Precise Zero-Knowledge in Concurrent Setting[*]

Ning Ding, Dawu Gu
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, 200240, China
{dingning, dwgu}@sjtu.edu.cn

Abstract

We present a stronger notion of zero-knowledge: precise concurrent zero-knowledge. Our notion captures the idea that the view of any verifier in concurrent interaction can be reconstructed in the almost same time (within a constant/polynomial factor). Precise zero-knowledge in stand-alone setting was introduced by Micali and Pass in STOC'06 (The original work used the term "local zero-knowledge".). Their notion shows that the view of any verifier can be reconstructed in the almost same time in stand-alone setting. Hence our notion is the generalization of their notion in concurrent setting.

Furthermore, we propose a $\omega(\log^2 n)$-round concurrent zero-knowledge argument for NP with linear precision, which shows that the view of any verifier in concurrent interaction can be reconstructed by the simulator with linear-time overhead. Our argument is Feige-Lapidot-Shamir type which consists of a proof-preamble and a proof-body for a modified NP statement. Our result assumes the restriction of adversarial scheduling the communication that the concurrent interaction of preambles of all sessions will be scheduled before any proof-body by the adversarial verifier.

1 Introduction

Zero-knowledge proofs were introduced by Goldwasser, Micali and Rackoff [17]. Their definition essentially states that an interactive proof of $x \in L$ provides zero (additional) knowledge if, for any efficient verifier V, the view of V in the interaction can be "indistinguishably reconstructed" by an efficient simulator S-interacting with no one- on just input x. Since efficiency is formalized as polynomial-time, a worst-case notion, zero-knowledge too automatically becomes a worst-case notion. The refinement of [14] calls for a tighter coupling between the expected running-time of V and that of S: a proof is zero-knowledge with tightness $t(\cdot)$ if there exists a fixed polynomial $p(\cdot)$ such that the expected running-time of $S(x)$ is upper-bounded by $t(|x|)$ times the expected running-time of $V(x)$ plus $p(|x|)$.

Micali and Pass [20] argued, however, that such coupling may still be insufficient, even when the tightness function is a constant and the polynomial $p(\cdot)$ is identically 0. Consider a malicious verifier V that, on input an instance $x \in \{0,1\}^n$, takes n^{10} computational steps with probability $\frac{1}{n}$

[*]This work is supported by the National Natural Science Foundation of China under Grant No.60573031 and New Century Excellent Talent Program of Education Ministry of China under Grant NCET–05–0398.

, and n steps the rest of the time. The expected running-time of V is n^9, and thus zero-knowledge with optimal tightness only requires that V be simulated in expected time $\Omega(n^9)$. [20] thinks that it is doubtful to take indifference for V to get out and interact with the prover or to stay home and run S for granted. Since by interacting with P, V will almost always execute n steps of computation, while (in absence of extra guarantees) running the simulator might always cause him to invest n^9 steps of computation. This discussion shows that we need a stronger notion of zero-knowledge.

Hence [20] put forward a stronger notion: precise zero-knowledge ([20] used the term "local zero-knowledge". [21] changed to use the term "precise zero-knowledge".). This notion captures the idea that prover provides a zero-knowledge proof of $x \in L$ if the view v of any verifier in an interaction with the prover about x can be reconstructed in the same time (within a constant/polynomial factor). Thus, precise zero-knowledge bounds the knowledge of the verifier in terms of its actual computation.

However, [20] investigated precise zero-knowledge only in stand-alone setting. A more realistic setting is the concurrent setting. A zero-knowledge protocol is concurrent zero-knowledge if for every polynomial-time verifier there exists a polynomial-time simulator that can output an indistinguishable view in concurrent execution of the protocol. So concurrent zero-knowledge is also formalized as a worst-case notion, which also suffers the same problems [20] proposed. Hence a natural consideration is to generalize precise zero-knowledge from stand-alone setting to concurrent setting, i.e., the view of the verifier can be re-constructed by the simulator in almost same time in concurrent setting. This is the very motivation of our work.

1.1 Previous Works

1.1.1 Concurrent zero-knowledge

The notion of concurrent zero knowledge [10] formalizes security in a scenario in which several verifiers access concurrently a prover and maliciously coordinate their actions so to extract information from the prover. In [7] it has been showed that in the black-box model $\widetilde{\Omega}(\log n)$ rounds are necessary for concurrent zero knowledge for non-trivial languages. The first concurrent zero-knowledge proof system for NP has been given by [25] that showed that $O(n^{\varepsilon})$ are sufficient for any $\varepsilon > 0$. Poly-logarithmic round-complexity was achieved in [19] and, finally, in [23] it is shown that $\widetilde{\Omega}(\log n)$ rounds are sufficient. The proof systems presented in [25][19][23] are black-box zero knowledge and the round-complexity of the proof system of [23] is almost optimal in view of the lower bound proved in [7]. Thus unlike the stand-alone case, black-box concurrent zero knowledge cannot be achieved in a constant number of rounds.

Different models have been presented in which round-efficient black-box concurrent zero knowledge is possible. In [10][11][15] constant-round concurrent zero-knowledge proof systems have been presented by relaxing the asynchrony of the model or the zero-knowledge property. In [9][6], constant-round concurrent zero-knowledge proof systems have been presented assuming the existence of a common reference string or a shared random string (i.e., a trusted third party) while in [8] a constant-round concurrent zero-knowledge with concurrent soundness argument system is shown by assuming that there exists a public repository that contains the public keys of the verifiers. Furthermore, Pass [22]] gave a constant-round concurrent zero-knowledge argument with a super-polynomial-time simulator. In [1], Barak presented a non-black-box constant-round bounded-concurrent zero-knowledge argument system. The construction of [1] assumes that the maximum number of concurrent sessions is known in advance.

1.1.2 Precise zero-knowledge

The first step towards precision was introduced in [14]. The refinement of [22], aimed at measuring the "actual security" of a zero-knowledge proof system, calls for a tighter coupling between the worst-case running time of verifier and the expected running time of simulator. [18] introduced the notion of a conservative proof of knowledge that bounds the knowledge of a player in terms of its expected running time. [20] proposed the notion of precise zero-knowledge in stand-alone setting and construct some precise stand-alone zero-knowledge protocols for NP using at least $\omega(1)$ rounds.

