Client-Server Concurrent Zero Knowledge with Constant Rounds and Guaranteed Complexity*

Ran Canetti[1]**, Abhishek Jain[2], and Omer Paneth[3]***

[1] Boston University and Tel-Aviv University, canetti@bu.edu
[2] Boston University and MIT, abhishek@csail.mit.edu
[3] Boston University, omer@bu.edu

Abstract. The traditional setting for concurrent zero knowledge considers a server that proves a statement in zero-knowledge to multiple clients in multiple concurrent sessions, where the server's actions in a session are *independent* of all other sessions. Persiano and Visconti [ICALP 05] show how keeping a limited amount of global state across sessions allows the server to significantly reduce the overall complexity while retaining the ability to interact concurrently with an unbounded number of clients. Specifically, they show a protocol that has only slightly super-constant number of rounds; however the communication complexity in each session of their protocol depends on the number of other sessions and has no a-priori bound. This has the drawback that the client has no way to know in advance the amount of resources required for completing a session of the protocol up to the moment where the session is completed. We show a protocol that does not have this drawback. Specifically, in our protocol the client obtains a bound on the communication complexity of each session at the start of the session. Additionally the protocol is *constant-rounds*. Our protocols is fully concurrent, and assumes only collision-resistant hash functions. The proof requires considerably different techniques than those of Persiano and Visconti. Our main technical tool is an adaptation of the "committed-simulator" technique of Deng et. al [FOCS 09].

1 Introduction

Concurrent security of a protocol means that security is preserved even when many copies of the protocol may be executed concurrently with each other and with other, potentially unknown protocols. Concurrent security is essential for protocols designed for modern networks, such as the Internet. However, it often imposes a cost on the complexity of the protocol. For example, stand-alone zero-knowledge protocols can be implemented in a constant number of rounds

* This paper is supported by the NSF EAGER grant, and NSF Algorithmic Foundations grant no. 1218461.
** Supported by the Check Point Institute for Information Security.
*** Supported by the Simons award for graduate students in theoretical computer science.

based on any one way function, while constant-round concurrent zero-knowledge protocols are not known without relying on non-standard assumptions or trusted setup.

Concurrent Zero Knowledge. The *concurrent zero knowledge* task [8] considers a natural and special case of concurrent security. Here there is a server that wants to prove theorems in zero-knowledge [9] to multiple clients (verifiers). For that purpose, the server runs an instance (i.e., a *session*) of a protocol with each client. There may be an unbounded (albeit polynomial) number of sessions, and sessions may execute concurrently with adversarially controlled delay and ordering of messages. Furthermore, the prover side of each session should execute without knowledge of any other session. This simplifies the design for the server and allows the prover side to be executed on separate machines without coordination. Still, for security we only consider two cases: one where all or some provers are corrupted, and one where all or some of the verifiers are corrupted. While the concurrent zero-knowledge setting is a substantial restriction of general composition it distills an important aspect of the general challenge of concurrent security. Indeed, this setting was extensively studied, with special attention to minimizing the number of rounds [18, 13, 17, 3, 12, 6, 14]. Furthermore, techniques developed for concurrent zero knowledge have been found useful in the study of more general concurrent systems (see e.g., [5]).

The state of the art for protocols based on standard assumptions is $\Omega(\log n)$ rounds, where n is the security parameter. Furthermore, for protocols with black-box simulation we know that $\tilde{\Omega}(\log n)$ is the best possible.

Correlated Provers. Persiano and Visconti [16] consider a relaxed variant of the classic concurrent zero knowledge model, where the server is allowed to somewhat correlate its strategies in the different sessions. Here one has to make sure that the correlation is on the one hand effectively implementable by the server, and on the other hand preserves the overall efficiency and performance from the point of view of the client. Specifically, they present a zero-knowledge protocol where the server keeps track of the *number* of currently open sessions at any time. It then starts off each session to have a constant number of messages whose length depends polynomially on the number of currently open sessions. If the number of sessions increases beyond some threshold before the session is over, the session has to be "re-done" with longer messages. Overall, it is guaranteed that if n^c sessions are executed concurrently to a session, then the protocols of [16] requires $O(c)$ rounds and $n^{O(c)}$ communication for that session.

The global state to be kept by the server in this protocol is indeed minimal and reasonable. Additionally, the number of rounds in every session grows very slowly with the number of sessions, significantly improving the best known "pure" concurrent zero-knowledge protocols (as long as the total number of sessions is polynomial). However, this protocol has the strong disadvantage that a client has no way of knowing, at any point during the protocol execution, how much communication it will need in order to complete the session.

This work. We present a new concurrent zero-knowledge protocol where, like the [16] protocol, the server keeps track of the number of sessions currently open. Our protocol improves upon the protocol of [16] in two ways:

- **Constant rounds.** Our protocol takes six messages, regardless of the number of concurrent sessions.
- **Guaranteed complexity.** In our protocol, the server announces in the beginning of every session the communication complexity of the session. The server cannot dynamically increase the communication complexity of a session to accommodate new clients that arrive during the session's execution.

