6.876/18.426: Advanced Cryptography 28.4.2003.

Lecture 19: Concurrent Zero Knowledge in Polylogarithmic Rounds

Scribed by: Nenad Dedić

1 Introduction

The subject of these notes is *concurrent zero knowledge*, in particular the construction given in [KP01].

Zero knowledgeness property of zero knowledge proof systems is defined with respect to an adversarial verifier that does not attempt to run multiple instances of a protocol concurrently. It is possible to prove that such protocols can be composed serially without substantial loss of security. However, it is unclear whether parallel (more generally, concurrent) composition preserves security. Simulating a protocol then seems to become computationally more demanding in the concurrent setting. Namely, there are protocols that, to the best of our current knowledge, take exponential time to simulate.

However, requiring *concurrent composability*, is natural, as there is no reason to limit the adversary's capability to run multiple protocols concurrently. Therefore, it is of interest to attempt to construct zero-knowledge proof systems that are concurrent composable.

2 Concurrent Composable Zero Knowledge: The Construction

In this section, we give an overview of the construction of [KP01].

2.1 Stating the Objective

It has been demonstrated in previous lectures that many statements that need to be proven in various protocols are NP. Furthermore, NP has complete problems, which is particularly convenient, as it often only need be demonstrated that a complete problem has a certain property, from which it follows that all other NP problems have it. This was certainly the case with zero knowledge proofs.

Therefore, we declare our goal to **construct a zero-knowledge proof system for any** NP **language that is concurrent composable**. Concurrent composability means that **we allow the verifier to interact with multiple independent provers, intertwining those interactions in any way it pleases** (the two extremes being fully serial and fully parallel).

2.2 Tools and Assumptions

Now we turn to describe which cryptographic tools (primitives) are used to construct concurrent composable zero knowledge proof systems. We also identify the necessary assumptions.

Zero knowledge proofs for NP

We assume that for any language in NP, a zero knowledge proof system can be constructed. As a brief reminder, zero knowledge proof system for a language L is a protocol between two parties, the prover P and the verifier V, that satisfies three properties (for detailed definitions, please refer to the previous lectures):

- **completeness:** any $x \in L$ is accepted with overwhelming probability
- **soundness:** any $x \notin L$ is accepted with negligible probability
- **zero knowledgeness:** for any $x \in L$ and any probabilistic polynomial time V^*, there is an oracle machine $\mathsf{S}^{\cdot}$ (simulator) such that $\mathsf{S}^{\mathsf{V}^*}(x)$ is indistinguishable from the view of V^*

Bit commitment schemes

We assume that commitment schemes exist. In a commitment scheme, there are two parties: one of them, Alice, wishes to commit to a message, but delay the revealing of it; the other one, Bob, wishes to be certain that, when Alice reveals the message, it indeed is the one she commited to.

A non-cryptographic solution to the problem is: Alice and Bob communicate; at some point, Alice puts a message in a safe visible by both her and Bob, locks it and keeps the key; they continue the conversation; when she wishes to decommit, she gives the key to Bob. When Bob unlocks the safe, he is reasonably certain that Alice could not have tampered with the message inside while it was locked.

In terms of cryptography, a bit commitment scheme is a triple of probabilistic polynomial time procedures ($\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}$) with the following properties:

- functional:
  - CK $\leftarrow$ ($\mathsf{Setup}(1^{\mathsf{k}})$ (given a security parameter, the setup procedure outputs a public commitment key)
  - $(c, d) \leftarrow \mathsf{Commit}_{\mathrm{CK}}(m)$ (given a message, the commit procedure outputs a commitment c and an opening value d)
  - $m \leftarrow \mathsf{Open}_{\mathrm{CK}}(c, d))$ (given a commitment and a decommitment value, the open procedure outputs m; m may either be a message, or a special symbol that denotes the invalidity of the input (c, d))
- security:
  - **Hiding:** For any adversary[1] A, it is infeasible to generate two messages m_0 and m_1 such that A can distinguish their commitments c_0 and c_1 ($(c_i, d_i) = \mathsf{Commit}_{\mathrm{CK}}(m_i)$).
  - **Binding:** For any adversary A, it is infeasible to generate a triple (c, d, d') such that (c, d) and (c, d') open to different messages, i.e, $m \leftarrow \mathsf{Open}_{\mathrm{CK}}(c, d)$, $m' \leftarrow \mathsf{Open}_{\mathrm{CK}}(c, d)$ and $m \neq m'$.

