Eye for an Eye: Efficient Concurrent Zero-Knowledge in the Timing Model

Rafael Pass*, Wei-Lung Dustin Tseng**, and Muthuramakrishnan Venkitasubramaniam

Cornell University, NY, USA

Abstract. We present new and efficient concurrent zero-knowledge protocols in the timing model. In contrast to earlier works—which through artificially-imposed delays require *every* protocol execution to run at the speed of the *slowest* link in the network—our protocols essentially only delay messages based on the *actual* response time of each verifier (which can be significantly smaller).

1 Introduction

Zero-knowledge (ZK) interactive proofs [GMR89] are paradoxical constructs that allow one player (called the prover) to convince another player (called the verifier) of the validity of a mathematical statement $x \in L$, while providing *zero additional knowledge* to the verifier. This is formalized by requiring that for every PPT adversary verifier V^*, there is a PPT *simulator* S that can simulate the view of V^* interacting with the honest prover P. The idea behind this definition is that whatever V^* might have learned from interacting with P could have been learned by simply running the simulator S. The notion of concurrent ZK (cZK), first introduced and achieved by Dwork, Naor and Sahai [DNS04] extends the notion of ZK protocols to a concurrent and asynchronous setting. More precisely, we consider a single adversary mounting a coordinated attack by acting as a verifier in many concurrent sessions, possibly with many independent provers. cZK protocols are significantly harder to construct and analyze, and are often less efficient than the "standalone" ZK protocols.

The original constant-round cZK protocol of [DNS04] is constructed in the timing model (also explored in [Gol02]). Informally speaking, the timing model assumes that every party (in our case every honest prover) has a local clock, and that all these local clocks are roughly synchronized (1 second is roughly the same on every clock). Also, all parties know a (pessimistic) upper-bound, Δ, on the time it takes to deliver a message on the network. As argued by Goldreich [Gol02], this assumption seems to be most reasonable for systems

* Supported in part by a Microsoft New Faculty Fellowship, NSF CAREER Award CCF-0746990, AFOSR Award FA9550-08-1-0197 and BSF Grant 2006317.

** Supported in part by a NSF Graduate Research Fellowship.

today. The problem, however, is that known constructions of cZK protocols in the timing model [DNS04,Gol02] are not very efficient in terms of execution time: Despite having a constant number of rounds (4 or 5 messages), the prover in these protocols delays the response of certain messages by time Δ. In other words, every instance of the protocol must take time longer than the pessimistic bound on the max latency of the network (rather than being based on the actual message-delivery time).

Leaving the timing model, Richardson and Kilian [RK99] (and subsequent improvments by Kilian and Petrank [KP01] and Prabhakaran, Rosen and Sahai [PRS02]) show how to construct cZK protocols in the standard model (without clocks). Here the protocols are "message-delivery" driven, but there is a significant increase in round-complexity: Whereas constant-round ZK protocols exists in the standalone setting, $\tilde{O}(\log n)$-rounds are both necessary and sufficient for (black-box) cZK protocols [PRS02,KPR98,Ros00,CKPR01]. Another related work of Pass and Venkitasubramaniam [PV08] gives a constant-round cZK protocol without clocks, but at the expense of having quasi-polynomial time simulators (against quasi-polynomial time adversaries).

In this work we revisit the timing model. Ideally, we want to construct cZK protocols that are efficient in all three manners mentioned so far: Small (constant) round-complexity, low imposed delays, and fast simulation. As communicated by Goldreich [Gol02], Barak and Micciancio suggested the following possible improvement to cZK protocols in the timing model: The prover may only need to impose a delay δ that is a linear fraction of Δ (say $\delta = \Delta/d$), at the expense of increasing the running time of the ZK simulator exponentially (around $n^{O(d)}$). In other words, there could be a compromise between protocol efficiency and knowledge security [Gol01,MP06] (i.e., simulator running-time). However, as discussed in [Gol02], this suggestion has not been proven secure. We show that such a trade-off is not only possible, but can be significantly improved.[1]

Trading rounds for minimum delays. The original work of Richardson and Kilian [RK99] shows that increasing the number of communication rounds can decrease the running-time of the simulator. Our first result shows that by only slightly increasing the number of rounds, but still keeping it constant (e.g., 10 messages), the prover may reduce the imposed delay to $\delta = \Delta/2^d$, while keeping the simulator running time at $n^{O(d)}$. This is accomplished by combining simulation techniques from both the timing model [DNS04,Gol02] (polynomial time simulation but high timing constraints) and the standard model [RK99,PV08] (quasi-polynomial time simulation but no timing constraints). As far as we know, this yeilds the first formal proof that constant-round concurrent zero-knowledge protocols are possible using a delay δ that is smaller than Δ.

"Eye-for-an-eye" delays. The traditional approach for constructing cZK protocols is to "penalize" all parties equally, whether it is in the form of added round

[1] It seems that traditional techniques can be used to demonstrate the Barak-Micciancio trade-off when the adversary employs a *static* scheduling of messages. However, complications arise in the case of *adaptive* schedules. See Section 3.1 for more details.

complexity or imposed timing delays. One may instead consider the notion of punishing only adversarial behaviour, similar to the well-known "tit-for-tat" or "eye-for-an-eye" technique of game theory (see e.g., [Axe84]). The work of Cohen, Kilian and Petrank [CKP01] first implemented such a strategy (with respect to cZK) using an iterated protocol where in each iteration, the verifier is given a time constraint under which it must produce all of its messages; should a verifier exceed this constraint, the protocol is restarted with doubled the allowed time constraint (the punishment here is the resetting); their protocol had $\tilde{O}(log^2 n)$ rounds and $\tilde{O}(\log n)$ "responsive complexity"—namely, the protocol takes time $\tilde{O}(\log n)T$ to complete if each verifier message is sent within time T. The work of Persiano and Visconti [PV05] and Rosen and shelat [Rs09] takes a different approach and punish adversaries that perform "bad" schedulings of messages by adaptively adding more rounds to the protocol; their approaches, however, only work under the assumption that there is a *single* prover, or alternatively that all messages on the network are exposed on a broadcast channel (so that the provers can check if a problematic scheduling of messages has occurred).

