CMSC 858K — Advanced Topics in Cryptography April 22, 2004

Lecture 23

Lecturer: Jonathan Katz

Scribe(s): Nicholas Sze, Ji Sun Shin, Kavitha Swaminathan

1 Introduction

We showed previously a zero-knowledge proof system for 3-colorability. Unfortunately, to achieve negligible soundness error while maintaining zero knowledge it was required to repeat the basic, 3-round protocol *sequentially* polynomially-many times (giving a protocol with polynomial round complexity). Here, we show a *constant-round* zero-knowledge proof system for $\mathcal{NP}$. We will also discuss the notion of *proofs of knowledge* and show a (non-constant-round) zero-knowledge proof of knowledge for languages in $\mathcal{NP}$.

1.1 A Brief Review

At a very high level, we review why our previous techniques did not suffice to give a constant-round zero-knowledge proof system (refer to previous lectures for more details). Recall the basic, 3-round protocol for 3-colorability: the prover sends commitments to the colorings of the vertices in the graph; the verifier sends a "challenge" (i, j) (where this is an edge in the graph); and the prover responds by "opening" the commitments to the colors for vertices i and j. The verifier accepts only if these colors are different.

In proving the zero-knowledge property of this basic protocol, we relied on the fact that a simulator could "guess" the verifier's challenge (i, j) in advance with noticeable (i.e., inverse polynomial) probability. So, having the simulator "rewind" polynomially-many times would be sufficient to allow the simulator to "guess correctly" at least once. But this very property also allows a cheating prover to guess the verifier's challenge in advance with noticeable probability, meaning that the soundness will not be negligible.

We can decrease the soundness error by repeating the basic protocol many times. But if we schedule to repetitions in parallel, the simulator has only negligible probability of guessing all the verifier's queries (simultaneously) in advance. On the other hand, if we schedule the repetitions sequentially then the simulator can guess each challenge (one-by-one) with noticeable probability.[1]

2 A Constant-Round Zero-Knowledge Proof for $\mathcal{NP}$

Goldreich and Kahan [3] suggested the first constant-round ZK proof system for $\mathcal{NP}$. The intuition behind their scheme is to force the verifier to commit to its challenges in advance. Then, once the simulator has learned the verifier's challenges, it can rewind and commit to a set of colorings that will allow it to answer the challenges correctly.

[1] Note that the simulator has more power than a cheating prover since it can rewind the verifier.

We now describe the protocol in detail:

Initialization The prover and verifier each have a graph $\mathcal{G}$. The prover also knows a 3-coloring of this graph. Let the common security parameter be k (this might be the number of vertices in $\mathcal{G}$, but it could also be independent of $\mathcal{G}$).

First stage The verifier chooses k edges $(i_1, j_1), \ldots, (i_k, j_k)$ uniformly at random from (the edge set of) $\mathcal{G}$. It then commits to these edges using a perfect commitment scheme, and sends these commitments to the prover. We saw in the last lecture that there are two-round protocols for perfect commitment based on some number-theoretic assumptions, so for convenience we will assume that the first stage is carried out in rounds 1 and 2.

Rounds 3–5 The prover and verifier now execute k *parallel* executions of the basic, 3-round protocol for graph 3-colorability. Sketching this in a bit more detail (but still assuming the reader is familiar with the basic protocol from a previous lecture):

- **Round 3** The prover commits to k different colorings of $\mathcal{G}$ using independent randomness for each of these k iterations, and where the colors in each iteration are committed to vertex-by-vertex (as usual). It is stressed that independent randomness is used in each of the k iterations, so in particular each of the k colorings is a random permutation of the coloring the prover started with.
- **Round 4** The verifier decommits the challenges that it committed to in the first stage. This results in a sequence of k edges (and the corresponding decommitments) that are sent to the prover.
- **Round 5** The prover first checks to make sure that the verifier opened his commitments correctly. If not, then the verifier is cheating so the prover aborts. Otherwise, the prover responds to the challenges as usual: in iteration ℓ, if the challenge is (i_ℓ, j_ℓ) then the prover reveals the colors of vertices i and j in the ℓ^{th} iteration.

Acceptance The verifier accepts only if all k iterations were successfully completed.

Note that the verifier uses a *perfect* commitment scheme to commit to its challenges in the first phase, while the prover uses a *standard* commitment scheme to commit to the colorings in round 3. The is necessary because a proof system requires soundness to hold against an *all-powerful* (cheating) prover. If the verifier used a standard commitment scheme, then an all-powerful prover would be able to figure out the verifier's challenges before round 3, and could then fool the verifier into accepting even if $\mathcal{G}$ were not a 3-colorable graph. Similarly, if the prover's commitments were not information-theoretically binding then it would be able to "change" its answers depending on the challenges of the verifier.

