mathsf{sol}})$.
   (g) Otherwise, $d = d + 1$ and go to step (a).
2. Delete the tuplet $(C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, (a_i, e'_i), t)$ from $\mathcal{Q}$, and return $(\mathcal{C}'_{t+1}, \emptyset)$.

Fig. 10. The procedure Solve.

$T < \log n$ classes of sessions *with respect to the argument* PZK_{KInstD} being simulated. This is crucial for our analysis because our argument $\mathsf{PZK}_{\mathsf{KInstD}}$ in those special-purpose $\mathsf{IDWIAOK}^i_v$ satisfies only $\log n$-class-bounded resettable zero knowledge property.

2. The procedure Solve does not update the history, and furthermore, all attempts within a call of Solve are *completely* independent. Observe that each attempt within the invocation of Solve at level t starts from exactly the same state: the current attempt uses only information about classes solved (stored in $\mathcal{S}$) in this attempt or solved before this invocation of Solve, and ignores the information about classes solved in previous attempts within the same invocation of Solve. Note that different attempts might solve different classes (at level lower than t) that are not in the waiting list $\mathcal{Q}'$ (as defined in the first remark), and hence the table $\mathcal{S}$ might differ at the beginning of each attempt. This property is due to the fact that the Simulate checks only the *local* table $\mathcal{C}_t$ (rather than the *global* table $\mathcal{S}$) in step 2 to decide what to do, and the local table $\mathcal{C}_t$ is the same at the beginning of each attempt within a call of Solve at level t. (see step 2 of Simulate and step 1(a) of Solve.) This helps us simplify our analysis much.
3. As in previous simulation strategies[6,22], the information about classes solved in all attempts within a call of Solve at level t is accessible to Simulate at level $t + 1$ (see step $2(d)$iA of Simulate and step $1(e)$ and $1(f)$iiof Solve).
4. The correctness of $\mathsf{Sim}^t_{\mathsf{KID}}$ follows from the following facts.
   – The underlying procedure $\mathsf{Sim}^t_{\mathsf{B}}$, which treats the joint code of the residual procedures $\mathsf{Intermed}^t$, Simulate at level t[24] and V^* as a (non-resetting) malicious verifier, handles only a single subsession with respect to some i^{th} iteration of a class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$ for which the current Simulate at level t was invoked.
   – All subsessions with respect the i^{th} iteration of a class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$ formes a single class of subsessions (see footnote 8), and when a subsession belonging to this class *first* reaches the end of stage 1 of the underlying simplified Pass-Rosen protocol, $\mathsf{Sim}^t_{\mathsf{B}}$ already know the corresponding "valid" witness (i.e., the joint code of the residual

[24] As mentioned before, the Simulate at level t includes all subroutines at lower level invoked by it directly or indirectly (except the current $\mathsf{Sim}^t_{\mathsf{KID}}$).

$\mathsf{Sim}^t_{\mathsf{KID}}(h, C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, t', t'')$:

Run the procedure consisting of an intermediator $\mathsf{Intermed}^t$ and a slightly modified Barak's simulator $\mathsf{Sim}^t_{\mathsf{B}}$ (defined below)on input $(h, C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, t', t'')$.

/*Recall that here t' indicates that the next prover message will be produced by $\mathsf{Sim}^{t'}_{\mathsf{KID}}$ invoked at level t' and t'' indicates the level at which $\mathsf{Simulate}$ made this query. Note that we always have $t' \geq t$ and $t'' \leq t$.
The queries to $\mathsf{Sim}^t_{\mathsf{KID}}$ comes directly from two procedures: the Simulate at level t and $\mathsf{Sim}^{t-1}_{\mathsf{KID}}$ (though $\mathsf{Sim}^{t-1}_{\mathsf{KID}}$ does not make query itself). The query from the former one has $t'' = t$, and the query from the latter has $t'' < t$./*

$\mathsf{Intermed}^t$: maintain a table R^t (initially being empty) and act as follows.

1. Upon receiving $(h, C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, t', t'')$, retrieve the subtranscript tr of the underlying $\mathsf{PZK}_{\mathsf{KInstD}}$ argument in the i^{th} $\mathsf{Iteration}$ (i.e., the messages sent in stage 2 of this $\mathsf{PZK}_{\mathsf{KInstD}}$ argument, together with the corresponding statement "challenge is correct") of the current session, and store tr in R^t.
2. If the next scheduled prover message to tr is stored in R^t (as we will see, this implies this query comes from the current $\mathsf{Simulate}$ at level t), forward the same message to the current $\mathsf{Simulate}$ at level t.
3. Else, do the following:
   (a) If $t'' \leq t < t'$, forward $(h, C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, t', t'')$ to $\mathsf{Sim}^{t+1}_{\mathsf{KID}}$, obtain the next prover message to tr, store this message in R^t and forward it to $\mathsf{Sim}^t_{\mathsf{B}}$, then act as follows. If $t'' < t$, forward it to $\mathsf{Sim}^{t-1}_{\mathsf{KID}}$ invoked at the level $t-1$; Otherwise ($t'' = t$), forward it to $\mathsf{Simulate}$ at this level t.

   /*_Comments:_ Note that $\mathsf{Intermed}^t$ stores the answer from $\mathsf{Sim}^{t+1}_{\mathsf{KID}}$, and this implies that it never repeats the same query to $\mathsf{Sim}^{t+1}_{\mathsf{KID}}$. It should also be noted that here $\mathsf{Intermed}^t$ does not forward $(h, C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, t', t'')$ to the $\mathsf{Sim}^{t'}_{\mathsf{KID}}$ invoked at level t' (which will produce the next prover message) directly, instead it forward it to $\mathsf{Sim}^{t+1}_{\mathsf{KID}}$ invoked at its neighbor level $t+1$. This means that $(h, C^{(l,m)}_{\mathsf{f\text{-}msg}}, i, t', t'')$ will go through $\mathsf{Sim}^{t+1}_{\mathsf{KID}}$, $\mathsf{Sim}^{t+2}_{\mathsf{KID}}$..., and finally arrives at $\mathsf{Sim}^{t'}_{\mathsf{KID}}$ ($t < t+1 < \cdots < t'$). Once the next prover message is produced by $\mathsf{Sim}^{t'}_{\mathsf{KID}}$, it will go through (and will be stored by) $\mathsf{Sim}^{t'-1}_{\mathsf{KID}}$, $\mathsf{Sim}^{t'-1}_{\mathsf{KID}}$,..., and finally arrive at $\mathsf{Sim}^{t''}_{\mathsf{KID}}$ and will be further forwarded to $\mathsf{Simulate}$ at level t''. This guarantees that for all $t' > t$, $\mathsf{Sim}^t_{\mathsf{B}}$ can obtain those prover messages produced by $\mathsf{Sim}^{t'}_{\mathsf{KID}}$ (and treats these messages as *external* messages)./*

   (b) If $t'' \leq t = t'$, forward tr to the modified Barak's simulator $\mathsf{Sim}^t_{\mathsf{B}}$, obtain the next prover message from $\mathsf{Sim}^t_{\mathsf{B}}$, store this message in R^t and then act in the same way as in the previous step: If $t'' < t$, forward it to $\mathsf{Sim}^{t-1}_{\mathsf{KID}}$ invoked at the level $t-1$; Otherwise, forward it to $\mathsf{Simulate}$ at this level t.

$\mathsf{Sim}^t_{\mathsf{B}}$: given procedures $\mathsf{Intermed}^t$, $\mathsf{Simulate}(t, h, \mathcal{Q}, \mathcal{C}_t, \mathcal{S})$, and V^* as input, act as follows.

1. Upon receiving a prover message from $\mathsf{Intermed}^t$, just store it.
2. Upon receiving tr, produce the next scheduled prover message as the Barak's simulator (for $\log n$-bounded current ZK argument) with the following natural modifications:
   - At the first prover step of this subsession (where the Barak's simulator needs to commit a code), $\mathsf{Sim}^t_{\mathsf{B}}$ computes a commitment $Com(h(\Pi(y_1, \cdot)))$ to hash value of $\Pi(y_1, \cdot)$, where Π is the joint code of $\mathsf{Intermed}^t$, $\mathsf{Simulate}(t, h, \mathcal{Q}, \mathcal{C}_t, \mathcal{S})$ (except the current subroutine $\mathsf{Sim}^t_{\mathsf{KID}}$, but including all procedures at lower level invoked by it directly or indirectly), and V^*, where h is the hash function sent by V^*, y_1 denotes the sequence of prover messages $\mathsf{Sim}^t_{\mathsf{B}}$ received (from $\mathsf{Sim}^{t'}_{\mathsf{B}}$ at higher level t') so far, i.e., $\Pi(y_1, \cdot)$ is the joint code of the *residual* procedures $\mathsf{Intermed}^t$, $\mathsf{Simulate}(t, h, \mathcal{Q}, \mathcal{C}_t, \mathcal{S})$, and V^*. $\mathsf{Sim}^t_{\mathsf{B}}$ forward $Com(h(\Pi(y_1, \cdot)))$ to $\mathsf{Intermed}^t$.
   - After the first prover step, $\mathsf{Sim}^t_{\mathsf{B}}$ proceeds exactly as Barak's simulator by treating the prover messages it received between its first step and the second verifier step of this subsession as external messages.