In our work, we instead suggest the following simple approach: Should a verifier provide its messages with delay t, the prover will delay its message accordingly so that the protocol completes in time $p(t)+\delta$, where p is some *penalty* function and δ is some small minimal delay. We note that, at a high-level, this approach is somewhat reminiscent of how message delivery is performed in TCP/IP.

As we show, such penalty-based *adaptive delays* may significantly improve the compromise between protocol efficiency and knowledge security. For example, setting $p(t) = 2t$ (i.e., against a verifier that responds in time $t < \Delta$, the prover responds in time $t + \delta$) has a similar effect as increasing the number of rounds: The prover may reduce the minimal imposed delay to $\delta = \Delta/2^d$, while keeping the simulator running time at $n^{O(d)}$. Moreover, if we are willing to use more aggressive penalty functions, such as $p(t) = t^2$, the minimal delay may be drastically reduced to $\delta = \Delta^{1/2^d}$, greatly benefiting "honest" parties that respond quickly, while keeping the same simulator running time. Note that, perhaps surprisingly, we show that such a "tit-for-tat" technique, which is usually employed in the setting of rational players, provides significant efficiency improvements even with respect to fully adversarial players.

Combining it all. Finally, we combine our techniques by both slightly increasing the round complexity and implementing penalty-based delays. We state our main theorem below for $p(t) = t$ (no penalty), ct (linear penalty), and t^c (polynomial penalty) (in the main text we provide an expression for a generic $p(t)$):

Theorem 1. *Let Δ be an upper-bound on the time it takes to deliver a message on the network. Let r and d be integer parameters, and $p(t)$ be a (penalty) function. Then, assuming the existence of claw-free permutations, there is a $(2r+6)$-message black-box perfect cZK argument for all of* NP *with the following properties:*

- *The simulator has running time $(rn)^{O(d)}$.*

– *For any verifier that cumulatively delays its message by time at most T, the prover will provide its last message in time at most $p(T) + \delta$, where*

$$\delta = \begin{cases} 2\Delta/r^d & \text{if } p(t) = t \quad \textit{(no penalty)} \\ 2\Delta/(cr)^d & \text{if } p(t) = ct \quad \textit{(linear penalty)} \\ \frac{(2\Delta)^{1/c^d}}{r^{1+1/c+\cdots+1/c^{d-1}}} \leq \frac{(2\Delta)^{1/c^d}}{r} & \text{if } p(t) = t^c \quad \textit{(polynomial penalty)} \end{cases}$$

Remark 1 (On the number of rounds). Even without penalty-based delays, if $r = 2$, we achieve an exponential improvement in the imposed delay ($\delta = \Delta/2^d$), compared to the suggestion by Barak and Micciancio (which required a delay of $\delta = \Delta/d$). Larger r (i.e., more rounds) allows us to further improve the delay.

Remark 2 (On adversarially controlled networks). If an adversary controls the whole network, it may also delay messages from the honest players. In this case, honest players (that answer as fast as they can) are also penalized. However, the adversary can anyway delay message delivery to honest players, so this problem is unavoidable. What we guarantee is that, if a pair of honest players are communicating over a channel that is not delayed (or only slightly delayed) by the adversary, then the protocol will complete fast.

Remark 3 (On networks with failure). Note that even if the network is not under adversarial control, messages from honest parties might be delayed due to network failures. We leave it as an open question to (experimentally or otherwise) determine the "right" amount of penalty to employ in real-life networks: Aggressive delays allow us to minimize the imposed delay δ, but can raise the expected protocol running time if network failures are common.

Remark 4 (On concurrent multi-part computation). [KLP05] and [LPV09] show that concurrent multi-party computation (MPC) is possible in the timing model using delays of length $O(\Delta)$. Additionally, [LPV09] shows that at least $\Delta/2$ delays are necessary to achieve concurrent MPC in the timing model. In retrospect, this separation between concurrent ZK and MPC should not be surprising since cZK can be constructed in the plain model [RK99,KP01,PRS02], but concurrent MPC cannot [CF01,Lin04].

1.1 Organization

In Sect. 2 we give definitions regarding the timing model and primitives used in our constructions. An overview of our protocol and zero-knowledge simulator, followed by their formal descriptions, is given in Sect. 3. Actual formal analyses are given in Sect. 4 and the appendix.

2 Preliminaries

We assume familiarity with indistinguishability, interactive proofs and arguments, and stand-alone (black-box) zero-knowledge. Let N denote the set of natural numbers. Given a function $g : \mathsf{N} \to \mathsf{N}$, let $g^k(n)$ be the function computed by composing g together k times, i.e., $g^k(n) = g(g^{k-1}(n))$ and $g^0(n) = n$.

2.1 Timing model

In the timing model, originally introduced by Dwork, Naor and Sahai [DNS04], we consider a model that incorporates a "timed" network. Informally, in such a network, a (known) maximum network latency Δ—the time it takes for a message to be computed and delivered over the network—is assumed. Moreover, each party (in our case the honest provers) possesses a local clock that is somewhat synchronized with the others (in the sense that a second takes about the same time on each clock).