[1]The restrictions on the computational resources of the adversary will be discussed later in the text.

We consider two flavours of commitment schemes: *perfectly binding* and *statistically hiding*. The difference lies in the limitations to the resources available to the adversaries.
A **perfectly binding commitment scheme** is a commitment scheme whose

- hiding property holds against a **probabilistic polynomial time** adversary, with **negligible insecurity**
- binding property holds against an **unbounded adversary**, with **information-theoretic security**

A **statistically hiding commitment scheme** is a commitment scheme whose

- hiding property holds against an **computationally unbounded** adversary, resources, with **negligible insecurity**
- binding property holds against a **probabilistic polynomial time** adversary, with **negligible insecurity**

It should be noted that the commitment schemes are concurrent composable, i.e, running multiple schemes concurrently does not jeopardize either hiding or binding property.

Witness Indistinguishability

Even though we mention zero knowledge proof systems as one of the key ingredients in building concurrent composable ones, it is in fact sufficient that we use witness indistinguishable proof systems.

A witness indistinguishable proof system must satisfy completeness and soundness as described for zero knowledge proof systems, but the last requirement is weaker. We only require that no probabilistic polynomial time machine can tell if w_1 or w_2 were used as a witness. For more details refer to [Gol01].

Assumptions

We have seen earlier in the course that existence of commitment schemes implies the existence of zero knowledge proof systems for NP. In addition to that, it is known that statistically hiding commitment schemes exist if collections of claw-free permutations exist [DPP93] and that perfectly binding commitment schemes exist if one-way functions do [Nao91].

But zero knowledge proof systems for NP, two round perfectly binding and two round statistically hiding commitment schemes are the only ingredients we need in constructing concurrent composable zero knowledge proof systems. Thus existence of collections of claw-free permutations suffices.

2.3 The Construction

We first state the main result of [KP01]:

Theorem [Main]: Assume that collections of claw-free permutations exist. Let k be a complexity parameter bounding the size of the input. The verifier is polynomial time in k and the concurrent proof may consist of polynomially many in k instances. Under this condition, a zero knowledge proof system exists for any $L \in \text{NP}$ that is **computational**, **black-box**, **concurrent composable** and **it has** $\omega(\log^2 k)$ **rounds**.

Let us now see the construction. Let $L \in \text{NP}$ and $T \in L$ (T is an NP statement that we wish to prove). The prover and the verifier first exchange some messages that facilitate concurrent composability, and then proceed to prove in a zero knowledge fashion a statement T' very similar to T. The messages exchanged prior to proving T' are called *preamble* while the messages that constitute the proof of T' are called *body*.

Protocol [CZK]:

1.	V → P:	commit to random	$v_1, \ldots, v_m$
2.	P → V:	commit to	p_1
3.	V → P:	reveal	v_1
4.	P → V:	commit to	p_2
5.	V → P:	reveal	v_2
...			
$2m-1$.	V → P:	reveal	v_{m-1}
$2m$.	P → V:	commit to	p_m
$2m+1$.	V → P:	reveal	v_m
$2m+2$	P ↔ V:	carry out a zero knowledge proof that:	$(T \in L) \vee (\exists i) p_i = v_i$
...		(T', the modified statement, is precisely	$(T \in L) \vee (\exists i) p_i = v_i$)

where $m = \omega(\log^2 k)$.

The verifier's commitments are statistically hiding, and the prover's are perfectly binding.

If the verifier fails to open any of its commitments correctly, the protocol should be terminated.

The idea behind this protocol is that the preamble makes it possible for the simulator to find i such that $p_i = v_i$ after a reasonable number of rewinds. This in turn constitutes a witness for T', so the simulator may act as a real prover in the body of the protocol, thus creating a view distributed as expected.

On the other hand, soundness is not significantly affected because the real prover is unable to rewind the protocol, so it is computationally infeasible for it to find i such that $v_i = p_i$. Thus in order to give a convincing proof, it must prove $T \in L$. Completeness is not affected at all, because T' can be proven by proving $T \in L$. The next section explains completeness and soundness in some more detail.

2.4 Completeness and Soundness of the Protocol

For any zero knowledge proof system, let us call the probability that the prover fails to prove a true statement *completeness error*, and the probability that the verifier accepts a false statement *soundness error*.

