On the Concurrent Composition of Zero-Knowledge Proofs

Ransom Richardson[1] and Joe Kilian[2]

[1] Groove Networks, 800 Cummings Center, Beverly, MA 01915
rrichard@groove.net
[2] NEC Research Institute
joe@research.nj.nec.com

Abstract. We examine the concurrent composition of zero-knowledge proofs. By concurrent composition, we indicate a single prover that is involved in multiple, simultaneous zero-knowledge proofs with one or multiple verifiers. Under this type of composition it is believed that standard zero-knowledge protocols are no longer zero-knowledge. We show that, modulo certain complexity assumptions, any statement in NP has k^{ϵ}-round proofs and arguments in which one can efficiently simulate any $k^{O(1)}$ concurrent executions of the protocol.

Key Words: Asynchronous Attacks, Zero Knowledge, Black-box Simulation.

1 Introduction

Zero-knowledge proofs [11] and arguments [1] are interactive protocols between a prover (or arguer), P, and a verifier, V, which informally yield no knowledge except for the validity of the assertion. The original formal definition of zero-knowledge considered a very minimal context, and almost immediately, unexpected problems emerged when attempting to apply the notion of zero-knowledge to more practical contexts; the notion of zero-knowledge has been refined accordingly. For example, to make zero-knowledge closed under sequential composition, a number of researchers ([18,20,12]) have proposed a modified definition, known as *auxiliary zero-knowledge*. A still cleaner model, motivated by these issue, is that of *black-box simulation zero-knowledge* [18]; all of the results we will discuss are for this model.

In practice, it is often desirable to run a zero-knowledge proof many times in parallel, so as to lower the error probability without increasing the round complexity. Unfortunately, it is not clear how to efficiently simulate an arbitrary zero-knowledge proof in parallel in polynomial time. Indeed, Goldreich and Krawczyk [10] have shown that for any language L outside of BPP, there is no 3-message protocol for L whose parallel execution can be simulated in black-box zero-knowledge. In their model, the verifier has oracle access to a truly random function; given the existence of cryptographically secure pseudorandom generators, the oracle can be reduced to simply a private string. However, based on

J. Stern (Ed.): EUROCRYPT'99, LNCS 1592, pp. 415–431, 1999.

reasonable computational assumptions, there exist constant-message (indeed, 4 messages suffice) interactive proofs and arguments whose parallel versions remain black-box simulatable.

1.1 Concurrent Repetition

Parallel repetition combines many versions of the same protocol in lock step. When V is supposed to send its ith message, it must send the ith message for all of the parallel runs of the interactive proof. It cannot, for example, delay sending the first message from Game 2 until it has seen the first response in Game 5.

However, in practice, one may wish to engage in many proofs simultaneously and concurrently. For example, one may conceivably give a zero-knowledge proof to establish ones identity whenever one accesses an internet-based service. Different processes may access a number of different services, with no synchronization. This scenario allows for an attack in which a verifier engages in many proofs with the prover, and arbitrarily interleaves the messages in these protocols. Intuitively, the verifier can run some of the protocols ahead in an attempt to gain information that will enable it to attack some of the other protocols.

Beth and Desmedt [3] first discussed such concurrent attacks in the context of identification protocols, and show how to defend against such attacks if parties have precisely synchronized clocks and the adversary is forced to delay its actions.

Dwork, Naor and Sahai [6] consider the role of concurrent attacks on zero-knowledge protocols. They give 4-round zero-knowledge protocols for NP, assuming a weak constraints on the synchrony of weak players: there exist a pair (α, β), where $\alpha \leq \beta$, such that when a good player has observed the passage of β units of time, then every other good player has observed the passage of at least α units of time. Dwork and Sahai [7] reduce (but do not eliminate) the timing constraints required by their defense.

A natural question is defend against arbitrary scheduling without any use of timing. A negative result by Kilian, Petrank and Rackoff [15] extends the Goldreich-Krawczyk result to concurrent attacks, for essentially the same model. They show that for any 4-message proof system for a language L, if one can black-box simulate polynomially many asynchronous proofs, then $L \in BPP$.

1.2 Our Model

Following [6], we consider a malicious verifier V that is allowed to run up to k interactive proofs with the prover P, where k is a free parameter. For our results, k may be replaced with $k^{O(1)}$. Within each proof, V must follow the proper order of the steps, but may arbitrarily interleave steps between different proofs. For example, V may execute the first step of Proof 1, then execute all of Proof 2 in order to obtain an advantage when it executes the second step of Proof 1.

For a given, presumably malicious verifier, V, the simulator is given access to V, but not to the details of its internal state. It is allowed to run V, receiving "requests" for different proofs, and send V responses for these proofs. We assume

without loss of generality that V waits for the response before continuing (it never hurts to receive as much information as possible from the prover before sending one's next message). V is allowed to schedule k proofs arbitrarily, subject to the constraint that within each proof the steps are properly ordered.