As in [DNS04,Gol02,KLP05], we model all the parties in the timing model as interactive Turing machines that have an extra input tape, called the **clock tape**. In an adversarial model, the adversary has full control of the content of everyone's clock tape (it can initialize and update the tape value at will), while each machine only has read access to its own clock tape. More precisely, when a party P_i is invoked, the adversary initializes the local clock of P_i to some time t of its choice. Thereafter the adversary may, at any time, overwrite the all existing clock tapes with new time values. To model that in reality most clocks are reasonably but not perfectly synchronized, we consider adversaries that are **ϵ-drift preserving**, as defined below:

Let $\sigma_1, \sigma_2, \ldots$ be a series of global states of all machines in play; these states are recorded whenever the adversary initiates a new clock or updates the existing clocks. Denote by $\text{CLK}_P(\sigma)$ the value of the local clock tape of machine P at state σ. We say that an adversary is **ϵ-drift preserving** if for every pair of parties P and P' and every pair of states σ and σ', it holds that

$$\frac{1}{\epsilon}(\text{CLK}_P(\sigma) - \text{CLK}_P(\sigma')) \leq \text{CLK}_{P'}(\sigma) - \text{CLK}_{P'}(\sigma') \leq \epsilon(\text{CLK}_P(\sigma) - \text{CLK}_P(\sigma'))$$

As in [DNS04,Gol02,KLP05], we use the following constructs that utilize the clock tapes. Below, by local time we mean the value of the local clock tape.

Delays: When a party is instructed to delay sending a message m by δ time, it records the present local time t, checks its local clock every time it is updated, and sends the message when the local time reaches $t + \delta$.

Time-out: When a party is instructed to time-out if a response from some other party P_i does not arrive in δ time, it records the present time t. When the message from P_i does arrive, it aborts if the local time is greater than $t + \delta$.

Measure: When a party is instructed to measure the time elapsed between two messages, it simply reads the local time t when the first message is sent/ received, and reads the local time t' again when the second message is sent/ received. The party then outputs the elapsed time $t' - t$.

Although the measure operator is not present in previous works, it is essentially the quantitative version of the time-out operation, and can be implemented without additional extensions of the timing model. For simplicity, we focus on the model where the adversary is 1-drift preserving, i.e. all clocks are synchronized, but our results easily extend to ϵ-drift preserving adversaries.

2.2 Black-Box Concurrent Zero-Knowledge in the Timing Model

The standard notion of concurrent zero-knowledge extends straightforwardly to the timing model; all machines involved are simply augmented with the aforementioned clock tape. The view of a party still consists of all incoming messages as well as the parties random tape. In particular, the view of the adversary determines the value of all the clocks. We repeat the standard definition of black-box concurrent zero-knowledge below.

Let $\langle P, V \rangle$ be an interactive proof for a language L, and let V^* be a concurrent adversarial verifier that may interact with multiple independent copies of P concurrently, without any restrictions over the scheduling of the messages in the different interactions with P. Let $\{\mathsf{VIEW}_2[P(x) \leftrightarrow V^*(x,z)]\}$ denote the random variable describing the view of the adversary V^* in an interaction with P on common input x and auxiliary input z.

Definition 1. *Let $\langle P, V \rangle$ be an interactive proof system for a language L. We say that $\langle P, V \rangle$ is* black-box concurrent zero-knowledge *if for every polynomials q and m, there exists a probabilistic polynomial time algorithm $S_{q,m}$, such that for every concurrent adversary V^* that on common input x and auxiliary input z opens up $m(|x|)$ sessions and has a running-time bounded by $q(|x|)$, $S_{q,m}(x,z)$ runs in time polynomial in $|x|$. Furthermore, it holds that the ensembles $\{\mathsf{VIEW}_2[P(x) \leftrightarrow V^*(x,z)]\}_{x \in L, z \in \{0,1\}^*}$ and $\{S_{q,m}(x,z)\}_{x \in L, z \in \{0,1\}^*}$ are computationally indistinguishable over $x \in L$. We say $\langle P, V \rangle$ is* black-box perfect concurrent zero-knowledge *if the above ensembles are identical.*

Remark: [Gol02] defines concurrent ZK in the timing model with the assumption (WLOG) that the adversary never trigger a time-out from any prover. [Gol02] also made the assumption that the adversary always delays the verifier messages as much as permitted, but is assumption is no longer WLOG for protocols with penalty-based delays. Therefore in our model, the adversary is given total control over all the clocks (subject to ϵ-drift preserving), similar to the definition of [KLP05] for the setting of concurrent multi-party computation.

2.3 Other primitives

We informally define other primitives used in the construction of our protocols.

Special-sound proofs: A 3-round public-coin interactive proof for the language $L \in \mathcal{NP}$ with witness relation R_L is special-sound with respect to R_L, if for any two transcripts (α, β, γ) and $(\alpha', \beta', \gamma')$ such that the initial messages α, α' are the same but the challenges β, β' are different, there is a deterministic procedure to extract the witness from the two transcripts that runs in polynomial time. Special-sound WI proofs for languages in NP can be based on the existence of non-interactive commitment schemes, which in turn can be based on one-way permutations. Assuming only one-way functions, 4-round special-sound WI proofs for NP exists[2]. For simplicity, we use

[2] A 4-round protocol is special sound if a witness can be extracted from any two transcripts $(\tau, \alpha, \beta, \gamma)$ and $(\tau', \alpha', \beta', \gamma')$ such that $\tau = \tau'$, $\alpha = \alpha'$ and $\beta \neq \beta'$.

3-round special-sound proofs in our protocol though our proof works also with 4-round proofs.

Proofs of knowledge: Informally an interactive proof is a proof of knowledge if the prover convinces the verifier not only of the validity of a statement, but also that it possesses a witness for the statement. If we consider computationally bounded provers, we only get a "computationally convincing" notion of a proof of knowledge (aka *arguments of knowledge*).

