CMSC 858K — Advanced Topics in Cryptography April 1, 2004

Lecture 17

Lecturer: Jonathan Katz *Scribe(s): Chiu Yuen Koo, Nikolai Yakovenko, Jeffrey Blank*

1 "Limitations" of NIZK Proof Systems

In the NIZK proof systems we have seen, we have assumed that the prover and verifier share a common random string (CRS). Although in many applications of NIZK such an assumption presents no problem (e.g., in the application to chosen-ciphertext-secure public-key encryption, the receiver chooses a random string and includes it in her public key), in the general case it is not clear where this string comes from (if no trusted third-party is assumed). A natural question is whether NIZK can be achieved *without* a common random string. In fact, it is easy to see that this is impossible for any "non-trivial" language:

Lemma 1 *NIZK proofs are impossible without a CRS for any language $L \notin \mathcal{BPP}$.*

Proof (Sketch) Let (P, V) be an NIZK proof system for some language $L \in \mathcal{NP}$, with polynomial-time simulator Sim. For simplicity, we assume the protocol has perfect completeness and negligible soundness error, although these assumptions are not necessary. Our goal is to prove that $L \in \mathcal{BPP}$. By definition of NIZK, for any $x \in L$ the distribution of proofs π output by $\mathcal{P}(1^k, x, w)$ (where w is a witness for x) is indistinguishable from the distribution of proofs π output by $\mathsf{Sim}(1^k, x)$. In particular, we have $\mathcal{V}(\mathsf{Sim}(1^k, x), x) = 1$ with probability negligibly close to 1 for $x \in L$ (this follows from perfect completeness).

We show how to decide membership in L using a PPT algorithm A (by definition, this implies $L \in \mathcal{BPP}$). A works as follows: given x, it computes $\pi \leftarrow \mathsf{Sim}(1^k, x)$ and outputs 1 iff $\mathcal{V}(\pi, x) = 1$. By what we have said above, the probability that A outputs 1 when $x \in L$ is negligibly close to 1. On the other hand, *soundness* of the proof system implies that the probability that A outputs 1 when $x \notin L$ is negligible (since otherwise we have an easy way to "fool" $\mathcal{V}$ into accepting a proof of a false statement). $\square$

It is instructive to see where the previous proof fails when a common random string r is available. In that case the simulator gets to choose r, so a simulator which outputs "valid-looking" proofs for false statements does not contradict the soundness of the NIZK proof system (since soundness holds with respect to the *fixed* string r that the cheating prover cannot change).

The above demonstrates that *interaction* is necessary if we want to have zero-knowledge protocols which do not rely on any set-up assumptions (such as a common random string).

2 Zero-Knowledge (Interactive) Proof Systems

Note: Definitions of zero-knowledge are quite subtle and, although we will do our best to be accurate, we may omit some details for simplicity. The interested reader is referred to other works [2, 3] for more details and a more rigorous and careful treatment.

We first define the notion of zero knowledge. Note that the definition is more complex than in the non-interactive case because in the latter case we only need to consider the proofs generated by a legitimate (honest) prover; in the interactive case, on the other hand, we must also consider the actions of a prover *when interacting with a dishonest verifier* who may not follow the protocol.

Definition 1 A pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ is called a zero-knowledge (ZK) proof system for a language $L \in \mathcal{NP}$ (with $L_k \stackrel{\text{def}}{=} L \cup \{0,1\}^{\leq k}$) if it satisfies the following properties:

1. **(Completeness)** For all k, all $x \in L_k$, and all witnesses w for x,
$$\Pr[\mathcal{V}(1^k, x) \text{ accepts when interacting with } \mathcal{P}(1^k, x, w)] = 1.$$
2. **(Soundness)** For all $x \notin L_k$ and all (even all powerful) $\mathcal{P}^*$, the following is negligible:
$$\Pr[\mathcal{V}(1^k, x) \text{ accepts when interacting with } \mathcal{P}^*(x)].$$
3. **(Zero knowledge)** For all PPT (cheating verifiers) $\mathcal{V}^*$, there exists an (expected) polynomial time simulator Sim such that the following are computationally indistinguishable for any $x \in L_k$ and witness w for x:
   - the view of $\mathcal{V}^*(1^k, x)$ when interacting with $\mathcal{P}(1^k, x, w)$;
   - the output of $\mathsf{Sim}(1^k, x)$.