1.2 Our Work

In this paper we generalize the notion of precise zero-knowledge from stand-alone setting to concurrent setting. That is, we present the stronger notion of precise concurrent zero-knowledge. Our notion captures the idea that the view of any verifier in concurrent interaction with different provers can be reconstructed by the simulator in the almost same time (within a constant/polynomial factor). Thus, our notion bounds the knowledge of the verifier in terms of its actual computation in concurrent interaction.

Following the notion we propose a $\omega(\log^2 n)$-round concurrent zero-knowledge argument for NP with linear precision in a timing assumption with respect to the static schedule. Linear precision means that the view of any verifier in concurrent interaction can be reconstructed by the simulator with linear-time overhead. The zero-knowledge argument we present relies on the existence of three-round public-coin zero-knowledge proofs of knowledge for NP without requiring negligible soundness-error probability, one of which is presented in [5] if we assume the existence of non-interactive perfectly-binding commitments [4][3]. Our argument is similar to the zero-knowledge protocols in [25][19][23], which consist of two phases: a preamble and a proof body, but we show a different construction of the preamble. The timing assumption we are using is that the concurrent interaction of preambles (excluding the last round) of all sessions will be scheduled before any proof-body by the adversarial verifier.

The key point on obtaining linear precision for the zero-knowledge argument in this work is our precise simulator. Actually for any zero-knowledge proof/argument the precision property is full determined by its simulator. Our simulator can output an indistinguishable view by adopting the rewinding strategy in [19] and adapting it to out argument to extract witnesses of all sessions. But the more important and complicated problem we should face is how to obtain (linear) precision during rewinding. Our simulator also solves the problem by applying the basic idea, i.e., counting the computational steps by the verifier, presented by Micali and Pass [20] to the rewinding strategy and thus obtain linear precision.

Comparison with previous works. As shown in [20], any black-box zero-knowledge is not precise. So all known concurrent black-box zero-knowledge protocols [8][9][10][19][25][23] are not precise. Furthermore, Barak's non-black-box bounded-concurrent zero-knowledge [1] is not precise too, observed by [21]. Hence our protocol is the first one that is provably precise in concurrent setting (under the timing assumption).

1.3 Guide to The Paper

In Section 2 we present some notations we are using. In Section 3 we state out new notion of precise concurrent zero-knowledge. In Section 4 we present the concurrent zero-knowledge argument for

NP with linear precision. In Section 5 we provide a (precise) simulator for the argument and in Section 6 we show why the simulator works in the timing model.

2 Preliminaries

Throughout our paper we use the terms "Turing machine" and "algorithm" interchangeably. We assume familiarity with notions of interactive Turing machines (ITMs for brevity), interactive proofs/arguments as well as basic notions of computational indistinguishability, next message functions, views etc. See [14] for standard descriptions of these notions.

Concurrent execution. Let (P, V) be a two-party protocol, V^* be any ITM, $\{(a_i, b_i)\}_{i=1}^u$ be a set of u inputs to the protocol (P, V). A u-time concurrent execution of (P, V) coordinated by V^* on inputs $\{(a_i, b_i)\}_{i=1}^u$ is the following experiment:

1. Run u independent copies of P with the i'th copy getting a_i as input;
2. Provide V^* with the $b_1, \cdots, b_u$;
3. On each step V^* outputs a message (i, m). The i'th copy of P is given with the message m. V^* is given the prover's response.

Counting TM/ITM steps. If M is a probabilistic machine, denote by M_r the deterministic one obtained by fixing the content of M's random tape to r, by $\text{STEPS}_{M_r(x)}$ the number of computational steps taken by M_r on input x.

Assume (P, V) uses q-round prover messages. In u-times concurrent execution of (P, V), for any ITM V^*, denote by $v = (x_1, \cdots, x_u, z, (m_1, m_2, ..., m_{qu}))$ the view of V^* coordinating u sessions, where V^* is a polynomial-time machine with the auxiliary input $z \in \{0, 1\}^{poly(n)}$ (W.l.o.g assume V^* is deterministic.). Then denote by $\text{STEPS}_{V^*}(v)$ the number of computational steps taken by V^* running on input $x_1, \cdots, x_u$, the auxiliary input z and letting the j'th message received be m_j. (In counting steps, we assume that an algorithm A, given the code of a second algorithm B and an input x, can simulate the computation of B on input x with linear-time overhead.)

3 The New Notion

In this section we put forward our new notion of precise concurrent zero-knowledge and give some remarks on the notion.

Definition 1 *(Precise concurrent zero-knowledge) Let (P, V) be an interactive proof/argument system for a language $L = L(R)$, $p : N \times N \to N$ be a monotonically increasing 2-variate polynomial. We say that (P, V) is a concurrent zero-knowledge proof/argument with precision p if there exists a probabilistic algorithm S such that for every probabilistic polynomial-time algorithm V^* and every auxiliary input $z \in \{0, 1\}^{poly(n)}$ and every polynomial $g(n)$ and every list $\{(x_i, y_i)\}_{i=1}^{g(n)}$, $(x_i, y_i) \in R$:*

1. The following two ensembles are computationally indistinguishable:

a) The view of V^* in a $g(n)$-time concurrent execution of (P, V) with inputs $\{(x_i, y_i), x_i\}_{i=1}^{g(n)}$.

b) $S(x_1, \cdots, x_{g(n)}, V^*, z)$.

2. For every sufficiently long $r \in \{0, 1\}^*$, let v be the view generated by S_r on input $x_1, \cdots, x_{g(n)}, V^*, z$. Then $\text{STEPS}_{S_r(x_1,\cdots,x_{g(n)},V^*,z)} \leq p(\max(n, g(n)), \text{STEPS}_{V^*}(v))$.

We refer to S as above as a precise simulator. We also say that (P, V) has polynomial precision or linear precision if $p(n, u)$ is a linear function in u.