   /*_Comments:_ It should be noted that the procedure $\mathsf{Sim}^t_{\mathsf{B}}$ handles only a single session, though it is able to simulate $\log n$ sessions. Note also that if the total length of all prover messages (being treated by $\mathsf{Sim}^t_{\mathsf{B}}$ as *external* messages), which are produced by those $\mathsf{Sim}^{t'}_{\mathsf{KID}}$ invoked at higher level $t' > t$, is "short", then this $\mathsf{Sim}^t_{\mathsf{B}}$ works./*

Fig. 11. The procedure $\mathsf{Sim}^t_{\mathsf{KID}}$.

procedures $\mathsf{Intermed}^t$, $\mathsf{Simulate}$ at level t and V^*, and some relevant prover messages sent in other sessions handled by $\mathsf{Sim}^{t'}_{\mathsf{KID}}$ at higher level t') to carry out this subsession. It is easy to verify that $\mathsf{Sim}^t_{\mathsf{KID}}$ succeeds as long as the total length of the *external* messages is "short". This is the case because there are only $T - t < \log n$ subsessions (that simulated by some $\mathsf{Sim}^{t'}_{\mathsf{KID}}$ at higher level $t' > t$.) for which $\mathsf{Sim}^t_{\mathsf{KID}}$ treats the prover's messages therein as *external* messages.

- The same prover message produced by $\mathsf{Sim}^t_{\mathsf{B}}$ can be used (by $\mathsf{Intermed}^t$) to answer any V^*'s query in the same step in any subsession belonging to this class. This is due to that the all subsessions belonging to this class share the same transcript of a single session of the underlying simplified Pass-Rosen protocol (see section 4 for the structure of a class of sessions) even in our case that γ is an YES instance (see the proof of lemma 2 and 3 in next section).

  performs well: It takes only t sessions (in concurrent model),

It should be noted that when the simulator commits to a code, this code needs to be well-defined, thus the simulator $\mathsf{Sim}^{t''}_{\mathsf{KID}}$ at lower level is not able to commit to the simulator $\mathsf{Sim}^t_{\mathsf{KID}}$ at higher level; In our setting, at the bottom level, all $\mathsf{Sim}^0_{\mathsf{KID}}$ is well defined; Note that $\mathsf{Sim}^1_{\mathsf{KID}}$ is well defined as long as all $\mathsf{Sim}^0_{\mathsf{KID}}$ is well defined; Repeating this, it is easy to see that when $\mathsf{Sim}^t_{\mathsf{KID}}$ commits to the code of $\mathsf{Simulate}$ at level t, all codes of $\mathsf{Sim}^{t'}_{\mathsf{KID}}$ at lower level $t' < t$ therein are well defined.

C.2 Analysis of simulator S

Roughly, our analysis of simulator **S** proceeds as follows. We first prove that, conditioned on the event that the output of **S** is not $\perp$, **S** is valid, and then we bound the probability that **S** outputs $\perp$. The latter is most challenging task in this analysis, and on a high level, its analysis somewhat similar to the one in [6]. However, due to the non-black box nature of the simulator and some subtleties arising therefrom, our techniques are very different.

We prove the following lemmas to conclude that our simulator works.

Lemma 1. Conditioned on not being $\perp$, the history output by simulator **S** is (computational) indistinguishable from the real interaction history.

Proof. Note that at the top level of recursion, $\mathsf{Simulate}(T, (x_1, \cdots, x_{poly}), \emptyset, \emptyset)$ acts exactly as the honest provers except that: 1) It generates YES instance γ; 2) It uses witnesses different from the ones used by honest prover to carried out those $\mathsf{IDWIAOK}^{\mathsf{S}}_p$ and $\mathsf{IDWIAOK}^{\mathsf{M}}_p$; 3) It uses truly random coins whenever the honest prover uses pseudorandom coins (by the **Convention** we made at the beginning of description of the simulator).

Note also that at level T, $\mathsf{Simulate}$ does not do any non-black-box simulation. This enables us to prove this lemma by simple hybrid arguments. Consider the following hybrid prover:

HProver acts exactly as honest prover except that, it generates YES instance γ in every session.

The indistinguishability between the interaction between V^* and HProvers and the real interaction follows from (using standard hybrid argument again) the pseudorandomness of G. Similarly, we consider the following hybrid simulator:

S′ acts exactly as **S** except that, it generates (pseudo)randomness in each prover step in the same way as honest prover.

Due to the resettable witness indistinguishability of $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ and $\mathsf{IDWIAOK}_p^{\mathsf{M}}$, the output of $\mathbf{S}'$ conditioned on not being "$\perp$"[25] is indistinguishable from the interaction between V^* and HProvers. Furthermore, it follows from the pseudorandomness of the pseudorandom function (defined by random tape of the honest prover) that, the output of $\mathbf{S}'$ is indistinguishable from the output of $\mathbf{S}$ (both conditioned on not being "$\perp$"). □

Observe that there are only four events that cause the simulator $\mathbf{S}$ to output $\perp$:

1(too many new classes): There are more than $(k/128)^T/16$ new classes of sessions initiated by V^* within the invocation of Simulate at the top level T.

2(unfortunate message): Simulate at some level of the recursion receives a unfortunate message with respect to the view of V^*.

3(false witness): Simulate at the top level T of the recursion obtains a *false* witness during some executions of stage Iteration of some session, i.e., it obtains the witness that it uses to execute the stage Setup in this session (class), rather than the preimage of the function value sent in V^* first step.

4(getting *stuck*). Simulate at the top level T of the recursion gets stuck on some session, i.e., it does not get any witness from executions of stage Iteration of any session in the same class to which the current session belongs.

Note that the total number of sessions (hence the total number of classes) initiated by V^* within a run of $\mathsf{Simulate}(T, (x_1, \cdots, x_{poly}), \emptyset, \emptyset)$ is at most K, and $K < K^2/16 < (k/128)^T/16$ for $K > 16$. Thus, we have

Fact. Event 1 will never happen.

The following lemmas say the latter three events occur with only negligible probability, this ends the analysis of our simulator $\mathbf{S}$.

Lemma 2. Event 2 occurs only with negligible probability.)

Proof. We start with a proof for unfortunate message of type 1.

Claim 2.1. The probability that $\mathbf{S}$ receives an unfortunate message of type 1 *with respect to the view of* V^* is negligible.

Proof. We order V^*'s steps according to the order of appearance in the whole simulation. Assume that in the s^{th} step of V^*, $\mathbf{S}$ receives an unfortunate message of type 1 with respect to the view of V^* with probability p.

Consider the following non-uniform hybrid simulator HybridS.

HybridS. Given all witnesses $(w_1, \cdots, w_{poly})$ for all common inputs $(x_1, \cdots, x_{poly})$ as an auxiliary input, HybridS acts exactly as $\mathbf{S}$ except that, for every i and every k, whenever $\mathbf{S}$ runs the non-black-box simulator $\mathsf{Sim}_{\mathsf{KID}}^t$ to carry out the subprotocol $\mathsf{PZK}_{\mathsf{KInstD}}$ of the special-purpose $\mathsf{IDWIAOK}_v^i$ in i^{th} iteration of session k, HybridS uses the witness for the relevant common input to carry out this subprotocol. (Recall that in this subprotcol of the special-purpose $\mathsf{IDWIAOK}_v^i$, the prover proves to verifier an OR statement: he knows the randomness that used in committing the current challenge in a previous step or $x \in L$.)

We will prove that, with the same probability p, HybridS receives an unfortunate message with respect to the view of V^* in the s^{th} step of V^*.

We define the following events:

[25] Note that the output of $\mathbf{S}'$ does not contain any simulated subsession.

- A_i: the event that $\mathbf{S}$ ($\mathsf{HybridS}$) never outputs $\perp$ in the first i steps of V^*. We denote $Pr_{\mathbf{S}}[A_i]$ ($Pr_{\mathsf{H}}[A_i]$, resp.) the probability that A_i happens to $\mathbf{S}$ ($\mathsf{HybridS}$, resp.).
- B_i^b, $b = 1, 2, 3, 4$: B_i^1 denotes the event that $\mathbf{S}$ ($\mathsf{HybridS}$) receives an $\mathsf{unfortunate}$ message of 1 with respect to the view of V^* in the i^{th} step of V^*; B_i^2 denotes the event that that $\mathbf{S}$ ($\mathsf{HybridS}$) receives an $\mathsf{unfortunate}$ message of 2 with respect to the view of V^* in the i^{th} step of V^*; B_i^3 denotes the event that that $\mathbf{S}$ ($\mathsf{HybridS}$) obtains an false witness in the i^{th} step of V^*; B_i^4 denotes the event that that $\mathbf{S}$ ($\mathsf{HybridS}$) gets stuck in the i^{th} step of V^*. We define $Pr_{\mathbf{S}}[B_i^b]$ and $Pr_{\mathsf{H}}[B_i^b]$ in the same way as above.

  Note that $\overline{\bigcup_{1\leq b\leq 4, j\leq i} B_j^b} = A_i$

We show that, for every b and i, $Pr_{\mathbf{S}}[B_i^b] = Pr_{\mathsf{H}}[B_i^b]$ by induction on i. Obviously, $Pr_{\mathbf{S}}[B_1^b] = Pr_{\mathsf{H}}[B_1^b]$. Assume that for all $i \leq s$ and all b, $Pr_{\mathbf{S}}[B_i^b] = Pr_{\mathsf{H}}[B_i^b]$ holds. Note that this implies $Pr_{\mathbf{S}}[A_s] = Pr_{\mathsf{H}}[A_s]$.