The importance of guaranteed complexity. The advantage of having guaranteed complexity is best explained by an analogy: Consider a customer that is placing a call to a call center and is being put on hold. The customer's waiting is likely to become more endurable and efficient if the call center commits to (or estimates) the required waiting time at the beginning of the call. In our setting, the client's resource is communication rather than waiting time. Clearly, clients benefit from knowing ahead of time how much communication is required from them to participate in the protocol. For example, a client with limited communication resources would prefer to learn ahead of time that its resources are insufficient to complete the protocol, rather than during the session after all its resources have already been spent.

The protocol of [16]. The protocol of [16] is based on the *bounded concurrent* protocol of Barak [1]. Barak's protocol is secure as long the number of concurrent sessions does not exceed some bound that depends on the communication complexity of the protocol. Very roughly, Persiano and Visconti show that it is possible to add rounds to the protocol and increase its communication "on-the-fly" as new occurrent sessions start. However, as a result, the round complexity of their protocol must depend on the number of sessions, and the server cannot guarantee the complexity of any session ahead of time.

It may seem that bounded concurrency is of no use for designing protocols with guaranteed complexity. Indeed, when the server commits the communication complexity of, say, the first session, it has no bound on the number of sessions that will be started concurrently to the first session.

Our protocol. Counter to the above intuition, our protocol does leverage bounded concurrency techniques of Barak. However, our approach departs from [16] in the following manner: we set the communication complexity of every session only based on the order in which the sessions *start*. The first n sessions to start execute a bounded concurrent protocol that is secure for n concurrent session. The following $n^2 - n$ sessions execute a bounded concurrent protocol that is secure for n^2 session, and so on. Importantly, the communication complexity of a session is not affected by sessions that start *after* it. This in particular means that the [1, 16] simulation technique is inadequate in our setting. Indeed, our security proof differs significantly from that in [1, 16].

1.1 Our Techniques

We start by recalling Barak's zero-knowledge protocol and its simulation. Barak's protocol starts with a preamble phase where the prover sends a commitment c and the verifier responds with a random challenge r. Any prover that can commit to a program that predicts r can obtain a "trapdoor" and cheat in the proof phase. The zero-knowledge simulator will be able to obtain a trapdoor by committing to the code of the verifier itself. Next we discuss two approaches for extending Barak's protocol to the concurrent setting.

Bounded concurrency. In the concurrent setting, the simulator cannot simply commit to the code of the verifier. Indeed, the verifier's code eventually predicts r, but might only do so after receiving convincing proofs in other sessions. Furthermore, when the simulator sends the commitment c in some session, it did not yet compute the proofs in upcoming sessions (in fact, these proofs might depend on c); therefore it cannot commit to such proof together with the verifier's code.

The approach in [1] is to change the protocol as follows: to obtain a trapdoor, the simulator must commit to a program that predicts r given some auxiliary information z (that may be chosen after r is sent). To maintain soundness, z must be much shorter then r. The simulator can now encode the simulated proofs in a bounded number of other sessions into z. This results in a bounded concurrent protocol. As argued above, this technique, on its own, is inadequate for our setting.

Committed simulator. A different approach, that we will refer to as the "committed simulator" approach, is as follows: even if the number of concurrent sessions is unbounded, the simulated proofs in all these sessions still have a short description, which is the code of the simulator itself. Concretely, in every session, the simulator will commit to a version of itself that simulates the interaction with the verifier in all other sessions until the verifier sends the challenge r in that session.

The problem with this approach is bounding the running time of the simulator. If the simulator commits to itself in the preamble phase, then in the proof phase the simulator will prove a statement on its own execution. This execution might contain the proof phase of in some other sessions where the simulator also proved a statement on its own execution. For some adversarial schedules, such recursive construction of proof becomes too expensive. Nonetheless, variants of the committed simulator approach were successfully applied in many different settings [7, 4, 11, 15, 10, 6].

Our approach. Our simulation combines these two approaches to obtain a protocol with constant rounds and guaranteed complexity, assuming only collision resistant hashing. In a nutshell, we leverage the bounded concurrent simulation technique to "flatten" the recursion tree, avoiding the blowup in the simulator's running time. A more detailed description follows.

We start by assigning a *level* to each session. All sessions that execute a bounded concurrent protocol for n^i sessions, are assigned level i. Our protocol is defined such that for every i, the total number of sessions at all levels $\leq i$ is

at most n^i. It follows that in every session at level i, the verifier's challenge is long enough to account for all the messages received by the verifier in sessions at levels at most i.

To deal with the messages sent in sessions at levels larger than i, we turn to the committed simulator approach. The main idea is that we can avoid the exponential blowup in the running time of the simulator by committing only to specific *parts* of the simulator that are in charge of simulating the sessions at levels larger than i rather than the entire code of the simulator.

The simulator. The simulator Sim is divided into multiple components $\{\mathsf{Sim}_i\}$ where the i'th component Sim_i is in charge of simulating sessions at level i. To simulate a session at level i, Sim_i will commit to a program Π_i that contains the verifier's code together with the code of all the simulator's components Sim_j for $j > i$. We can think of the program Π_i as a new verifier that simulates all sessions at levels $> i$ internally and forwards externally the messages in sessions at level $\leq i$. Since sessions at level i execute a bounded concurrent protocol for n^i sessions, and the total number of sessions at levels $< i$ is at most n^i, we have that Sim_i can encode all the messages sent to Π_i as auxiliary input.

