Precise Concurrent Zero Knowledge

Omkant Pandey[1], Rafael Pass[2⋆], Amit Sahai[1⋆⋆], Wei-Lung Dustin Tseng[2⋆⋆⋆], and Muthuramakrishnan Venkitasubramaniam[2]

[1] {omkant, sahai}@cs.ucla.edu
University of California Los Angeles, California.
[2] {rafael, wdtseng, vmuthu}@cs.cornell.edu
Cornell University, New York.

Abstract. *Precise zero knowledge* introduced by Micali and Pass (STOC'06) guarantees that the view of any verifier V can be simulated in time closely related to the *actual* (as opposed to worst-case) time spent by V in the generated view. We provide the first constructions of precise concurrent zero-knowledge protocols. Our constructions have essentially optimal precision; consequently this improves also upon the previously tightest non-precise concurrent zero-knowledge protocols by Kilian and Petrank (STOC'01) and Prabhakaran, Rosen and Sahai (FOCS'02) whose simulators have a quadratic worst-case overhead. Additionally, we achieve a statistically-precise concurrent zero-knowledge property—which requires simulation of unbounded verifiers participating in an unbounded number of concurrent executions; as such we obtain the first (even non-precise) concurrent zero-knowledge protocols which handle verifiers participating in a super-polynomial number of concurrent executions.

1 Introduction

Zero-knowledge interactive proofs, introduced by Goldwasser, Micali and Rackoff [GMR85] are constructs allowing one player (called the Prover) to convince another player (called the Verifier) of the validity of a mathematical statement $x \in L$, while providing no additional knowledge to the Verifier. The zero-knowledge property is formalized by requiring that the view of any PPT verifier V in an interaction with a prover can be "indistinguishably reconstructed" by a PPT simulator S, interacting with no one, on input just x. Since whatever V "sees" in the interaction can be reconstructed by the simulator, the interaction does not yield anything to V that cannot already be computed with just

⋆ This material is based upon work supported under a I3P Identity Management and Privacy Grant.

⋆⋆ This research was supported in part by NSF ITR and Cybertrust programs (including grants 0627781, 0456717, 0716389, and 0205594), a subgrant from SRI as part of the Army Cyber-TA program, an equipment grant from Intel, an Okawa Research Award, and an Alfred P. Sloan Foundation Research Fellowship. First author was supported in part from third author's grants.

⋆⋆⋆ This material is based upon work supported under a National Science Foundation Graduate Research Fellowship, and a NSERC Canada Julie-Payette Fellowship.

the input x. Because the simulator is allowed to be an arbitrary PPT machine, this traditional notion of ZK only guarantees that the *class* of PPT verifiers learn nothing. To measure the knowledge gained by a particular verifier, Goldreich, Micali and Wigderson [GMW87] (see also [Gol01]) put forward the notion of *knowledge tightness*: intuitively, the "tightness" of a simulation is a function relating the (worst-case) running-time of the verifier and the (expected) running-time of the simulator—thus, in a knowledge-tight ZK proof, the verifier is guaranteed not to gain more knowledge than what it could have computed in time closely related to its *worst-case* running-time.

Micali and Pass [MP06] recently introduced the notion of *precise zero knowledge* (originally called *local* ZK in [MP06]). In contrast to traditional ZK (and also knowledge-tight ZK), precise ZK considers the knowledge of an individual verifier in an *individual execution*—it requires that the view of any verifier V, in which V takes t computational steps, can be reconstructed in time closely related to t—say $2t$ steps. More generally, we say that a zero-knowledge proof has precision $p(\cdot,\cdot)$ if the simulator uses at most $p(n,t)$ steps to output a view in which V takes t steps on common input an instance $x \in \{0,1\}^n$.

This notion thus guarantees that the verifier does not learn more than what can be computed in time closely related to the *actual* time it spent in an interaction with the prover. Such a guarantee is important, for instance, when considering knowledge of "semi-easy" properties of the instance x, considering proofs for "semi-easy" languages L, or when considering *deniability* of interactive protocols (see [MP06,Pas06] for more discussion).

The notion of precise ZK, however, only considers verifiers in a stand-alone execution. A more realistic model introduced by Dwork, Naor and Sahai [DNS98], instead considers the execution of zero-knowledge proofs in an asynchronous and concurrent setting. More precisely, we consider a single adversary mounting a coordinated attack by acting as a verifier in many concurrent *sessions* of the same protocol. Concurrent zero-knowledge proofs are significantly harder to construct and analyze.

Richardson and Kilian [RK99] constructed the first concurrent zero-knowledge argument in the standard model (without any extra set-up assumptions). Their protocol requires $O(n^{\epsilon})$ number of rounds. Kilian and Petrank [KP01] later improved the round complexity to $\tilde{O}(\log^2 n)$. Finally, Prabhakaran, Rosen and Sahai [PRS02] provided a tighter analyis of the [KP01] simulator showing that $\tilde{O}(\log n)$ rounds are sufficient. However, none of the simulators exhibited for these protocols are precise, leaving open the following question:

Do there exist precise concurrent zero-knowledge proofs (or arguments)?

In fact, the simulators of [RK99,KP01,PRS02] are not only imprecise, but even the overhead of the simulator with respect to the *worst-case* running-time of the verifier—as in the definition of knowledge tightness—is high. The simulator of [RK99] had *worst-case precision* $p(n,t) = t^{O(\log_n t)}$—namely, the running-time of their simulator for a verifier V with *worst-case* running-time t is $p(n,t)$ on input a statement $x \in \{0,1\}^n$. This was significantely improved by [KP01] who obtained a quadratic worst-case precision, namely $p(n,t) = O(t^2)$; the later result by [PRS02] did not improve upon this, leaving open the following question:

> *Do there exist concurrent zero-knowledge arguments (or proofs) with sub-quadratic worst-case precision?*

Our Results. Our main result answers both of the above questions in the affirmative. In fact, we present concurrent zero-knowledge protocols with essentially optimal precision. Our main lemma shows the following.

Lemma 1 (Main Lemma) *Assuming the existence of one-way functions, for every $k, g \in \mathbb{N}$ such that $k/g \in \omega(\log n)$, there exists an $O(k)$-round concurrent zero knowledge argument with precision $p(t) \in O(t \cdot 2^{\log_g t})$ for all languages in $\mathcal{NP}$.*

By setting k and g appropriately, we obtain a simulation with near-optimal precision.