We now consider the case $i = s + 1$. We order the threads according to the order of appearance, and suppose that the $(s+1)^{th}$ step message of V^* sent in thread m.

Thinking of $\mathsf{HybridS}$ as honest prover in those subsessions that are simulated by $\mathbf{S}$, and $\mathbf{S}$ as the simulator for these sessions. So, if A_s happens to both $\mathbf{S}$ and $\mathsf{HybridS}$, by the **Convention** (the simulator $\mathbf{S}$ and $\mathsf{HybridS}$ use truly random and independent coins in the abovementioned subsessions), we have that the **Conditions for PZK** (see section 3.2) hold. Note also that the commitment scheme used for committing the challenge message in $\mathsf{IDWIAOK}_v^i$ are *perfect hiding*, thus we can conclude that the first thread produced by $\mathbf{S}$ and the first thread simulated by $\mathsf{HybridS}$ are identical. Furthermore, the (random) tables $\mathcal{S}$ and $\mathcal{C}$ maintained by $\mathbf{S}$ and those maintained by $\mathsf{HybridS}$ at the beginning of thread 2 are identical as long as the first threads are identical. Repeating the same reasoning, we have that, if A_s happens to both $\mathbf{S}$ and $\mathsf{HybridS}$, then the thread m before the $(s+1)^{th}$ step of V^* produced $\mathsf{HybridS}$ and the one simulated by $\mathbf{S}$ are identical. Thus, we have $Pr_{\mathbf{S}}[B_{s+1}^b|A_s] = Pr_{\mathsf{H}}[B_{s+1}^b|A_s]$ for every b.

Observe that $Pr_{\mathbf{S}}[B_{s+1}^b|\overline{A_s}] = Pr_{\mathsf{H}}[B_{s+1}^b|\overline{A_s}] = 0$. Hence, we have $Pr_{\mathbf{S}}[B_{s+1}^b] = Pr_{\mathbf{S}}[B_{s+1}^b|A_s] \cdot Pr_{\mathbf{S}}[A_s]$, and $Pr_{\mathsf{H}}[B_{s+1}^b] = Pr_{\mathsf{H}}[B_{s+1}^b|A_s] \cdot Pr_{\mathsf{H}}[A_s]$. As showed above, $Pr_{\mathbf{S}}[B_{s+1}^b|A_s] = Pr_{\mathsf{H}}[B_{s+1}^b|A_s]$, and by the hypothesis on $i = s$ ($Pr_{\mathbf{S}}[A_s] = Pr_{\mathsf{H}}[A_s]$), we conclude that $Pr_{\mathbf{S}}[B_i^b] = Pr_{\mathsf{H}}[B_i^b]$. This complete the induction, and in particular, $Pr_{\mathbf{S}}[B_{s+1}^1] = Pr_{\mathsf{H}}[B_{s+1}^1]$.

We now draw the conclusion that if $\mathbf{S}$ ($\mathsf{Simulate}$) receives an $\mathsf{unfortunate}$ message of type 1 with respect to the view of V^* during the simulation with probability p, then, with the same probability p, $\mathsf{HybridS}$ receives an $\mathsf{unfortunate}$ message with respect to the view of V^* during its interaction with V^*.

We now show p is negligible. Consider the following procedures.

$\mathsf{HybridS}^1$. Given all witnesses $(w_1, \cdots, w_{poly})$ for all common inputs $(x_1, \cdots, x_{poly})$ as an auxiliary input, $\mathsf{HybridS}^1$ acts exactly as $\mathsf{HybridS}$ except that, for every $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ and $\mathsf{IDWIAOK}_p^{\mathsf{M}}$, $\mathsf{HybridS}^1$ uses the witness for the relevant common input to carry out this subprotocol.

$\mathsf{HybridS}^2$. Given all witnesses $(w_1, \cdots, w_{poly})$ for all common inputs $(x_1, \cdots, x_{poly})$ as an auxiliary input, $\mathsf{HybridS}^2$ acts exactly as $\mathsf{HybridS}^1$ except that, $\mathsf{HybridS}^2$ generates a NO instance γ in its first step in every session.

It is easy to see that, the probability p_1 that $\mathsf{HybridS}^1$ receives an $\mathsf{unfortunate}$ message of type 1 with respect to the view of V^* during its simulation is negligibly close to p; Otherwise, we can construct an V^{**} (incorporating V^*) to break the resettable WI property of these $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ and $\mathsf{IDWIAOK}_p^{\mathsf{M}}$: Given all witnesses $(w_1, \cdots, w_{poly})$ for all common inputs

$(x_1, \cdots, x_{poly})$ as an auxiliary input, V^{**} invokes many independent *external* provers of the arguments $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ and $\mathsf{IDWIAOK}_p^{\mathsf{M}}$[26], simply forwards any V^*'s message belonging to these arguments to a relevant prover and replies to V^* with the response from this prover, while simulating V^*'s view (note that it is a *black-box* simulation.) using the strategy $\mathsf{HybridS}^1$ or $\mathsf{HybridS}$ in the remaining part of a session. It is easy to see that and if p_1 and p are not negligibly close, then we can setup a standard hybrid argument to break the resettable WI property of one of these arguments.

Again by hybrid argument, we can conclude that the probability p_2 that $\mathsf{HybridS}^2$ receives an unfortunate message of type 1 with respect to the view of V^* during its simulation is negligibly close to p_1 due to the pseudorandomness of the pseudorandom generator G. Note that the messages of type 1 with respect to the view of V^* are the output of InstD-VRF functions used by V^* and γ serves as the key_instance of this function. Since all γ are NO instances and therefore all these InstD-VRF functions satisfies the uniqueness, p_2 is negligible. Thus p_1 and p are negligible. □

Remark on special-purpose IDWIAOK and perfect ZK of $\mathsf{PZK}_{\mathsf{KInstD}}$. The major step in proofs of this and next lemma is to construct $\mathsf{HybridS}$, which does not do any non-black-box simulation. This important feature relies on the special-purpose IDWIAOK and allows us to handle arbitrary external messages in the analysis of our simulator. It seems that it is necessary to require perfect ZK from $\mathsf{PZK}_{\mathsf{KInstD}}$ in our case. If $\mathsf{PZK}_{\mathsf{KInstD}}$ satisfies only computational ZK, we don't know how to justify the (even computational) indistinguishability between $\mathsf{HybridS}$ and $\mathbf{S}$. Note that in $\mathbf{S}$, the non-black-box simulator at higher level commits to the code of those at lower level. It is really hard to deal with this complex situation with only computational ZK.

Remark on handling external message. We stress that we cannot apply the above reasoning on $\mathsf{HybridS}^1$ and $\mathsf{HybridS}^2$ to $\mathbf{S}$. Note that, as showed above, in order to break the resettable WI of $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ or $\mathsf{IDWIAOK}_p^{\mathsf{M}}$, or to break the pseudorandomness of G, we need to construct many external provers or make query to an external oracle (that will reply with pseudorandom bits or truly random bits), while simulating the view of V^*. In this setting, the simulator needs to handle many (cannot be bounded by any fixed polynomial) external messages, however, our simulator $\mathbf{S}$ cannot do this because the current non-black-box technique requires that there is a priori bound on the total length of all external messages. This observation holds in **Claim 2.2** and **Lemma 3**.

As for unfortunate message of type 2, we prove the following stronger claim to give a close look at the non-malleable issue.

Claim 2.2. The probability that $\mathbf{S}$ an unfortunate message of type 2 *with respect to the view of* $\mathbf{S}$ is negligible.

Consider the non-uniform hybrid simulator $\mathsf{HybridS}$ above. Let p be the probability that $\mathbf{S}$ receives an unfortunate message of type 2 *with respect to the view of* $\mathbf{S}$.

We let $B_s'^2$ denote the event that that $\mathbf{S}$ ($\mathsf{HybridS}$) receives an unfortunate message of 2 with respect to the view of $\mathbf{S}$ ($\mathsf{HybridS}$) in the s^{th} step of V^*, and let A_s as defined in **Claim 2.1**. As showed in **Claim 2.1**, we have $Pr_{\mathbf{S}}[A_s] = Pr_{\mathsf{H}}[A_s]$ for any i. Using exactly the same induction on i as in **Claim 2.1**, we can conclude that $p = Pr_{\mathbf{S}}[B_s'^2] = Pr_{\mathsf{H}}[B_s'^2]$ for all s, i.e., with the same probability p, the procedure $\mathsf{HybridS}$ receives an unfortunate message of type 2 *with respect to the view of* $\mathbf{S}$during its simulation.

We construct the following procedure to show that p is negligible.

[26] We assume these provers are provided with two witness sequences. For the argument $\mathsf{IDWIAOK}_p^{\mathsf{M}}$, we have V^{**} feed it with the witness α (such that $\beta = f(\alpha)$) once V^{**} extracts from V^*.

HybridS^3. Given all witnesses $(w_1, \cdots, w_{poly})$ for all common inputs $(x_1, \cdots, x_{poly})$ as an auxiliary input, HybridS^3 acts exactly as HybridS except that, in every IDWIAOK$_p^{\mathsf{S}}$ and IDWIAOK$_p^{\mathsf{M}}$, whenever HybridS applies his InstD-VRF function to generate the next message, HybridS^3 chooses a truly random string (complying with the consistence restriction) and uses the witness for the common input (that serves as the key_instance for his InstD-VRF) to prove the correctness of this random string via a ZAP.