Remarks.
1. Our definition differs from the standard definition of concurrent zero-knowledge in that we additionally require that the actual running-time of the simulator is "close" to the actual running-time of V^* in a true concurrent interaction with different provers.
2. Our definition requires that the simulator is an universal machine. Actually, all known simulators of zero-knowledge protocols are universal machines. One might weaken the definition by requiring that for every polynomial-time V^* there exists a simulator that can satisfy the two conditions.
3. Our definition allows the simulator to use the code of V^*. Actually, [20] shows that any black-box zero-knowledge is not precise zero-knowledge. Hence it manifests an inherent property that any simulator satisfying our definition must access V^* in a non-black-box way.
4. The expression of $\max(n, g(n))$ in condition 2 is used to handle the case that $g(n) = o(n)$. When $g(n) = 1$, it means only one session is running, i.e., this is the stand-alone setting. In this case our definition equals the definition in [20] on computational zero-knowledge. (Besides computational zero-knowledge, [20] also presented the definitions on perfect/statistical zero-knowledge.)

4 A Concurrent Zero-Knowledge Argument with Linear Precision

In this section we construct a concurrent zero-knowledge argument for NP with linear precision. Firstly we present some components to be used in the construction of the zero-knowledge argument. Then we describe the argument.

4.1 Components

4.1.1 Public-coin zero-knowledge proofs of knowledge

Without requiring that soundness error probability is negligible, there exist three-round public-coin zero-knowledge proofs of knowledge for NP languages. One construction was given in [5] for Directed Hamiltonian Cycle (DHC) language if assuming the existence of non-interactive perfectly-binding commitments, some construction of which were given in [4][3]. In the proof for DHC the prover runs the 1'st round and sends a non-interactive perfectly-binding commitment $\texttt{com}$ to the verifier. Then the verifier computes a random bit challenge $\texttt{ch}$ in 2'nd round and the prover sends a response $\texttt{resp}$ for the challenge to the verifier in the 3'rd round. Denote these three messages in interaction by $\texttt{com}$, $\texttt{ch}$ and $\texttt{resp}$. Further, the proof for DHC is special-sound, i.e., for a common input and $\texttt{com}$ if $\texttt{ch}_1, \texttt{resp}_1$ and $\texttt{ch}_2, \texttt{resp}_2$ are two different interaction both of which can convince the verifier, i.e., $\texttt{resp}_1$ and $\texttt{resp}_2$ are right responses, then the extractor of the proof, on input these messages, can output the witness.

Moreover, for any langue L in NP L can be Levin reduced to DHC. That is, there exist three polynomial-time functions f, g, h such that if $(x, y) \in R_L$, then $(f(x), g(x, y)) \in R_{DHC}$ and if $(f(x), z) \in R_{DHC}$, then $(x, h(x, z)) \in R_L$. Actually, Cook-Levin's proof of NP-completeness of SAT [24] shows that L can be Levin reduced to SAT that can be Levin reduced to DHC [26]. Then according to the transitivity of Levin reduction, L can be Levin reduced to DHC. Hence we have that there exists a three-round public-coin zero-knowledge proof of knowledge for L(without requiring negligible soundness error probability). We also denote the three interactive messages by $\texttt{com}$, $\texttt{ch}$ and $\texttt{resp}$, where $\texttt{ch}$ is just a random bit.

4.1.2 Round-efficient witness-indistinguishable proofs of knowledge

Witness-indistinguishability (WI) was introduced in [13]. Informally, a protocol is witness indistinguishable if verifier cannot distinguish between two executions of the protocols which differ in the specific witness prover is using. Clearly, zero-knowledge protocols are witness-indistinguishable and [13] shows that WI property can be preserved in concurrent setting. Hence n parallel composition of the zero-knowledge proof of knowledge for DHC is one construction of round-efficient WI proofs of knowledge.

4.2 The Zero-Knowledge Argument

Our zero-knowledge argument for NP is similar to the zero-knowledge protocols in [25][19][23], following the ideas suggested by Feige, Lapidot and Shamir [12]. The protocol consists of a proof-preamble phase and a proof-body being any "standard" zero-knowledge proof for a modified NP statement. The preamble of our argument consists of $m = \omega(\log^2 n)$ rounds for the security parameter n. The proof body consists of a round-efficient witness-indistinguishable proof of knowledge for NP. Thus, the number of rounds is dominated by the preamble.

Let us concentrate now on the preamble. Let x be the statement to be proven in a NP language L. We use a preamble to start the argument. Let $f : \{0,1\}^n \rightarrow \{0,1\}^n$ be a (non-uniform) one-way function with the relation R_{L_1}, where $(y, x) \in R_{L_1}$ if $f(x) = y$, characterize the language L_1. In this preamble V chooses a random string c and computes $d = f(c)$ and sends d to prover in the first round. Denote the three-round public-coin zero-knowledge proof of knowledge for L_1 by $(P, V)_{\text{atom}}$. Also we use `com`, `ch` and `resp` to denote the three round messages of $(P, V)_{\text{atom}}$.

The preamble is the m-time repetition of $(P, V)_{\text{atom}}$ and the verifier proves to the prover that there exists a c such that $(d, c) \in R_{L_1}$ in the i'st $(P, V)_{\text{atom}}$. The concurrent zero-knowledge argument (with linear precision) goes as follows:

Protocol LinPreciConZK
The common input: $x \in \{0,1\}^n$ to be proved in a NP language L.
The private input to prover: w, the witness for x in L.
The preamble
$V \rightarrow P$: Verifier randomly chooses a string c and computes $d = f(c)$, and sends d to prover.
$V \rightarrow P$: Verifier sends the commitment $\texttt{com}_1$ of the 1'st $(P, V)_{\text{atom}}$ to prover.
$P \rightarrow V$: Prover randomly chooses a bit challenge $\texttt{ch}_1$ and sends it to verifier.
$V \rightarrow P$: Verifier sends a response $\texttt{resp}_1$ and thus completes the 1'st $(P, V)_{\text{atom}}$.
$\cdots\cdots$
$V \rightarrow P$: Verifier sends the commitment $\texttt{com}_m$ of the m'st $(P, V)_{\text{atom}}$ to prover.
$P \rightarrow V$: Prover randomly chooses a bit challenge $\texttt{ch}_m$ and sends it to verifier.
$V \rightarrow P$: Verifier sends a response $\texttt{resp}_m$ and thus completes the m'st $(P, V)_{\text{atom}}$.
The body
$P \leftrightarrow V$: Prover proves to verifier in the round-efficient witness-indistinguishable proof of knowledge the OR of the following statements:
1. There exists a w such that $(x, w) \in R_L$.
2. There exists a c such that $(d, c) \in R_{L_1}$.