Theorem 1 *Assuming the existence of one-way functions, for every $\varepsilon > 0$, there exists a $\omega(\log n)$-round concurrent zero knowledge argument for all languages in $\mathcal{NP}$ with precision $p(t) = O(t^{1+\varepsilon})$.*

Theorem 2 *Assuming the existence of one-way functions, for every $\varepsilon > 0$, there exists an $O(n^\varepsilon)$-round concurrent zero knowledge argument for all languages in $\mathcal{NP}$ with precision $p(t) = O(t 2^{\frac{2}{\epsilon}\log_n t})$. As a corollary, we obtain the following: For every $\epsilon > 0$, there exists an $O(n^\varepsilon)$-round protocol $\langle P, V\rangle$ such that for every $c > 0$, $\langle P, V\rangle$ is a concurrent zero knowledge argument with precision $p(n,t) = O(t)$ with respect to verifiers with running time bounded by n^c for all languages in $\mathcal{NP}$.*

Finally, we also construct statistically-precise concurrent ZK arguments for all of $\mathcal{NP}$, which requires simulation of *all* verifiers, even those having a priori unbounded running time.

Theorem 3 *Assume the existence of* claw-free permutations*, then there exists a poly(n)-round statistically precise concurrent zero-knowledge argument for all of $\mathcal{NP}$ with precision $p(t) = t^{1+\frac{1}{\log n}}$.*

As far as we know, this is the first (even non-precise) concurrent ZK protocol which handles verifiers participating in an unbounded number of executions. Previous work on statistical concurrent ZK also considers verifiers with an unbounded running-time; however, those simulations break down if the verifier can participate in a super-polynomial number of executions.

Our Techniques. Micali and Pass show that only trivial languages have black-box simulator with polynomial precision [MP06]. To obtain precise simulation, they instead "time" the verifier and then try to "cut off" the verifier whenever it attempts to run for too long. A first approach would be to adapt this technique to the simulators of [RK99,KP01,PRS02]. However, a direct application of this cut-off technique breaks down the correctness proof of these simulators.

To circumvent this problem, we instead introduce a new simulation technique, which rewinds the verifier *obliviously* based on *time*. In a sense, our simulator is not only oblivious of the content of the messages sent by the verifier (as the simulator by [KP01]), but also oblivious to when messages are sent by the verifier!

The way our simulator performs rewindings relies on the rewinding schedule of [KP01], and our analysis relies on that of [PRS02]. However, obtaining our results requires us to both modify and generalize this rewinding schedule and therefore also change the analysis. In fact, we cannot use the same rewinding schedule as KP/PRS as this yields at best a quadratic *worst-case* precision.

2 Definitions and Preliminaries

Notation. Let L denote an NP language and R_L the corresponding NP-relation. Let $(\mathcal{P},\mathcal{V})$ denote an interactive proof (argument) system where $\mathcal{P}$ and $\mathcal{V}$ are the prover and verifier algorithms respectively. By $\mathcal{V}^*(x, z, \bullet)$ we denote a non-uniform *concurrent adversarial* verifier with common input x and auxiliary input (or advice) z whose random coins are fixed to a sufficiently long string chosen uniformly at random; $\mathcal{P}(x, w, \bullet)$ is defined analogously where $w \in R_L(x)$.

Note that $\mathcal{V}^*$ is a *concurrent adversarial* verifier. Formally, it means the following. Adversary $\mathcal{V}^*$, given an input $x \in L$, interacts with an unbounded number of independent copies of $\mathcal{P}$ (all on common input x)[3]. An execution of a protocol between a copy of $\mathcal{P}$ and $\mathcal{V}^*$ is called a *session*. Adversary $\mathcal{V}^*$ can interact with all the copies at the *same* time (i.e., concurrently), interleaving messages from various sessions in any order it wants. That is, $\mathcal{V}^*$ has control over the scheduling of messages from various sessions. In order to implement a scheduling, $\mathcal{V}^*$ concatenates each message with the session number to which the next scheduled message belongs. The prover copy corresponding to that session then immediately replies to the verifier message as specified by the protocol. The *view* of concurrent adversary $\mathcal{V}^*$ in a concurrent execution consists of the common input x, the sequence of prover and verifier messages exchanged during the interaction, and the contents of the random tape of $\mathcal{V}^*$.

Let $\textsc{view}_{\mathcal{V}^*(x,z,\bullet)}$ be the random variable denoting the view of $\mathcal{V}^*(x, z, \bullet)$ in a *concurrent* interaction with the copies of $\mathcal{P}(x, w, \bullet)$. Let $\textsc{view}_{\mathcal{S}_{\mathcal{V}^*}(x,z,\bullet)}$ denote the view output by the simulator. When the simulator's random tape is *fixed* to r, its output is instead denoted by $\textsc{view}_{\mathcal{S}_{\mathcal{V}^*}(x,z,r)}$. Finally, let $T_{\mathcal{S}_{\mathcal{V}^*}(x,z,r)}$ denote the *steps* taken by the simulator and let $T_{\mathcal{V}^*}(\textsc{view})$ denote the steps taken by

[3] We remark that instead of a single fixed theorem x, $\mathcal{V}^*$ can be allowed to adaptively choose provers with different theorems x'. For ease of notation, we choose a single theorem x for all copies of $\mathcal{P}$. This is not actually a restriction and our results hold even when $\mathcal{V}^*$ adaptively chooses different theorems.

$\mathcal{V}^*$ in the view $\textsc{view}$. For ease of notation, we will use $\textsc{view}_{\mathcal{V}^*}$ to abbreviate $\textsc{view}_{\mathcal{V}^*(x,z,\bullet)}$, and $\textsc{view}_{\mathcal{S}_{\mathcal{V}^*}}$ to abbreviate $\textsc{view}_{\mathcal{S}_{\mathcal{V}^*}(x,z,\bullet)}$, whenever it is clear from the context.

Definition 1 (Precise Concurrent Zero Knowledge) *Let $p : N \times N \to N$ be a monotonically increasing function. $(\mathcal{P}, \mathcal{V})$ is a* concurrent zero knowledge *proof (argument) system with* precision p *if for every non-uniform* probabilistic polynomial time $\mathcal{V}^*$*, the following conditions hold:*

1. *For all $x \in L$, $z \in \{0,1\}^*$, the following distributions are computationally indistinguishable over L:*
   $\left\{\textsc{view}_{\mathcal{V}^*(x,z,\bullet)}\right\}$ *and* $\left\{\textsc{view}_{\mathcal{S}_{\mathcal{V}^*}(x,z,\bullet)}\right\}$
2. *For all $x \in L$, $z \in \{0,1\}^*$, and every sufficiently long $r \in \{0,1\}^*$, it holds that:*
   $T_{\mathcal{S}_{\mathcal{V}^*}(x,z,r)} \leq p(|x|, T_{\mathcal{V}^*}(\textsc{view}_{\mathcal{S}_{\mathcal{V}^*}(x,z,r)}))$.

When there is no restriction on the running time of $\mathcal{V}^*$ and the first condition requires the two distributions to be statistically close (resp., identical), we say $(\mathcal{P}, \mathcal{V})$ is statistical (resp., perfect) zero knowledge.