Observe that, due to the pseudorandomness of the InstD-VRF on YES instance x, with probability negligibly close to p, HybridS^3 receives an unfortunate message of type 2 *with respect to the view of* **S** during its simulation. Suppose that this happens in two sessions with the same commitment c^e (to challenge e) in which V^* convinced HybridS^3 to accept the two different challenges e and e' via the underlying PZK$_{\mathsf{KInstD}}$ in IDWIAOK$_p^{\mathsf{S}}$ (or IDWIAOK$_p^{\mathsf{M}}$).

We now construct an cheating prover P^* (incorporating a slightly modified HybridS^3 and V^*) of the underlying simplified Pass-Rosen protocol. P^* has HybridS^3 return to the point where e is just sent, and guesses a session[27] for which V^* will succeed in proving the correctness of e (note that the probability that this guess is right is non-negligible, say p'), and forward V^*'s message in this session to an external verifier V of the simplified Pass-Rosen protocol; For every V's message (that is truly random string), P^* uses the witness for the common input (that serves as the key_instance prove the "correctness" of this message via a ZAP; For every V^*'s message sent to HybridS^3 in this session, P^* forwards it to V; For any other sessions, HybridS^3 remains unchanged.

It is clear that the V^*'s view in the above experiment is identical to that in an interaction with HybridS^3. Thus, P^* will succeed to convince V to accept e with probability negligibly close to pp'. By argument of knowledge property of the simplified Pass-Rosen protocol, we can extract an s such that $c^e = Com(e, s)$ with probability pp'; Note that when we have HybridS^3 return to the point where e' is just sent and have P^* do the above experiment again, we can extract another s' such that $c^e = Com(e', s')$ with probability pp''. Thus if p is non-negligible (note that p' is non-negligible), we break the binding property of the commitment scheme Com with non-negligible probability. □

Remark. Observe that this claim (and the next lemma) essentially states that our protocol is immune to the non-malleable attack from the malicious resetting verifier V^*: V^* does not learn how to simulate a correctness proof for an "incorrect" challenge from our simulator. □

Lemma 3. Event 3 occurs only with negligible probability.

Proof. We have showed in **Claim 2.1** that $Pr_{\mathbf{S}}[B_s^3] = Pr_{\mathsf{H}}[B_s^3]$ for all s, that is, if event 3 occurs with non-negligible probability p, then it occurs on HybridS with the same probability p.

Using the same reasoning in proof of **Claim 2.1**, we have the probability p_1 that event 3 occurs on HybridS^1 is negligibly close to p due to the resettability WI property of the argument IDWIAOK$_p^{\mathsf{S}}$ or IDWIAOK$_p^{\mathsf{M}}$, and the probability p_2 that event 3 occurs on HybridS^2 is negligibly close to p_1 due to that the pseudorandomness of G. However, there is *no* "false" witness (more precisely, there exists no δ such that $\gamma = G(\delta)$ except with exponentially small probability) in HybridS^2. This leads to a contradiction that p is negligible. □

Lemma 4. Event 4 occurs only with negligible probability.

We will prove this lemma in next section.

[27] Here we simply assume that V^* never repeats its query, i.e., it never sends the same subsession-prefix with respect to the underlying simplified Pass-Rosen protocol.

C.3 Proof of Lemma 4

In this section we mainly prove that at any level $t \geq 1$ of the recursion, Simulate will get stuck on a *new* session (class) only with negligible probability. This leads to the conclusion of **Lemma** 4 because there is no *old* session (class) for the initial state of the run of Simulate at level T.

We refer to a *state* of $\mathbf{S}$ at some point as the configuration of $\mathbf{S}$ (including the current history) at the current point, *excluding* the randomness used in its run after this point. An execution of the simulator consists of a sequence of consecutive states. Given randomness r, we denote an invocation of Simulate with randomness r by *invocation* r, and say that a class of sessions is *complete* in *invocation* r if some session in this class reaches the end of the stage Iteration before the end of this invocation. For more precise statement, we rephrase the *new* (*old*) in terms of *state*: a class of sessions is said to be *new* for a state σ if the verifier's first message of this class does not appear in the history h specified in σ; Otherwise, it is called *old* for state σ. We also say class of sessions is *new for an invocation* of Simulate if this class is new for the start state of this invocation.

Now let's consider a random invocation r of Simulate starting with initial state σ_{init} at level $t > 1$, and the j^{th} *new* (for σ_{init}) class[28] appearing in this invocation.

Let σ_i be the state at which the first message of i^{th} iteration of the j^{th} class is just reached (i.e., some session in this class reaches the challenge message of $i - 1^{th}$ iteration.) during this invocation r of Simulate at level t. Note also that at state σ_i Solve at level $t-1$ is invoked. Let $r = r_{\mathsf{init}} \parallel r_1 \parallel \cdots \parallel r_i \parallel \cdots$, where r_i (resp., r_{init}) is the randomness used in the execution segment from state σ_i (resp., σ_{init}) to σ_{i+1} (resp., σ_1) of this invocation r of Simulate at level t (excluding the randomness used in invocations of Solve at level $t-1$.), and all these segments of randomness are chosen *independently* (by **Convention**). We call such a segment with randomness r_i execution r_i.

Our mission is to bound the probability that this invocation r of Simulate at level t gets stuck on the j^{th} new class for any $t \geq 1$. This yields **Lemma** 4 immediately. First, we introduce some notations.

- Successful attempt. Let the waiting list for which the above invocation r of Simulate at level t is initiated be $\mathcal{Q}$. We call an attempt within an invocation of Solve with start state σ_i at level $t-1$ successful if at the end of this attempt, either one class listed in $\mathcal{Q}$ or the current j^{th} class is solved; Otherwise, we say this attempt fails; We say an invocation of Solve successful if an attempt therein is successful.
- The random variables $h^{\mathsf{sim}}_{\sigma_i,r_i}$ and $h^{\mathsf{att}}_{\sigma_i,r'_i}$. For an execution r_i of Simulate from state σ_i to state σ_{i+1}[29] at level t, we denote by $h^{\mathsf{sim}}_{\sigma_i,r_i}$ the segment of transcript of this execution[30]; We denote by $h^{\mathsf{att}}_{\sigma_i,r'_i}$ the transcript of a *single* attempt using uniformly chosen randomness r'_i within an invocation of Solve starting with state σ_i at level $t-1$.
- The probability $\lambda^{\mathsf{att}}_{\sigma_i}$ and $\lambda^{\mathsf{sim}}_{\sigma_i}$. Let $\mathcal{Q}$ as above. For a random execution r_i of Simulate from state σ_i to state σ_{i+1} at level t, we denote by $\lambda^{\mathsf{sim}}_{\sigma_i}$ the probability that Simulate during this random execution fails to solve any class in the waiting list of $\mathcal{Q}$ and the j^{th} new class does not reach the last verifier message of i^{th} iteration (i.e., no session in this class reaches the last verifier message of i-th iteration) due to V^* halts or Simulate gets

[28] As mentioned before, we order the new and complete classes in this invocation according to the order of appearance.

[29] Here by "to state σ_{i+1}" we mean that the execution r_i does not go past the state σ_{i+1}, rather than ending with state σ_{i+1}. (Note that it is possible Simulate returns before reaching σ_{i+1}.)

[30]

stuck on an *old* class for σ_i, and denote by $\lambda^{\mathsf{att}}_{\sigma_i}$ the probability that a *single* attempt within an invocation of Solve with start state σ_i at level $t-1$ fails due to V^* halts or this attempt gets stuck on an *old* class for σ_i.

– The random variables $\mu^{\mathsf{sim}}_{\sigma_i,r_i}$, $\mu^{\mathsf{att}}_{\sigma_i,r'_i}$. We let $\mu^{\mathsf{sim}}_{\sigma_i,r_i}$ denote the number of *new* and *complete* classes recorded in $h^{\mathsf{sim}}_{\sigma_i,r_i}$, and let $\mu^{\mathsf{att}}_{\sigma_i,r'_i}$ denote the number of *new* and *complete* classes recorded in $h^{\mathsf{att}}_{\sigma_i,r'_i}$.

Our mission is accomplished as follows. We start with the observation that if we impose the same restriction on the number of new classes allowed in the (segment) execution r_i of Simulate at level t as in an attempt within an invocation of Solve starting with state σ_i at level $t-1$, then $h^{\mathsf{sim}}_{\sigma_i,r_i}$ and $h^{\mathsf{att}}_{\sigma_i,r'_i}$ are indistinguishable (**Claim 4.1**), $|\lambda^{\mathsf{sim}}_{\sigma_i} - \lambda^{\mathsf{att}}_{\sigma_i}|$ is negligible and $\mu^{\mathsf{sim}}_{\sigma_i,r_i}$ is indistinguishable from $\mu^{\mathsf{att}}_{\sigma_i,r'_i}$. We then prove that, if for many i, the probability that $\lambda^{\mathsf{att}}_{\sigma_i}$ is high or $\mu^{\mathsf{att}}_{\sigma_i,r'_i}$ is large is high, then the j^{th} new class (for state σ_{init}) reaches the end of stage Iteration only with negligible probability (**Claim 4.2** and **Claim 4.3**). This enable us to conclude that this invocation of Simulate gets stuck on j^{th} new class only with negligible probability (**Claim 4.4**), and therefore yields **Lemma 4.3** immediately (**Claim 4.5**).