Following the standard notion of black-box simulatability, the simulator S is allowed to save V's state and rewind V back to a previously saved state. For ease of explication, we do not explicitly state when S is saving V's state, but speak only in terms of rewinding (S may save V's state after every message). Without loss of generality, we assume that V's state includes all of the messages sent to it, though when restored to a previously saved state, no messages are sent since the state was saved are remembered (i.e., we use the reasonable notion of V's "memory"). Given V's initial state, S's interaction with V induces a distribution on V's final state. S's goal is for this distribution to be statistically or computationally indistinguishable from the distribution on V's final state after interacting with P.

Note that in our modeling of the adversary, we are considering *ordering* attacks, but not *timing* attacks [16] in which one uses the actual response time from the prover to obtain information. There are implementation-specific defenses to such attacks [16]; these methods and concerns are orthogonal to our own.

Similarly, we assume that while the verifier can delay a given message M so that other messages are received before M, it cannot delay M so as to make it unclear whether M is actually going to arrive. That is, the prover and simulator can at some point know that no further messages are arriving. Without this stipulation, even a single execution of most protocols seem impossible to simulate: a malicious verifier V might with probability n^{-2C} wait for n^C time-steps before giving its next answer, where C is either ∞ or a large constant unknown to S. This attack forces S to either keep on waiting or risk giving a slightly (but non-negligibly) distorted simulation.

1.3 Results of This Paper

For ease of exposition, we assume the existence of a certain publicly agreed upon bit commitment schemes, both from the prover to the verifier and from the verifier to the prover. We use an unconditionally binding, computationally private bit commitment scheme from the prover to the verifier. We use a computationally binding, unconditionally private bit commitment scheme from the verifier to the prover. The former can be based on one way functions [14,17], and the latter can be based on collision-resistant hash functions [4].

Our main result is a transformation on zero-knowledge protocols for statements in NP. Our transformed protocol for a statement T (or a proof of knowledge) has two parts: an $O(m)$-message *preamble*, for some parameter m and a *main body*. The main body consists of a zero-knowledge proof of knowledge for a witness to a statement T', which is a modified version of T. A witness for T is also a witness for T'. The longer the preamble, the more resistant the resulting argument is to concurrent attacks.

Theorem 1. *Assume the existence of the commitment schemes described above, and a proof system or argument for $T \in NP$ as described above. Let ϵ be an arbitrary positive constant and let $m = k^{\epsilon}$. The transformed protocol remains a proof of knowledge for T. Furthermore, there exists a polynomial-time black-box simulation for any concurrent attack using at most $k^{O(1)}$ versions of the proof. This simulation achieves computational indistinguishability.*

As we mention in Section 4, there is no need for a public bit-commitment scheme; this convention simply drops some easily handled cases from our simulation and proof.

Quite recently, Rafail Ostrovsky and Giovanni Di Crescenzo have proposed a different solution for defeating concurrent attacks without out timing [19]. Their solution requires a round complexity that is greater than m, an a priori upper bound on the number of attacker; hence, m must be known and bounded in advance. In our solution, $m = k^{\epsilon}$ is possible, and more to the point m need not really be known in advance, though the larger m is, the longer the simulation takes. However, the result in [19] uses no additional complexity assumptions, and is thus an incomparable result.

1.4 Techniques Used

We use a technique of Feige, Lapidot and Shamir [8] in order to convert witness indistinguishable protocols into zero-knowledge protocols. Instead of proving Theorem T, the prover proves a technically weaker theorem, $T \vee W$, where W is a statement that will fail to hold (or for which the prover will fail to have a witness of) with extremely high probability. However, in the simulation, S obtains a witness for W, and may then act as an ordinary prover. Similarly, we set up our proof system so that the simulator will have a "cheating" witness to the statement being proven.

Discussion Indeed, at first glance it may appear that the method from [8] can be used unchanged. Recall that in the scenario of [8], the world begins with an agreement on a pseudorandom generator $g : \{0,1\}^{\ell} \rightarrow \{0,1\}^{2k}$ and the generation of a random string $R \in \{0,1\}^{2k}$ (for ℓ suitably large). Then, any proof of T is replaced with a proof that T is true or $g^{-1}(R)$ exists. To simulate the world from its creation, the simulator S generates g and $R = g(Q)$ for a random $Q \in \{0,1\}^{\ell}$. Then S has a witness, (Q), for any statement of the form "T or $g^{-1}(R)$ exists." By an appeal to the witness indistinguishability of the underlying zero-knowledge proof, S is indistinguishable from any other prover for this statement, despite the fact that its witness is quite different than that used by an actual prover.

Space precluded a detailed discussion, but we note that our construction gives a simulatability result that is more "standard" in the zero-knowledge framework. Also, we do not need a common string, guaranteed to be random.Although we, for ease of exposition, assume that a suitable bit commitment scheme has been

standardized, we can relax this assumption with only a trivial change to the protocol and no substantive change to the simulator and its proof. On a high level, our methods don't try to "break" or alter the commitment scheme in any way, and thus this scheme can be decided on at the beginning of the protocol.

1.5 Guide to the Rest of the Paper

In Section 2 we describe our transformation and how to simulate it. In Section 3, we analyze the efficiency and efficacy of our simulator. In 4 we discuss some simple extensions of our technique, and some open questions.