Theorem 1 *Protocol LinPreciConZK is an interactive argument.*

Proof. We show that completeness and computational soundness are satisfied.
Completeness: Straightforward. If $(x, w) \in R_L$, the truthful prover can always make the verifier accepted in the body using the witness w regardless of whatever happened in the preamble.
Computational soundness: Suppose computational soundness doesn't hold. That is, there exists a polynomial-sized circuit prover-strategy P^* that can cheat V to accept a $x \notin L$ with non-negligible probability. We can construct a polynomial-sized circuit R based on P^* to find pre-images of the one-way function f, which contradicts the onewayness of f.

Assume $d = f(c)$ for an unknown random c. The task of R on input d is to compute c. Let R and P^* play protocol LinPreciConZK. For each n assume R is given an auxiliary input $x \in \{0,1\}^n, x \notin L$. Since the preamble is the m-time repetition of $(P,V)_{\text{atom}}$, then we know the preamble is also zero-knowledge to the prover. R works as follows: Firstly R calls the simulator of the preamble, on input x, to generate an indistinguishable view, denoted `PreambleView`. Let P^{**} denote P^* with x and `PreambleView` hardwired into. In the following R adopts the honest-verifier strategy to play the proof body of LinPreciConZK with P^{**}. If P^* can cheat V with probability ε, then we claim that P^{**} can cheat R with probability ε' such that $|\varepsilon - \varepsilon'|$ is negligible. (Otherwise (P^{**}, R) is a distinguisher for the output of the simulator of the preamble.). Hence R calls the knowledge extractor of the WI proof of knowledge to find the witness for the OR of the statements of (1) x is true or (2) P^{**} knows c. Since x is not true, we have the extracted witness is the c. Thus, R succeeds in computing the preimage of d with non-negligible probability, which contradicts the onewayness of f. Hence computational soundness is satisfied. Thus, this completes the proof.

More naturally, we combine every two adjacent verifier-messages into one verifier-message in the preamble as follows without hurting completeness and soundness of the protocol.
The preamble'
$V \to P$: Verifier randomly chooses a string c and computes $d = f(c)$, and sends d, to prover. Then send the commitment $\texttt{com}_1$ of the 1'st $(P,V)_{\text{atom}}$ to prover.
$P \to V$: Prover randomly chooses a bit challenge $\texttt{ch}_1$ and sends it to verifier.
$V \to P$: Verifier sends a response $\texttt{resp}_1$, thus completes the 1'st $(P,V)_{\text{atom}}$ and sends the commitment $\texttt{com}_2$ of the 2'st $(P,V)_{\text{atom}}$ to prover.
$\cdots\cdots$
$V \to P$: Verifier sends a response $\texttt{resp}_{m-1}$, thus completes the $m-1$'st $(P,V)_{\text{atom}}$ and sends the commitment $\texttt{com}_m$ of the m'st $(P,V)_{\text{atom}}$ to prover.
$P \to V$: Prover randomly chooses a bit challenge $\texttt{ch}_m$ and sends it to verifier.
$V \to P$: Verifier sends a response $\texttt{resp}_m$ and thus completes the m'st $(P,V)_{\text{atom}}$.

In the following we adopt the natural representation for the preamble. We sometimes call the verifier round that contains a response and a commitment the verifier-response for concision.

5 The Precise Simulator

In this section we construct a simulator that can provide zero-knowledgeness and linear precision simultaneously in concurrent setting under our timing assumption.

5.1 The High Level Description of The Simulator

The problems the simulator should solve is how to gain the witnesses of all sessions and obtain linear precision via an efficient rewinding strategy. First of all we adopt the oblivious recursive rewinding strategy presented by Kilian and Petrank [19] and adapt it to our protocol. Assume that an adversarial verifier executes $g(n)$ sessions concurrently. We assume that the real prover (and so also the simulator) always answers immediately. Namely, the adversarial verifier may delay its answer as it pleases, but the prover answers at once. Thus, we get that the adversarial verifier V^* may schedule an overall number of $m \cdot g(n)$ slots in the preambles. Every slot consists of a response by the verifier, followed by a challenge by the prover. Note that the last verifier-response in each preamble is not followed by a prover-challenge, so this verifier-response cannot be included in any slot. But for concision we still say the overall number of slots are $m \cdot g(n)$. Hence our timing assumption can be stated more precisely that all these $m \cdot g(n)$ slots (excluding the last verifier-response) should be scheduled before any body of all sessions.

We consider the $m \cdot g(n)$ slots by their order in time. We will not need to rewind the bodies: the simulator will behave like the real prover in the bodies. Very briefly, the adapted oblivious rewinding on the $m \cdot g(n)$ slots is described by the following procedure $\texttt{Schedule}(1, m \cdot g(n))$.

```
Procedure Schedule$(a, b)$
if $b - a = 1$ then execute slot $a$ and slot $b$ sequently and return.
execute Schedule$(a, \frac{a+b-1}{2})$.
rewind $\frac{a+b-1}{2} \rightarrow a$, i.e., re-execute Schedule$(a, \frac{a+b-1}{2})$.
execute Schedule$(\frac{a+b+1}{2}, b)$.
rewind $b \rightarrow \frac{a+b+1}{2}$, i.e., re-execute Schedule$(\frac{a+b+1}{2}, b)$.
return.
End Schedule
```

[19] showed how to obtain witnesses of all sessions via the above rewinding strategy. But the more complicated problem is how to obtain precision simultaneously. Micali and Pass [20] investigated the problem on obtaining precision in stand-alone setting and presented an approach to solve this problem. Their simulator keeps track of the number of computational steps taken by verifier in many sub-protocols. Soon after, the simulator will attempt to compute a witness by rewinding each of the sub-protocol and will abort each such rewinding as soon as verifier's running-time exceeds its running-time in the first execution of that very sub-protocol. When they succeed in extracting a witness in this fashion, it is guaranteed that running-time of the simulator is close to that of verifier in its first execution.