Let m_i be the maximal number of new classes for state σ_i allowed to be initiated in the (segment) execution r_i of Simulate at level t, which is inherited from the bound (which is $(k/128)^t/16$) on the number of new classes for state σ_{init} allowed to be initiated in the current invocation of Simulate.

Set $l_i = min\{m_i, (k/128)^{t-1}/16\}$. (Recall that $(k/128)^{t-1}/16$ is the upper bound on the number of new classes for state σ_i allowed to be initiated in an attempt within a call of Solve starting with state σ_i at level $t-1$.) We denote by $h^{\mathsf{sim}}_{\sigma_i,r_i}|_{l_i}$ ($h^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i}$) the prefix of $h^{\mathsf{sim}}_{\sigma_i,r_i}$ ($h^{\mathsf{att}}_{\sigma_i,r'_i}$, resp.) truncated at the point where the $l+1^{th}$ new class for state σ_i appears. We define $\lambda^{\mathsf{att}}_{\sigma_i}|_{l_i}$, $\lambda^{\mathsf{sim}}_{\sigma_i}|_{l_i}$, $\mu^{\mathsf{sim}}_{\sigma_i,r_i}|_{l_i}$, and $\mu^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i}$ in the obvious way.

In what follows, We denote by $neg(n)$ a negligible function.

Claim 4.1. For any state σ_i, the following hold.

1. $h^{\mathsf{sim}}_{\sigma_i,r_i}|_{l_i}$ and $h^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i}$ are indistinguishable.
2. $|\lambda^{\mathsf{sim}}_{\sigma_i}|_{l_i} - \lambda^{\mathsf{att}}_{\sigma_i}|_{l_i}| < neg(n)$.
3. $\mu^{\mathsf{sim}}_{\sigma_i,r_i}|_{l_i}$ and $\mu^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i}$ are indistinguishable.

Proof. Note that item 2 and 3 follow immediately from item 1. Therefore, it is sufficient to prove that item 1 holds.

We first stress that, given a state σ_i, the simulated history before reaching this state is also given to V^* for V^* to continue. Hence, we need to take this prior history into account when proving this claim.

Fix the simulated history prior to the state σ_i, denoted h^{σ_i}. We claim that the pair $(h^{\sigma_i}, h^{\mathsf{sim}}_{\sigma_i,r_i}|_{l_i})$ is indistinguishable form $(h^{\sigma_i}, h^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i})$. This implies claim 1.

Observe that the only difference between $(h^{\sigma_i}, h^{\mathsf{sim}}_{\sigma_i,r_i}|_{l_i})$ and $(h^{\sigma_i}, h^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i})$ is that, for the i^{th} iteration of the j^{th} new class in $(h^{\sigma_i}, h^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i})$, the challenge of this iteration is random (therefore is unlikely to be correct) and the proof that this challenge is correct is been simulated, while in $(h^{\sigma_i}, h^{\mathsf{sim}}_{\sigma_i,r_i}|_{l_i})$, the messages in this iteration are produced by following the honest prover strategy. (Notice that at any level of recursion, Solve never updates the execution history.) We also observe that, in the history $(h^{\sigma_i}, h^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i})$, there are at most T

classes (with respect to the global protocol) that have each only one iteration in which the subexecutions of the underlying argument $\mathsf{PZK}_{\mathsf{KInstD}}$ have been simulated. Since all these subexecutions of argument $\mathsf{PZK}_{\mathsf{KInstD}}$ belonging to a single class of sessions of the global protocol form a *sigle* class of sessions with respect to the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument[31], we conclude that there are at most T classes of subsessions with respect to the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument that have been simulated in $(h^{\sigma_i}, h^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i})$.

We now use the following procedure $\mathsf{HybirdSolve}$ (that will be called at state σ_i) to prove that $(h^{\sigma_i}, h^{\mathsf{sim}}_{\sigma_i,r_i}|_{l_i})$ and $(h^{\sigma_i}, h^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i})$ are indistinguishable.

$\mathsf{HybridSolve}(t-1, h^{\sigma_i}, \mathcal{Q}, \mathcal{C}_t, C^{(l,m)}_{\mathsf{f\text{-}msg}}, i)$: act exactly as $\mathsf{Solve}(t-1, h^{\sigma_i}, \mathcal{Q}, \mathcal{C}_t, C^{(l,m)}_{\mathsf{f\text{-}msg}}, i)$, except that for the i^{th} iteration of class $C^{(l,m)}_{\mathsf{f\text{-}msg}}$, $\mathsf{HybridSolve}$ computes the challenge message e_i *honestly* in each attempt.

Let $h^{\mathsf{hatt}}_{\sigma_i,r''_i}$ denote the transcript of a single attempt using randomness r''_i within invocation of $\mathsf{HybridSolve}(t-1, h^{\sigma}, \mathcal{Q}, \mathcal{C}_t, C^{(l,m)}_{\mathsf{f\text{-}msg}}, i)$. We define $h^{\mathsf{hatt}}_{\sigma_i,r''_i}|_{l_i}$ in the same way as $h^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i}$. It is easy to see:

1. $(h^{\sigma_i}, h^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i})$ and $(h^{\sigma_i}, h^{\mathsf{hatt}}_{\sigma_i,r''_i}|_{l_i})$ are identical due to the perfect hiding property of the commitment scheme Com_v used to produce c^e_i in the prover's seconde message.
2. If event 2 does not occur, $(h^{\sigma_i}, h^{\mathsf{hatt}}_{\sigma_i,r''_i}|_{l_i})$ and $(h^{\sigma_i}, h^{\mathsf{sim}}_{\sigma_i,r_i}|_{l_i})$ are indistinguishable due to $\log n$-class bounded resettable zero knowledge property of $\mathsf{PZK}_{\mathsf{KInstD}}$ argument. Note that the total number of classes of subsessions of the underlying argument $\mathsf{PZK}_{\mathsf{KInstD}}$ is at most T as long as event 2 does not occur, and $T < \log n$.

Note also that event 2 occurs with only negligible probability. This leads to the conclusion that, for any state σ_i, given the history h^{σ_i} prior to σ_i, $h^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i}$ and $h^{\mathsf{sim}}_{\sigma_i,r_i}|_{l_i}$ are indistinguishable. □

Claim 4.2. The probability that the following two events that occur *simultaneously* is exponentially small.

1. $\lambda^{\mathsf{att}}_{\sigma_i} > 7/8$ for at least $k/2$ states σ_i;
2. the j^{th} new (for state σ_{init}) class reaches the end of its stage $\mathsf{Iteration}$ (i.e., no session belonging to this class reaches the end of stage $\mathsf{Iteration}$) before the end of the random invocation r of $\mathsf{Simulate}$.

Proof. Let I be the set of these iterations i of the j^{th} new class for which $\lambda^{\mathsf{att}}_{\sigma_i} > 7/8$ holds. We distinguish the following two cases depending on whether $m_i \geq (k/128)^{t-1}/16$ or not, and prove this claim by case analysis.

Case 1: $m_i \geq (k/128)^{t-1}/16$ for all $i \in I$. In this case, we have $l_i = (k/128)^{t-1}/16$, $\lambda^{\mathsf{att}}_{\sigma_i} = \lambda^{\mathsf{att}}|_{l_i}$, and $\lambda^{\mathsf{sim}}_{\sigma_i} \geq \lambda^{\mathsf{sim}}_{\sigma_i}|_{l_i}$. By **Claim 4.1**, the occurrence of event 1 implies that, for at least $k/2$ iterations i of the j^{th} class, $\lambda^{\mathsf{sim}}_{\sigma_i} \geq \lambda^{\mathsf{att}}_{\sigma_i} - neg(n) > 7/8 - neg(n)$, in other words, for each of these i, the probability that j^{th} class completes its i^{th} iteration in the simulation thread at level t is less than $1/8 + neg(n)$.

[31] As mentioned earlier, this class of subsessions with respect to the $\mathsf{PZK}_{\mathsf{KInstD}}$ argument is specified by the incarnation of prover in the global protocol (which also play the role of prover in the underlying argument $\mathsf{PZK}_{\mathsf{KInstD}}$) and the first message of this $\mathsf{PZK}_{\mathsf{KInstD}}$ argument sent in the verifier's first message of the global protocol.

Obviously, The probability that event 1 and 2 occur simultaneously is less than the probability that the j^{th} class completes all these $|I| > k/2$ iterations conditional on the occurrence of event 1 in the simulation thread at level t, and the latter is less than (note that those r_i are chosen independently.)

$$\begin{aligned}
&\binom{k}{k/2}[(1-7/8+neg(n))]^{k/2}\\
<&\binom{k}{k/2}[(1-7/8)]^{k/2}\\
\leq&[ke/(k/2)]^{k/2}[(1-7/8+neg(n))]^{k/2}\\
<&(e/4+neg(n))^{k/2}.
\end{aligned}$$

Case 2: $m_i < (k/128)^{t-1}/16$ for some $i \in I$. Let N ($N \subseteq I$) be the set of these iterations i. In this case, we have $l_i = m_i < (k/128)^{t-1}/16$. Set $m'_i = (k/128)^{t-1}/16$. For those $i \in N$, we relax the restriction on number of *new* (for σ_i) classes allowed to be initiated after state σ_i in the simulation thread at level t by extending its upper bound m_i to m'_i. We call the simulation thread at level t done by the original simulator the *normal* simulation thread, and call the one with the above extension the *extended* simulation thread.