2 Transforming the Protocol

2.1 The Protocol

Let T be the statement that P is attempting to prove. We insert an $O(m)$ message preamble to the proof. Instead of simply giving a proof of T, P and V will each randomly choose and commit to m numbers, $p_1, p_2, ...p_m$ and $v_1, v_2, ...v_m$ respectively. P will then prove that either T is true or that for some i $p_i = v_i$.

$V \rightarrow P$: Commit to v_1, v_2,... v_m
$P \rightarrow V$: Commit to p_1

$V \rightarrow P$: Reveal v_1
$P \rightarrow V$: Commit to p_2
...
$V \rightarrow P$: Reveal v_i
$P \rightarrow V$: Commit to p_{i+1}
...
$V \rightarrow P$: Reveal v_m

$P \leftrightarrow V$: Zero-Knowledge Proof that ($\exists i$ s.t. $v_i = p_i$) $\vee$ (T is true)

The protocol begins with $m + 1$ message exchanges. First V sends P a commitment to uniformly chosen $v_1, \ldots, v_m \in \{0,1\}^q$, for some suitably large q. For simplicity, we assume that this commitment is information-theoretically secure. P responds by sending a commitment to p_1. In exchange $i + 1$, for $1 \leq i < m$, V reveals v_i and P commits to p_{i+1}. Finally, V reveals v_m. At the conclusion of these exchanges, P responds by giving a zero-knowledge proof that either T is true or that for some i $p_i = v_i$. In the argument model, P gives a statistical zero-knowledge proof that it knows either:

- a witness for T, or
- a witness for a pair (i, REVEAL) such that on seeing REVEAL in the revelation of p_i, V would accept that $p_i = v_i$.

Note that P doesn't reveal which witness it knows, just that it knows one or the other. The general protocols of [13] and [1] may be used for this step (conceivably, more efficient protocols may be designed for useful special cases). The details of this interactive proof (argument) are unimportant.

There are two ways in which P may cause V to accept. Either it proves that T is true or it takes the "easy option" by showing that some $p_i = v_i$. However, regardless of a (possibly malicious) prover $\hat{P}$'s strategy, the easy option will be available with probability at most $m2^{-q}$; by setting q sufficiently large, this option occurs with negligible probability. Hence, the protocol remains a proof (of knowledge) of T.

2.2 Why We Can Simulate the Proof

Since there is so little chance of guessing v_i, P's strategy is to choose p_i at random, or 0^q, and simply proceed with the proof of T. Thus, for the correct prover, the preamble is irrelevant and for a malicious prover, the preamble is not useful. However, the simulator, S, can use the preamble to its advantage. After seeing v_i, it can rewind the conversation to the point where it is required to send p_i, and choose $p_i = v_i$. Because V committed to these v_i in the first message, S need not worry that the v_i change after the rewind as long as it doesn't rewind past the first message of the proof (which it might do while simulating a different proof).

Once S has ensured that for some i $p_i = v_i$, we say it has *solved* the protocol. It can complete the rest of the simulation (of this proof) without any further rewinding. When the actual proof begins, S has an actual witness to the statement being proved, and can therefore proceed according to the algorithm used by the actual prover. Appealing to the witness indistinguishability of the zero-knowledge proof, it is impossible to distinguish whether S used this witness or a witness for T.

2.3 Caveates

We mention three (of many) caveats regarding this approach. First, rewinding a single step in one proof can render irrelevant the simulations of many other proofs; nesting effects can cause exponential blowups in the simulation (as discussed in [6]). However, since S has m places it can rewind in order to fix a proof's simulation, it can choose good times to rewind.

Second, an improper use of rewinding can alter the distribution on the verifiers' questions, rendering the simulation invalid. Our simulation runs in two modes: *normal* and *look-ahead*. The normal mode is a step by step simulation of the k concurrent proofs. A step made in a normal proof is never rewound, facilitating the analysis of the distribution of the verifiers' messages. The look-ahead mode is invoked when the simulator, running in normal mode, is required to commit to p_i to the verifier, for one of the simulated proofs. In look-ahead mode, the simulator will explore many possible simulation paths and return with either the value of v_i, allowing S to solve this proof, or a statement that

this is an unsuitable time to solve the proof. Once the look-ahead is complete, the simulator continues the normal-mode simulation. We show that S can use the information obtained in its look-ahead mode yet still maintain a faithful simulation.

We must also take care to avoid malleability attacks [5], where one links a commitment to the value of another parties commitment. For example, the prover might try to commit to the verifier's value, always achieving a match, or the verifier might try to foil the simulation by somehow opening up values different than those committed by the prover. Our assymetric choice of commitment protocols prevents these attacks.

2.4 Preliminaries

Let $v_{i,j}$ and $p_{i,j}$ denote the values of v_i and p_i committed to in the simulation of the jth proof. These values depend greatly on where we are in the simulation. In particular, they may be defined and then undefined when S rewinds the simulation.

Within a simulation path, we number the protocols in order of appearance. Thus, orderings may differ between different paths, but this will not affect our analysis.

During the preamble of a simulated protocol j, the verifier commits to m strings, $v_{1,j}, \ldots, v_{m,j}$. By a standard argument, the probability that $v_{i,j}$ is successfully revealed to be different values at different times after being committed to is negligible. Thus, we'll speak of the "value" of $v_{i,j}$. However, if the simulator rewinds past the point where the verifier committed to $v_{1,j}, \ldots, v_{m,j}$, these values become undefined.