Let (P,V) be a zero knowledge proof system for L (for the original language, not for the modified statement). Denote with ϵ_c and ϵ_s the completeness and soundness errors of (P,V).

Claim [Completeness]: The completeness error of the protocol CZK is no greater than ϵ_c, that of the original zero knowledge proof system.

This is so, because if $T \in L$, the prover simply proves that statement in the body. The probability of failing is ϵ_c.

Claim [Soundness]: The soundness error of the protocol CZK is no greater than $\epsilon_s + \nu(k)$, where ν is a negligible function.

Let us first see where the soundnes might go wrong. The prover might conceivably get an advantage by commiting to p_i such that both $\mathsf{Commit}(v_i)$ and $\mathsf{Commit}(p_i)$ open to the same value. Note that to do that, the prover need not necessarily learn all the information about some v_i. But if the verifier uses a non-malleable commitment scheme, then it is not in prover's power to construct commitments that are correlated to $\mathsf{Commit}(v_i)$

However instead of using a non-malleable commitment scheme, we achive the desired security by having the verifier use a statistically hiding commitment scheme, and prover use a perfectly hiding one. This eliminates any but negligible chance that prover can open one of its commitments to the same value as the verifier. Namely, because of statistically hiding property, the value p_i cannot be non-negligibly correlated with v_i. On the other hand, because of the perfectly binding property, $\mathsf{Commit}(p_i)$ can be open in only one way. Therefore the protocol is sound.

2.5 Zero Knowledgeness of the Protocol in Concurrent Setting

As usual, we demonstrate the zero knowledgeness of the protocol by constructing a simulator. The simulator is a probabilistic polynomial time and it interacts in a black box manner with the verifier. Its goal is to produce a view whose distribution is indistinguishable from that of the real verifier. Of course we allow the verifier to execute multiple interactions with the prover concurrently.

We already said that the preamble is the simulator's fulcrum. It suffices that for each proof, the simulator finds one index i such that $p_i = v_i$. That index is a witness for T', so the simulator may proceed to prove T'. A naive way to find a suitable index is to simply rewind after the verifier opens $\mathsf{Commit}(v_i)$ for some i, and then set $p_i \leftarrow v_i$ and continue. After rewinding and setting p_i, the verifier must open the same value as it did before rewinding, since it commited to it even earlier. Seemingly, this achieves our goal. However, rewinding by need, as we may call this schedule, may lead to exponential time simulation. This is so because many other instances may be nested in between $\mathsf{Commit}(p_i)$ and $\mathsf{Open}(\mathsf{Commit}(v_i))$.

We now observe that several problems need to be addressed:

- a more clever rewinding schedule must be employed
- the rewinding schedule must enable the simulator to learn the verifiers' secrets

- it needs to be proven that proving that some v_i is equal to the corresponding p_i does not noticably slant the simulator's distribution away from the verifier's view (remember, the real prover proves that $T \in \text{NP}$, while the simulator proves that $(\exists i) v_i = p_i$)
- it needs to be proven that the simulator runs in polynomial time

The following sections describe how these issues are solved, if static scheduling is assumed. That means that we assume that the adversary, who starts the proof sessions as it pleases, and intersperses their messages, determines the schedule of the messages in advance, independently of the content of the messages. In the paper [KP01], dynamic scheduling is addressed as well.

Rewinding

The simulator needs to be able to provide a "good" portion of the view for each proof (remember, we are considering concurrent setting and many proofs may be running concurrently). The simulator does so by rewinding after learning v_i, so that it is able to set p_i equal to v_i, which is the desired event for each proof. Thus we say that the simulator *solves a proof* if it is able to set $p_i = v_i$ for some i.

Cleary, in order to solve a proof the simulator must, after seeing the verifier's decommitment $\mathsf{Open}(\mathsf{Commit}(v_i))$, rewind past the point where it commited to p_i. However, the simulator must not rewind too far, because it might rewind past the verifier's commitments, thereby rendering useless v_i's that it had learned. Also the simulator must not let the first run (the one before rewinding) go past the preamble, because otherwise the verifier might just notice that something has gone wrong and stop responding.

Once again we stress that a naive solution: picking an index i and, after learning v_i rewinding just enough to set $p_i = v_i$, does not necessarily work. Instead we employ an oblivious rewinding schedule.