Finally, we argue that the running time of the simulator is polynomial. Using the analysis of the bounded concurrent protocol, we have that the running time of the component Sim_i is polynomial in the running time of the program Π_i. Since Π_i simply emulates all the simulator components Sim_j for $j > i$, we have that the running time of Sim_i is only polynomially larger than the total running time of all the components Sim_j for $j > i$. Since the total number of concurrent sessions started by an efficient adversary is bounded by some polynomial n^c, we get that the total number of levels is constant and therefore the running time of all the simulator's components is bounded by a polynomial.

Avoiding circular use of randomness. We note that by using the above leveled simulation strategy we do not only avoid the blowup in the simulator's running time, but also avoid some of the technical complications that arise when the simulator commits to its own code. For example, in [4, 10, 6], the simulator needs to commit to its own code together with the randomness that it will use to simulate the rest of the protocol. The aforementioned works develop additional techniques to deal with this problem. In our setting, since every component only commits to the randomness used by the *higher* level component, such circular use of randomness is avoided, resulting in simpler protocol and analysis.

Taking advantage of terminating sessions. It is natural to require that, as existing sessions terminate and the load on the server decreases, the complexity of the protocol in new sessions decreases as well. We note that extending our simulation strategy to satisfy this requirement is not straight-forward. The problem is that our simulation strategy assumes that for every session at level i, the total number of concurrent sessions at levels $\leq i$ is bounded by n^i. However, consider the scheduling where all sessions at levels $\leq i$ terminate and a new session starts. If we choose to decrease the protocol complexity in the new session, then the total number of sessions at levels $\leq i$ may exceed n^i. We demonstrate a

slightly more complicated server strategy where the complexity of new sessions does decrease as old sessions terminate (while preserving overall simulatability).

1.2 Related Work

Concurrent zero-knowledge in the Plain Model. Improving the round-complexity of concurrent zero-knowledge proofs in the plain model has been an active area of research. The round complexity of concurrent zero-knowledge with black-box simulation was studied in [18, 13, 17], resulting in protocols with logarithmic round-complexity (which is essentially optimal [3]). Constant-round protocols with non-black-box simulation where constructed based on different non-standard assumption such as interactive knowledge assumptions [12], statistically sound P-certificates [6] and differing input (or extractable) obfuscation [14].

Optimistic concurrent zero-knowledge. The work of Rosen and Shelat [19] also studies the round complexity of concurrent zero-knowledge proofs with a correlated prover in the client-server setting. Their focus is on improving the round complexity of concurrent zero-knowledge with respect to "optimistic" adversarial schedules. That is, the round complexity of their protocol significantly decreases when the scheduling of messages does not include too many nested sessions. However, for a worst-case adversarial schedules, [19] give no improvement over logarithmic round-complexity of [17] while our protocol has constant rounds in the worst cast. However, unlike in our protocol, the communication complexity in [19] has a fixed upper bound that is independent of the adversary.

2 The Guaranteed Complexity Model

In this section we formally define a protocol in the guaranteed complexity model. We start by describing the general syntax and the model of communication. We then consider the specific case of zero-knowledge proof systems in the guaranteed complexity model and present a security definition for the same.

Let Server be interactive PPT machine that interacts with multiple clients in concurrent sessions and let $\{\langle \mathcal{S}_\ell, \mathcal{C}_\ell \rangle\}_{\ell \in \mathbb{N}}$ be a family of protocols parameterized by a *load parameter* ℓ where for every $\ell \in \mathbb{N}$, $\mathcal{S}_\ell$ and $\mathcal{C}_\ell$ are PPT machines. A protocol in the guaranteed complexity model is defined by the tuple $\Pi = (\mathsf{Server}, \{\langle \mathcal{S}_\ell, \mathcal{C}_\ell \rangle\})$.

(Honest) Protocol Execution. The execution of a protocol $\Pi = (\mathsf{Server}, \{\langle \mathcal{S}_\ell, \mathcal{C}_\ell \rangle\})$ consists of a single server executing the algorithm Server while interacting with multiple clients concurrently. To initiate a new session a client sends a special *session initiation message* to the server. In response to the session initiation message, the server chooses a load parameter ℓ for the session and sends it to the client. In the rest of the session we require that the algorithm Server follows the strategy $\mathcal{S}_\ell$ while the client follows the strategy $\mathcal{C}_\ell$.

An execution of the protocol Π with $p(n)$ sessions is defined by the randomness of all the clients and the schedule of messages across all the sessions. Even though for every fixed load parameter ℓ, the strategies $\mathcal{S}_\ell, \mathcal{C}_\ell$ are efficient, the server algorithm may choose ℓ to be very large, increasing the running time of the concurrent execution. Therefore we explicitly require the efficiency of a concurrent execution.