At some point in the protocol, the simulated verifier will send a string that is supposed to reveal $v_{i,j}$. This string will either actually reveal this unique value or fail to reveal any value. Note that in the actual protocol, P aborts in the latter case.

During a path in the simulation, we say that a simulated protocol j is *solved* if, for some i, $v_{i,j}$ has been determined and $p_{i,j}$ has not yet been sent. We say that a simulated protocol j is *aborted* if the verifier fails to reveal $v_{i,j}$ when scheduled to do so. Note that rewinding and choosing a new path can change whether a simulated protocol is solved or is aborted.

If protocol j has been solved, the simulator simulates the prover's messages as follows. If the prover is supposed to send $p_{i,j}$ then it sends $v_{i,j}$ if it is known and an arbitrary string otherwise. During the main body of the proof, the simulator has a witness to the statement being proved, and acts according to the algorithm used by an honest prover. In particular, no rewinding is ever needed.

2.5 The Simulator

We let k_0 be a constant set to the initial value of k, which denotes the number of concurrent proofs. We show that if the number of message exchanges in the

preamble is $m = {k_0}^{\epsilon}$ for any $\epsilon > 0$ then the above protocol can be simulated in time $k^{O(1/\epsilon)}$. This section describes the look-ahead procedure used by the simulator and how the simulator works in normal mode.

Look-Ahead Mode An *n-proof look-ahead* is a procedure used by S to gather information about the messages V is likely to send in the future. In the look-ahead phase, the simulation is allowed to proceed until certain events occur that cause it to be (prematurely) ended. The limited duration of the look-ahead makes it much more efficient than a full simulation, and indeed it is called many times during the simulation.

The n-proof look-ahead is called when S is required to commit to some $p_{i,j}$. The main simulator runs many ($100{k_0}^2$, to be precise) look-ahead simulations; we call these *threads*. We first describe one of one such thread, then describe how to use the results from many threads.

In the course of the simulation, the simulator is required to commit to strings $p_{a,b}$ and to engage in the main body of proofs. Along the way, it receives the values of strings $v_{a,b}$. A particular run of the n-proof look-ahead terminates when either $v_{i,j}$ has been revealed or the $n+1^{st}$ new proof, which started since the look-ahead began, is seen. The former case means that the mission is accomplished: S can set $p_{i,j} = v_{i,j}$. The latter case means that the simulation is proceeding too far and risks becoming too complicated; it may not be cost effective to keep waiting for $v_{i,j}$ to be revealed.

We differentiate between the protocols $1, \ldots, z$ that have already begun and the protocols $z+1, \ldots, z+n$ that begin during the look-ahead simulation. The look-ahead simulator recursively starts a normal-mode simulator (described later). The normal-mode simulator requires a parameter specifying the maximum number of simultaneous proofs; this parameter is set to n. All messages and requests related to proofs $z+1, \ldots, z+n$ are forwarded to this recursive simulation.

The normal-mode simulation has the property that, with all but negligible probability, by the time $p_{m,a}$ has been committed to, proof a will have been solved (see lemma 8). The look-ahead mode is less careful about solving proofs which began before the look-ahead. In this mode, S may commit to $p_{m,a}$ for an unsolved proof, and subsequently be unable to enter the main body of the proof. However, S will only get stuck if $v_{m,a}$ is revealed. Whenever this happens, S aborts the look-ahead and rewinds. We note that this rewinding will take S to before it committed to $p_{m,a}$, so proof a is now solved. Since less than k_0 proofs can begin before any given look-ahead, only k_0 of the look-ahead simulation paths will be aborted (the effect of these aborted paths is dealt with in lemma 7). Because these simulation paths are always rewound they will not affect the distribution of the normal-mode simulation.

We formally describe the n-proof look-ahead simulation by a case analysis of how the simulator responds to various messages. Only the first three cases are related to the purpose of the look-ahead; the rest are simply to keep the

simulation going in a faithful fashion. By convention, the simulation takes as its first message the message being handled at the time it was called.

$\mathbf{V} \to \mathbf{S}$: Valid revelation of $v_{i,j}$.
$\mathbf{S} \to \mathbf{V}$: Terminate the simulation. *(proof j has been solved)*

$\mathbf{V} \to \mathbf{S}$: A commitment to $v_{1,a}, \ldots, v_{m,a}$, for $a = z + n + 1$.
$\mathbf{S} \to \mathbf{V}$: Terminate the simulation. *(look-ahead is finished)*

$\mathbf{V} \to \mathbf{S}$: Invalid revelation of $v_{i,j}$.
$\mathbf{S} \to \mathbf{V}$: Terminate the simulation. *(no chance of recovering $v_{i,j}$)*

$\mathbf{V} \to \mathbf{S}$: Any message related to protocol a, for $z < a \leq z + n$.
$\mathbf{S} \to \mathbf{V}$: Forward the message to the recursive simulation.
(We assume $1 \leq a \leq z$ in the remaining)

$\mathbf{V} \to \mathbf{S}$: Any message related to a solved proof a.
$\mathbf{S} \to \mathbf{V}$: Answer according to the standard fashion for solved proofs.