Next, we briefly describe some of the cryptographic tools used in our construction.

Special Honest Verifier Zero Knowledge (HVZK). A (three round) protocol is special-HVZK if, given the verifier's challenge in advance, the simulator can construct the first and the last message of the protocol such that the simulated view is computationally indistinguishable from the real view of an honest verifier. The Blum-Hamiltonicity protocol [Blu87] used in our construction is special-HVZK. When the simulated view is identical to the real view, we say the protocol is perfect-special-HVZK.

View Simulation. We assume familarity with the notion of "simulating the verifier's view". In particular, one can fix the random tape of the adversarial verifier $\mathcal{V}^*$ during simulation, and treat $\mathcal{V}^*$ as a deterministic machine.

Perfectly/Statistically Binding Commitments. We assume familiarity with "perfectly/statistically binding and computationally hiding" commitment schemes. Such commitment schemes are known based on the existence of one way function [Nao91,HILL99]. Naor's scheme has a two round commit phase where the first message is sent by the receiver. Thereafter, the sender can create the commitment using a randomized algorithm, denoted $c \leftarrow \textsc{com}_{\textsc{pb}}(v)$. The decommitment phase is only one round, in which the sender simply sends v and the randomness used, to the receiver. This will be denoted by $(v, r) \leftarrow \textsc{dcom}_{\textsc{pb}}(c)$. More on commitment schemes appears in the full version of this paper [PPS+07].

3 Our Protocol

We describe our Precise Concurrent Zero-Knowledge Argument, PCZK, in Figure 1. It is a slight variant of the PRS-protocol [PRS02]; in fact, the only difference is that we pad each verifier message with the string 0^l if our zero knowledge simulator (found in Figure 5) requires l steps of computation to produce the next message (l grows with the size of x). For simplicity, we use perfectly binding commitments in PCZK, although it suffices to use statistically binding commitments, which in turn rely on the existence of one way functions. The parameter k determines the round complexity of PCZK.

PCZK(k): A Protocol for Precise Concurrent Zero Knowledge Arguments.
All verifier messages are padded with the string 0^l where l is the running time required by our simulator (Figure 5) to compute the next prover message.

(Stage 1)

P0: Select the first message of the *perfectly binding* commitment scheme ($\text{COM}_{\text{PB}}, \text{DCOM}_{\text{PB}}$) and send it to $\mathcal{V}$.

V0: Select $\sigma \xleftarrow{R} \{0,1\}^n$ and set $\beta \leftarrow \text{COM}_{\text{PB}}(\sigma)$. Now select strings $\sigma^0_{i,j} \xleftarrow{R} \{0,1\}^n$ and $\sigma^1_{i,j} \leftarrow \{0,1\}^n$ such that $\sigma^0_{i,j} \oplus \sigma^1_{i,j} = \sigma$, for $i, j = 1, 2, \ldots, k$ (total $2k^2$ strings). Create commitments $\beta^b_{i,j} \leftarrow \text{COM}_{\text{PB}}(\sigma^b_{i,j})$ for all values of i, j and $b = 0, 1$. Send $\beta, \{\beta^0_{i,j}\}^k_{i,j=1}, \{\beta^1_{i,j}\}^k_{i,j=1}$, to $\mathcal{P}$.

For $j = 1, 2, \ldots, k$ proceed as follows.

Pj: Select $r_j \in \{0,1\}^k$ uniformly at random and send it to the verifier.

Vj: Let $r_{i,j}$ denote the i^{th} bit of r_j. Then, send $\sigma^0_{i,j}$ (resp., $\sigma^1_{i,j}$) and the decommitment information of $\beta^0_{i,j}$ (resp., $\beta^1_{i,j}$), if $r_{i,j} = 0$ (resp., if $r_{i,j} = 1$) to the prover.

(Stage 2)

p1: If $\mathcal{V}$ failed to properly decommit in Step Vj, for *any* $j \in [k]$, abort the protocol. Otherwise, run n parallel and *independent* copies of BH-prover (Figure 2) and send the n prover messages $\widehat{\text{p1}}$ to the verifier.

v1: Reveal the challenge σ and send decommitment information for *all* commitments which are unopened so far. Each bit of σ can be thought of as verifier's challenge in Step $\widehat{\text{v1}}$ of BH-protocol.

p2: Prover verifies that all the decommitments are proper and that $\sigma = \sigma^0_{i,j} \oplus \sigma^1_{i,j}$. If yes, execute the step $\widehat{\text{p2}}$ for each of the n parallel copies of the BH-protocol.

v2: Verify each of the n parallel proofs as described in $\widehat{\text{v2}}$. If all n $\widehat{\text{v2}}$ steps are accepting, accept the proof, otherwise reject the proof.

Fig. 1. Our Precise Concurrent Zero Knowledge Protocol.

Since our PCZK-protocol is just an instantiation of the PRS-protocol (with extra padding), it is both complete and sound.

The Blum-Hamiltonicity(BH) Protocol [Blu87].

$\widehat{\text{p1}}$: Choose a random permutation π of vertices V. Commit to the adjacency matrix of the permuted graph, denoted $\pi(G)$, and the permutation π, using a *perfectly binding* commitment scheme. Notice that the adjacency matrix of the permuted graph contains a 1 in position $(\pi(i), \pi(j))$ if $(i, j) \in E$. Send both the commitments to the verifier.

$\widehat{\text{v1}}$: Select a bit $\sigma \in \{0, 1\}$, called the *challenge*, uniformly at random and send it to the prover.

$\widehat{\text{p2}}$: If $\sigma = 0$, send π along with the decommitment information of *all* commitments. If $\sigma = 1$ (or anything else), decommit all entries $(\pi(i), \pi(j))$ with $(i, j) \in C$ by sending the decommitment information for the corresponding commitments.

$\widehat{\text{v2}}$: If $\sigma = 0$, verify that the revealed graph is identical to the graph $\pi(G)$ obtained by applying the revealed permutation π to the common input G. If $\sigma = 1$, verify that all the revealed values are 1 and that they form a cycle of length n. In both cases, verify that all the revealed commitments are correct using the decommitment information received. If the corresponding conditions are satisfied, accept the proof, otherwise reject the proof.

Fig. 2. The Blum-Hamiltonicity protocol used in PCZK

4 Our Simulator and its Analysis

4.1 Overview

At a high level, our simulator receives several opportunities to rewind the verifier and extract the "trapdoor" σ that will allow it to complete the simulation. More precisely, our simulator will attempt to rewind the verifier in one of the k "slots" (i.e. a message pair $\langle(\text{Pj}), (\text{Vj})\rangle$) in the first stage. If at any point it obtains the decommitment information for two different challenges (Pj), the simulator can extract the secret σ (that the verifier sends in the Stage 2) and simulate the rest of the protocol using the special-HVZK property of the BH-protocol.