For those $i \in N$, we consider the following two events:

A: the j^{th} class completes its i^{th} iteration in the *normal* simulation thread at level t.

B: the j^{th} class completes its i^{th} iteration in the *extended* simulation thread at level t.

It is easy to verify that $Pr[A] \leq Pr[B]$, and $Pr[B] \leq 1-\lambda^{\mathsf{sim}}_{\sigma_i}|_{m'_i}$. By **Claim 4.1** (Notice also that $m'_i = (k/128)^{t-1}/16$), the occurrence of event 1 implies

$$\lambda^{\mathsf{sim}}_{\sigma_i}|_{m'_i} > \lambda^{\mathsf{att}}_{\sigma_i}|_{m'_i} - neg(n) = \lambda^{\mathsf{att}}_{\sigma_i} - neg(n) > 7/8 - neg(n)$$

Thus, the probability that A occurs conditional on occurrence of event 1 is less than $1/8 + neg(n)$.

For those $i \in I \setminus N$, as showed in case 1, the upper bound $1/8 + neg(n)$ on the above conditional probability holds.

In sum, for every $i \in I$, the probability that the j^{th} class completes its i^{th} iteration conditional on occurrence of event 1 in the simulation thread at level t is less than $1/8 + neg(n)$. Applying the same reasoning of case 1, we conclude that this claim holds for **Case** 2.

Claim 4.3. The probability that the following two events that occur *simultaneously* is exponentially small.

1. $Pr[\mu^{\mathsf{att}}_{\sigma_i,r'_i} < (k/128)^{t-1}/16] < 15/16$ holds for at least $3k/8$ states σ_i;
2. the j^{th} new (for state σ_{init}) class reaches the end of its stage Iteration before the end of the random invocation r of Simulate.

Proof. First note that for any invocation r of Simulate with start state σ_{init} at level t, there are at most $k/128$ iterations i of the j^{th} new class such that $\mu^{\mathsf{sim}}_{\sigma_i,r_i} > (k/128)^{t-1}/16$ for each of these i, otherwise there are more than $(k/128)^{t-1}/16 \times k/128 = (k/128)^t/16$ new (for state σ_{init}, observe that classes new for σ_i is also new for σ_{init}) classes during this invocation r, which contradicts the upper bound on the number of new class allowed to be initiated in an invocation of Simulate at level t.

Let I be the set of these iterations i of the j^{th} new class for which $Pr[\mu^{\mathsf{att}}_{\sigma_i,r'_i} < (k/128)^{t-1}/16] < 15/16$ holds. Again, we prove this claim by case analysis.

Case 1: $m_i \geq (k/128)^{t-1}/16$ for all $i \in I$. In this case, we have $l_i = (k/128)^{t-1}/16$, $\mu^{\mathsf{att}}_{\sigma_i,r'_i} = \mu^{\mathsf{att}}_{\sigma_i,r'_i}|_{l_i}$, By **Claim 4.1**, the occurrence of event 1 implies that $Pr[\mu^{\mathsf{sim}}_{\sigma_i,r_i} < (k/128)^{t-1}/16] < 15/16 + neg(n)$ holds for every $i \in I \setminus N$. We define

$$X_i = \begin{cases} 0 & \text{if } \mu^{\mathsf{sim}}_{\sigma_i,r_i} < (k/128)^{t-1}/16 \\ 1 & \text{otherwise} \end{cases}$$

Let $S = \sum_{i \in I} X_i$. Note that those X_i are independent, and $Pr[X_i = 1] > 1/16 - neg(n)$ for those $i \in I \setminus N$. Thus, conditioning on occurrence of event 1, we have $E(S) > 1/16 \times 3k/8 - neg(n) = 3k/128 - neg(n)$.

However, as mentioned above, for the j^{th} class to complete its stage Iteration, S must be less than $k/128$ due to the upper bound on the number of new class allowed to be initiated in an invocation of Simulate at level t. Hence, the probability that the j^{th} class reaches the end of stage Iteration is less than the probability that S is less than $k/128$. Note that $Pr[S < k/128] < Pr[|S - E(S)| > 3k/128 - k/128 - neg(n)]$, and by Chernoff inequality, we have

$$Pr[|S - E(S)| > 3k/128 - k/128 - neg(n) > k/128] < Pr[|S - E(S)| > k/128] < e^{-ck}$$

as desired, where c is a constant.

Case 2: $m_i < (k/128)^{t-1}/16$ for some $i \in I$. Let N ($N \subseteq I$) be the set of these iterations i. Note that for $i \in N$, $l_i = m_i < (k/128)^{t-1}/16$. As in **Claim 4.2**, we set $m'_i = (k/128)^{t-1}/16$, and relax the restriction on number of *new* classes allowed to be initiated after state σ_i in the simulation thread at level t by extending its upper bound m_i to m'_i for every $i \in N$.

It is easy to see that the probability that the j^{th} class completes its stage Iteration in the *normal* simulation thread at level t is less than the probability that the j^{th} class completes its stage Iteration in the *extended* simulation thread at level t. Also note that, in the *extended* simulation thread, $m'_i \geq (k/128)^{t-1}/16$ for all $i \in I$, and the analysis of case 1 shows the probability that the j^{th} class completes its stage Iteration in the *extended* simulation thread at level t is exponentially small. Thus we arrive at the conclusion of **Claim 4.3** in this case.

Claim 4.4. For any $t \geq 1$, $1 \leq j \leq K$, a random invocation of Simulate starting from σ_{init} at level t gets stuck on the j^{th} new class only with negligible probability.

Proof. We prove this claim by induction on t.

When $t = 1$. Let σ_i be defined as above. By **Claim 4.2** and **Claim 4.3**, we have that, except with exponentially small probability, there are at least $k/2 + 5k/8 - k = k/8$ states σ_i (in fact, a single one suffices), such that:

- $\lambda^{\mathsf{att}}_{\sigma_i} < 7/8$, and
- $Pr[\mu^{\mathsf{att}}_{\sigma_i,r'_i} > (k/128)^{1-1}/16 = 0] < 1/16$. (Note that $\mu^{\mathsf{att}}_{\sigma_i,r'_i}$ is an integer-valued random variable.)

Fix a state σ_i satisfying the above two conditions. Let the waiting list for which Simulate at level 1 is called be $\mathcal{Q}$. Recall that, at the state σ_i, the i^{th} iteration of the j^{th} new class is appended to Q, and this forms a new waiting list, denoted $\mathcal{Q}_i$, for which Solve at level 0 is invoked.

We analyze the probability that a single attempt within a call of Solve at level 0 with start state σ_i fails to solve any class listed on $\mathcal{Q}_i$. Observe that there are two events that cause an attempt within a call of Solve fail:

1. Within this attempt, Simulate at level 0 does not solve any class listed on $\mathcal{Q}_i$ due to V^* halts or it gets *stuck* on an *old* class for σ_i;
2. Within this attempt, there are at least $(k/128)^{1-1}/16 + 1 = 1$ new (for state σ_i) class being initiated;

Note that by property 1 of state σ_i mentioned above, $Pr[event\ 1\ ocurs] = \lambda_{\sigma_i}^{\mathsf{att}} < 7/8$; By property 2 of state σ_i, $Pr[event\ 2\ ocurs] < 1/16$. Thus, except with probability 15/16, a random single attempt within a call of Solve at level 0 with start state σ_i will be successful, and this means the expectation of the number of failure attempts within a call of Solve is $24K \cdot 15/16 = 22.5K$. By Chernoff Inequality, we have that the probability that more than $23K$ attempts within this call of Solve fails is less than e^{-cK} for some constant c. In other words, there are at least K attempts (in fact, a single one suffices) within this call of Solve will be successful except with exponentially small probability.

We have proved that the invocation of Solve at the state σ_i will succeed to solve a class appeared on $\mathcal{Q}_i$ except with exponentially small probability. This means a random execution of Simulate at level 1 does not get stuck on j^{th} new class except with negligible probability[32],

Hypothesis: Claim 4.4 holds for $t \leq t'$.

When $t = t' + 1$. We prove **Claim 4.5** holds for $t = t' + 1$ in a similar way to that in case $t = 1$. First, for at least $k/2 + 5k/8 - k = k/8$ states σ_i, the **Claim 4.2** and **Claim 4.3** guarantee (except with exponentially small probabitlity) the following two properties of these σ_i:

- $\lambda_{\sigma_i}^{\mathsf{att}} < 7/8$, and
- $Pr[\mu_{\sigma_i, r_i'}^{\mathsf{att}} > (k/128)^{t'}/16] < 1/16$.

Fix such a state σ_i. Let $\mathcal{Q}$ and $\mathcal{Q}_i$ be as above. Observe that there are only three events that cause an attempt within a call of Solve to fail[33]:

1. Within this attempt, Simulate at level t' does not solve any class listed on $\mathcal{Q}_i$ due to that V^* halts or it gets *stuck* on an *old* class for state σ_i;
2. Within this attempt, there are at least $(k/128)^{t'}/16 + 1$ new (for state σ_i) class being initiated;
3. This attempt gets stuck on a new (for state σ_i) class.