$\mathbf{V} \to \mathbf{S}$: Valid revelation of $v_{b,a}$, where a is unsolved and $b < m$.
$\mathbf{S} \to \mathbf{V}$: Commit to an arbitrary (random) value of $p_{b+1,a}$.

$\mathbf{V} \to \mathbf{S}$: Invalid message related to proof a
$\mathbf{S} \to \mathbf{V}$: Sign-off message from simulated prover for proof a.

$\mathbf{V} \to \mathbf{S}$: The value of $v_{m,a}$ for an unsolved proof a.
$\mathbf{S} \to \mathbf{V}$: Abort this line of the simulation. *(the simulator cannot simulate the main body of the proof of an unsolved proof)* However, note that proof a is now solved at the point where the look-ahead simulation began, which allows us to bound how often this bad case occurs.

Combining the Result of the Look-Aheads. Each run of the n-proof look-ahead simulation either returns a solution to the proof ($v_{i,j}$) or announces a failure to do so. To give a faithful simulation, the simulator must flip coins (e.g., when committing to $p_{b,a}$. Thus, there is a probability distribution on these results. In the normal-mode simulation, whenever the look-ahead simulation is invoked it is in fact invoked ${100k_0}^2$ times. If a solution is found in any of these invocations, the proof is solved. Otherwise, we will argue that with high (but not overwhelming) probability, at least n proofs in the actual (normal-mode) simulation will be started before $v_{i,j}$ is revealed.

Normal Mode Simulation Our simulator, working in normal mode, services the requests for up to k asynchronous proofs; the parameter k will be changed during recursive calls. Valid responses from S can take the following form:

- S signs off due to an invalid message.
- S engages in the main body of a proof.
- S commits to some $p_{i,j}$.

Handling invalid messages is trivial. Once S has solved a proof a, it can easily engage in the main body of a proof as a prover, since it has a witness for this proof. We will ensure that with all but negligible probability, S will always have solved a proof before it enters into the main body.

When S must commit to some $p_{i,j}$, it runs the n-proof look-ahead procedure ${100k_0}^2$ times, where $n = \lceil 2k/m \rceil$, in an attempt to recover $v_{i,j}$. If it succeeds it commits to $p_{i,j} = v_{i,j}$; otherwise, it commits to an arbitrary value of $p_{i,j}$.

As before, we describe the behavior of S by its response to various messages.

V $\rightarrow$ **S**: Any message related to a solved proof a.
S $\rightarrow$ **V**: Answer according to the standard fashion for solved proofs.

V $\rightarrow$ **S**: A commitment to $v_{1,a}, \ldots, v_{m,a}$.
S $\rightarrow$ **V**: Invoke the $n = \lceil 2k/m \rceil$-proof look-ahead simulation ${100k_0}^2$ times. If $v_{1,a}$ is recovered, set $p_{1,a} = v_{1,a}$, else set $p_{1,a}$ arbitrarily. Commit to $p_{1,a}$.

V $\rightarrow$ **S**: valid revelation of $v_{b,a}$, $b < m$.
S $\rightarrow$ **V**: Invoke the $n = 2k/m$-proof look-ahead simulation ${100k_0}^2$ times. If $v_{b+1,a}$ is recovered, set $p_{b+1,a} = v_{b+1,a}$, else set $p_{b+1,a}$ arbitrarily. Commit to $p_{b+1,a}$.

V $\rightarrow$ **S**: Any invalid message related to proof a
S $\rightarrow$ **V**: Sign-off message from simulated prover for proof a.

3 Analysis of the Simulation

Theorem 2. *The simulator, S, described in Section 2 is a black box simulator for the protocol in Section 2 that runs in time $k^{O(1/\epsilon)}$ on k non-synchronized proofs when $m = k^{\epsilon}$ for $\epsilon > 0$.*

Proof. (Sketch) This theorem will follow from the following lemmata. Lemma 3 and Lemma 4 show that it runs in time $k^{O(1/\epsilon)}$. Lemma 6 shows that the simulator produces a valid output as long as it never gets stuck. Lemma 8 shows that the chance of getting stuck is negligibly small in m and the security parameter for the bit commitment schemes. □

3.1 Bounding the Running Time

We assume that a simulator can handle a single message and give a response in *unit time*. We note that since we consider the main body of the proof to be a single message, we consider that proof to be given in unit time. A more precise (and more cumbersome) statement is that the running time is $k^{O(1/\epsilon)}$ times the amount of time it takes to perform the main body of the proof.

Lemma 3. *The running time of the simulator is bounded by the function*

$$t(k) = 100m{k_0}^3 t(\lceil \frac{2k}{m} \rceil) + {k_0}^{O(1)}.$$

Proof. (Sketch) First we note that each look-ahead thread begins a recursive simulation that handles up to $\lceil \frac{2k}{m} \rceil$ further proofs. This takes time bounded by $t(\lceil \frac{2k}{m} \rceil)$. Each look-ahead is repeated up to ${100k_0}^2$ times and S may attempt to solve each of the k_0 games by performing these look-aheads in m different places. This results in the coefficient of ${100mk_0}^3$. Each look-ahead thread also handles messages from previous game, whether solved or not. This takes unit time for each message. The number of messages, games and look-aheads are all polynomial in k_0. So the cost of this in all look-aheads is bounded by the ${k_0}^{O(1)}$ term. □

Lemma 4. *The recurrence* $t(k) = {100mk_0}^3 t(\lceil \frac{2k}{m} \rceil) + {k_0}^{O(1)}$ *is* $k^{O(1/\epsilon)}$ *whenm* $= {k_0}^{\epsilon}$.