We apply this idea to the above rewind strategy and make our simulator can solve these problems simultaneously. Now let us concentrate on the simulation strategy. First of all, there exists an essential difference between our simulator and one in [19]. That is, when the simulator in [19] rewinds the verifier, the rewind run is the one that the simulator uses to continue the interaction. The first run is only used to get information and is then abandoned. But for our simulator the first run is the one that the simulator uses to continue the interaction. The rewind run is only used to get information for extraction.

Since we assume that the preambles of all sessions are scheduled before any body proof, we pay attention to the rewind strategy of preambles in the first. In the following we describe the simulator's behavior at the bottom of the recursion, i.e., how the simulator executes two sequent slots at the bottom of recursion. Assume that the simulation is now entering the interval $[a, b]$. The run of interval $[a, b]$ will be divided into three cases.

If it is the first (not rewind) run of the interval $[a, b]$ the simulator adopts the honest-prover strategy to interactive with the verifier. The simulator stores the verifier-responses and prover-challenges in slot a and slot b respectively and counts the running-time of the verifier on outputting the verifier-responses. From this fact we know that the simulator uses non-black-box access to the adversarial verifier since it counts the running-time of verifier, even in a very minimal sense.

If it is the first rewind run of interval $[a, b]$, the prover-challenge of slot a is the rewind entry. The simulator chooses a random challenge once more and stores it. Recall that the first run is used to continue the simulation, while the rewind run is used to gather information for extraction. That is, the prover-challenge in this rewind run will be not used until the point where the simulation entries a slot which verifier-response is the response for this prover-challenge. The simulator will use the new prover-challenge for extraction when entering that slot, just like what it will do in slot b, to be described in the following.

After choosing the prover-challenge in slot a the simulator is about to entry slot b. Before computing the verifier-response of slot b, the simulator prepares the input messages (view) to verifier's next-message function. That is, the simulator gains all prover-messages from the stored transcript up until the point that the verifier is about to compute the response, but to replace the prover-challenge chosen in the first run that corresponds to the verifier-response of slot b by the one chosen in some rewind run of some slot. Actually there must exist a previous slot in which the simulator chose and stored the random challenge in the rewind, just like what the simulator did in slot a. Note that up until the point the verifier is about to compute the response the views to the verifier are same except for the two random prover-challenges in the first run and rewind run respectively. Also note that by our timing assumption the simulation in preambles only involves the messages in the preambles in which all prover-messages are public-coin. So V^* cannot distinguish the two views at all.

When computing the verifier-response of slot b, the simulator put the artificial view to the verifier and use the running-time gathered in the first run to bound the V^*'s computing. That is, if the verifier cannot output its response within the running-time, then abort its computing. Otherwise when the verifier succeeds in outputting its response, if both the two verifier-responses in the first run and rewind run is an accepting response, the simulator calls the knowledge extractor of $(P, V)_{\mathrm{atom}}$ to extract the witness. In this case we call the rewind run succeeds. In any other case we call the rewind run fails. Using the running-time gathered in the first run to bound the rewind run can guarantee that the running-time of the rewind run is no more than that in the first run.

If it is a rewind run, but not the first rewind run, of interval $[a, b]$, we just let the simulator jumps over the interval $[a, b]$. Hence every interval will be rewound once, which will result in obtaining linear precision.

When has finished the recursive rewinding, the simulator plays the bodies of all sessions with the verifier since we assume all preambles are scheduled before the bodies. In the bodies the simulator just adopts honest-prover strategy using the witnesses extracted in the rewind to convince the verifier. If the simulator cannot extract a witness for a session, then it aborts the simulation.

5.2 The Actual Description of The Simulator

We now turn to formally describing the simulator algorithm. The simulator algorithm consists of two procedures: `Schedule` and `ExecuteSlot`. `Schedule` is the recursive rewind of $m \cdot g(n)$ slots. At the bottom of recursion, if it is the first (not rewind) run of interval $[a, b]$, `Schedule` calls `ExecuteSlot` to execute slot a and slot b sequently. W.l.o.g assume that every verifier message has the form of $(t, message)$ where $message$ is the verifier message and t is the number of the session $message$ belongs to.

Algorithm: The simulator S
Input: $x_1, \cdots, x_{g(n)} \in \{0,1\}^n$: the statement to be proved in the i'th session is that $x_i \in L$. V^*: description of a polynomial-time verifier coordinating an $g(n)$-time concurrent execution. $z \in \{0,1\}^{poly(n)}$: the auxiliary input. W.l.o.g assume V^* is deterministic. For concision we sometimes use V^* to denote V^* with input $x_1, \cdots, x_{g(n)}, z$.
Initialization: Construct such tables and variables as follows:
`InputMsgToVrf`: A variable denotes the input messages to V^*'s next message function.
`OutputMsg`: A variable denotes V^*'s output message in a round.
`SimMsg`: A variable denotes S's prover-message in a round.
`Trans`: A table of length $q \cdot g(n)$, where q is the number of rounds of LinPreciConZK. `Trans` is used to record the transcript in interaction between S and V^*. That is, `Trans`$[i]$ is the i'th round message, $1 \le i \le q \cdot g(n)$, in concurrent interaction. Initial `Trans` is empty.
`VrfMsgRcvOfSes`: A table of length of $g(n)$. `VrfMsgRcvOfSes`$[i]$ records the number of verifier-responses belonging to session i S received. Initial `VrfMsgRcvOfSes` is empty.
`SrlNumInTransOfSlots`: A table of length $m \cdot g(n)$. `SrlNumInTransOfSlots`$[i]$ is used to store the serial number of the cell in `Trans` where the first message of slot i, i.e. V^*'s response, is stored. Initial `SrlNumInTransOfSlots[i]` is 0.
`InfoOfSlots`: A table of length $m \cdot g(n)$. `InfoOfSlots`$[i]$ contains some useful information of slot i, which consists of six elements: `IsRwd`, `HasBeenRwd`, `PrvMsgSrlNum`, `PrvMsgSesNum`, `VrfMsgSrlNum` and `VrfMsgSesNum`. `IsRwd` is a boolean variable. If a run of slot i is a rewind run, put `IsRwd`=`True`. Otherwise put `IsRwd`=`False`. `HasBeenRwd` is also a boolean variable denoting whether or not slot i has been rewound before. If it has, put `HasBeenRwd=True`. Otherwise put `HasBeenRwd=False`. `PrvMsgSrlNum` and `PrvMsgSesNum` denote the serial number of the atomic protocol and session number which the prover-challenge of slot i belongs to respectively. Analogously, `VrfMsgSrlNum` and `VrfMsgSesNum` denote the serial number of the atomic protocol and session number which the verifier-response of slot i belongs to respectively. Initial `IsRwd=HasBeenRwd=False`.
`PrvMsgSrlNumInTrans`: A two-dimension table of length $g(n) \times m$. `PrvMsgSrlNumInTrans`$[i][j]$ records the serial number of the cell in `Trans` where the prover-challenge of the j'th atomic protocol of the i'th session is stored.
`MsgOfExt`: A two-dimension table of length $g(n) \times m$. `MsgOfExt`$[i][j]$ contains all information on witness extraction of the j'th atomic protocol of the i'th session, which consists of eight elements: `com`, $\mathtt{ch}_1$, $\mathtt{resp}_1$, `RunTime`, $\mathtt{ch}_2$, $\mathtt{resp}_2$, `IsExtSuc` and `ExtValue`. $\mathtt{ch}_1$ and $\mathtt{resp}_1$ are the messages in the first run of the j'th atomic protocol of the i'th session. `RunTime` is the computational steps by V^* to output $\mathtt{resp}_1$. $\mathtt{ch}_2$ and $\mathtt{resp}_2$ are the messages in the rewind run of the atomic protocol. If in the rewind run after running `RunTime` steps V^* doesn't terminate to output $\mathtt{resp}_2$, or $\mathtt{ch}_1 = \mathtt{ch}_2$, or at least one of $\mathtt{resp}_1$ and $\mathtt{resp}_2$ is not a right response. In this case put the boolean