$\diamondsuit$

Note that the simulator is given exactly what the verifier knows. Thus, the intuition behind a ZK proof is that "anything a poly-time cheating verifier can learn from its interaction with $\mathcal{P}$ it could learn on its own, anyway (in essentially the same amount of time)". In the definition above, we allowed Sim to run in *expected* polynomial time rather than requiring it to run in (strict) polynomial time.[1] Also, the above definition considers only *uniform* poly-time cheating verifiers; the "standard" definition, however, requires the zero-knowledge condition to hold even for *non-uniform* poly-size cheating verifiers (i.e., poly-time verifiers with *auxiliary inputs*) for good reason. (We have chosen to ignore the issue for ease of presentation.) See [2, 3] for further details.

We first show a ZK proof system for the language of *graph isomorphism*.

Definition 2 Let $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ be two graphs, where V_i is the set of vertices and E_i is the set of edges of G_i. The graphs G_1 and G_2 are *isomorphic* (denoted by $G_1 \approx G_2$) if there exists a permutation $\varphi : V_1 \to V_2$, such that $(u, v) \in E_1 \Leftrightarrow (\varphi(u), \varphi(v)) \in E_2$. Such a φ is called an *isomorphism* from G_1 to G_2. We also let $\varphi(G_1)$ denote the graph obtained by "permuting" the vertices of G_1 according to φ and preserving edges; clearly, $G_2 = \varphi(G_1)$ (hopefully, using φ to refer to a permutation of the vertex set of G_1 as well as to a permutation of the graph G_1 will not cause confusion). $\diamondsuit$

[1] Allowing the simulator to run in expected poly-time is somewhat "controversial", and certainly strict polynomial time simulation is preferable. However, the only currently-known way to achieve "reasonably-efficient" ZK protocols is to allow expected poly-time simulation. For more discussion, see [2] or [1].

Define the language *graph isomorphism* by $L \stackrel{\text{def}}{=} \{(G_1, G_2) \mid G_1 \approx G_2\}$. Clearly, we have $L \in \mathcal{NP}$ (since it is easy to check that a given isomorphism is valid). Consider the following proof system for L: the prover begins with G_1, G_2, and an isomorphism φ from G_1 to G_2; the verifier knows only G_1, G_2. The protocol proceeds as follows:

1. $\mathcal{P}$ chooses a random permutation φ', and sends $H = \varphi'(G_1)$ to $\mathcal{V}$.

2. The verifier chooses a random $b \in \{1, 2\}$ and sends b to $\mathcal{P}$.

3. We assume the verifier sends a $b \in \{1, 2\}$ (any other response can be taken, by default, to represent a "1"), and the prover will send an isomorphism from G_b to H. In particular, if $b = 1$ then the prover sends φ'; if $b = 2$, the prover sends $\varphi' \circ \varphi^{-1}$ (i.e., the composed permutation).

4. The verifier receives a permutation φ'' and accepts iff $\varphi''(G_b) \stackrel{?}{=} H$.

Theorem 2 *The above is a zero-knowledge protocol (with soundness error* $1/2$*) for graph isomorphism.*

Proof Clearly, $\mathcal{P}$ and $\mathcal{V}$ can be implemented in polynomial time. We now argue completeness. Assume the prover acts honestly, and so $H = \varphi'(G_1)$ for some permutation φ'. If $b = 1$ then the verifier clearly accepts. If $b = 2$, then the verifier also accepts since in this case $\varphi''(G_2) = \varphi' \circ \varphi^{-1}(G_2) = \varphi'(G_1) = H$.

We now show that the above protocol achieves soundness error $1/2$ (meaning that if G_1, G_2 are *not* isomorphic, then no prover can make a verifier accept with probability better than $1/2$). Assume $G_1 \not\approx G_2$, and let H be the graph sent by the prover in the first round. Note that at most one of $G_1 \approx H$ or $G_2 \approx H$ can be true (if they are both true, then by transitivity we have $G_1 \approx G_2$ which we know is not the case). So the prover can respond correctly to at most one of the verifier's possible challenges sent in the third round, and thus will succeed in making the verifier accept with probability at most $1/2$. (The soundness error is not negligible, as required by Definition 1; however, we discuss below an easy way to modify the protocol to achieve negligible soundness error.)