3 Our Protocol and Simulator

3.1 Protocol Overview

Following the works of [FS90,GK96], later extended to the concurrent setting by [RK99,KP01,PRS02,PV08], we consider ZK protocols with two stages:

Stage 1: First the verifier V "commits to a trapdoor" (the start message). This is followed by one or multiple **slots**; each slot consists of a prover challenge (the opening of the slot) followed by a verifier response (the closing of the slot). A rewinding black-box ZK simulator can rewind any one of these slots to extract the verifier trapdoor.

Stage 2: The protocol ends with a modified proof of the original statement that can be simulated given the verifier trapdoor.

To generate the view of an adversarial verifier V^* in the standalone setting, a black-box simulator simply rewinds a slot to learn the trapdoor, and use it to simulate the final modified proof.

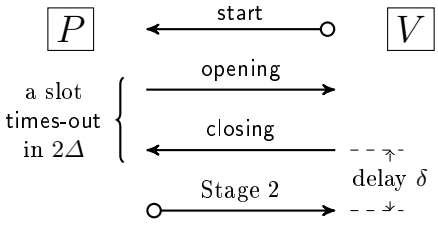
In the concurrent setting, however, V^* may *fully nest* another session inside a slot (i.e., after the prover sends the opening message, V^* schedules a full session before replying with closing message). In order for the simulator to rewind this slot, it would need to simulate the view of the nested session twice. Therefore, repeated nesting may cause a naive simulator to have super-polynomial running time [DNS04]. Different techniques were employed in different models to circumvent this difficulty caused by nesting. In the timing model, [DNS04,Gol02] shows that by delaying the Stage 2 proof and limiting the time allowed between the opening and closing of any slot, we can avoid the nesting situation all together. On the other hand, [RK99] showed that if the protocol has enough slots, the simulator can always find a slot that isn't "too nested" to rewind.

The work of Pass and Venkitasubramaniam describes a simulator (based on the work of [RK99]) that works also for constant-round protocols. Its running time (implicitly) depends on the maximum nesting level/depth of the least nested slot. Specifically, the running time of the simulator is $n^{O(d)}$ when this maximum depth of nesting is d. Building upon this, we now focus on reducing the maximum depth of nesting in the timing model.

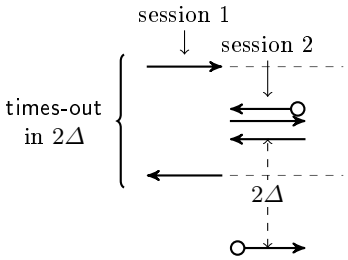
In the following overview of our techniques, we assume that V^* interleaves different sessions in a *static* schedule; the full generality of dynamic scheduling is left for our formal analysis. Additionally, we keep track of the running time of

our protocols as a function of T—the total amount of accumulated delay caused by the verifier in all the messages.

Imposing traditional timing delays with one slot. We first review the works of [DNS04,Gol02]. Recall that Δ is the maximum network latency—the time it takes for a message to be computed and delivered over the network. We require that the time between the opening and closing of each slot be bounded by 2Δ (otherwise the prover aborts); this is the smallest time-out value that we may ask of the honest verifier. At the same time, the prover delays the Stage 2 proof by δ time (after receiving the closing message of the last slot), where δ is a parameter (Fig. 1(a)). It is easy to see that if $\delta = 2\Delta$, then no nesting can occur (Fig. 1(b)). In this case the running time of the protocol is $T + \Delta$.



(a) 1 slot protocol with traditional timing constraints



(b) $\delta = 2\Delta$ prevents nesting.

Fig. 1. Traditional timing delays with **1 slot**.

If we consider the suggestion of Barak and Micciancio and set $\delta = 2\Delta/d$, then up to d levels of nesting can occur (Fig. 2). In this case, the running time of the protocol is $T + 2\Delta$ and $T + 2\Delta/d$, respectively.

Increasing the number of slots. This idea was first explored by [RK99] in the standard model where intuitively, more slots translates to more rewinding opportunities for the simulator. In the timing model, the effect of multiple slots is much more direct. Let us look at the case of 2 slots. Suppose in some session, V^* delays the closing of a slot by the maximum allowed time, 2Δ. Further suppose

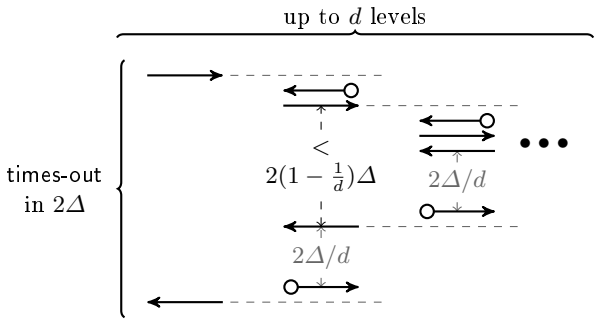


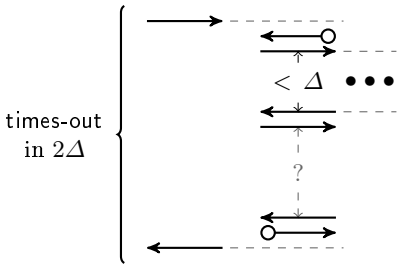
Fig. 2. $\delta = 2\Delta/d$ gives at most d levels of nesting.

that V^* nests an entire session inside this slot. Then in this nested session, one of the slots must have taken time less than Δ (Fig. 3(a)). Continuing this argument, some fully nested session at level d must take time less than $2\Delta/2^d$. Therefore if we set $\delta = 2\Delta/2^d$, V^* cannot fully nest every slot beyond depth d, and the running time of the protocol becomes $T + 2\Delta/2^d$.