Definition 1. *A protocol* $(\mathsf{Server}, \{\langle \mathcal{S}_\ell, \mathcal{C}_\ell \rangle\})$ *in the guaranteed complexity model is efficient if for every polynomial* p *there exists another polynomial* q *such that the running time of* Server *in every execution with* $p(n)$ *sessions is bounded by* $q(n)$*.*

Zero Knowledge in the Guaranteed Complexity Model. Let $\Pi = (\mathsf{Server}, \{\langle \mathcal{S}_\ell, \mathcal{C}_\ell \rangle\})$ be a protocol in the guaranteed complexity model and let $\mathcal{L}$ be an NP language with witness relation $\mathcal{R}_\mathcal{L}$. We say that Π is an interactive proof system for $\mathcal{L}$ if for every $\ell \in \mathbb{N}$, the protocol $\langle \mathcal{S}_\ell, \mathcal{C}_\ell \rangle$ is an interactive proof for $\mathcal{L}$. Next we define the zero-knowledge property.

Let Π be an interactive proof for language $\mathcal{L}$ in the guaranteed complexity model. Let n be the security parameter. Consider a concurrent adversary V^* that start $m(n)$ concurrent session with the server for some polynomial m. Let $\boldsymbol{x} \in \mathcal{L}^m$ be the vector of instances used in the different session and let $\boldsymbol{w}$ be a vector of the corresponding witnesses used by the server. We allow V^* to control the scheduling of the messages across all the sessions. Let $\mathsf{View}_{V^*}(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{z})$ be the random variable describing the output of V^* in the above experiment when executed with auxiliary input z.

Definition 2 (Concurrent Zero-Knowledge in the Guaranteed Complexity Model). *Let* $\Pi = (\mathsf{Server}, \{\langle \mathcal{S}_\ell, \mathcal{C}_\ell \rangle\})$ *be an interactive proof system for language* $\mathcal{L}$ *in the guaranteed complexity model. We say that* Π *is* zero knowledge *if for every polynomial* m*, and for every PPT concurrent adversary* V^* *starting* $m(n)$ *sessions there exists a PPT algorithm* $\mathcal{S}$*, such that for every instances vector* $\boldsymbol{x} \in \mathcal{L}^{m(n)}$*, every witnesses vector* $\boldsymbol{w}$ *such that* $(x_i, w_i) \in \mathcal{R}_\mathcal{L}$ *for all* $i \in [m(n)]$*, and for every auxiliary input* $z \in \{0,1\}^{\mathrm{poly}(n)}$ *the following ensembles are computationally indistinguishable,*

$$\{\mathsf{View}_{V^*}(\boldsymbol{x}, \boldsymbol{w}, z)\}_{n \in \mathbb{N}} \approx_c \{\mathcal{S}(x, z)\}_{n \in \mathbb{N}}\,.$$

3 Constant-Round Zero-Knowledge in the Guaranteed Complexity Model

In this section we describe a constant-round ZK protocol $\Pi_{\mathsf{zk}} = (\mathsf{Server}, \{\langle P_\ell, V_\ell \rangle\})$ in the guaranteed complexity model. We start by defining a family of protocols $\{\langle P_\ell, V_\ell \rangle\}_{\ell \in \mathbb{N}}$ where, roughly speaking, the protocol $\langle P_\ell, V_\ell \rangle$ is simply Barak's bounded-concurrent ZK protocol [1] with n^ℓ as the a priori bound on the number of sessions. We then define the server algorithm Server to complete the description of Π_{zk}.

The protocol $\langle P_\ell, V_\ell \rangle$. The protocol will make use of the following primitives: a statistically binding commitment Com, a family $\mathcal{H} = \{\mathcal{H}_n\}_{n\in\mathbb{N}}$ of collision-resistant hash functions such that $h \in \mathcal{H}_n$ maps strings in $\{0,1\}^*$ to strings in $\{0,1\}^n$, and a witness-indistinguishable universal argument UA for an $\mathbf{NTIME}(T(n))$-complete language where $T : \mathbb{N} \to \mathbb{N}$ is a "slightly" super-polynomial function, for example $T(n) = n^{\log\log n}$ [2]. In the description of the protocol, the length of the verifier's messages will depend on a parameter m that denotes the total length of the *prover's* messages in the protocol.

Common Input: $x \in \mathcal{L}$.
Auxiliary Input to P: A witness w for $x \in \mathcal{L}$.

Initiation Stage:
V_ℓ samples $h \leftarrow \mathcal{H}_n$ and sends h to P_ℓ.
Preamble Stage:
1. P_ℓ sends $c = \mathsf{Com}(h(0^n))$ to V_ℓ.
2. V_ℓ samples $r \leftarrow \{0,1\}^{n^\ell \cdot m + n}$ and sends r to P_ℓ.

Proof Stage:
P_ℓ and V_ℓ execute the protocol UA where P_ℓ proves that $x \in \mathcal{L} \vee (h, c, r) \in \mathcal{L}_U$.

The language $\mathcal{L}_U$ is defines as follows: $(h, c, r) \in \mathcal{L}_U$ iff there exist a program $\Pi \in \{0,1\}^*$, a string $y \in \{0,1\}^*$, and randomness s for Com such that:

1. $|y| \leq |r| - n$.
2. $c = \mathsf{Com}(h(\Pi); s)$.
3. $\Pi(y)$ outputs r within $T(n)$ steps.

Fig. 1: Protocol Family $\langle P_\ell, V_\ell \rangle$ for ZK in the Guaranteed Complexity Model (Protocol 1)

Remark 1. The relation $\mathcal{L}_U$ presented in Protocol 1 is slightly oversimplified. For this relation, we can prove the security of Protocol 1 when $\mathcal{H}$ is collision-resistant against "slightly" super-polynomial sized circuits. For simplicity of exposition, in this manuscript, we will work with this assumption. We stress, however, that as discussed in several prior works (see e.g., [2]), this assumption can be removed by using an appropriate error-correcting code.