variable IsExtSuc False. Otherwise put IsExtSuc True and in this case we can call the knowledge extractor of $(P, V)_{\text{atom}}$ to compute the witness and let ExtValue denote the extracted witness.

Begin simulation
S Informs V^* to start the interaction. Then V^* sends the first round message of session t to S. Store the message in Trans. Set VrfMsgRcvOfSes$[t] = 1$. The simulator S randomly chooses a bit ch$_1$, sends ch$_1$ to V^* and stores ch$_1$ in Trans. Set com and ch$_1$ of MsgOfExt$[t][1]$ the commitment in the verifier round and ch$_1$ respectively and set PrvMsgSrlNumInTrans$[t][1]$ the serial number of cells in Trans where ch$_1$ is stored. Then S starts the procedure Schedule.
Schedule$(1, m \cdot g(n))$.
In the remainder of simulation, S adopts the honest-prover strategy to play with V^* and stores interaction in Trans. In the proof body of the i'th session S searches a j such that IsExtSuc in MsgOfExt$[i][j]$ is True. If there doesn't exist such j, S aborts the remainder simulation. Otherwise, S uses the ExtVal as the witness to make V^* accept the combined statement. At last Trans contains all prover-messages that are indistinguishable from the view of V^* in real interaction but all common input and the auxiliary input.
End Simulation

```
Procedure Schedule$(a, b)$
If b-a=1 then we consider two cases according to values of IsRwd and HasBeenRwd of InfoOfSlots$[a]$.
    Case 1 of IsRwd=False and HasBeenRwd=False: It means this is the first run of interval $[a, b]$.
        Then execute slot $a$ and slot $b$ sequently. ExecuteSlot$(a)$. ExecuteSlot$(b)$.
    Case 2 of IsRwd=True and HasBeenRwd=False: It means this is the first rewind run of $[a, b]$.
        Set HasBeenRwd=True. $S$ randomly chooses a bit, denoted ch$_a$, and obtains PrvMsgSrlNum
        and PrvMsgSesNum from InfoOfSlots$[a]$. Put ch$_2$ of MsgOfExt [PrvMsgSesNum] [PrvMsgSrlNum]
        ch$_a$.
            Also $S$ obtains VrfMsgSrlNum and VrfMsgSesNum of slot $b$ from InfoOfSlots$[b]$ and
        thus gains com$_2$ and RunTime of MsgOfExt [VrfMsgSesNum] [VrfMsgSrlNum]. Let $k$ be
        PrvMsgSrlNumInTrans [VrfMsgSesNum] [VrfMsgSrlNum].
            Let ch$_b$ be ch$_2$ of MsgOfExt [VrfMsgSesNum] [VrfMsgSrlNum], $j =$ SrlNumInTransOfSlots$[b]$.
        Let InputMsgToVrf be the prover-messages in Trans from the 1'st cell to the $j-1$'st
        cell and then replace the $\frac{k}{2}$'th element in InputMsgToVrf by ch$_b$.
            Compute $(t,$ OutputMsg$) \leftarrow V^*($InputMsgToVrf$)$ within the time RunTime. If $V^*$ cannot
        finish computing, then put resp$_2$ of MsgOfExt$[t]$[VrfMsgSrlNum] $\perp$. Otherwise set resp$_2$
        the verifier-response in OutputMsg. If ch$_1 \neq$ ch$_2$, and both resp$_1$ and resp$_2$ is a right
        response, $S$ calls the extractor of $(P, V)_{\text{atom}}$, on input $x_t$, com, ch$_1$, resp$_1$, ch$_2$, resp$_2$, to
        compute the witness. At last $S$ sets ExtVal and IsExtSuc of MsgOfExt$[t]$[VrfMsgSrlNum]
        the witness and True respectively.
Else, i.e., $b - a \neq 1$
    Schedule$(a, \frac{a+b-1}{2})$. Schedule$(a, \frac{a+b-1}{2})$.
    Schedule$(\frac{a+b+1}{2}, b)$. Schedule$(\frac{a+b+1}{2}, b)$.
End Schedule
```

```
Procedure ExecuteSlot$(i)$
    Set IsRwd of InfoOfSlots$[i]$ True.
```

Let `InputMsgToVrf` be all prover-messages in `Trans`. Compute $(t, \texttt{OutputMsg}) \leftarrow V^*(\texttt{InputMsgToVrf})$ and counts the running-time of V^*. Store `OutputMsg` in `Trans` and increase `VrfMsgRcvOfSes`$[t]$ by 1. S, on input `OutputMsg`, computes the next prover message, denoted `SimMsg`. Store `SimMsg` in `Trans`.

