6.876/18.426: Advanced Cryptography April 23, 2003

Lecture 18: Concurrent Zero Knowledge

Scribed by: Jonathan Derryberry

1 Motivation

Up to this point, our proofs of zero knowledgeness of protocols have been based on a simple model of communication in which there are two agents, a prover and a verifier. However, in the real world, situations arise in which there is one prover P (e.g. a web server) and many verifiers V_i (e.g. clients) that have different delays but are all trying to interact with the prover at the same time. Suppose an adversary, A controls many of these clients, and starts many sessions with the prover. In particular, suppose A deviously interleaves and nests the execution of its clients. If this is the case, is "normal" zero knowledge enough to prevent A from extracting knowledge from P?

2 Review

To see why normal ZK may not be enough, first recall [GMW91]'s ZKPS for the graph 3-colorability problem.

$$\mathsf{P} \xrightarrow{c(\pi(G))} \mathsf{V}$$

$$\mathsf{P} \xleftarrow{e} \mathsf{V}$$

$$\mathsf{P} \xrightarrow{d(\pi(G)_e)} \mathsf{V},$$

where c is a commitment, d is a decommitment, $\pi(G)$ is the coloring of G subject to a permutation of the 3 colors used, and $d(\pi(G)_e)$ is shorthand for the decommitment of the colors of the nodes of the queried edge. This protocol is zero knowledge because a simulator S can give all of the edges except one garbage colors, then reveal if and only if V asks for the one edge that is properly colored and rewind otherwise. Clearly, for a protocol to have zero knowledge against many clients, it is important for the protocol to be parallelizable, so that an adversary who begins many sessions in parallel cannot learn anything from P.

Can the above scheme be parallelized? The answer is no. If S has to commit to k colorings at once, there is a low probability that it will guess all of the edges that V will query (the probability is $|E|^{-k}$). Thus, S runs in exponential expected time.

However, [GK96] fixed this by adding an extra round to the protocol at the beggining

that forces V to commit to the particular edges it will query as follows

$$\mathsf{P} \xleftarrow{c(e_1),\ldots,c(e_k)} \mathsf{V}$$

$$\mathsf{P} \xrightarrow{c(\pi_1(G)),\ldots,c(\pi_k(G))} \mathsf{V}$$

$$\mathsf{P} \xleftarrow{d(e_1),\ldots,d(e_k)} \mathsf{V}$$

$$\mathsf{P} \xrightarrow{d(\pi_1(G)_{e_1}),\ldots,d(\pi_k(G)_{e_k})} \mathsf{V}.$$

By adding the first round of commitments to the edges that will be queried, S can now rewind any V' after its first query and know exactly which edge to fix for each graph.

3 The Problem with Parallelizability

However, even though parallelizable protocols such as [GK96]'s seems to suffice for a server to use to handle multiple clients simultaneously, consider the following problem, which involves clients maliciously nesting sessions so that the simulator S that [GK96] present runs in exponential time.

Suppose an adversary A controls verifiers $\mathsf{V}_1, \ldots, \mathsf{V}_k$. A behaves as follows: "For $i = 1 \ldots k$, make V_i send commitments to the edges to query and wait for the response from P. For $i = k \ldots 1$, make V_i reveal the edges queried and wait for the response from P." A visual depiction of these interactions with $k = 3$ appears in Figure 1.

Now consider what S has to do to simulate this interaction, assuming that V_i's messages depend on the messages between P and V_{i-1}, so that the simulations cannot be carried out separately. $\mathsf{S}(i, n)$, which simulates n nested sessions starting with V_i, works as follows:

1. If $i = n + 1$ return because there are no more nested simulations.
2. Get V_i's commitment and respond with a bogus commitment to graphs.
3. Recursively call $\mathsf{S}(i + 1, n)$.
4. Retrieve the edges to be queried.
5. Rewind V_i.
6. Recursively call $\mathsf{S}(i + 1, n)$.
7. Retrieve the edges to be queried and respond with decommitments to the colors of the edges.

A call to $\mathsf{S}(1, n)$ simulates the real proof. However, this simulation requires $O(2^k)$ time as evident from the two recursive calls with a maximum depth of k. Note that both of the recursive calls are necessary because the queries that V_i makes depend on the responses to V_{i-1}, which change after the correct edges to fix are learned. Thus, the natural way of simulating the nested sessions using the simulator for the parallel sessions does not work.