To handle concurrency and precision, consider first the KP/PRS simulator. This simulator relies on a static and *oblivious* rewinding schedule, where the simulator rewinds the verifier after some fixed number of messages, independent of the message content. Specifically, the total number of verifier messages over all sessions are divided into two halves. The KP/PRS-rewinding schedule recursively invokes itself on each of the halves twice (completing two runs of the first half before proceeding to the two runs of the second half). The first run of each half is called the *primary* thread, and the latter is called the *secondary* thread. As shown in [KP01,PRS02], after the verifier commits to σ in any given session s, the KP/PRS-simulator gets several opportunities to extract it *before* Stage 2 of session s begins. We also call the thread of execution in the final output by the simulator the *main thread*. The KP/PRS-simulator keeps uses the secondary threads (recursively) as the main thread; all other threads, used to gather useful information for extracting σ, are called *look-ahead* threads. However, since the verifier's running time in look-ahead threads could be significantly longer than its running time in the main thread, the KP/PRS-simulator is not precise.

On the other hand, consider the precise simulation by Micali and Pass [MP06]. When rewinding a verifier, the MP simulator cuts off the second run of the verifier if it takes more time than the first run, and outputs the view of the verifier on the first run. Consequently, the running time of the simulator is proportional to the running time of the verifier on the output view. In order to apply the MP "cut" strategy on top of the KP/PRS-simulator, we need to use the primary thread (recursively) as the main output thread, and "cut" the secondary thread with respect to the primary thread. However, this cut-off will cause the simulator to abort more often, which significantly complicates the analysis.

To circumvent the above problems, we introduce a new simulation technique. For simplicity, we first present a simulator that knows an upper bound to the running-time of the verifier. Later, using a standard "doubling" argument, we remove this assumption. Like the KP/PRS-rewinding schedule, our simulator is oblivious of the verifier. But instead of rewinding based on the number of messages sent, we instead rewind based on the number of *steps* taken by the verifier (and thus this simulator is oblivious not only to the content of the messages sent by the verifier, but also to the time when these messages are sent!). In more detail, our simulator divides the total *running time* T of $\mathcal{V}^*$ into two halves and executes itself recursively on each half twice. In each half, we execute the primary and secondary threads in *parallel*. As we show later, this approach results in a simulation with quadratic precision.

To improve the precision, we further generalize the KP/PRS rewinding schedule. Instead of dividing T into *two* halves, we instead consider a simulator that divides T into g parts, where g is called the *splitting factor*. By choosing g appropriately, we are able to provide precision $p(t) \in O(t^{1+\epsilon})$ for every constant ϵ. Furthermore, we show how to achieve essentially *linear* precision by adjusting both k (the round complexity of our protocol) and g appropriately.

4.2 Worst Case Quadratic Simulation

We first describe a procedure that takes as input a parameter t and simulates the view of the verifier for t steps. The SIMULATE procedure described in Figure 3 employs the KP rewinding method with the changes discussed earlier. In Stage 1, SIMULATE simply generates uniformly random messages. SIMULATE attempts to extract σ using rewindings, and uses the special honest-verifier ZK property of the BH protocol to generate Stage 2 messages. If the extraction of σ fails, it outputs $\perp$. The parameter st is the state of $\mathcal{V}^*$ from which the simulation should start, and the parameter $\mathcal{H}$ is simply a global history of all "useful messages" for extracting σ.[4]

[4] For a careful treatment of $\mathcal{H}$, see [Ros04].

Let st_0 be the initial state of $\mathcal{V}^*$ and $d = d_t$ be the maximum recursion depth of $\textsc{simulate}(t, \mathsf{st}_0, \emptyset)$. The actual precise simulator constructed in the next section uses SIMULATE as a sub-routine, for which we show some properties below. In Proposition 2, we show that $\textsc{simulate}(t, \mathsf{st}_0, \emptyset)$ has a worst case running time of $O(t^2)$, and in Proposition 3 we show that SIMULATE outputs $\perp$ with negligible probability.

The $\textsc{simulate}(t, \mathsf{st}, \mathcal{H})$ Procedure.

1. If $t = 1$,
   (a) If the next scheduled message, p_u, is a first stage prover message, choose p_u uniformly. Otherwise, if p_u is a second stage prover message, compute p_u using the PROVE procedure (Figure 4). Feed p_u to the verifier. If the next scheduled message is verifier's message, run the verifier from its current state st for exactly 1 step. If an output is received then set $\mathrm{v}_u \leftarrow \mathcal{V}^*(\mathsf{hist}, \mathrm{p}_u)$. Further, if v_u is a first stage verifier message, store v_u in $\mathcal{H}$.
   (b) Update st to the current state of $\mathcal{V}^*$. Output $(\mathsf{st}, \mathcal{H})$.
2. Otherwise (i.e., $t > 1$),
   (a) Execute the following two processes in *parallel*:
      i. $(\mathsf{st}_1, \mathcal{H}_1) \leftarrow \textsc{simulate}(t/2, \mathsf{st}, \mathcal{H})$. (primary process)
      ii. $(\mathsf{st}_2, \mathcal{H}_2) \leftarrow \textsc{simulate}(t/2, \mathsf{st}, \mathcal{H})$. (secondary process)

      Merge $\mathcal{H}_1$ and $\mathcal{H}_2$. Set the resulting table equal to $\mathcal{H}$.
   (b) Next, execute the following two processes in *parallel*, starting from st_1,
      i. $(\mathsf{st}_3, \mathcal{H}_3) \leftarrow \textsc{simulate}(t/2, \mathsf{st}_1, \mathcal{H})$. (primary process)
      ii. $(\mathsf{st}_4, \mathcal{H}_4) \leftarrow \textsc{simulate}(t/2, \mathsf{st}_1, \mathcal{H})$[5]. (secondary process)
   (c) Merge $\mathcal{H}_3$ and $\mathcal{H}_4$. Set the resulting table equal to $\mathcal{H}$.
      Output $(\mathsf{st}_3, \mathcal{H})$ and the view of $\mathcal{V}^*$ on the thread connecting st, st_1, and st_3.

Fig. 3. The time-based oblivious simulator

Proposition 2 (Running Time of simulate) $\textsc{simulate}(t, \cdot, \cdot)$ *has worst-case running time $O(t^2)$.*

Proof We partition the running time of SIMULATE into the time spent emulating $\mathcal{V}^*$, and the time spent simulating the prover (i.e. generating prover messages). By construction, $\textsc{simulate}(t, \cdot, \cdot)$ spends time at most t emulating $\mathcal{V}^*$ on main thread. Furthermore, the number of parallel executions double per level of recursion. Thus, the time spent in simulating $\mathcal{V}^*$ by $\textsc{simulate}(t, \cdot, \cdot)$ is $t \cdot 2^d$, where the d is the maximum depth of recursion. Since $d = d_t = \lceil \log_2 t \rceil \leq 1 + \log_2 t$, we conclude that SIMULATE spends at most $2t^2$ steps emulating $\mathcal{V}^*$. To compute

[5] In the case where t does not divide evenly into two, we use $\lfloor t/2 \rfloor + 1$ in step (2a), and $\lfloor t/2 \rfloor$ in step (2b).