Put `SrlNumInTransOfSlots`$[i]$ the number of cells used in `Trans`. Set $j =$ `VrfMsgRcvOfSes`$[t]$. Put $\texttt{resp}_1$ of `MsgOfExt`$[t][j]$ the verifier-response in `OutputMsg`. Let `VrfMsgSrlNum` and `VrfMsgSesNum` of `InfoOfSlots`$[i]$ be j and t respectively. Put `RunTime` of `MsgOfExt`$[t][j]$ the counted running-time. If `SimMsg` is still a prover-challenge of an atomic protocol, then let `PrvMsgSrlNumInTrans`$[t][j+1]$ be `SrlNumInTransOfSlots`$[i]+1$. Also set `PrvMsgSrlNum` and `PrvMsgSesNum` of `InfoOfSlots`$[i]$ $j+1$ and t respectively. Set `com` and $\texttt{ch}_1$ of `MsgOfExt`$[t][j+1]$ the verifier-commitment in `OutputMsg` and `SimMsg` respectively.
End `ExecuteSlot`

6 Why The Simulator Works

Since Theorem 1 has shown that LinPreciConZK is an interactive argument, we only need to show that the simulator S satisfies definition 1. That is, we should explain (1) that the outputs generated by the simulator S are computationally indistinguishable from the view of V^* in real interaction and (2) that there exists a monotonically increasing 2-variate polynomial $p(\cdot,\cdot)$ such that for $v = S_r(x_1, \cdots, x_{g(n)}, V^*, z)$ $\text{STEPS}_{S_r(x_1,\cdots,x_{g(n)},V^*,z)} \leq p(\max(x, g(n)), \text{STEPS}_{V^*}(v))$.

(1). As shown in [25][19][23], the ability of extracting witnesses of the simulator implies that the simulator can generate an indistinguishable view. Hence the main goal of this part is to show that the simulator S can extract witnesses for all sessions with very high probability. So in the following we will analyze the probability of extracting witnesses of S. Before presenting the details, we refer to the notion of a rewind that may solve a proof, introduced in [19] firstly.

Definition 2 *(A rewind that may solve a proof [19]) We say that a rewind $(l \to k)$ may solve a proof Π if the following four conditions hold:*
1. *Exactly two rounds of the preamble of Π take place during round slots $k, k+1, \cdots, l$,*
2. *The first round of Π takes place at a round slot $i < k$,*
3. *The last round of Π takes place at a round slot $j > l$, and*
4. *The first round of Π appears in the first half the rewind interval $(l \to k)$ and the second round of Π appears in the second half of the rewind interval $(l \to k)$.*

This notion captures a necessary condition that a rewind is the smallest rewind that has a chance of extracting a witness. Note that Π has two rounds in the same half, then there is a dominated interval that may solve Π. This the reason for condition 4 of the definition.

Lemma 1 *[19]. For any schedule of k copies of the proof preambles (in the mk round slots), if a preamble of a specific proof Π completes in round l, then there are at least $\left\lceil \frac{m}{\log(mk)+1} \right\rceil - 2$ rewind intervals that complete by round l and that may solve Π.*

For a specific proof Π of $g(n)$ sessions, let us compute the probability that a rewind that may solve Π fails to solve it. Actually, it is not always the case that a proof is always solved in a rewind that may solve it. There are two reasons. One reason is that V^* may sometimes abort to give a right response. The other one is that even though V^* wants to give right responses in both the first run and rewind run, there is no sufficient time for V^* to output the response in the rewind run. Note that if V^* chooses not to send a right response in the first run, we take for granted that the proof is solved since in this case the real prover will abort the interaction immediately. Further, if V^* chooses to give right responses in both the first run and rewind run and V^* can finish the computing within the running-time gathered in the first run, then it is clearly that Π is solved. In other cases the rewind may solve Π fails to solve it.

Now let us compute the probability that Π is solved in the rewind may solve it in the case that V^* chooses to give right responses in both the first run and rewind run. Assume S sends a random bit `ch` to V^* and obtains the V^*'s response within a running-time, denoted `RunTime`, in the first run of the rewind may solve Π. In the rewind run S send a random bit `ch`' to V^* once more. If `ch`'=`ch`, then the the rewind fails. If `ch`'$\neq$`ch`, V^* may still fail to output a response because V^* needs more time than `RunTime` to finish the computing. However, we have that if S chooses `ch`' and sends it to V^* in the first run and sends `ch` to V^* in the rewind run, then V^* has enough time to output two responses for `ch`' and `ch` respectively. Note that the views to V^* in the first run and rewind run, up until the point that V^* is about to send its response, are same except for `ch`' and `ch`. And further, by the timing assumption all prover-messages are public-coin, so V^* cannot distinguish the two views at all. Hence for the rewind the probability V^* can finish its computing is $\frac{1}{2}$ conditioned that `ch`'$\neq$`ch` which happens with probability $\frac{1}{2}$. In a word, a rewind may solve Π solves it with probability $\frac{1}{4}$ in this case.

Suppose V^* chooses not to send a right response in a run with probability α. From the above analysis we have that a rewind may solve the proof solves it with probability $\alpha+\frac{1}{4}(1-\alpha)(1-\alpha) \geq \frac{1}{4}$. By Lemma 1 there are at least $\left\lceil \frac{m}{\log(m\cdot g(n))+1} \right\rceil - 2$ rewind intervals that may solve Π. Hence using $m = \omega(\log^2 n)$, we get that there are at least $\frac{\omega(\log^2 n)}{\log g(n)+\log m} = \omega(\log n)$ rewinds in which Π may be solved. Now because there are no more than $\frac{3}{4}$ probability of a proof not being solved during any of these rewinds, and all of them are disjoint, we have that with probability at most $(\frac{3}{4})^{\omega(\log n)} = \varepsilon(n)$ the proof is not solved during any rewind, where $\varepsilon(n)$ is a negligible function. Now using the union bound we get there is only negligible probability that any proof is not solved at all. Hence as shown in [25][19][23], the simulated view is indistinguishable from the view in real interaction of all sessions.