The server algorithm Server**.** We start by describing a simple server algorithm that only assigns monotonically increasing values of the load parameter to new sessions. In Section 3.2, we describe a better server algorithm that decreases the load parameter when some of the concurrent sessions terminate.

The algorithm Server maintains a variable $\mathsf{SessionCount}$ that counts the number of concurrent sessions started so far. Whenever a client initiates a new ses-

sion, Server increases the value of $\mathsf{SessionCount}$. When a new clients sends a session initiation message to the server, Server sets the load parameter ℓ for that session such that $n^{\ell-1} \leq \mathsf{SessionCount} \leq n^{\ell}$.

In the next section we prove the following theorem:

Theorem 1. *Assuming h is a hash function ensemble that is collision-resistent against circuits of size $n^{\log n}$, Com is a statistically binding commitment, and UA is a witness-indistinguishable universal argument for $\boldsymbol{NTIME}(n^{\log \log n})$, the protocol $\Pi_{\mathsf{zk}} = (\mathsf{Server}, \{\langle P_\ell, V_\ell \rangle\})$ is concurrent zero-knowledge in the guaranteed complexity model.*

3.1 Proof of Theorem 1

The proof that for every $\ell \in \mathbb{N}$, the protocol $\langle P_\ell, V_\ell \rangle$ is complete and sound, follows directly from the analysis of the bounded-concurrent ZK protocol in [1]. In this section we first show that for every $\ell \in \mathbb{N}$, Protocol 1 is efficient according to Definition 1. We then show that Π_{zk} is ZK in the guaranteed complexity model.

Protocol 1 is efficient. Let p be a polynomial and let ℓ_{max} be such that for large enough values of n, $p(n) < n^{\ell_{\mathsf{max}}}$. By the definition of the server algorithm Server, in an execution with $p(n)$ sessions, the load parameter of every session is at most ℓ_{max}. Since the running time of P_ℓ only grows with ℓ, we have that the running time of Server in every session is at most the running time of $P_{\ell_{\mathsf{max}}}$ and therefore the total running time of Server is bounded by a polynomial that depends only on p.

Protocol Π_{zk} is ZK in the guaranteed complexity model. Let V^* be a malicious verifier that starts at most $n^{\ell_{\mathsf{max}}}$ sessions for some constant ℓ_{max}. By the definition of the server algorithm Server, the load parameter of every session in an honest execution is at most ℓ_{max}. We construct a simulator $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{load}}, \{\mathsf{Sim}_\ell\})$ consisting of $\mathsf{Sim}_{\mathsf{load}}$ and ℓ_{max} other components $\{\mathsf{Sim}_\ell\}_{\ell \in [\ell_{\mathsf{max}}]}$. Roughly speaking, the component $\mathsf{Sim}_{\mathsf{load}}$ simulates the servers responses to the clients session initiation message in all sessions. The component Sim_ℓ simulates all the executions of $\langle P_\ell, V_\ell \rangle$ in sessions with load parameter ℓ. We now give more details.

The component $\mathsf{Sim}_{\mathsf{load}}$. This component simulates the server's responses to the clients session initiation message in all sessions. This simulation involves assigning a load parameter for every session started by V^*. Since the honest server Server selects the load parameter in each session based only on the (public) adversarial scheduling, $\mathsf{Sim}_{\mathsf{load}}$ can use the exact same algorithm as Server, resulting in a perfect simulation of these messages.

The component Sim_ℓ. This component simulates the interaction of $\langle P_\ell, V_\ell \rangle$ in all the sessions with load parameter ℓ. At a high-level, the simulation will follow the simulation strategy of Barak's bounded-concurrent ZK protocol [1]. According to this strategy, the simulator sends a commitment c to the code of

the verifier and then uses this code as a trapdoor witness, proving that c is commitment to a code Π that outputs the random string r sent by the verifier. All the messages simulated in concurrent sessions are given to Π as auxiliary input. The main problem is that in order to guarantee that the protocol is sound, the program Π is only allowed to get an auxiliary input of bounded length; however, the number of concurrent sessions in our setting are not bounded.

We fix this problem in the following manner. Instead of simply committing to the code of V^*, Sim_ℓ will commit to a program V^*_ℓ that includes the code of V^* as well as the code of the simulation components $\mathsf{Sim}_{\mathsf{load}}$ and $\mathsf{Sim}_{\ell+1}, \ldots, \mathsf{Sim}_{\ell_{\mathsf{max}}}$. Roughly speaking, the program V^*_ℓ will simulate all the sessions with load parameter $\ell' > \ell$ internally, and therefore Sim_ℓ will need to provide as auxiliary input only the messages of concurrent sessions where the load parameter is at most ℓ. It follows from the description of Server that the number of concurrent sessions where the load parameter is at most ℓ is bounded by some polynomial (that depends on ℓ). Therefore, it is possible to include all of these messages as an auxiliary input to V^*_ℓ.

Next we formally describe the simulator component Sim_ℓ, starting with the definition of the program V^*_ℓ.