The PROVE Procedure.

Let $s \in [m]$ be the session for which the prove procedure is invoked. The procedure outputs either p1 or p2, whichever is required by $\mathcal{S}_{\mathcal{V}^*}$. Let hist denote the set of messages exchanged between $\mathcal{S}_{\mathcal{V}^*}$ and $\mathcal{V}^*$ in the *current* thread. The PROVE procedure works as follows.

1. If the verifier has aborted in any of the k first stage messages of session s (i.e., hist contains Vj=`ABORT` for $j \in [k]$ of session s), abort session s.
2. Otherwise, search the table $\mathcal{H}$ to find values $\sigma^0_{i,j}, \sigma^1_{i,j}$ belonging to session s, for some $i, j \in [k]$. If no such pairs are found, output $\perp$ (indicating failure of the simulation). Otherwise, extract the challenge $\sigma = \sigma_1\sigma_2 \dots \sigma_n$ as $\sigma^0_{i,j} \oplus \sigma^1_{i,j}$, and proceed as follows.
   (a) If the next scheduled message is p1, then for each $h \in [n]$ act as follows. If $\sigma_h = 0$, act according to Step $\widehat{\text{p1}}$ of BH-protocol. Otherwise (i.e., if $\sigma_h = 1$), commit to the entries of the adjacency matrix of the complete graph K_n and to a random permutation π.
   (b) Otherwise (i.e., the next scheduled message is p2), check (in hist) that the verifier has properly decommitted to all relevant values (and that the h^{th} bit of $\sigma^0_j \oplus \sigma^1_j$ equals σ_h for all $j \in [k]$) and abort otherwise.
   For each $h \in [n]$ act as follows. If $\sigma_h = 0$, decommit to all the commitments (i.e., π and the adjacency matrix). Otherwise (i.e., if $\sigma_h = 1$), decommit only to the entries $(\pi(i), \pi(j))$ with $(i, j) \in C$ where C is an arbitrary Hamiltonian cycle in K_n.

Fig. 4. The PROVE Procedure used by SIMULATE for Stage 2 messages

the time spent simulating the prover, recall that the verifier pads each messages with 0^l if the SIMULATE requires l steps of computation to generate the next message. Therefore, SIMULATE always spends less time simulating the prover than $\mathcal{V}^*$ giving us a bound of $2 \cdot 2t^2 = 4t^2$ on the total running time. □

Proposition 3 *The probability that* SIMULATE *outputs* $\perp$ *is negligible in* n.

Proof The high-level structure of our proof follows the proof of PRS. We observe that SIMULATE outputs $\perp$ only when it tries to generate Stage 2 messages. We show in Lemma 4 that for each session, the probability of outputting $\perp$ for the first time on any thread is negligible. Since SIMULATE only runs for polynomial time, there are at most polynomial sessions and threads.[6] Therefore, we conclude using the union bound that SIMULATE outputs $\perp$ with negligible probability.

Lemma 4 *For any session* s_0 *and any thread* l_0 *(called the reference session and the reference thread), the probability that session* s_0 *and thread* l_0 *is the first time* SIMULATE *outputs* $\perp$ *is negligible.*

[6] We will reexamine this claim in section 5, where simulation time is (a priori) unbounded.

Proof Recall that for SIMULATE to extract σ, $\mathcal{V}^*$ needs to reply to two *different* challenges (Pj) with corresponding (Vj) messages ($j \geq 1$) (after $\mathcal{V}^*$ has already committed to σ). Since SIMULATE generates only polynomially many uniformly random (Pj) messages, the probability of any two challenge being identical is exponentially small in n. Therefore, it is sufficient to bound the probability conditioned on SIMULATE never repeating the same challenge.[7]

We now proceed using a *random-tape counting* argument similar to PRS. For a fixed session s_0 and thread l_0, we call a random tape ρ *bad*, if running SIMULATE with that random tape makes it output $\perp$ first on session s_0 in thread l_0. The random tape is called good otherwise. As in PRS, we show that every bad random tape can be mapped to a set of super-polynomially many good random tapes. Furthermore, this set of good random tapes is unique. Such a mapping implies that the probability of a random tape being bad is negligible. Towards this goal, we provide a mapping f that takes a bad random tape to a set of good random tapes.

To construct f, we need some properties of good and bad random tapes. We call a *slot* (i.e. a message pair $\langle (\mathrm{Pj}), (\mathrm{Vj}) \rangle$) *good* if the verifier does not `ABORT` on this challenge. Then:

1. When SIMULATE uses a *bad* random tape, all k slots of session s_0 on thread l_0 are good. (Otherwise, SIMULATE can legitimately abort session s_0 without outputting $\perp$.)
2. A random tape is *good* if there is a good slot such that (1) it is on a non-reference thread $l \neq l_0$, (2) it occurs after $\mathcal{V}^*$ has committed to σ with message (V0) on thread l_0, and (3) it occurs before the Stage 2 message (p1) takes place on thread l_0. This good slot guarantees that SIMULATE can extract σ if needed.

Properties 1 and 2 together give the following insight: Given a bad tape, "moving" a good slot from the reference thread l_0 to a non-reference thread produces a good random tape. Moreover, the rewind-schedule of SIMULATE enables us to "swap" slots across threads by swapping segments of SIMULATE's random tape. Specifically, whenever SIMULATE splits into primary and secondary processes, the two processes share the same start state, and are simulated for the same number of steps in parallel; swapping their random tapes would swap the simulation results on the corresponding threads[8].

We define a *rewinding interval* to be a recursive execution of SIMULATE on the reference thread l_0 that contains a slot, i.e. a $\langle (\mathrm{Pj}), (\mathrm{Vj}) \rangle$-pair, but does not contain the initial message (V0) or the Stage 2 message (p1). A *minimal rewinding interval* is defined to be a rewinding interval where none of its children intervals (i.e. smaller recursive executions of SIMULATE on l_0) contain the same slot (i.e. both (Pj) and (Vj)). Following the intuition mentioned above, swapping

[7] As in footnote 6, we will reexamine this claim in section 5, where simulation time is unbounded.

[8] $\mathcal{V}^*$ is assumed to be deterministic.

the randomness of a rewinding interval with its corresponding intervals on non-reference threads will generate a good tape (shown in Claim 3).