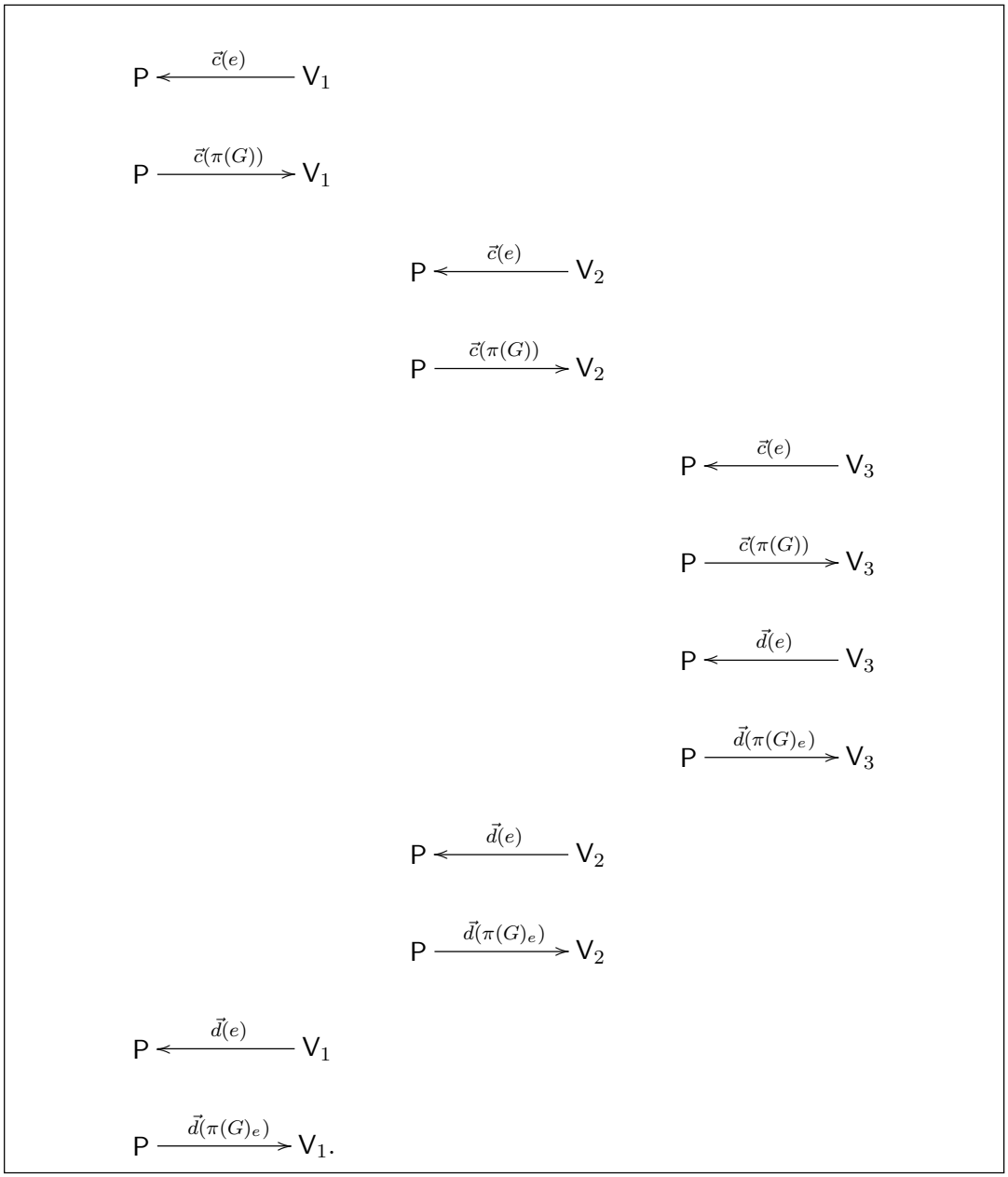


Figure 1: The nested interactions that A makes. The symbols $\vec{c}$ and $\vec{d}$ are shorthand for the $1 \ldots k$ commitments and decommitments in the [GK96] protocol.

4 Adding a Timing Assumption

[DNS99]'s solution to this problem was to add a "timing assumption," which prevents V from nesting sessions. In the setting they propose, any two communicating parties M_1 and M_2 must obey an (α, β) timing constraint that implies that if M_2 begins its measurement after

M_1 according to "God's clock," then M_2 will finish after M_1 finishes. This is accomplished, for example, by having P *delay* β units of time on its local clock after receiving a request from V_i before sending a response. Moreover, once P is expecting a response from V_i, it only waits α units of time on its local clock before *timing out*.