Again, the property 1 and property 2 of state σ_i mentioned above guarantee $Pr[event\ 1\ ocurs] = \lambda_{\sigma_i}^{\mathsf{att}} < 7/8$ and $Pr[event\ 2\ ocurs] < 1/16$; By the Hypothesis, we have that the third event happens on a specific class only with negligible probability. Since there are at most K class being initiated in this attempt, we conclude that $Pr[event\ 3\ ocurs] \leq K \cdot neg(n) = neg(n)$. Thus, except with probability $15/16 + neg(n)$, a random single attempt within a call of Solve at level t' with start state σ_i will be successful. Using the same reasoning of the base case when $t = 1$, we get that there are at least K attempts within this call of Solve will be

[32] Note that if the j^{th} new class is solved within this call of Solve, Simulate at level 1 does not get stuck on this class; If a class listed in $\mathcal{Q}$ (for which Simulate at level 1 is invoked) is solved within this call of Solve, then Simulate at level 1 accomplishes its mission and returns (thus does not get stuck on the j^{th} new class).

[33] Note that in the base case $t = 1$, non-occurrence of event 2 already implies non-occurrence of the event 3.

successful except with exponentially small probability, and thus draw the conclusion that a random execution of $\mathsf{Simulate}$ at level $t'+1$ gets stuck on the j^{th} new class only with negligible probability.

Now we are ready to prove the **Lemma** 4.

Claim 4.5. **S** gets stuck only with negligible probability.

Proof. Notice that the total number of sessions (hence the total number of classes) that opened in a random execution of $\mathsf{Simulate}(T,(x_1,\cdots,x_{poly}),\emptyset,\emptyset)$ is at most K, and $K < K^2/16 < (k/128)^T/16$. By **claim 4.4**, we have that a random execution of $\mathsf{Simulate}$ at level T gets stuck on a new class with probability less than $K \cdot neg(n) = neg(n)$, which is negligible. Since there are no old class for the initial state, we conclude $\mathsf{Simulate}(T,(x_1,\cdots,x_{poly}),\emptyset,\emptyset)$ does not get stuck except with negligible probability. □

D Proof of resettable-soundness of our protocol

Resettable-soundness of our protocol can be proved by showing an algorithm B that uses the power of a cheating prover P^* to invert the one-way function f.

Let Int_k denote the set of sessions having the first P^*'s first message γ_k. Assume in the real interaction, P^* sends t distinct γs. Given a challenge β (supposed to be an image under f), B plays the role of the verifier $V^{(j)}(x)$, and guesses P^* will cheat on a session belonging to Int_k. Then, for all sessions outside Int_k, B acts as an honest verifier; for sessions belonging to Int_k, B sets β to be the corresponding image of f and invokes the extractor E described in last section and extracts δ_k (such that $\gamma_k = G(\delta_k)$) from the executions of $\mathsf{IDWIAOK}_p^{\mathsf{S}}$, uses this δ_k as witness to complete stage $\mathsf{Iteration}$, then it extracts α (such that $\beta = f(\alpha)$) from the executions of $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ using the same extraction strategy E.

Intuitive reasons why proof of resettable-soundness is much simpler. Before proceeding further, we (intuitively) explain why our protocol is not symmetric (in the sense that the underlying $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ and $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ is not special) and it admits simple proof of soundness. Typically, the fact that proof of soundness is simpler than proof of zero knowledge is mainly due to that the proof of soundness just needs to focus on a single session (class of sessions) and it is somewhat easy for the reduction algorithm to play the role of honest verifier. In our case, as we have seen, there is *false witness* issue arising in the proof of resettable ZK, and handling this issue requires the resettable WI property of $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ and $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ holds even in the whole simulation process (where V^* may see many non-black-box simulation for the underlying $\mathsf{PZK}_{\mathsf{KInstD}}$ in the argument $\mathsf{special\text{-}purpose}$ $\mathsf{IDWIAOK}_v^i$). However, in the proof of resettable-soundness where $x \notin L$ is assumed, there is no false witness issue: any witness extracted by B is satisfactory.

We describe the algorithm B in figure 12.

We now turn to show that if P^* can convince the honest verifier on $x \notin L$ with non-negligible probability p, then B can find a preimage α of β under f with non-negligible probability. This breaks the one-wayness of f.

Note that the probability that P^* convinces the honest verifier in a session belonging to Int_k is p/t, where t is a polynomial. Moreover, in stage 3 of B, a single run of step 2 of E succeeds in obtaining a accepting transcript of the underlying 3-round WI of $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ with probability at least p/t, and therefore will extract δ_k with probability at least $p/t - neg(n)$. (Observe that in the stage 3, B plays the role of honest verifier.)

The invertor $\mathsf{B}(\beta)$:

1. B selects a random string for P^*, and plays the role of honest verifier.
2. B uniformly chooses k from $\{1, ..., t\}$. Throughout this inverting process, B adopts honest verifier strategy in every session in Int_j, $j \neq k$; In the execution of Int_k, B sets β to be the corresponding image of f, and play the role of honest verifier until stage Setup is first reached by a session in Int_k.
3. When one session in Int_k first reaches stage Setup, B applies the extractor E to the executions of $\mathsf{IDWIAOK}_p^{\mathsf{S}}$ with the following natural modifications:
   - In a single run of E's step 4, if a *non-target* session in Int_k reaches its stage $\mathsf{Iteration}$, B aborts the current attempt (in this case, B cannot proceed further without knowledge of δ_k.) and chooses independent randomness to make another one.
   - In a single run of E's step 4, when the underlying non-black-box simulator $\mathsf{Sim}_{\mathsf{KID}}^{\mathsf{E}} = (\mathsf{Intermed}^{\mathsf{E}}, \mathsf{Sim}_{\mathsf{B}}^{\mathsf{E}})$ is instructed to commit to hash value of a code, the corresponding subroutine $\mathsf{Sim}_{\mathsf{B}}^{\mathsf{E}}$ commits to the *joint* code of the residual procedures $\mathsf{Intermed}^{\mathsf{E}}$ and B(except the current subroutine $\mathsf{Sim}_{\mathsf{KID}}^{\mathsf{E}}$) and P^*.
4. If the above stage fails to extract a witness, B outputs $\perp$; If the corresponding δ_k (such that $\gamma_k = G(\delta_k)$) was obtained, B uses it as witness to carry out all the $\mathsf{special\text{-}purpose\ IDWIAOK}_v^i$ in Int_k until the stage $\mathsf{Mainproof}$ is first reached by a session in Int_k.
5. When one session in Int_k first reaches stage $\mathsf{Mainproof}$, B applies the extractor E to the executions of $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ with the following natural modifications:
   - In a single run of E's step 4, when the underlying non-black-box simulator $\mathsf{Sim}_{\mathsf{KID}}^{\mathsf{E}} = (\mathsf{Intermed}^{\mathsf{E}}, \mathsf{Sim}_{\mathsf{B}}^{\mathsf{E}})$ is instructed to commit to hash value of a code, the corresponding subroutine $\mathsf{Sim}_{\mathsf{B}}^{\mathsf{E}}$ commits to the *joint* code of the residual procedures $\mathsf{Intermed}^{\mathsf{E}}$ and B(except the current subroutine $\mathsf{Sim}_{\mathsf{KID}}^{\mathsf{E}}$) and P^*.
     Note that here the residual procedure B already knew δ_k (such that $\gamma_k = G(\delta_k)$) which may be used as witness to carry out some $\mathsf{special\text{-}purpose\ IDWIAOK}_v^i$ in Int_k (keep in mind that P^* is a resetting adversary).
6. If the above stage fails to extract a witness, B outputs $\perp$; If the corresponding α (such that $\beta = f(\alpha)$) was obtained, B outputs α.

Fig. 12. The invertor B.

Let p_1 be the probability that, in stage 5 of B, a single run of step 2 of E succeeds in obtaining a accepting transcript of the underlying 3-round WI of $\mathsf{IDWIAOK}_p^{\mathsf{M}}$ (i.e., the probability that E enters its step 3). Note that the only difference between a single run of E's step 2 and real interaction is that, in the former, B may use δ_k as witness to carry out some special-purpose $\mathsf{IDWIAOK}_v^i$ in Int_k, and note also that E does not do any non-black-box simulation in this step. It follows from the resettable WI property of those special-purpose $\mathsf{IDWIAOK}_v^i$ when $x \notin L$ that p_1 is at least $p/t - neg(n)$ (This can be proved in the same way as in **Claim 2.1**). Thus, we conclude that, conditioned on B entering its stage 5, E will extract α with probability at least $p/t - neg(n)$.

Thus, if if P^* can convince the honest verifier on $x \notin L$ with non-negligible probability p, then B finds a preimage α of β under f with probability at least $(p/t - neg)^2$, which is non-negligible.

Remark. It should be noted that in the analysis of stage 5 of B, it suffices to analyze a single run of E's step 2, and we do not care whether the resettable WI property of those special-purpose $\mathsf{IDWIAOK}_v^i$ when $x \notin L$ remains in the whole stage 5 (mainly due to that there is no false witness issue).