We next construct the mapping f to carry out the swapping of rewinding intervals in a structured way. Intuitively, f finds disjoint subsets of minimal rewinding intervals and performs the swapping operation on them. The f we use here is exactly the same mapping constructed in PRS (see Figure 5.4 of [Ros04], or the appendix for a more detailed description). Even though our simulator differs from that of PRS, the mapping f works on any simulator satisfying the following two properties: (1) Each rewinding is executed twice. (2) Any two rewindings are either disjoint or one is completely contained in the other.

We proceed to give four properties of f. Claim 1 bounds the number of random tapes produced by f based on the number of minimal rewinding intervals, while Claim 2 shows that f maps different bad tapes to disjoint sets of tapes. Both these properties of f syntactically follows by using the same proof of PRS for any simulator that satisfy the two properties mentioned above and we inherit them directly. In the following claims, ρ denotes a bad random tape.

Claim 1 (f produces many tapes) $|f(\rho)| \geq 2^{k'-d}$*, where* k' *is the number of minimal rewinding intervals and* d *is the maximum number of intervals that can overlap with each other.*

Remark: We reuse the symbol d since the maximum number of intervals that can overlap each other is just the maximum depth of recursion.

Claim 2 (f produces disjoint sets of tapes) *If* $\rho' \neq \rho$ *is another bad tape,* $f(\rho)$ *and* $f(\rho')$ *are disjoint.*

Proof These two claims were the crux of [PRS02] [Ros04]. See Claim 5.4.12 and Claim 5.4.11 in [Ros04], for more details. We remark that Claim 1 is proved with an elaborate counting argument. Claim 2, on the other hand, is proved by constructing an “inverse” of f based on the following observation. On a bad tape, good slots appear only on the reference thread l_0. Therefore, given a tape produced by f, one can locate the minimal intervals swapped by f by searching for good slots on non-reference threads, and invert those swappings. $\square$

In Claim 3 we show that, the tapes produced by f are good, while Claim 4 counts the number of minimal rewinding intervals. These two claims depend on how SIMULATE recursively calls itself and hence we cannot refer to PRS for the proof of these two claims; nevertheless, they do hold with respect to our simulator as we prove below.

Claim 3 (f produces good tapes) *The set* $f(\rho)\backslash\{\rho\}$ *contains only good tapes (for* SIMULATE*).*

Proof This claim depends on the order in which simulate executes its recursive calls, since that in turn determines when σ extracted. The proof of this claim by PRS (see Claim 5.4.10 in [Ros04]) requires the main thread of the simulator to be executed after the look-ahead threads. SIMULATE, however, runs the two executions in parallel. Nevertheless, we provide an alternative proof that handles such a parallel rewinding.

Consider $\rho' \in f(\rho)$, $\rho' \neq \rho$. Let I be the first minimal rewinding interval swapped by f, and let J be the corresponding interval where I is swapped to. Since I is the first interval to be swapped, the contents of I and J are exchanged on ρ' (while later intervals may be entirely changed due to this swap). Observe that after swapping, the $\langle(\text{Pj}), (\text{Vj})\rangle$ message pair that originally occurred in I will now appear on a non-reference thread inside J. Now, there are two cases depending on J:

Case 1: J does not contain the first Stage 2 message (p1) before the swap. After swapping the random tapes, (p1) would occur on the reference thread after executing both I and J. By property 2, we arrive at a good tape.

Case 2: J contains the first Stage 2 message (p1) before the swap. By the definition of a bad random tape, SIMULATE gets stuck for the first time on the reference thread after I and J are executed; Consequently, after swapping the random tape, SIMULATE will not get stuck during I. SIMULATE also cannot get stuck later on thread l_0, again due to property 2. In this case, we also arrive at a good tape.

□

Claim 4 *There are at least $k' = k - 2d$ minimal rewinding intervals for session s_0 on thread l_0 (for* SIMULATE*).*

Proof This claim depends on the number of recursive calls made by SIMULATE. For now, SIMULATE$(t, \cdot, \cdot)$ splits t into two halves just like in PRS, thus this result follows using the same proof as in PRS. Later, in Claim 7, we provide a self-contained proof of this fact in a more general setting. □

Concluding proof of Lemma 4: It follows from Claims 1, 2, 3 and 4 that every bad tape is mapped to a unique set of at least 2^{k-3d} good random tapes. Hence, the probability that a random tape is bad is at most

$$\frac{1}{2^{k-3d}}$$

Recall that $d = \lceil \log_2 T \rceil \in O(\log n)$, since T is a polynomial in n. Therefore, the probability of a bad tape occurring is negligible if $k \in \omega(\log n)$. □

This concludes the proof of Proposition 3. □

4.3 Precise Quadratic Simulation

Recall that SIMULATE takes as input t, and simulates the verifier for t steps. Since the actual simulator $\mathcal{S}_{\mathcal{V}^*}$ (described in Figure 5) does not know a priori the running time of the verifier, it calls SIMULATE with increasing values of $\hat{t}$, doubling every time SIMULATE returns an incomplete view. On the other hand, should SIMULATE ever output $\perp$, $\mathcal{S}_{\mathcal{V}^*}$ will immediately output $\perp$ as will and terminate. Also, $\mathcal{S}_{\mathcal{V}^*}$ runs SIMULATE with two random tapes, one of which is used exclusively whenever SIMULATE is on the main thread. Since, $\mathcal{S}_{\mathcal{V}^*}$ uses the same tape every time it calls SIMULATE, the view of $\mathcal{V}^*$ on the main thread proceeds identically in all the calls to SIMULATE.

$\mathcal{S}_{\mathcal{V}^*}(\rho_1, \rho_2)$, where ρ_1 and ρ_2 are random tapes.

1. Set $\hat{t} = 1, \mathsf{st} =$ initial state of $\mathcal{V}^*, \mathcal{H} = \emptyset$.
2. While SIMULATE did not generate a full view of $\mathcal{V}^*$:
   (a) $\hat{t} \leftarrow 2\hat{t}$
   (b) run SIMULATE$(\hat{t}, \mathsf{st}, \emptyset, (\rho_1, \rho_2))$, where random tape ρ_1 is exclusively used to simulate the verifier on the main thread, and random tape ρ_2 is used for all other threads.
   (c) output $\perp$ if SIMULATE outputs $\perp$
3. Output the full view $\mathcal{V}^*$ (i.e., random coins and messages exchanged) generated on the final run of SIMULATE$(\hat{t}, \mathsf{st}, \emptyset)$

Fig. 5. The Quadratically Precise Simulator.

Lemma 5 (Concurrent Zero Knowledge) *The ensembles* $\{\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}}(x, z)\}_{x \in L, z \in \{0,1\}^*}$ *and*
$\{\text{VIEW}_{\mathcal{V}^*}(x, z)\}_{x \in L, z \in \{0,1\}^*}$ *are computationally indistinguishable over* L.