The program V^*_ℓ. V^*_ℓ is an interactive algorithm that includes the code of V^* together with the code of the simulation components $\mathsf{Sim}_{\mathsf{load}}$ and $\mathsf{Sim}_{\ell+1}, \ldots, \mathsf{Sim}_{\ell_{\mathsf{max}}}$. V^*_ℓ uses the same randomness as Sim to execute V^* and all the other simulation components. V^*_ℓ will emulate the execution of V^*, and will use the mentioned simulator components to internally simulate the responses to the session initiation messages in all sessions as well the prover messages of the protocols $\langle P_{\ell'}, V_{\ell'} \rangle$ executed in the sessions with load parameter $\ell' > \ell$. In the sessions with load parameter $\ell' \leq \ell$, V^*_ℓ will forward the messages of the protocol $\langle P_{\ell'}, V_{\ell'} \rangle$ externally.

In every session with load parameter ℓ, Sim_ℓ will simulate the execution of $\langle P_\ell, V_\ell \rangle$ as follows:

1. Sim_ℓ receives the description of a hash function h from V^*.
2. Sim_ℓ sends a commitment c to the hash of the code of a program Π that given auxiliary input $y = (m_1, \ldots, m_t)$, emulates an execution of V^*_ℓ when receiving the messages $m_1, \ldots, m_t$, and outputs V^*_ℓ's next message.
3. Sim_ℓ receives the the random string r from V^*.
4. Sim_ℓ sends a UA proof using a trapdoor witness that contains the code of the program Π and an appropriate auxiliary input string y. The string y is a list of all the prover messages that were simulated by Sim in all sessions with load parameter at most ℓ and sent before V^* sent the random string r in the present session.

This completes the description of the simulator. Next, we turn to its analysis.

Analysis of Sim We start by showing that Sim_ℓ constructs a valid witness for the statement $(h, c, r) \in \mathcal{L}_U$. This amounts to proving that $\Pi(y)$ outputs r and that $|y| \leq |r| - n$. We also need to show that the running time of $\Pi(y)$ is at

most $T(n)$. We will show that the last statement is correct when we analyze the running time of the simulation. Finally, we will prove the indistinguishability of the adversary's view in the real and ideal world.

Proof that $\Pi(y)$ outputs r. The program $\Pi(y)$ outputs the next message of V_ℓ^* given the external messages in y. V_ℓ^* emulates V^* using the same randomness as Sim. It is left to show that the messages sent to V^* emulated by V_ℓ^* and by Sim are identical. Recall that the messages sent to V^* in the execution emulated by V_ℓ^* are as follows: in sessions with load parameter larger than ℓ, the messages are generated by the internal simulation of V_ℓ^*, and the messages sent in sessions with load parameter at most ℓ are specified in y. For sessions with load parameter larger than ℓ, the messages sent to V^* in the emulation of V_ℓ^* and of Sim are identical since they are generated using the same simulation algorithm and using the same randomness (by the construction of V_ℓ^*). For sessions with load parameter at most ℓ, the messages sent to V^* in the emulation of V_ℓ^* and of Sim are identical by the way the auxiliary input string y is constructed.

Proof that $|y| \leq |r| - n$. The auxiliary input string y constructed by Sim_ℓ contains only prover messages in sessions with load parameter at most ℓ. By the definition of the server algorithm Server there could be at most n^ℓ such sessions, and the total length of all the prover messages in every session is bounded by the parameter m. Therefore we have $|y| \leq n^\ell \cdot m$. Since V_ℓ samples $r \in \{0,1\}^{n^\ell \cdot m + n}$ we have that $|y| \leq |r| - n$.

Proof that the simulation is polynomial time. It is enough to show that all components of Sim are polynomial time. Since $\mathsf{Sim}_{\mathsf{load}}$ just follows the honest server algorithm, the efficiency of $\mathsf{Sim}_{\mathsf{load}}$ follows from the efficiency of the protocol. For every $\ell \in [\ell_{\mathsf{max}}]$ we show that the running time of Sim_ℓ is bounded by a polynomial in the security parameter (that depends on ℓ and on V^*). Since Sim_ℓ constructs the program V_ℓ^*, commits to its code, and provides a UA proof of its execution, the running time of Sim_ℓ is polynomial in the size and running time of V_ℓ^*. Additionally, since Sim_ℓ reads the entire transcript of the execution and uses it to construct the auxiliary input y in every session it simulates, the running time of Sim_ℓ is polynomial in the total length of the transcript. Note that the total length of the transcript is always bounded by the running time of V^* which is polynomial in the security parameter.

We start by bounding the running time of $\mathsf{Sim}_{\ell_{\mathsf{max}}}$. The program $V^*_{\ell_{\mathsf{max}}}$ only consists of the code of V^* and the code of $\mathsf{Sim}_{\mathsf{load}}$ and therefore, the running time of $V^*_{\ell_{\mathsf{max}}}$ is a polynomial. It follows that the running time of $\mathsf{Sim}_{\ell_{\mathsf{max}}}$ is also a polynomial. Now, for every $\ell \in [\ell_{\mathsf{max}}]$, the program V_ℓ^* only consists of the code of V^*, the code of $\mathsf{Sim}_{\mathsf{load}}$, and the code of $\mathsf{Sim}_{\ell'}$ for every $\ell \leq \ell' < \ell_{\mathsf{max}}$. Since ℓ_{max} is a constant depending only on V^*, and assuming that for all $\ell \leq \ell' < \ell_{\mathsf{max}}$ the running time of every $\mathsf{Sim}_{\ell'}$ is polynomial, the running time of V_ℓ^* and therefore also of Sim_ℓ must be polynomial. By induction we have that for every $\ell \in [\ell_{\mathsf{max}}]$ the running time of Sim_ℓ is bounded by a polynomial, and therefore the entire simulation is polynomial time.