Proof. (Sketch) At each recursive step, k is divided by ${k_0}^{\epsilon}/2$. Thus the total depth of the recursion is $O(1/\epsilon)$. Both the coefficient of for the recursive term and the cost at each level of the recursion are bounded by ${k_0}^{O(1)}$. Therefore, the total cost is ${k_0}^{O(1/\epsilon)}$. □

3.2 The Simulation Is Valid

Note that (S, V) doesn't just simulate the conversation, it implicitly simulates the internal state of V - that is, the state of the "black-box" V that S is interacting with. We can consider the conversation generated thus far to be part of V's internal state. Thus, the process of S interacting with V constitutes a sequence of transformation on V's state. We can similarly consider the interaction of P and V to be a sequence of transformations on V's state.

We say that S becomes *stuck* if it enters the main body of a proof that hasn't been solved. We designate all other moves made by the simulator as *safe*. Lemma 6 says that S will produce a valid simulation as long as S only performs safe moves.

Lemma 5. *Any sequence of safe operations performs the identical (up to computational indistinguishability) transformations on* V *as the corresponding operations performed by* P.

Proof. (Sketch) Whenever S interacts in a solved proof, it has a witness for the statement to be proven. Due to the witness indistinguishability of the zero-knowledge proof in the main body, and the security of the bit commitment scheme used by the prover, all actions taken by S are computationally indistinguishable from those taken by any other prover.

When S commits to $p_{i,j}$, it may first launch into many recursive subsimulations involving many backtrackings. However, at the end of all these subsimulations, S restores V to its initial state, chooses a value for $p_{i,j}$ and commits to $p_{i,j}$. The value of $p_{i,j}$ depends on the results of these subsimulations; its distribution may be completely different from that generated by P (indeed, whenever a proof is solved, it's distribution is quite different). However, the distribution

of messages sent for the commitment is the same (up to computational indistinguishability), regardless of this value.

Finally, by inspection, S responds to any illegal messages the same way as does P. □

Note that the notion of a corresponding operations makes sense, because neither S nor P can control which *type* of operation it must make in response to V. Here, we are appealing to the computational indistinguishability of the commitment scheme from the prover to the verifier and the witness indistinguishability of the zero-knowledge protocols.

Now, given a particular configuration of V, S may run many simulations, due to the look-ahead mode. However, these simulations are ultimately thrown away. If one ignores all simulation threads arising from *further* recursive calls to the look-ahead mode (a currently active look-ahead mode may be continued), one obtains a unique sequence of transformations on V. This holds regardless of whether the particular configuration of V is encountered in normal mode or look-ahead mode. We call this sequence the *main line* of the evolution of V.

The main line of V from its starting configuration constitutes the simulation of the proofs. The main line from the beginning of a look-ahead thread goes to the point where S has finished this line or has been forced to discontinue the simulation (or get stuck).

Lemma 6. *Consider the evolution of V's configuration along its main line. Assuming that S never gets stuck, this evolution will be indistinguishable from the corresponding evolution obtained by interacting with P.*

Proof. (Sketch) The evolution of V consists of it sending messages to S $\{P\}$ and then having S $\{P\}$ perform operations. As long as S doesn't get stuck, all of its operations will be safe, and by Lemma 5, their effect on V will be indistinguishable from the effect of the corresponding operations performed by P.

It remains to be shown that the evolution of V when it generates its next message in the simulation is faithful to that in the actual protocol. Note that this is a nontrivial statement: S could conceivable run V many times and pick a path in which V sends a message that is amenable to S. However, by inspection of the simulation algorithms, S never selects which path to follow based on what V says. Indeed, it's selection process is completely rigid: Paths taken in look-ahead mode are ultimately not pursued; the main line path is pursued, at least locally (it may be thrown away later if it is part of a larger look-ahead path). Along any mainline path, S obtains V's message exactly once, by running V in the normal matter. Hence, V's internal evolution is also identical to that obtained by interacting with P. □

3.3 Bounding the Probability of Getting Stuck

Lemma 6 implies that the main line from a configuration of V is indeed simulated correctly, as far as it goes (since in look-ahead mode, a simulation is typically

ended prematurely) and as long as S doesn't get stuck. We now show that S gets stuck along any main line with negligible probability.

While in look-ahead mode, S never becomes stuck on a proof which began before the look-ahead, since it simply aborts the thread if it is about to become stuck. From then on that proof is solved, so the number of times S aborts is limited. This strategy cannot be employed in normal mode (at least on the top level of the recursion) since such stopping would constitute a failure to finish the overall simulation.

Recall that for any proof started in normal mode, the simulator tries m times to solve some proof a, by going into look-ahead mode in order to determine $v_{i,a}$ for each i. We characterize the various outcomes of this attempt.