Proof We consider the following "intermediate" simulator $\mathcal{S}'$ that on input x (and auxiliary input z), proceeds just like $\mathcal{S}$ (which in turn behaves like an honest prover) in order to generate messages in Stage 1 of the view. Upon entering Stage 2, $\mathcal{S}'$ outputs $\perp$ if $\mathcal{S}$ does; otherwise, $\mathcal{S}'$ proceeds as an honest prover in order to generate messages in Stage 2 of the view. Indistinguishability of the simulation by $\mathcal{S}$ then follows from the following two claims:

Claim 5 *The ensembles* $\{\text{VIEW}_{\mathcal{S}'_{\mathcal{V}^*}}(x, z)\}_{x \in L, z \in \{0,1\}^*}$ *and* $\{\text{VIEW}_{\mathcal{V}^*}(x, z)\}_{x \in L, z \in \{0,1\}^*}$ *are statistically close over* L.

Proof We consider another intermediate simulator $\mathcal{S}''$ that proceeds identically like $\mathcal{S}'$ except that whenever $\mathcal{S}'$ outputs $\perp$ in a Stage 2 message, $\mathcal{S}'$ instead continues simulating like an honest prover. Essentially, $\mathcal{S}''$ never fails. Since $\mathcal{S}''$ calls SIMULATE for several values of t, this can skew the distribution. However, recall that the random tape fed by $\mathcal{S}''$ into SIMULATE to simulate the view on the main thread is identical for every call. Therefore, the view on the main thread of SIMULATE proceeds identically in every call to SIMULATE. Thus, it follows from the fact that the Stage 1 messages are generated uniform at random and that $\mathcal{S}''$ proceeds as the honest prover in Stage 2, the view output by $\mathcal{S}''$ and the view of $\mathcal{V}^*$ are identically distributed.

It remains to show that view output by $\mathcal{S}'$ and $\mathcal{S}''$ are statistically close over L. The only difference between $\mathcal{S}'$ and $\mathcal{S}''$ is that $\mathcal{S}'$ outputs $\perp$ sometimes. It suffices to show that $\mathcal{S}'$ outputs $\perp$ with negligible probability. From Proposition 3, we know that SIMULATE outputs $\perp$ only with negligible probability. Since SIMULATE is called at most logarithmically many times due to the doubling of t, using the union bound we conclude that $\mathcal{S}'$ outputs $\perp$ with negligible probability. $\square$

Claim 6 *The ensembles* $\{\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}}(x,z)\}_{x \in L, z \in \{0,1\}^*}$ *and* $\{\text{VIEW}_{\mathcal{S}'_{\mathcal{V}^*}}(x,z)\}_{x \in L, z \in \{0,1\}^*}$ *are computationally indistinguishable over L.*

Proof The only difference between $\mathcal{S}$ and $\mathcal{S}'$ is in the manner in which the Stage 2 messages are generated. Indistinguishability follows from the special honest-verifier ZK property using a standard hybrid argument, as given below.

Assume for contradiction that there exists a verifier V^*, a distinguisher D and a polynomial $p(\cdot)$ such that D distinguishes the ensembles $\{\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}}(x,z)\}$ and $\{\text{VIEW}_{\mathcal{S}'_{\mathcal{V}^*}}(x,z)\}$ with probability $\frac{1}{p(n)}$. Furthermore, let the running time of V^* be bounded by some polynomial $q(n)$. We consider a sequence of hybrid simulators, S_i for $i = 0$ to $q(n)$. S_i proceeds exactly like S, with the exception that in the first i proofs that reach the second stage, it proceeds using the honest prover strategy in the second stage for those proofs. By construction $S_0 = S$ and $S_{q(n)} = \mathcal{S}'$ (since there are at most $q(n)$ sessions, bounded by the running time of the simulators). By assumption the output of S_0 and $S_{q(n)}$ are distinguishable with probability $\frac{1}{p(n)}$, so there must exist some j such that the output of S_j and S_{j+1} are distinguishable with probability $\frac{1}{p(n)q(n)}$. Furthermore, since S_j proceeds exactly as S_{j+1} in the first j sessions that reach the second stage, and by construction they proceed identically in the first stage in all sessions, there exists a partial view v of S_j and S_{j+1}—which defines an instance for the protocol in the second stage of the $j+1$ session—such that the output of S_j and S_{j+1} are distinguishable, conditioned on the event that S_j and S_{j+1} feed V^* the view v. Since the only difference between the view of V^* in S_j and S_{j+1} is that the former is a simulated view, while the later is a view generated using an honest prover strategy, this contradicts the special honest-verifier ZK property of the BH-protocol in the second stage of the protocol. $\square$ $\square$

Lemma 6 (Quadratic Precision) *Let* $\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}}$ *be the output of the simulator* $\mathcal{S}_{\mathcal{V}^*}$*, and t be the running time of* $\mathcal{V}^*$ *on the view* $\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}}$*. Then,* $\mathcal{S}_{\mathcal{V}^*}$ *runs in time* $O(t^2)$.

Proof Recall that, $\mathcal{S}_{\mathcal{V}^*}$ runs SIMULATE with increasing values of $\hat{t}$, doubling each time, until a view is output. We again use the fact that the view on the main thread of SIMULATE proceeds identically (in this case, proceeds as $\text{VIEW}_{\mathcal{S}_{\mathcal{V}^*}}$) since the random tape used to simulate the main thread is identical in every call to SIMULATE. Therefore, the final value of $\hat{t}$ when v is output satisfies,

$$t \leq \hat{t} < 2t$$

The running time of $\mathcal{S}_{\mathcal{V}^*}$ is simply the sum of the running times of $\textsc{simulate}(t, \mathsf{st}, \emptyset)$ with $t = 1, 2, 4, \ldots, \hat{t}$. By Lemma 2, this running time is bounded by

$$c1^2 + c2^2 + c4^2 + \cdots + c\hat{t}^2 \leq 2c\hat{t}^2 \leq 8ct^2$$

For some constant c. $\square$

4.4 Improved Precision

We now consider a generalized version of $\textsc{simulate}$. Let $g \geq 2$ be an integer; $\textsc{simulate}_g(t, \cdot, \cdot)$ will now divide t in g smaller intervals. If t does not divide into g evenly, that is if $t = qg + r$ with $r > 0$, let the first r sub-intervals have length $\lfloor t/g \rfloor + 1$, and the rest of the $g - r$ sub-intervals have length $\lfloor t/g \rfloor$. We call g the splitting factor, and assume $k/g \in \omega(\log n)$ as stated in Theorem 1. Due to the lack of space most of the details of this section are given in the full version [PPS$^+$07]. We only state our main claims here.

In the full version, we demonstrate that the running time of our new simulator is given by the following lemma.