By employing this strategy, the nesting depth of an adversary A's sessions are limited. Suppose A starts the [GK96] protocol with V_1 and waits for the first response, m_1, before starting V_2. Note that when P sends m_1 it starts a timer with α units of time for V_1's response. V_1 wants to wait a long time to send this response. However, if α is set less than β as it is supposed to be, V_1 must respond before V_2 receives m_2 because P waits β units of time before sending back m_2. Thus, it is impossible to nest sessions as described in Section 3. To summarize this sequence of events among P, V_1, and V_2:

- V_1 one starts the protocol
- P waits β units of time before responding
- P responds with m_1 and starts a timer with an α time countdown, after which it will time out
- V_2 starts a new session with a message that depends on m_1
- P waits β units of time and responds with m_2 (in the meanwhile, V_1's session times out)
- V_2 responds immediatly to try to complete the nesting, but V_1's session has already timed out with P

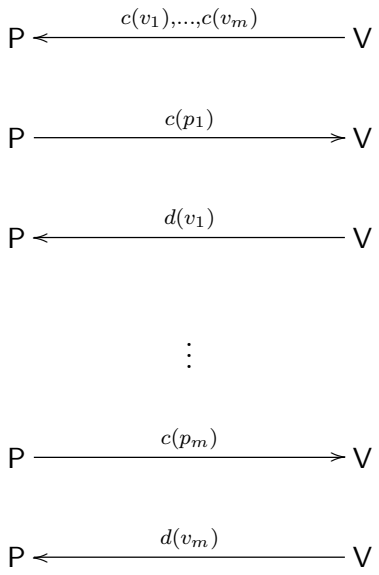
The added timing assumption allows a parallelizable ZKPS to remain zero knowledge, even when an adversary tries to nest sessions. Moreover, the soundness is unaffected. However, completeness may be affected because an honest verifier may have limited computational resources and be unable to respond.

5 Adding a Preamble to Protect Zero Knowledge ([FLS90], [KPR01], [RK99])

An alternative to adding a timing assumption is to add additional rounds in the beginning of the protocol to give S a back door to performing the simulation. Suppose the goal is to prove $x \in L$ and a NIZKPS exists for L.

To overcome any adversary A's interleaving strategy, the proof system for L is modified. A preamble is added with some communication between P and V. Then P gives a WI proof of the related theorem $x \in L \vee y \in \Lambda$. The statement $x \in L$ is what is proved in the real system, while S proves $y \in \Lambda$. The idea is that because the proof is witness indistinguishable, it is difficult to tell which part of the disjunction was proven $x \in L$ or $y \in \Lambda$. However, it is important that a cheating P' is not able to prove $y \in \Lambda$, otherwise soundess would be destroyed because P' would have a trap door. On the other hand, S will be able to do this. Note that an convenient property of a witness indistinguishable proof is that it is parallel composable.

So what communication goes before the proof that $x \in L$? First, V sends commitments $c(v_1), \ldots, c(v_m)$ where $v_i \in \{0,1\}^k$. Then P chooses random $p_1 \in \{0,1\}^k$ and sends $c(p_1)$ to V. The V responds by opening up v_1, sending $d(v_1)$. These exchanges are continued for m rounds where $m = \omega(\log^2 k)$. Graphically, this looks like



After this preamble, P proves $x \in L \vee \exists i\ p_i = v_i$. It is obviously overwhelmingly unlikely that that any $p_i = v_i$, so P is forced to prove $x \in L$, and soundness is not significantly affected. However, note that S can rewind the tape after learning V''s decommitments and easily set $p_i = v_i$ for some i. Because the proof of $x \in L \vee \exists i\ p_i = v_i$ is WI, there is no probabilistic polynomial time algorithm that can determine which statement was proved. One caveat is that a cheating prover could simply echo one of V's commitments by setting $c(p_i) = c(v_i)$. This is bad, but can be overcome by using mutually independent commitments or if V uses a perfectly hiding commitment and P uses a perfectly binding commitment. To see why this works, if V uses a perfectly hiding commitment, P cannot have any knowledge about what the committed value is when it must bind itself perfectly to one value.

To see that this scheme helps acheive zero knowledge regardless of A's interleaving strategy, note that the S is *oblivious* to A's interleaving strategy. Some reasoning for this will be given in the next lecture.

References

[DNS99] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge.

[GK96] O. Goldreich and H. Krawczyk. On the Composition of Zero Knowledge Proof Systems. SIAM J. on Computing, Vol. 25, No. 1. pp. 169-192, 1996.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing But their Validity, and a Methodology of Cryptographic Protocol Design. J. of the ACM 38. pp. 691-729, 1991.

[FLS90] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero Knowledge Proofs Based on a Single Random String. Proceedings of 31st IEEE Symposium on Foundations of Computer Science, pp. 308-317, 1990.

[KPR01] J. Kilian, E. Petrank, and R. Richardson. Concurrent Zero-Knowledge Proofs for NP Thirty-Third Annual ACM Symposium on the Theory of Computing July 6-8, 2001.

[RK99] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. Lecture Notes in Computer Science, Vol. 1592, Springer-Verlag, pp. 415-431, 1999.