- (complete success) The look-ahead recovers $v_{i,a}$, solving proof a.
- (win by forfeit) During the main line, the next mesage from V regarding proof a is ill formed (does not reveal $v_{i,a}$ when it should have).
- (honorable failure) The look-ahead fails to solve proof a, and during the main line more than $2k/m$ new proofs are begun before V reveals $v_{i,a}$.
- (dishonorable failure) The look-ahead fails to solve proof a, then during the main line, at most $2k/m$ new proofs are begun, after which V then sends a correct revelation of $v_{i,a}$.

Clearly, a single complete success or a win by forfeit will cause the game to be solved. We must show that with high probability, one of the m attempts will result in a complete success or a win by forfeit.

We next observe that an honorable failure can happen at most $m/2$ times. Since the normal-mode only handles k games, $m/2$ honorable failures result in more than $(2k/m) \cdot (m/2) = k$ new games, a contradiction. Thus, it remains to bound the probability of $m/2$ dishonorable failures.

We will prove that the chance of getting stuck at any level in the recursion is negligibly small by using induction on k (the number of proofs in a call to the simulator). The base case, $k = 1$, is when the simulator is solving a single proof. The look-aheads will never encounter another proof and as a result can never become stuck. The following lemma will be needed to complete the inductive step.

Lemma 7. *During any attempt to solve proof a the probability of a dishonorable failure is at most* $1/10$ *as long as the chance of getting stuck in a look-ahead is negligibly small.*

Proof. (Sketch) S attempts to solve proof a by performing $100{k_0}^2$ look-aheads after being asked to commit to $p_{i,a}$. We note that since these look-aheads have a negligibly small chance of getting stuck, lemma 6 implies that they give a valid sampling of the possiblie paths of the conversation. In order for a dishonorable failure to happen V must not reveal $v_{i,a}$ during any of those look-aheads but then reveal it when S continues on in normal mode. We may assume that the chance of V revealing $v_{i,a}$ is at least $p = 1/10$.

Now we must show that the chance of S not learning $v_{i,a}$ in any of the look-aheads is smaller than p. We must remember that some of the look-aheads could

have been aborted if V revealed $v_{m,b}$ for some unsolved proof b. But each time a look-ahead is aborted we solve proof b. So the maximum number of times the look-ahead is aborted is $k_0 - 1$. Thus we need to show that the chance of seeing $v_{i,a}$ at least k_0 times is greater than p.

It is easy to verify that for any $b < a/3$, $\binom{a}{b-1} \leq \binom{a}{b}/2$. Therefore the chance of seeing $v_{i,a}$ at most k_0 times is at most twice the cost of seeing it exactly k_0 times. This cost is less than

$$\binom{100{k_0}^2}{k_0} p^{k_0}(1-p)^{100{k_0}^2-k_0}$$

$$< 100^{k_0} {k_0}^{2k_0} (1/10)^{k_0} (9/10)^{99{k_0}^2}$$

Which is dominated by the $(9/10)^{99{k_0}^2}$ and therefore (much) less than $1/10$.

Note that the above argument glossed over the fact that the safe steps are only computationally close to "real" steps. By a standard argument, this does not affect the analysis by more than a negligible amount. □

Lemma 8. *The chance of S getting stuck is negligibly small in m and the security parameter of the bit commitment scheme.*

Proof. (Sketch) We use induction on k. In the base case of $k = 1$ the simulator will trivially never get stuck because there are no proofs to get stuck on. By induction we may assume that all look-aheads (which have smaller values of k) get stuck with negligibly small probability. The total number of look-aheads must be polynomial because the total running time of S is polynomial (see Lemma 3 and Lemma 4). Therefore the total chance of getting stuck in any look-ahead is also negligibly small. By lemma 7 the chance of each dishonorable failure is less than $1/10$. We note that for S to get stuck during a proof it must have had at least $m/2$ dishonorable failures. The chance of this is less than $\binom{m}{m/2}(1/10)^{m/2} < 2^{m/2}(1/10)^{m/2} < 2^{-m/2}$. There are a total of at most k proofs on which the simulator can get stuck, so the total chance of getting stuck is negligibly small.

Note that as with the previous argument, the above argument glossed over the fact that the safe steps are only computationally close to "real" steps. For this reason, the probability of getting stuck is negligibly small, not exponentially small. □

4 Extensions and Open Questions

4.1 Extensions

Our proof assumes that k is known. In the case where k is unknown, S may start by assuming that $k = 1$ and double k and restart the interaction each

time it discovers that k is larger than it assumed. It is easy to verify that this has no effect on the output distribution and that the total running time is still polynomial.

We do not really need to have globally agreed upon commitment schemes. The modification is to add two messages to the protocol in which each party specifies the commitment scheme that should be used to commit to it (first the verifier, then the prover). The property we desire is that the commitments are unconditionally guaranteed to be zero-knowledge, regardless of how it is specified (illegal specifications are treated as invalid messages). Thus, a party can use the other party's bit commitment system without any loss of its security. The party specifying the protocol has no obvious reason to make it easy to break, but this is not enforced. Such bit commitment schemes are easily constructed based on [4] and [14,17]. Due to space limitations, details are omitted from this manuscript.