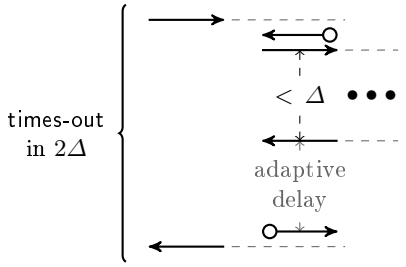
Penalizing the adversarial verifier with adaptive delays. Here we implement our "eye-for-an-eye" approach of penalizing adversarial verifiers that delay messages. Let $p(t)$ be a **penalty function** that satisfies $p(t) > t$ and is monotonically increasing. During Stage 1 of the protocol, the prover measures t, the total time elapsed from the opening of the first slot to the closing of the last slot. Based on this measurement, the prover delays Stage 2 by time $p(t) - t$ or by the minimal imposed delay δ, whichever is greater. As a result, Stage 2 only starts after $p(t)$ time has elapsed starting from the opening of the first slot. For example, suppose $p(t) = 2t$ and that the protocol has 1 slot. Then for V^* to fully nest a session inside a slot that took time 2Δ, the slot of the nested session must have taken time at most Δ, giving the same effect as having 2 slots (Fig. 3(b)). Furthermore, if we implement more aggressive penalties, such as $p(t) = t^2$,[3] then the slot of the nested session is reduced to time $\sqrt{2\Delta}$. Therefore if we set $\delta = (2\Delta)^{1/2^d}$, V^* cannot fully nest every slot beyond depth d, and the running time of the protocol becomes $T^2 + (2\Delta)^{1/2^d}$.

Combining the techniques. In general, we can consider concurrent ZK protocols that both contain multiple slots and impose penalty-based delays (e.g., Fig. 4). If we have r slots and impose $p(t)$ penalty on delays, and define

[3] Formally we may use $p(t) = t^2 + 1$ to ensure that $p(t) > t$.



(a) **2 slots, no penalty.** One of the nested slot must have half the delay.



(b) **1 slot,** $2t$ **penalty**. The nested slot must have half the delay as well.

Fig. 3. Our main techniques of restricting the nesting depth of V^*.

$g(t) = p(rt)$, then δ can be decreased to

$$d \text{ times } \left\{ \frac{p^{-1}\left(\cdots \frac{p^{-1}\left(\frac{p^{-1}(2\Delta)}{r}\right)}{r}\right)}{r} = (g^{-1})^d(2\Delta) \right.$$

$$= \begin{cases} 2\Delta/r^d & \text{if } p(t) = t \quad \text{(no penalty)} \\ 2\Delta/(cr)^d & \text{if } p(t) = ct \text{ (linear penalty)} \\ \frac{(2\Delta)^{1/c^d}}{r^{1+1/c+\cdots+1/c^{d-1}}} \le \frac{(2\Delta)^{1/c^d}}{r} & \text{if } p(t) = t^c \text{ (polynomial penalty)} \end{cases}$$

while keeping the simulator running time at $(rn)^{O(d)}$. The running time of the protocol is then $p(T) + \delta$.

Handling dynamic scheduling. So far we have discussed our analysis (and have drawn our diagrams) assuming that V^* follows a static schedule when interleaving multiple sessions. In general though, V^* may change the scheduling dynamically based on the content of the prover messages. As a result, the schedule (and nesting) of messages may change drastically when a black-box simulator rewinds V^*. This phenomenon introduces many technical difficulties

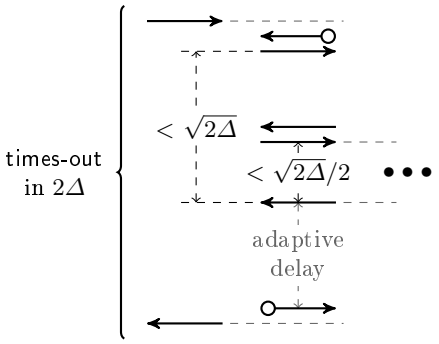


Fig. 4. 2 slots and t^2 penalty. Slots of nesting sessions decrease in size very quickly.

into the analysis, but fortunately the same difficulties were also present and resolved [PV08]. By adapting the analysis in [PV08], we give essentially the same results in the case of dynamic scheduling, with one modification: An additional slot is needed whenever $\delta < 2\Delta$ (this includes even the case illustrated in Fig. 2). For example, a minimal of 2 slots is needed to implement penalty-based delays, and a minimum of 3 slots is needed to reap the improvements that result from multiple slots.

Handling ϵ-drifts in clock tapes. As in the work of [DNS04,Gol02] we merely need to scale the time-out values in our protocols when the local clocks are not perfectly synchronized. Specifically, if the adversary is ϵ-drift preserving for some $\epsilon \geq 1$, then our protocol will impose a minimal delay of $\epsilon\delta$ and an adaptive delay of $\epsilon p(t)$ (when applicable) between the closing of the last slot and Stage 2.

3.2 Description of the protocol

Our concurrent ZK protocol is a slight variant of the precise ZK protocol of [MP06], which in turn is a modification of the Feige-Shamir protocol [FS90]. Given a one-way function f, a parameter r, a penalty function $p(t)$, and a minimal delay δ, our protocol for language $L \in \mathsf{NP}$ proceeds in the following two stages on common input $x \in \{0,1\}^*$ and security parameter n:

Stage 1: The verifier picks two random strings $s_1, s_2 \in \{0,1\}^n$ and sends $c_1 = f(s_1), c_2 = f(s_2)$ to the prover. The verifier also sends $\alpha_1, \ldots, \alpha_{r+1}$, the first messages of $r+1$ invocations of a WI special-sound proof of the statement "c_1 and c_2 are in the image set of f". These proofs are then completed sequentially in $r+1$ iterations.

In the j^{th} iteration, the prover first sends $\beta_j \leftarrow \{0,1\}^{n^2}$, a random second message for the j^{th} proof (opening of the j^{th} slot), then the verifier replies with the third message γ_j of the j^{th} proof (closing of the j^{th} slot). The prover times-out the closing of each slot with time 2Δ, and measures the time that elapsed between the opening of the first slot and the closing of the $r+1^{\text{st}}$ slot as t.