The most interesting property to consider is the zero knowledge of the above protocol. To that end, we show the following simulator for any cheating verifier $\mathcal{V}^*$ (recall from the definition that the simulator is allowed to depend — in fact, must depend — on the verifier):

$\underline{\mathsf{Sim}(1^k, (G_1, G_2))}$
fix random coins ω for V^*
for $i = 1$ to k:
  choose a random permutation φ'
  $b \leftarrow \{1, 2\}$
  $H = \varphi'(G_b)$
  run $\mathcal{V}^*(1^k, (G_1, G_2), H; \omega)$ to obtain its response b'
  if $b = b'$ output view (ω, H, b, φ') and stop
if none of the above iterations have succeeded, output $\perp$

Fix $(G_1, G_2) \in L$ (recall that Definition 2 only requires simulation in this case). We first analyze the probability that Sim outputs $\perp$. In any given iteration of the inner loop and for any value of ω, note that the value of b chosen by the simulator is *perfectly independent* of H: the graph H is a random isomorphic copy of G_b, but that has the same distribution as a random isomorphic copy of $G_{\bar{b}}$ (since $G_1 \approx G_2$). That means that the value of b is independent of the view of V^* in line 4 of the inner loop. Therefore, the probability that $b' = b$ is exactly 1/2. Overall, then, the probability that we *never* have $b = b'$ is 2^{-k}, which is negligible. So, Sim outputs $\perp$ with only negligible probability.

We next claim that, conditioned on Sim not outputting $\perp$, the output of Sim is identically distributed to the view of $\mathcal{V}^*$. To see this, let us consider the elements of the view (ω, H, b, φ') one-by-one: clearly, ω is a random string having the same distribution as ω in a real interaction of $\mathcal{V}^*$ with $\mathcal{P}$. The next component, H, is a random isomorphic copy of G_b for some b, but we have already noted above that this is distributed identically to a random isomorphic copy of G_1 (which is what $\mathcal{P}$ sends in the real protocol). The challenge bit b is then a deterministic function of ω and H (since we have already fixed the coins ω of $\mathcal{V}^*$), and in both the real and simulated experiments is equal to $\mathcal{V}^*(1^k, (G_1, G_2), H)$. Finally, in the simulation φ' is uniformly distributed among isomorphisms mapping G_b to H; again, this is exactly as in a real execution of the protocol.

We remark that the above simulator runs in *strict* polynomial time. ■

The proof system presented above has soundness error 1/2, but it is easy to make the soundness error as small as desired (and, in particular, negligible): simply repeat the above protocol ℓ times to achieve soundness error $2^{-\ell}$. Setting $\ell = \omega(\log k)$, the protocol can still be implemented in polynomial time and the soundness error becomes negligible.

We need to be careful, however, that in changing the protocol we do not destroy its zero-knowledge property! There are two natural ways to implement the ℓ-fold repetition suggested above: the first is to run the ℓ instances of the protocol *in parallel*: in round 1 the prover sends ℓ different graphs $H_1, \ldots, H_\ell$ (each computed using an independent isomorphism); in round 2 the verifier replies with ℓ challenges $b_1, \ldots, b_\ell$, and in round 3 the prover responds to each challenge as in the original protocol. The second natural possibility is to run ℓ instances of the protocol *sequentially* in the obvious way. A drawback of the latter method is that the round complexity becomes $O(\ell)$, whereas when using the first method the round complexity remains the same (i.e., three rounds) as in the original protocol.

Unfortunately, *running the above protocol* **in parallel** *for $\ell = \omega(\log k)$ is not known to result in a zero knowledge protocol.* (It is not known definitively that the protocol is *not*[2] zero knowledge, but there is no known way of proving that the protocol *is* zero knowledge, either.) To see the difficulties that arise, imagine adapting the simulator given earlier for the case of ℓ-fold parallel repetition. Now, the simulator will "guess" in advance the challenge bits $b_1, \ldots, b_\ell$, construct graphs $H_1, \ldots, H_\ell$ in the appropriate way, and then hope that the output $b'_1, \ldots, b'_\ell$ of $\mathcal{V}^*$ satisfies $b'_i = b_i$ for all i. But the probability that this occurs is $2^{-\ell}$ (note that we cannot assume that $\mathcal{V}^*$ uses the same challenge bits every time — for all we know, $\mathcal{V}^*$ may choose its challenge bits based on all the graphs $H_1, \ldots, H_\ell$ sent by $\mathcal{P}$ in the first round). If $\ell = O(\log k)$ then $2^{-\ell}$ is inverse polynomial, and so by repeating this process some sufficiently-large polynomial number of times the simulator will succeed with all but negligible probability (as in the case of the original simulator). On the other hand, if $\ell = \omega(\log k)$ then $2^{-\ell}$ is negligible and so any simulator of this sort running in (expected)

[2] Although there are reasons to believe that constructing a simulator for this protocol will be "difficult" [4].

polynomial time will only succeed in outputting a valid view with negligible probability. But we need $\ell = \omega(\log k)$ to obtain negligible soundness error...