We also note that there is essentially no reason why our construction doesn't work even if the k proofs are different.

4.2 Open Questions

It is unknown whether there is a perfect simulation for non-synchronized composition of zero-knowledge proofs. We note that in the case when V always follows the protocol and successfully reveals v_i we can modify the simulator so that it never gets stuck. We do this by having the simulator look-ahead from the point it is forced to commit to p_m until v_m is revealed if the proof has not yet been solved. If V is required to reveal v_m this is always successful. Then instead of being stuck, S is just in a bad case which may take longer to simulate, but it is still possible to do in polynomial time. Assuming the existence of a perfect commitment scheme, this non-cheating verifier allows us to provide a perfect simulation.

It would also be useful to show that it is possible to simulate non-synchronized composition with a constant number of message exchanges in the preamble. Again, assuming that the verifier always reveals v_i, our protocol can be modified so that it runs in time $k^{O(\log k)}$ with a constant number of messages. This is interesting because it shows that the techniques used in [15] to show that any four message protocol takes time $2^{O(k)}$ to simulate can not be extended to any constant round proof.

As mentioned in the introduction, we do not address timing issues in the verifier's attack. Even modeling what zero-knowledge should mean in this context, in a way that is both useful and possible, is an interesting open question.

Finally, it is paradoxical that such seemingly meaningless alterations in the protocol can restore zero-knowledge. Intuitively, it seems implausible that the protocol has been made more secure in practice. Ideally, one would like to have a notion of security that is more or less invariant under such transformations. The notions of witness hiding and witness indistinguishable protocols are good steps in this direction.

5 Acknowledgments

Kilian would like to thank Cynthia Dwork, Uri Feige, Moni Naor, Amit Sahai and Erez Petrank for many illuminating conversations on this subject.

References

1. G. Brassard, D. Chaum, C. Crépeau. Minimum Disclosure Proofs of Knowledge. Journal of Computer and System Sciences, Vol. 37, 1988, pp. 156–189.
2. C. Brassard, C. Crepeau and M. Yung, "Constant-Round Perfect Zero-Knowledge Computationally Convincing Protocols", *Theoretical Computer Science*, Vol. 84, 1991, pp. 23-52.
3. T. Beth and Y. Desmedt. Identification tokens - or: Solving the chess grandmaster problem. In A. J. Menezes and S. A. Vanstone, editors, *Proc. CRYPTO 90*, pages 169–177. Springer-Verlag, 1991. Lecture Notes in Computer Science No. 537.
4. Damgård, Torben P. Pedersen, and Birgit Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In Douglas R. Stinson, editor, *Proc. CRYPTO 93*, pages 250–265. Springer, 1994. Lecture Notes in Computer Science No. 773.
5. D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In ACM, editor, *Proceedings of the twenty third annual ACM Symposium on Theory of Computing, New Orleans, Louisiana, May 6–8, 1991*, pages 542–552, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1991. IEEE Computer Society Press.
6. Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero knowledge. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 409–418, New York, May23–26 1998. ACM Press.
7. C. Dwork and A. Sahai. Concurrent zero-knowledge: Reducing the need for timing constraints. *Lecture Notes in Computer Science*, 1462, 1998.
8. U. Feige, D. Lapidot, and A. Shamir. Multiple non-interactive, zero-knowledge proofs based on a single random string. In *Proc. 31st Ann. IEEE Symp. on Foundations of Computer Science*, pages 308–317, 1990.
9. U. Feige and A. Shamir, "Zero Knowledge Proofs of Knowledge in Two Rounds", *Advances in Cryptology – Crypto 89 proceedings*, pp. 526-544, 1990.
10. O. Goldreich, H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. SIAM J. on Computing, Vol. 25, No.1, pp. 169-192, 1996
11. S. Goldwasser, S. Micali, C. Rackoff. The Knowledge Complexity of Interactive Proofs. Proc. 17th STOC, 1985, pp. 291-304.
12. S. Goldwasser, S. Micali, C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. SIAM J. on Computing, Vol. 17, 2(1988), pp. 281-308.
13. S. Goldwasser, S. Micali, A. Wigderson. Proofs that Yield Nothing But their Validity or All Languages in NP have Zero-Knowledge Proofs. J. of the ACM, Vol. 38, No. 3, July 1991, pp. 691-729.
14. Johan Hastad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Construction of a pseudo-random generator from any one-way function. Technical Report TR-91-068, International Computer Science Institute, Berkeley, CA, December 1991.
15. Kilian, Petrank, and Rackoff. Lower bounds for zero knowledge on the internet. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1998.

16. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 18–22 August 1996.
17. Moni Naor. Bit commitment using pseudo-randomness. In *Advances in Cryptology: CRYPTO '89*, pages 128–137, Berlin, August 1990. Springer.
18. Y. Oren. On the cunning powers of cheating verifiers: Some observations about zero knowledge proofs. In Ashok K. Chandra, editor, *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 462–471, Los Angeles, CA, October 1987. IEEE Computer Society Press.
19. R. Ostrovsky and G. Di Crescenzo. Personal Communication, September 15, 1998.
20. M. Tompa and H. Woll. Random self-reducibility and zero-knowledge interactive proofs of possession of information. In *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science*, pages 472–482, 1987.