Stage 2: The prover delays by time $\max\{p(t) - t, \delta\}$, and then provides a WI proof of knowledge of the statement "either $x \in L$, or that (at least) one of c_1 and c_2 are in the image set of f".

More precisely, let L' be the language characterized by the witness relation $R_{L'}(c_1, c_2) = \{(s_1, s_2) \mid f(s_1) = c_1 \text{ or } f(s_2) = c_2\}$. Let f be a one-way function, r and δ be integers, $p(t) : \mathsf{N} \to \mathsf{N}$ be a monotonically increasing function satisfying $p(t) > t$, and L be a language in NP. Our ZK argument for L, ConcZKArg, is depicted in Figure 5.

Protocol ConcZKArg

Common Input: an instance x of a language L with witness relation R_L.
Auxiliary Input for Prover: a witness w, such that $(x, w) \in R_L(x)$.
Parameters: r (round complexity), p (penalty function), Δ (max delay), δ (min delay)
Stage 1:
V uniformly chooses $s_1, s_2 \in \{0,1\}^n$.
V $\to$ P: $c_1 = f(s_1)$, $c_2 = f(s_2)$, and $r+1$ first messages $\alpha_1, \ldots, \alpha_{r+1}$ of WI special-sound proofs of knowledge of the statement $(c_1, c_2) \in L'$ (called the start message). The proof of knowledge is with respect to the witness relation $R_{L'}$.
For $j = 1$ to $r+1$ do
P $\to$ V [opening of slot j]: Select a second message $\beta_j \leftarrow \{0,1\}^{n^2}$ for the j^{th} WI special-sound proof. P times-out if the next verifier message is not received in time 2Δ.
V $\to$ P [closing of slot j]: Third message γ_j for the j^{th} WI special-sound proof.
P measures the time elapsed between the opening of the first slot and the closing of the $r+1^{\text{st}}$ slot as t.
Stage 2:
P delays the next message by time $\max\{p(t) - t, \delta\}$.
P $\leftrightarrow$ V: A perfect-WI argument of knowledge of the statement $(x, c_1, c_2) \in L \vee L'$, where $L \vee L'$ is characterized by the witness relation

$$R_{L \vee L'}(x, c_1, c_2) = \{(w, s_1', s_2') \mid w \in R_L(x) \vee (r_1', r_2') \in R_{L'}(c_1, c_2)$$

The argument of knowledge is with respect $R_{L \vee L'}$.

Fig. 5. Concurrent Perfect ZK argument for NP

The soundness and the completeness of the protocol follows directly from the proof of Feige and Shamir [FS90]; in fact, the protocol is an instantiation of theirs. Intuitively, to cheat in the protocol a prover must "know" an inverse to either c_1 or c_2, which requires inverting the one-way function f.

3.3 Simulator Overview

At a very high-level our simulator follows that of Feige and Shamir [FS90]. The simulator will attempt to rewind one of the special-sound proofs (i.e., the slots), because whenever the simulator obtains two accepting proof transcripts, the special-soundness property allows the simulator to extract a "fake witness" r_i such that $c_i = f(r_i)$. This witness can later be used in the second phase of the protocol. At any point in the simulation, we call a session of the protocol **solved** if such a witness has been extracted. On the other hand, if the simulation reaches Stage 2 of a session without extracting any "fake witnesses", we say the simulation is **stuck**.

In more detail, our simulator is essentially identical to that of [PV08], which in turn is based on the simulator of [RK99]. The general strategy of the simulator is to find and rewind the "easiest" slot for each session; during a rewind, the simulator recursively invokes itself on any nested sessions when necessary. The main difference between our work and that of [RK99,PV08] lies in determining which slot to rewind. In [RK99,PV08], a slot that contains a "small" amount of start messages (freshly started sessions) is chosen, whereas in our simulation, a slot with *"little" elapsed time* (between the opening and the closing) is rewound. As we will see, part of the analysis from [PV08] applies directly to our simulator modulo some changes in parameters; we only need to ensure that our definition of "little" elapsed time allows the simulator to always find a slot to rewind (formally argued in Claim 2).

3.4 Description of the simulator

Our simulator is defined recursively. Intuitively on recursive level 0, the simulator's goal is to generate a view of V^*, while on all other recursive levels, the simulator's goal is to rewind a particular slot (from a previous recursion level). On recursive level ℓ, the simulator starts by feeding random Stage 1 messages to V^*. Whenever a slot s closes, S decides whether or not to rewind s depending on the time elapsed between the opening and the closing of s. If the elapsed time is "small" (where the definition of small depends on the level ℓ), S begins to rewind the slot. That is, S recursively invokes itself on level $\ell + 1$ starting from the opening of slot s with a new (random) message β, with the goal of reaching the closing message of slot s. While in level $\ell + 1$, S continues the simulation until one of the following happens:

1. *The* closing *message γ for slot s occurs*: S extracts a "fake" witness using the special-sound property and continues its simulation (on level ℓ).
2. *V^* aborts or delays "too much" in the rewinding:* S restarts its rewinding using a new challenge β for s. We show in expectation, S only restarts $O(1)$ times (intuitively, this follows since during the execution at level ℓ, S only starts rewinding a slot if V^* did not abort and only took "little time").
3. *S is "stuck" at Stage 2 of an unsolved session that started at level $\ell + 1$:* S halts and outputs fail (we later show that this never happens).

4. *S is "stuck" at Stage 2 of an unsolved session that started at level ℓ:* Again, S restarts its rewinding. We show that this case can happen at most $m-1$ times, where m is the total number of sessions.
5. *S is "stuck" at Stage 2 of an unsolved session that started at level $\ell' < \ell$: S* returns the view to level ℓ' (intuitively, this is just case 4 for the recursion at level ℓ').