The good news is that ℓ-fold *sequential* repetition of the above protocol *does* result in a zero-knowledge protocol for any $\ell = \text{poly}(k)$ (note that we must have $\ell = \text{poly}(k)$ in order for the protocol to run in polynomial time).[3] The simulator for this protocol will essentially run the simulator given earlier, ℓ times sequentially. For completeness, we describe the simulator here but leave the analysis to the reader:

$\underline{\mathsf{Sim}(1^k, (G_1, G_2))}$
fix random coins ω for V^*
let $\mathsf{view}_0 = \omega$
for $i = 1$ to ℓ:
  for $j = 1$ to k:
    choose a random permutation φ'
    $b \leftarrow \{1, 2\}$
    $H = \varphi'(G_b)$
    run $\mathcal{V}^*(1^k, (G_1, G_2), \mathsf{view}_{j-1}, H)$ to obtain its response b'
    if $b = b'$, let $\mathsf{view}_j = \mathsf{view}_{j-1} || (H, b, \varphi')$ and exit loop
  if none of the above iterations have succeeded, output $\perp$
output view_ℓ

3 Honest-Verifier Zero Knowledge

A weaker definition of zero knowledge which is often useful is that of *honest-verifier zero knowledge* (HVZK). In this case, we only require simulation for a verifier who honestly follows the protocol, but may later try to learn some information (that it wasn't supposed to) from the transcript. This models so-called "honest-but-curious" adversaries who act in this way, and also models adversaries who eavesdrop on an honest execution of a protocol (between an honest verifier and the prover).

Definition 3 A pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ is called a zero-knowledge (ZK) proof system for a language $L \in \mathcal{NP}$ (with $L_k \stackrel{\text{def}}{=} L \cup \{0,1\}^{\leq k}$) if it satisfies the following properties:

1. **(Completeness and soundness)** As in Definition 1.

2. **Honest-verifier zero knowledge** There exists a polynomial time simulator Sim such that the following are computationally indistinguishable for any $x \in L_k$ and witness w for x:

   - the view of $\mathcal{V}(1^k, x)$ when interacting with $\mathcal{P}(1^k, x, w)$;
   - the output of $\mathsf{Sim}(1^k, x)$.

$\diamondsuit$

We do not allow for expected poly-time simulation since it is not needed to get efficient protocols (since the zero-knowledge property was weakened).

[3]In fact, *auxiliary-input* zero-knowledge (mentioned earlier but not formally defined in these notes) is always preserved under sequential composition; see [2, 3].

We now show that ℓ-fold parallel repetition of the previously-shown protocol for graph isomorphism is honest-verifier zero knowledge (for any polynomial ℓ):

$\underline{\mathsf{Sim}(1^k, (G_1, G_2))}$
choose random permutations $\varphi'_1, \ldots, \varphi'_\ell$
choose random $b_1, \ldots, b_\ell \leftarrow \{1, 2\}$
for all $i \in \{1, \ldots, \ell\}$: $H_i = \varphi'_i(G_{b_1})$
output $(H_1, \ldots, H_\ell; b_1, \ldots, b_\ell; \varphi'_1, \ldots, \varphi'_\ell)$

Note that the simulator's job here is much easier — because we are only concerned with simulating the view of an honest verifier, we may assume that the challenge bits $b_1, \ldots, b_\ell$ are chosen uniformly and independently of the graphs $H_1, \ldots, H_\ell$ sent in the first round.

References

[1] B. Barak and Y. Lindell. Strict Polynomial Time in Simulation and Extraction. STOC 2002. Full version available at http:
eprint.iacr.org.

[2] O. Goldreich. *Foundations of Cryptography, vol. 1: Basic Tools*. Cambridge University Press, 2001.

[3] O. Goldreich. Tutorial on Zero Knowledge. Available at http://www.wisdom.weizmann.ac.il/~oded/zk-tut02.html

[4] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM J. Computing* 25(1): 169–192 (1996).