Lemma 7 (Improved Precision) *Let* $\textsc{view}_{\mathcal{S}_{\mathcal{V}^*}}$ *be the output of the simulator* $\mathcal{S}_{\mathcal{V}^*}$ *using* $\textsc{simulate}_g$*, and* t *be the running time of* $\mathcal{V}^*$ *on the view* $\textsc{view}_{\mathcal{S}_{\mathcal{V}^*}}$*. Then,* $\mathcal{S}_{\mathcal{V}^*}$ *runs in time* $O(t \cdot 2^{\log_g t}) = O(t^{1+\log_g 2})$*.*

Thereafter, we show there the indistinguishability of the simulator's output.

Lemma 8 (Concurrent Zero Knowledge) $\{\textsc{view}_{\mathcal{S}_{\mathcal{V}^*}}(x, z)\}_{x \in L, z \in \{0,1\}^*}$ *and* $\{\textsc{view}_{\mathcal{V}^*}(x, z)\}_{x \in L, z \in \{0,1\}^*}$ *are computationally indistinguishable over* L*.*

Finally, in order to deduce our main lemma, we demonstrate the following important claim regarding the number of minimum intervals with respect to our new simulator. This claim is analogous to claim 4. It, however, depends on the splitting factor g and is modified as follows:

Claim 7 (Number of Minimal Rewinding Intervals) *There are at least* $k' = \frac{k}{g-1} - 2d$ *minimal rewinding intervals for session* s_0 *on thread* l_0 *(for* $\textsc{simulate}_g$*), where* d *is the recursion depth.*

The main use of this lemma is in deducing that our new simulator outputs $\perp$ with only negligible probability.

4.5 Proof of Main Lemma and Consequences

Lemma 1 (Main Lemma) *Assuming the existence of one-way functions, then for every* $k, g \in \mathbb{N}$ *such that* $k/g \in \omega(\log n)$*, there exists an* $O(k)$*-round concurrent zero knowledge argument with precision* $p(t) \in O(t \cdot 2^{\log_g t})$ *for all languages in* $\mathcal{NP}$*.*

Proof Using Lemmata 7 and 8, we conclude that the simulator $\mathcal{S}_{\mathcal{V}^*}$ (using SIMULATE$_g$) outputs a verifier view of the right distribution with precision $O(t \cdot 2^{\log_g t})$. □

By setting $g = 2^{1/\varepsilon}$ and $k \in \omega(\log n)$ in our main lemma, we get our first theorem.

Theorem 1 *Assuming the existence of one-way functions, for every $\varepsilon > 0$, there exists a $\omega(\log n)$-round concurrent zero knowledge argument for all languages in $\mathcal{NP}$ with precision $p(t) = O(t^{1+\varepsilon})$.*

Finally, by setting $g = n^{\epsilon/2}$ and $k = n^{\epsilon}$ in our main lemma, we get our next theorem.

Theorem 2 *Assuming the existence of one-way functions, for every $\varepsilon > 0$, there exists an $O(n^{\varepsilon})$-round concurrent zero knowledge argument for all languages in $\mathcal{NP}$ with precision $p(t) = O(t 2^{\frac{2}{\epsilon}\log_n t})$. As a corollary, we obtain the following: For every $\epsilon > 0$, there exists an $O(n^{\varepsilon})$-round protocol $\langle P, V\rangle$ such that for every $c > 0$, $\langle P, V\rangle$ is a concurrent zero knowledge argument with precision $p(n,t) = O(t)$ with respect to verifiers with running time bounded by n^c for all languages in $\mathcal{NP}$.*

5 Statistically Precise Concurrent Zero-Knowledge

In this section, we construct a statistically precise concurrent ZK argument for all of $\mathcal{NP}$. Recall that statistically precise ZK requires the simulation of all malicious verifiers (even those having a priori unbounded running time) and the distribution of the simulated view must be statistically close to that of the real view. A first approach is to use the previous protocol and simulator with the splitting factor fixed appropriately. However this approach does not work directly; briefly the reason being that we will need k to superpolynomial in n. We thus present a slightly modified simulator, which appears shortly.

Theorem 3 *Assume the existence of* claw-free permutations*, then there exists a poly(n)-round statistically precise concurrent zero-knowledge argument for all of $\mathcal{NP}$ with precision $p(n,t) = O(t^{1+\frac{1}{\log n}})$.*

Description of protocol: We essentially use the same protocol described in Section 3 setting the number of rounds $k = 5n^2 \log n$ (n is the security parameter), with the following exception: In Stage 2 of the protocol, the prover uses perfectly hiding commitments in the BH-protocol instead of computational hiding. This makes the BH-protocol perfect-special-HVZK.

Description of simulator S: The simulator S executes $\mathcal{S}_{\mathcal{V}^*}$ with $g = n$ and outputs whatever $\mathcal{S}_{\mathcal{V}^*}$ outputs, with the following exception: while executing SIMULATE$_n$ (inside $\mathcal{S}_{\mathcal{V}^*}$), if the verifier in the main thread runs for more than $2^{n \log_2 n}$ steps, it terminates the execution of SIMULATE$_n$ and retrieves the partial history hist simulated in the main thread so far. Then, it continues to simulate the verifier from hist in a "straight-line" fashion—it generates uniformly random messages for the Stage 1 of the protocol, and when it reaches Stage 2 of the protocol for some session, it runs the brute-force-PROVE procedure, given in the full version. An analysis of this simulator appears in [PPS+07].

Acknowledgements. We would like to thank Alon Rosen for several helpful discussions.

References

[Blu87] Manuel Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, pages 1444–1451, 1987.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero knowledge. In *Proc.* 30*th STOC*, pages 409–418, 1998.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc.* 17*th STOC*, pages 291–304, 1985.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In ACM, editor, *Proc.* 19*th STOC*, pages 218–229, 1987. See [Gol04, Chap. 7] for more details.

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

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications.* Cambridge University Press, 2004.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999. Preliminary versions appeared in STOC' 89 and STOC' 90.

[KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in poly-logarithm rounds. In *Proc.* 33*th STOC*, pages 560–569, 2001. Preliminary full version published as cryptology ePrint report 2000/013.

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In Jon M. Kleinberg, editor, *STOC*, pages 306–315. ACM, 2006.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991. Preliminary version in CRYPTO' 89.

[Pas06] Rafael Pass. *A Precise Computational Approach to Knowledge*. PhD thesis, MIT, July 2006.

[PPS+07] Omkant Pandey, Rafael Pass, Amit Sahai, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Precise concurrent zero knowledge. Cryptology ePrint Archive, Report 2007/451, 2007. `http://eprint.iacr.org/2007/451.pdf`.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *Proc.* 43*rd FOCS*, 2002.

[RK99] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In *Eurocrypt '99*, pages 415–432, 1999.

[Ros04] Alon Rosen. *The Round-Complexity of Black-Box Concurrent Zero-Knowledge*. PhD thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 2004.