E The running time of our simulator

Recall that a run of the non-black-box simulation strategy $\mathsf{Sim}_{\mathsf{KID}}^t(r)$[34] commits to the following joint procedures Π to produce some relevant prover messages to prove that this joint code can predict the second verifier's message w.r.t Barak's protocol in the session it is currently handling. Π consists of:

1. the procedure $\mathsf{Simulate}$ at level t (except the current subroutine $\mathsf{Sim}_{\mathsf{KID}}^t(r)$) by which $\mathsf{Sim}_{\mathsf{KID}}^t(r)$ is invoked. This procedure consists of two parts:
   (a) The HONEST part of the procedure $\mathsf{Simulate}$ at level t by which $\mathsf{Sim}_{\mathsf{KID}}^t(r)$ is invoked, i.e., this $\mathsf{Simulate}$ except the subroutine $\mathsf{Sim}_{\mathsf{KID}}^t(r)$ and all subroutines Solve at level $t-1$;
   (b) The procedures Solve at level $t-1$ invoked by this $\mathsf{Simulate}$ at level t. Note that These procedure Solve may include many $\mathsf{Sim}_{\mathsf{KID}}^{t'}(r')$ at lower level $t' < t$.
2. The verifier V^*;
3. The procedure $\mathsf{Intermed}^t$, which just repeats some prover messages that our non-black-box simulators have produced earlier.

A key observation that leads the simulator $\mathbf{S}$ to run in *polynomial* time is that, Π does not include any other $\mathsf{Sim}_{\mathsf{KID}}^t(r')$ with independent randomness r' at the same level t, though it may include many $\mathsf{Sim}_{\mathsf{KID}}^{t'}(r')$ at lower level $t' < t$, and for the same reason, all these $\mathsf{Sim}_{\mathsf{KID}}^{t'}(r')$ at the *same* level $t' < t$ are *independent*: each of them uses independent randomness and *does not* commit to any code that includes any other $\mathsf{Sim}_{\mathsf{KID}}^{t'}(r')$ at the same level t'.

Assume that the number of the different sessions appearing in level t' that need $\mathsf{Sim}_{\mathsf{KID}}^t(r)$ to handle internally is s. For these s sessions, there are s *independent* $\mathsf{Sim}_{\mathsf{KID}}^{t'}(r')$s that would be invoked by s independent $\mathsf{Simulate}$ at level t' (which are already included in Π), each of them handles only one single simulated session (appearing in level t') *independently*.

As we will see later, this is crucial in proving that our simulator runs in polynomial time.

[34] Hereafter we denote $\mathsf{Sim}_{\mathsf{KID}}^t(r)$ an invocation of $\mathsf{Sim}_{\mathsf{KID}}^t$ with randomness r.

We now show that the simulator $\mathbf{S}$ runs in polynomial time.

Let running time of the honest part of $\mathsf{Simulate}$ at any level is bounded by a polynomial p_1, and the running time of V^* is a polynomial p_2. Note that the $\mathsf{Simulate}$ at level t makes at most $k(k/128)^t/16$ invocations of Solve at level $t-1$ (since there are at most $(k/128)^t/16$ new (classes of[35]) sessions during this invocation of $\mathsf{Simulate}$, each of them consists of k slot), and each Solve at level $t-1$ consists of $24K$[36] independent invocations of $\mathsf{Simulate}$ at level $t-1$ recursively. Denote the running time of $\mathsf{Simulate}$ at level t with $Time(\mathsf{Simulate}^t)$, and the running time of $\mathsf{Sim}^t_{\mathsf{KID}}$ with $Time(\mathsf{Sim}^t_{\mathsf{KID}})$, and the running time of Solve at level t with $Time(\mathsf{Solve}^t)$

Thus, we have, for $t < T$,

$$\begin{aligned} Time(\mathsf{Simulate}^t) &\leq p_1 + p_2 + k(k/128)^t/16 Time(\mathsf{Solve}^{t-1}) + Time(\mathsf{Sim}^t_{\mathsf{KID}}) \\ &\leq p_1 + p_2 + k(k/128)^t/16 \cdot 24K \cdot Time(\mathsf{Simulate}^{t-1}) + Time(\mathsf{Sim}^t_{\mathsf{KID}}) \quad (1) \end{aligned}$$

Let's bound $Time(\mathsf{Sim}^t_{\mathsf{KID}})$. We first show that the code Π committed by $\mathsf{Sim}^t_{\mathsf{KID}}(r)$ can output the same V^*'s second message of the simplified Pass-Rosen protocol (in the relevant session) in polynomial time (It is easy to verify the correctness of Π). Let $Time(\Pi^t)$ denote the running time of Π. Recall that the joint code Π committed by $\mathsf{Sim}^t_{\mathsf{KID}}(r)$ consists of four parts, and that the procedure $\mathsf{Intermed}^t$ just repeats some relevant prover messages that have been produced by our non-black-box simulator earlier, and thus we can simply assume that its running time is bounded by the running time p_2 of V^*. (i.e., assume each reply to V^*'s query takes one step.) Hence, We have, for $t < T$

$$\begin{aligned} Time(\Pi^t) &\leq p_1 + p_2 + k(k/128)^t/16 Time(\mathsf{Solve}^{t-1}) + p_2 \\ &\leq p_1 + 2p_2 + (k(k/128)^t/16) \cdot 24K \cdot Time(\mathsf{Simulate}^{t-1}) \\ &\leq p_1 + 2p_2 + (k(k/128)^t/16) \cdot 24K \cdot [p_1 + p_2 \\ &\quad + (k(k/128)^{t-1}/16) \cdot 24K \cdot Time(\mathsf{Simulate}^{t-2}) + Time(\mathsf{Sim}^{t-1}_{\mathsf{KID}})] \quad (by(1)) \\ &\leq ... \\ &\leq poly + (24Kk/16)^t (k/128)^{t-1+t-2+...+1} Time(\mathsf{Simulate}^0) \\ &\quad + (24Kk/16)(k/128)^t Time(\mathsf{Sim}^{t-1}_{\mathsf{KID}}) + (24Kk/16)^2 (k/128)^{t+t-1} Time(\mathsf{Sim}^{t-2}_{\mathsf{KID}}) ... \\ &\quad + (24Kk/16)^{t-1} (k/128)^{t+t-1+...+2} Time(\mathsf{Sim}^1_{\mathsf{KID}}) \quad (2) \end{aligned}$$

Where *poly* is a polynomial.

Note that the above inequalities follows from the fact that all those Solve at level $t-1$ invoked by Π are *independent*, which leads to that those $\mathsf{Sim}^{t-1}_{\mathsf{KID}}$s invoked at the *same* level $t-1$ are *independent*. This holds for all levels. Thus, in the inequality (2), the item w.r.t. $Time(\mathsf{Sim}^{t'}_{\mathsf{KID}})$ for all $t' < t$ appears in the form $Poly \cdot Time(\mathsf{Sim}^{t'}_{\mathsf{KID}})$ for some polynomial $Poly$. (Notice that $t < T = 2\log^K_{k/128}$.)

Claim. For all $t < T$, $Time(\Pi^t)$ is polynomial.

[35] With the observation on the structure of a class of sessions, we can simply think of a class of sessions with resect to the underlying simplified Pass-Rosen protocol as a single session

[36] Recall that K is the total number of (classes of) sessions that V^* may initiate

proof. We prove this claim by induction on the level i. Observe that for any t, if $Time(\Pi^t)$ is a polynomial p, then $Time(\mathsf{Sim}^t_{\mathsf{KID}})$ is $p'(p)$ for some polynomial p' (This is guaranteed by the universal argument of Barak and Goldreich), which is also polynomial.

When $i = 1$. Note that Π^1 only consists of $\mathsf{Simulate}$ at level 0 and V^* and $\mathsf{Intermed}^t$, note that $\mathsf{Simulate}$ at level 0 performs in honest way and does not make further calls to Solve, thus $Time(\Pi^1)$ can be bounded by $p_1 + 2p_2$. By the above observation, we also have $Time(\mathsf{Sim}^1_{\mathsf{KID}})$ runs in polynomial time.

Hypothesis. Assume that for any $i < t$, $Time(\Pi^i)$ is polynomial, and so is $Time(\mathsf{Sim}^i_{\mathsf{KID}})$.

Induction. When i=t, by inequality (2) and the hypothesis that $Time(\mathsf{Sim}^i_{\mathsf{KID}})$ is polynomial for all $i < t$, and note that $Time(\mathsf{Simulate}^0)$ is polynomial and $t < T = 2log^K_{k/128}$, we have that $Time(\Pi^t)$ is polynomial. □

Again, we have $Time(\mathsf{Sim}^t_{\mathsf{KID}})$ is also polynomial for any t. Let the longest running time of $Time(\mathsf{Sim}^t_{\mathsf{KID}})$ for all t be $Poly$.

Thus by inequality (1), we have that for $t < T$

$$Time(\mathsf{Simulate}^t) \leq p_1 + p_2 + k(k/128)^t/16 \cdot 24KTime(\mathsf{Simulate}^{t-1}) + Poly \quad (3)$$

$$Time(\mathsf{Simulate}^o) \leq p_1 \quad (4)$$

and note also that

$$\begin{aligned} Time(\mathsf{Simulate}^T) &\leq p_1 + p_2 + K \cdot k \cdot Time(\mathsf{Solve}^{T-1}) \\ &\leq p_1 + p_2 + K \cdot k \cdot 24K \cdot Time(\mathsf{Simulate}^{T-1}) \end{aligned} \quad (5)$$

using the fact $T = 2log^K_{k/128}$, it is easy to verify that the running time of our simulator $\mathbf{S}$, i.e., $Time(\mathsf{Simulate}^T)$, is polynomial.