Using the above proof, we complete the proof that Sim_ℓ constructs a valid trapdoor witness. Sim_ℓ constructs a program Π and auxiliary input y, and we need to show that the running time of $\Pi(y)$ is bounded by some super-polynomial function $T(n)$. The running time analysis above implies that for every $\ell \in [\ell_{\mathsf{max}}]$, the running time of V_ℓ^* and the size of the auxiliary input y constructed by Sim_ℓ are polynomial. The simulator component Sim_ℓ constructs a program Π that simulates V_ℓ^* sending it messages from y. It follows that the running time of $\Pi(y)$ is polynomial and therefore bounded by $T(n)$.

Proof that the simulated view and the real view are indistinguishable. For $0 \leq \ell \leq \ell_{\mathsf{max}}$, let H_i be the hybrid experiment that is identical to the execution of Sim except that every session executing the the protocol $\langle P_{\ell'}, V_{\ell'} \rangle$ for $\ell' \leq \ell$ follow the honest prover strategy using the valid witness w_j for the statement $x_j \in \mathcal{L}$ in that session. of that session. Since $\mathsf{Sim}_{\mathsf{load}}$ simulates the responses to the sessions initiation messages perfectly we have that:

$$H_{\ell_{\mathsf{max}}} = \mathsf{View}_{V^*}(\boldsymbol{x}, \boldsymbol{w}, z), \quad H_0 = \mathcal{S}(\boldsymbol{x}, z) \ .$$

It is therefore sufficient to prove that for every $0 \leq \ell < \ell_{\mathsf{max}}$, $H_\ell \approx_c H_{\ell+1}$. By the definition of the server algorithm Server, the number of sessions with load parameter ℓ is at most n^ℓ. For $0 \leq i \leq n^\ell$, let $H_{\ell,i}$ be the hybrid experiment that is identical to H_ℓ except that the first i sessions executing of the protocol $\langle P_\ell, V_\ell \rangle$ follow the honest prover strategy using a the valid witness w_j for the statement $x_j \in \mathcal{L}$ in that session. It follows that:

$$H_{\ell,n^\ell} = H_{\ell+1}, \quad H_{\ell,0} = H_\ell \ .$$

It is therefore sufficient to prove that for every $0 \leq i < n^\ell$, $H_{\ell,i} \approx_c H_{\ell,i+1}$.

Let $H'_{\ell,i}$ be the hybrid experiment that is identical to the $H_{\ell,i}$ except that the execution of the witness-indistinguishable universal argument UA in the proof stage of the i^{th} execution of the protocol $\langle P_\ell, V_\ell \rangle$ uses a valid witness w_j for the session's statement $x_j \in \mathcal{L}$ instead of the trapdoor witness. Note that in an execution of Sim, the randomness of the component Sim_ℓ used for the UA prover executed in the proof stage of the protocol $\langle P_\ell, V_\ell \rangle$ is also used by the components $\mathsf{Sim}_{\ell'}$ for $\ell' < \ell$ in the construction of the program $V_{\ell'}^*$. However, in the experiment $H_{\ell,i}$, all the simulator components $\mathsf{Sim}_{\ell'}$ for $\ell' < \ell$ are replaced by executions of the honest prover. Since the randomness of the component Sim_ℓ used for the simulation of the UA prover in the protocol $\langle P_\ell, V_\ell \rangle$ is not used in any other part of the simulation, it follows from the indistinguishability property of UA that $H_{\ell,i} \approx_c H'_{\ell,i}$.

Note that the experiment $H_{\ell,i+1}$ is identical to the experiment $H'_{\ell,i}$ except that in the experiment $H_{\ell,i+1}$, the prover commitment c given in the preamble stage of the i'th execution of the protocol $\langle P_\ell, V_\ell \rangle$ is a commitment to the all zero string, following the honest prover strategy. As before, the randomness of the component Sim_ℓ used for the simulation of c sent in the protocol $\langle P_\ell, V_\ell \rangle$ is not used in any other part of the simulation and therefore it follows from the computational-hiding property of Com that $H_{\ell,i+1} \approx_c H'_{\ell,i}$.

Overall we have that for every $0 \leq \ell \leq \ell_{\mathsf{max}}, 0 \leq i \leq n^{\ell}$, $H_{\ell,i} \approx_c H_{\ell,i+1}$. Since $\ell \leq \ell_{\mathsf{max}}$ is a constant, n^{ℓ} is a polynomial and therefore we have that for every $0 \leq \ell \leq \ell_{\mathsf{max}}$, $H_{\ell+1} \approx_c H_{\ell}$ and also that $H_{\ell_{\mathsf{max}}} \approx_c H_0$ as required.