Since we are assuming static scheduling by the adversary, and we only care about preambles, we may view the adversary's schedule as mk slots which contain pairs of messages. Each pair consists of a commitment by the simulator (prover) to p_i, followed by the verifier revealing v_i. Determining a rewinding strategy now amounts to "walking back and forth" on these mk slots.

The oblivious rewinding strategy that gives all the desired properties is described by the following procedure:

```
$\mathsf{Schedule}(a..b)$:
    if $b - a = 1$ then return the following schedule:
        execute $a$
        execute $b$
    compute $s_1$ as $\mathsf{Schedule}(a..\frac{a+b-1}{2})$
    compute $s_2$ as $\mathsf{Schedule}(\frac{a+b+1}{2}..b)$
    return the following schedule:
        execute schedule $s_1$ and remember the values $v_i$
        rewind $\frac{a+b-1}{2} \to a$
        execute schedule $s_1$ again, this time setting $p_i$ appropriately
        execute schedule $s_2$ and remember the values $v_i$
        rewind $b \to \frac{a+b+1}{2}$
        execute schedule $s_2$ again, this time setting $p_i$ appropriately
```

The simulator's rewind strategy is then $\mathsf{Schedule}(1..mk)$. For example, when $mk = 8$, the slots executed are: 1, 2, 1, 2, 3, 4, 3, 4, 1, 2, 1, 2, 3, 4, 3, 4, 5, 6, 5, 6, 7, 8, 7, 8, 5, 6, 5, 6, 7, 8, 7, 8.

The Rewinding Schedule Solves All Proofs With High Probability

We already said that the simulator solves a proof if it learns some v_i of the proof in the first run (before rewinding). Then after rewinding, it sets $p_i = v_i$. However, it is not obvious that the above rewinding schedule guarantees that every proof is solved.

Let us first identify a necessary condition that during a particular rewind, a proof is solved.

Definition: We say that a rewind $l \to k$ *may solve a proof* Π if:

- the first round of Π is in a slot before k
- the last round of Π is in a slot after l
- exactly two rounds of Π[2] are in slots $k, k+1, \ldots, l$
- the first round of Π appears in the first half of $l \to k$ and the second round in the second half

This definition captures a necessary condition that the rewind $l \to k$ is the smallest rewind that has a chance of solving Π. The first two bullets guarantee that the rewind has a chance of solving Π, and the other two that it is the minimal one.

The distinction between the notions "solve a proof" and "may solve a proof" is justified, because not every rewind that may solve a proof actually solves it. That can happen if the verifier, whether maliciously or erroneously, decides not to disclose v_i during the first run. However, [KP01] demonstrate that there are enough rewinds that may solve any particular proof, to compensate for the verifier's malice or flaws.

Lemma [Rewinds]: For any schedule of k copies of the proof preambles, each with m pairs of messages, if the preamble of a specific proof Π completes in slot l, then there are at least $\lceil \frac{m}{\log(mk)+1} \rceil - 2$ rewinds that complete by slot l and that may solve Π.

In addition to that, it can be computed that each rewind that may solve a proof, indeed solves it with probability at least $\frac{2}{3}$. Roughly, the computation is based on the fact that the only way in which the verifier might force the simulator to not solve a proof, when it may, is to hide v_i in the first run and reveal it in the second one. Because of the hiding property of the commitment scheme used by the prover, the verifier is not able to tell which run is

[2]In this definition, by "proof" and Π we mean "preamble" and "preamble of Π".

currently happening, and thus cannot determine when to hide v_i and when not to v_i better than by flipping a coin.

By the previous lemma, using $m = \omega(\log^2 k)$ we get that there are

$$a \geq \frac{\omega(\log^2 k)}{\log k + \log m} = \omega(\log k)$$

rewinds in which a particular proof Π may be solved. Now because there are less than $\frac{1}{3}$ chance of a proof not being solved during any of these rewinds, and all of them are disjoint, we have that with probability at most

$$\left(\frac{1}{3}\right)^a = \left(\frac{1}{3}\right)^{\omega(\log k)} = \nu(k)$$

the proof Π is not solved during any rewind (ν is a negligible function). Now using the union bound we get that there is only negligible probability that any proof is not solved at all.

The Indistinguishability of the Views

Theorem [Indistinguishability of the Views]: If the simulator $\mathsf{S}^{\mathsf{V}^*}$ solves each proof, then the view of the verifier V^* (in the proof system $(\mathsf{P}, \mathsf{V}^*)$) is indistinguishable from the view generated by S.