(2). Assume the running-time of V^* on the view v is $\mathrm{STEPS}_{V^*}(v)$. The goal of this part is to provide an upper-bound for the running-time of S in computing v and obtaining witnesses during the simulation. Firstly we consider the running-time of S in the preambles. According to the description of S, we have that each interval at the bottom of recursion runs twice, in which one is the first run and the other one is the first rewind run. For the j'th atomic protocol of the i'th session, denote the computational steps by V^* in the verifier round to output its response in the first run by $\mathtt{RunTime}_{ij}$. Note that actually all computing of V^* in simulation is run by S. Denote by l the linear-time overhead for S simulating V^*. Then the running-time of S in simulating V^* in the recursive rewinding is no more than $2l$ multiplied by the sum of all $\mathtt{RunTime}_{ij}$, $1 \leq i \leq g(n), 1 \leq j \leq m$. Moreover, we know that the running-time of S not spending in simulating V^* in the preambles is a polynomial, denoted $p_1(n)g(n)$.

In the body phases S doesn't rewind V^* and the running-time of S in the WI proofs of knowledge is also a polynomial, denoted $p_2(n)g(n)$. And it is obvious that the sum of all $\mathtt{RunTime}_{ij}$, $1 \leq i \leq$

$g(n), 1 \leq j \leq m$ is less than $\mathrm{STEPS}_{V^*}(v)$. Hence we have that $\mathrm{STEPS}_{S_r}(x_1, \cdots, x_{g(n)}, V^*, z) \leq p_1(n)g(n) + p_2(n)g(n) + 2l \cdot \sum \texttt{RunTime}_{ij}$, $1 \leq i \leq g(n), 1 \leq j \leq m$.

Hence, $\mathrm{STEPS}_{S_r}(x_1, \cdots, x_{g(n)}, V^*, z) \leq p_1(n)g(n) + p_2(n)g(n) + 2l \cdot \mathrm{STEPS}_{V^*}(v)$. Therefore, there is a monotonically increasing 2-variate polynomial $p(n, u)$ that is linear in u such that

$$\mathrm{STEPS}_{S_r}(x_1, \cdots, x_{g(n)}, V^*, z) \leq p(\max(n, g(n)), \mathrm{STEPS}_{V^*}(v))$$

The linear precision is obtained. Thus we have Theorem 2.

Theorem 2 *Protocol LinPreciConZK is a concurrent zero-knowledge argument with linear precision (under the timing assumption).*

References

[1] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. In *42nd FOCS*, pages 106-115, 2001.

[2] B. Barak and O. Goldreich. Universal Arguments and their Applications. In *17th CCC*, pages 194-203, 2002.

[3] B. Barak, S. J. Ong, and S. Vadhan. Derandomization in Cryptography, In *Proc. Crypto'03*, 2003.

[4] M. Blum. Coin Flipping by Phone. In *Proc. 24th IEEE Computer Conference (Comp- Con)*, pages 133-137, 1982.

[5] M. Blum. How to prove a Theorem So No One Else Can Claim It. *Proc. of the International Congress of Mathematicians*, Berkeley, California, USA, pages 1444-1451, 1986.

[6] M. Blum, A. De Santis, S. Micali and G. Persiano. Non-Interactive Zero-Knowledge. *SIAM J. on Computing* 20, pages 1084-1118, 1991.

[7] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires $\tilde{\Omega}(\log n)$ Rounds. In *33rd STOC*, pages 570-579, 2001.

[8] G. Di Crescenzo, G. Persiano, I. Visconti. Constant-Round Resettable Zero Knowledge with Concurrent Soundness in the Bare Public-Key Model. In *Proc. of Crypto'04*. Springer-Verlag, LNCS 3152, pages 237-253, 2004.

[9] I. Damgard. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. In *Proc. of Eurocrypt'00.* Springer-Verlag LNCS 1807, pages 418-430, 2000.

[10] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero Knowledge. In *30th STOC*, pages 409-418, 1998.

[11] C. Dwork and A. Sahai. Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints. In *Proc. of Crypto'98.* Springer-Verlag, LNCS 1462, pages 442-457, 1998.

[12] Feige, Lapidot, and Shamir. Multiple Noninteractive Zero Knowledge Proofs Under General Assumptions. *SIAM J. Comput.*, 29, 1999.

[13] U. Feige and A. Shamir. Witness Indistinguishability and Witness Hiding Protocols. In *22nd STOC*, pages 416-426, 1990.

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

[15] O. Goldreich. Concurrent Zero-Knowledge with Timing, Revisited. In *Proc. of STOC'02*, ACM, pages 332-340, 2002.

[16] O. Goldreich and L. A. Levin. A Hard-Core Predicate for All One-Way Functions. In *Proc. 21st STOC*, pages 25-32. ACM, 1989.

[17] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems. In *17th STOC*, pages 291-304, 1985.

[18] S. Halevi and S. Micali. Conservative Proofs of Knowledge. MIT/LCS/TM-578, May 1998.

[19] J. Kilian and E. Petrank. Concurrent Zero-Knowledge in Poly-logarithmic Rounds. Cryptology ePrint Archive, Report 2000/013, 2000. Extended abstract appeared in STOC'01.

[20] S. Micali and R. Pass. Local Zero Knowledge. In *38th STOC*, 2006.

[21] R. Pass. A Precise Computational Approach to Knowledge. Ph.D Thesis, MIT, 2006.

[22] R. Pass. Simulation in Quasi-polynomial Time and its Application to Protocol Composition. In *EuroCrypt'03*, Springer LNCS 2656, pages 160-176, 2003.

[23] M. Prabhakaran, A. Rosen, and A. Sahai. Concurrent Zero Knowledge with Logarithmic Round-Complexity. In *43rd FOCS*, pages 366-375, 2002.

[24] S. Rudich and A. Wigderson, editors. *Computational Complexity Theory*. IAS/Park City mathematics series, 2004.

[25] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *Eurocrypt'99*, LNCS 1592, 1999.

[26] I. Wegener. *Complexity Theory*. Springer-Verlag, Berlin Heidelberg, 2005.