3.2 Decreasing the Load Parameter

In this section, we describe a different server algorithm that takes into account the termination of sessions and decreases the load parameter for new sessions accordingly. We start by describing the new server algorithm Server', and then describe the required changes to the simulation.

We identify the technical condition required for the simulation to work, and design a server algorithm Server' that always gives new sessions the lowest possible load parameter such that the technical condition still satisfies. The validity of our simulation relies on the validity of the following technical condition: for a session with load parameter ℓ_i, the number of sessions concurrent to it with load parameters at most ℓ_i is bounded by n^i. Before describing the algorithm Server' let us first introduce some notation. Let t be the number of open sessions at the moment a new client sends its session initiation message. For $i \in [t]$, let ℓ_i be the load parameter for the i'th open session. For $i \in [t]$, let t_i be the total number of sessions with load parameters at most i that are concurrent to session i. First note that if we set the load parameter of the new session to ℓ then for every session i such that $\ell_i \geq \ell$, the value t_i increases by 1. This will contradict the technical condition only if the value of t_i was already at its maximal allowed value n^{ℓ_i}.

Using the above notation, the algorithm Server' is easy to describe: Server' will set the load parameter of a new session to be the minimal value ℓ such that for every session i with $\ell_i \geq \ell$ we have $t_i < n^{\ell_i}$. While the behavior of the server algorithm Server' is not obvious, we can prove that it satisfies some natural conditions. For example we can show that if no sessions with load parameter ℓ are currently active, then the load parameter assigned to the next session to start cannot exceed ℓ.

Modifying the simulator. Next we discuss the necessary changes to the simulator. In the current description of the simulator, every program V^*_{ℓ} that Sim commits to, internally emulates V^* starting from its initial state. As a result, we must give V^*_{ℓ} auxiliary input z that consists of the messages in all concurrent sessions with load parameter at most ℓ starting from the beginning of the concurrent execution. The problem is that the definition of the server algorithm Server' does not guarantee that such auxiliary input z is sufficiently short. Instead it only gives a bound on the number sessions with load parameter at most ℓ that are executed *concurrently* to the current session. In particular, Server' does not guarantee anything about the number of sessions that terminated before the current session had started. The solution is based on the observation that providing V^*_{ℓ} auxiliary input z that contains messages sent before the current session had started is wasteful. Instead, Sim can commit the a program $\tilde{V}^*_{\ell}$ that already contains these messages hardwired into it.

References

1. Barak, B.: How to go beyond the black-box simulation barrier. In: FOCS. pp. 106–115 (2001)
2. Barak, B., Goldreich, O.: Universal arguments and their applications. SIAM J. Comput. 38(5), 1661–1694 (2008)
3. Canetti, R., Kilian, J., Petrank, E., Rosen, A.: Black-box concurrent zero-knowledge requires (almost) logarithmically many rounds. SIAM J. Comput. 32(1), 1–47 (2002)
4. Canetti, R., Lin, H., Paneth, O.: Public-coin concurrent zero-knowledge in the global hash model. In: TCC. pp. 80–99 (2013)
5. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC. pp. 494–503 (2002)
6. Chung, K.M., Lin, H., Pass, R.: Constant-round concurrent zero knowledge from p-certificates. In: FOCS (2013)
7. Deng, Y., Goyal, V., Sahai, A.: Resolving the simultaneous resettability conjecture and a new non-black-box simulation strategy. In: FOCS. pp. 251–260 (2009)
8. Dwork, C., Naor, M., Sahai, A.: Concurrent zero-knowledge. In: STOC. pp. 409–418 (1998)
9. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. 18(1), 186–208 (1989)
10. Goyal, V.: Non-black-box simulation in the fully concurrent setting. In: STOC. pp. 221–230 (2013)
11. Goyal, V., Jain, A., Ostrovsky, R., Richelson, S., Visconti, I.: Concurrent zero knowledge in the bounded player model. In: TCC. pp. 60–79 (2013)
12. Gupta, D., Sahai, A.: On constant-round concurrent zero-knowledge from a knowledge assumption. IACR Cryptology ePrint Archive 2012, 572 (2012)
13. Kilian, J., Petrank, E.: Concurrent and resettable zero-knowledge in poly-loalgorithm rounds. In: STOC. pp. 560–569 (2001)
14. Pandey, O., Prabhakaran, M., Sahai, A.: Obfuscation-based non-black-box simulation and four message concurrent zero knowledge for np. IACR Cryptology ePrint Archive 2013, 754 (2013)
15. Pass, R., Rosen, A., Tseng, W.L.D.: Public-coin parallel zero-knowledge for np. J. Cryptology 26(1), 1–10 (2013)
16. Persiano, G., Visconti, I.: Single-prover concurrent zero knowledge in almost constant rounds. In: ICALP. pp. 228–240 (2005)
17. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent zero knowledge with logarithmic round-complexity. In: FOCS. pp. 366–375 (2002)
18. Richardson, R., Kilian, J.: On the concurrent composition of zero-knowledge proofs. In: EUROCRYPT. pp. 415–431 (1999)
19. Rosen, A., Shelat, A.: Optimistic concurrent zero knowledge. In: ASIACRYPT. pp. 359–376 (2010)