Let us first see what are the issues in proving this theorem. First, the preambles generated by the simulator are different than the ones generated by the real proof system. That is because in simulator generated preambles, there are some values of i for which $p_i = v_i$, whereas that is improbable for the real proof system. Second, the witness used by the simulator is different than the one used by the real proof system: the simulator proves that $p_i = v_i$, while the real system proves that $x \in L$.

The first observation presents no problem, because if anyone were able to distinguish the preambles, that would imply breaking a commitment scheme. The second one also presents no problem, because any zero knowledge proof system is witness indistinguishable too, so noone can tell which witness was used to prove the theorem.

The full proof can be found in [KPR01], but let us just say here that a hybrid argument is used, with one hybrid point. The hybrind point is a view generated by a modified simulator $\mathsf{S}'^{\mathsf{V}^*}$ that gets a witness w, to the statement "$x \in L$", as extra input. It generates preambles as the usual simulator S, i.e. it tries to solve the proofs by setting $v_i = p_i$ for some i. But when it executes the body of the proof, it behaves as the real prover P, and proves that $x \in L$ using w.

Distinguishing the view generated by $\mathsf{S}'^{\mathsf{V}^*}$ from that generated by S then implies that the zero knowledge proof system for L is not witness indistinguishable, which in turn implies that it is not zero knowledge at all.

On the other hand, distinguishing the view generated by $\mathsf{S}'^{\mathsf{V}^*}$ from that generated by $(\mathsf{P}, \mathsf{V}^*)$ implies that a commitment scheme is not semantically secure.

Running Time of the Simulator

It can proven that the number of rewinds is polynomial in the security parameter k. On the other hand, a polynomial in k number of operations takes place during each rewind. Hence the simulator runs in polynomial time.

3 Conclusion

[CKPR01] prove that $\Omega(\frac{\log k}{\log\log k})$ rounds are needed for black box concurrent zero knowledge proofs for any language outside BPP. Thus the result of [KPR01] is not far from this lower bound. However, [PS02] further improve the round complexity of concurrent zero knowledge by giving an $O(\log k)$ round protocol.

References

[CKPR01] R. Canetti, J. Kilian, E. Petrank, A. Rosen. Black-box Concurrent Zero-Knowledge Requires Omega(log n) Rounds In *ACM Symposium on Theory of Computing*, pp. 570-579, 2001.

[DPP93] I. Dåmgard, T. Pedersen, B. Pfitzmann. On the Existence of Statistically Hiding Bit Commitment Schemes and Fail-Stop Signatures. Advances in Cryptology - CRYPTO '93. Proceedings, pp. 250-265. LNCS 773, Berlin: Springer-Verlag, 1994.

[Gol01] O. Goldreich. Foundations of Cryptography: Basic Tools. Cambridge University Press, 2001.

[Gol] O. Goldreich. Foundations of Cryptography - Fragments of a Book Available from O. Goldreich's web page: `http://www.wisdom.weizmann.ac.il/~oded`

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. Proceedings of 17th ACM Symposium on the Theory of Computing (STOC), pp. 291-304, 1985.

[KP01] J. Kilian, E. Petrank. Concurrent and Resettable Zero-Knowledge in Poly-logarithmic Rounds Proceedings of the thirty-third annual ACM symposium on Theory of computing, 2001, pp. 560-569

[KPR01] J. Killian, E. Petrank, R. Richardson. Concurrent Zero-Knowledge Proofs for NP. Available from E. Petrank's web page: `http://www.cs.technion.ac.il/~erez`

[Nao91] M. Naor. Bit Commitment Using Pseudo-Randomness. Journal of Cryptology, vol.4, 1991, pp. 151-158

[PS02] M. Prabhakaran, A. Sahai. Concurrent Zero Knowledge Proofs with Logarithmic Round-Complexity. Available from Electric Colloqium on Computational Complexity, ECCC, 2002. `http://citeseer.nj.nec.com/prabhakaran02concurrent.html`

[RK99] R. Richardson, J. Killian. On the Concurrent Composition of Zero-Knowledge Proofs. Proceedings of *Advances in Cryptology - EUROCRYPT '99*, May 1999, LNCS Vol. 1592, Springer 1999, pp. 415-431

[Sha90] A. Shamir. IP = PSPACE. Proceedings, 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 11-15, 1990.