In the unlikely event that S asks the same challenge β twice, S performs a brute-force search for the witness. Furthermore, to simplify the analysis of the running-time, the simulation is cut-off if it runs "too long" and S extracts witnesses for each session using brute-force search.

The basic idea behind the simulation is similar to [PV08]: We wish to define "little time" appropriately, so that some slot of every session is rewound and that expected running time is bounded. For a technical reason (used later in Claim 2), we actually want the simulator to rewind one of the first r (out of $r+1$) slots of each session.

Take for example $p(t) = 2t$ and $r = 2$ (3 slots). Based on our intuition from Sect. 3.1, a good approach would be to ensure that the simulation at recursive level ℓ finishes within time $2\Delta/4^{\ell}$, and define "little time" on level ℓ to be $2\Delta/4^{\ell+1}$. Then, we know that any session that is fully executed at recursive level ℓ must have taken time less than $2\Delta/(4^{\ell} \cdot 2)$ in Stage 1 (due to penalty-based delays), and therefore one of the first two slot must have taken time less than $2\Delta/4^{\ell+1}$, making it eligible for rewind. To show that the expected running time is bounded, we simply set δ appropriately (as a function of d, Δ and r) as in Sect. 3.1, and this would guarantee that the recursion depth of the simulator is bounded.

A formal description of our simulator can be found in Figure 6. We rely on the following notation.

- Define the function $g : \mathsf{N} \to \mathsf{N}$ by $g(n) = p(rn)$. Recall that $g^k(n)$ be the function computed by composing g together k times, i.e., $g^k(n) = g(g^{k-1}(n))$ and $g^0(n) = n$. Let d (the maximum depth of recursion) be $\min_d\{g^d(\delta) > 2\Delta\}$. Note that if $\delta = (g^{-1})^k(2\Delta)$, then $\mathsf{d} = k$.
- $\mathsf{slot}\ (i, j)$ will denote slot j of session i.
- W is a repository that stores the witness for each session. The $\mathsf{update\ W}$ command extracts a witness from two transcripts of a slot (using the special-sound property). If the two transcripts are identical (i.e. the openings of the slot are the same), the simulator performs a brute-force search to extract a "fake" witness s_i s.t. $c_i = f(s_i)$ for $i \in \{1, 2\}$.
- R is a repository that stores the transcripts of slots of unsolved sessions. Transcripts are stored in R when the simulator gets stuck in a rewinding (cases 4 and 5 mentioned in the high-level description).

4 Analysis of the Simulator

To prove correctness of the simulator, we show that the output of the simulator is correctly distributed and its expected running-time is bounded. We first prove

$\underline{\text{SOLVE}_d^{V^*}(x, \ell, h_{initial}, s, \mathsf{W}, \mathsf{R})}$:

Let $h \leftarrow h_{initial}$. Note that $h_{initial}$ contains all sessions that are started on previous recursion levels.

Repeat forever:

1. If v is a Stage 2 verifier message of some session, continue.
2. If V^* aborts in the sessions of slot s, or the time elapsed since $h_{initial}$ exceeds $g^{\mathsf{d}+1-\ell}(\delta)$, restart SOLVE from $h_{initial}$.
3. If the next scheduled message is a Stage 2 prover message for session i and $\mathsf{W}(i) \neq \perp$, then use $\mathsf{W}(i)$ to complete the WI proof of knowledge; if $\mathsf{W}(i) = \perp$ and start message of session i is in $h_{initial}$ return h, otherwise halt with output **fail**.
4. If the next scheduled message is a Stage 1 prover message for slot s', pick a random message $\beta \leftarrow \{0,1\}^{n^2}$. Append β to h. Let $v \leftarrow V^*(h)$.
5. Otherwise, if v is the $\mathsf{closing}$ message for $s' = \mathsf{slot}\ (i', j')$, then update W with v (using R) and proceed as follows.
   (a) If $s = s'$, then return h.
   (b) Otherwise, if session i' starts in $h_{initial}$, then return h.
   (c) Otherwise, if $\mathsf{W}(i') \neq \perp$ or the time elapsed since the opening of $\mathsf{slot}\ (i', j')$ exceeds $g^{\mathsf{d}-\ell}$, then continue.
   (d) Otherwise, let h' be the prefix of the history h where the prover message for s' is generated. Set $\mathsf{R}' \leftarrow \phi$. Repeat the following m times:
      i. $h^* \leftarrow \text{SOLVE}_d^{V^*}(x, \ell+1, h', s', \mathsf{W}, \mathsf{R}')$
      ii. If h^* contains an accepting proof transcript for slot s', extract witness for session i' from h and h^* and update W.
      iii. Otherwise, if the last message in h^* is the $\mathsf{closing}$ message for the last slot of an session that started in $h_{initial}$ return h^*.
      iv. Otherwise, add h^* to R'.

$\underline{S^{V^*}(x, z)}$:

Let $\mathsf{d} \leftarrow \min_d\{g^d(\delta) > 2\Delta\}$. Run $\text{SOLVE}_d^{V^*}(x, 0, , , ,)$ and output whatever SOLVE outputs with one exception. If an execution of $\text{SOLVE}_d^{V^*}(x, 0, , , ,)$ queries V^* more that 2^n times, proceed as follows:

Let h denote the view reached in the "main-line" simulation (i.e., in the top-level of the recursion). Continue the simulation in a "straight-line" fashion from h by using a brute-force search to find a "fake" witness each time Stage 2 of an session i is reached.

Fig. 6. Description of our black-box ZK simulator.

in Claim 2 that the simulator never outputs fail. Using Claim 2, we show that the output distribution of the simulator is correct in Prop. 3, and that the expected running time of the simulator is at most $poly(m^{\mathsf{d}} r^{\mathsf{d}})$ in Prop. 4. Theorem 1 then follows from Prop. 3 and 4, together with the fact that if $\delta = (g^{-1})^k(2\Delta)$ then $\mathsf{d} = k$.

Claim 2. *For every $x \in L$, $S^{V^*}(x, z)$ never outputs* fail.

Proposition 3. *The ensembles $\{\mathsf{VIEW}_2[P(x, w) \leftrightarrow V^*(x, z)]\}$ and $\{S^{V^*}(x, z)\}$ are identical over $x \in L, w \in R_L(x), z \in \{0, 1\}^*$.*

Proposition 4. *For all $x \in L, z \in \{0, 1\}^*$, and all V^* such that $V^*(x, z)$ opens up at most m sessions, $E[\mathsf{time}_{\tilde{S}^{V^*}(x,z)}] \leq poly(m^{\mathsf{d}} r^{\mathsf{d}})$*

The proof of Claim 2 is given below, while the proofs of Prop. 3 and 4 are given in the full version of the paper; in any case, the proofs of Prop. 3 and 4 are essentially identical to [PV08], modulo a change of parameters. Throughout the analysis we assume without loss of generality that the adversary verifier V^* is deterministic (as it can always get its random coins as part of the auxiliary input).

Proof: (Claim 2) Recall that $S^{V^*}(x, z)$ outputs fail only if $\text{SOLVE}_d^{V^*}(x, 0, , ,)$ outputs fail. Furthermore, SOLVE outputs fail at recursive level ℓ only if it reaches Stage 2 of an unsolved session that started at level ℓ (see Step 3 of SOLVE). We complete the proof in two parts. First we show $\text{SOLVE}_d^{V^*}$ will rewind at least one of the first r slots of every session at level ℓ. Then, we show that SOLVE always extracts a witness when it rewinds a slot.

In order for SOLVE to be stuck at a session i that starts at recursive level ℓ, session i must reach Stage 2 within $g^{(\mathsf{d}-\ell)}(\delta)$ time-steps (otherwise SOLVE would have rewound as per Step 2). This implies that t, the time between the opening of the first slot and the closing of th last slot of session i, must satisfy $p(t) \leq g^{(\mathsf{d}-\ell)}(\delta)$ (due to penalty-based delays). This in turn implies that one of the first r slots of session i must have taking time at most

$$\frac{t}{r} \leq \frac{p^{-1}(g^{(\mathsf{d}-\ell)}(\delta))}{r} \leq g^{(\mathsf{d}-\ell-1)}(\delta)$$

(here we use the monotonicity of p). By construction, SOLVE would have rewound this slot (i.e., execute Step 5.(d)).

Next we show that whenever SOLVE rewinds a slot, a witness for that session is extracted. Assume for contradiction that SOLVE fails to extract a witness after rewinding a particular slot. Let level ℓ and slot j of session i be the first time this happens. This means at the end of Step 5.(d), m views are obtained, yet none of them contained a second transcript for slot j. Observe that in such a view, SOLVE most have encountered Stage 2 of some unsolved session i' (i.e., stuck). Yet, we can show that the $m - 1$ other sessions can each cause SOLVE to be stuck at most once; this contradicts the fact that SOLVE is stuck on all m good views.

For every session i' that SOLVE gets stuck on, both the opening and the closing of the last slot occurs inside the rewinding of slot (i, j); otherwise, SOLVE would have rewound one of the r slots that occurred before the opening of slot (i, j) successfully and extracted a witness for session i' (l, i, j was the first "failed" slot). Furthermore, the transcript of this slot enables SOLVE to never get stuck on session i' again, since the next time that the last slot of session i' closes will allow SOLVE to extract a witness for session i'. □

5 Acknowledgments

We would like to thank the anonymous TCC reviewers for their helpful comments.

References

[Axe84] R. Axelrod. *The evolution of cooperation.* New York: Basic Books, 1984.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO '01*, pages 19–40, 2001.

[CKP01] Tzafrir Cohen, Joe Kilian, and Erez Petrank. Responsive round complexity and concurrent zero-knowledge. In *ASIACRYPT '01*, pages 422–441, 2001.

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires $\tilde{\omega}(\log n)$ rounds. In *STOC '01*, pages 570–579, 2001.

[DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *STOC '90*, pages 416–426, 1990.

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

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

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

[Gol02] Oded Goldreich. Concurrent zero-knowledge with timing, revisited. In *STOC '02*, pages 332–340, 2002.

[KLP05] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In *STOC '05*, pages 644–653, 2005.

[KP01] Joe Kilian and Erez Petrank. Concurrent and resettable zero-knowledge in poly-logarithmic rounds. In *STOC '01*, pages 560–569, 2001.

[KPR98] Joe Kilian, Erez Petrank, and Charles Rackoff. Lower bounds for zero knowledge on the internet. In *FOCS '98*, pages 484–492, 1998.

[Lin04] Yehuda Lindell. Lower bounds for concurrent self composition. In *TCC '04*, pages 203–222, 2004.

[LPV09] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *STOC '09*, pages 179–188, 2009.

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In *STOC '06*, pages 306–315, 2006.

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *FOCS '02*, pages 366–375, 2002.

[PV05] Giuseppe Persiano and Ivan Visconti. Single-prover concurrent zero knowledge in almost constant rounds. In *Automata, Languages and Programming*, pages 228–240, 2005.

[PV08] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. On constant-round concurrent zero-knowledge. In *TCC '08*, pages 553–570, 2008.

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

[Ros00] Alon Rosen. A note on the round-complexity of concurrent zero-knowledge. In *CRYPTO '00*, pages 451–468, 2000.

[Rs09] Alon Rosen and abhi shelat. A rational defense against concurrent attacks. Manuscript., 2009.