Given the above discussion, it is easy to see that the above scheme satisfies correctness and has negligible soundness error even for an all-powerful prover. The difficult part is to show that the protocol is zero knowledge. In fact, a full proof is quite involved and we will not give one here (see [3]). Instead, we will only give some of the intuition for the proof by considering the case of a verifier who *always opens the commitments correctly in round 4*. Also, we will be relatively informal here (since we are not giving a complete proof anyway)

but the interested reader will be able to derive a proof for this restricted case from the proof given earlier for the basic, 3-round protocol.

A simulator for the type of verifier considered here proceeds as follows:

1. For the first phase, the simulator runs the perfect commitment scheme normally and obtains a sequence of commitments from the verifier.

2. Simulating round 3, the simulator sends k "garbage" commitments to colorings of $\mathcal{G}$. Namely: for each of k iterations, commit to "red" for each vertex of the graph. (The simulator must commit to "garbage" because it does not know a coloring. But by indistinguishability of the commitments, a poly-timer verifier can't distinguish these "garbage" commitments from commitments that would be sent by a real prover.)

3. The verifier then decommits to the challenges that it committed to in the first stage. (Recall we assume that the verifier always decommits properly.) Denote these challenges by $(i_1, j_1), \ldots, (i_k, j_k)$.

4. Now, the simulator "rewinds" the verifier and sends k commitments to colorings of $\mathcal{G}$ *for which it can answer the challenges of the verifier*. That is: for the ℓ^{th} iteration, the simulator chooses random, distinct colors for vertices i_ℓ and j_ℓ, commits to these colors for i_ℓ and j_ℓ, and commits to "red" for all other vertices (in that iteration). Denote these commitments by $\mathsf{com}_1, \ldots, \mathsf{com}_k$ (each com_ℓ is composed of commitments for each vertex of $\mathcal{G}$).

5. The verifier again decommits to the challenges that it committed to in the first stage. Although we assume that the verifier always decommits properly, we do not necessarily assume that the decommitted values now are the same as they were before! However, they *do* in fact have to be the same *with all but negligible probability*; this follows from the (computational) binding of the commitment scheme used in the first phase.

6. Assuming the commitments were again opened in the same way, the simulator can easily decommit the relevant vertices correctly.

7. The final "view" output by the simulator includes the verifier's random coins, the messages sent to the verifier during the first stage, the second sequence of commitments $\mathsf{com}_1, \ldots, \mathsf{com}_k$, and the decommitments for the appropriate vertices.

Informally, the simulated transcript is indistinguishable from a real transcript because of the hiding property of the commitment scheme used by the prover in the 3^{rd} round. For a careful proof in the general case and much more discussion and details, see [3].

3 Proofs of Knowledge

Proofs of knowledge may be viewed as formalizing an even stronger notion of soundness. Very informally, a proof system may be viewed as demonstrating that a particular statement is true; a proof of knowledge may be viewed as demonstrating that the prover "knows" why the statement is true. Although it is fair to say that the notion of a proof of knowledge was introduced for (very important) technical reasons, there are some practical examples of

why proofs of knowledge are necessary. As an example of the latter, let G be a finite cyclic group of order q in which the discrete logarithm assumption is believed to hold, and let g be a generator of G. Consider the language $L_G = \{h \mid \exists x \in \mathbb{Z}_q \text{ s.t. } g^x = h\}$. On the one hand, L_G is just G itself since for every element $h \in G$ we know that $g^{\log_g h} = h$. So a proof that $h \in L_G$ is trivial (assuming that deciding membership in G is trivial). On the other hand, a proof *of knowledge* that $h \in L_G$ implies that the prover "knows" the value of $\log_g h$, something that is not implies by a proof alone. Similarly, if f is a one-way permutation and we define $L_f = \{y \mid \exists x \text{ s.t. } f(x) = y\}$, then a proof for L_f is trivial (since L_f contains all strings) but a proof of knowledge that $y \in L_f$ is not (as it implies that the prover "knows" $f^{-1}(y)$).

Of course, this leaves us with a vague sense of discomfort: what does it mean for a machine to "know" something? We define this in terms of the ability to *extract* the knowledge from the machine: i.e., a machine M "knows" something if there is a poly-time process by which we can extract this knowledge from M. We do not give a formal definition here (see [2] or [1] instead), but give the following informal definitions instead:

Definition 1 A *relation* R is a set of pairs of strings. A relation is said to be "polynomial-time" if: (1) there exists a polynomial $p(\cdot)$ such that $(x, y) \in R$ implies $|y| \leq p(|x|)$, and (2) given a pair (x, y), one can decide in polynomial time (in $|x|$) whether $(x, y) \in R$. $\diamondsuit$

Any relation R defines a language $L_R \stackrel{\text{def}}{=} \{x \mid \exists y \text{ s.t. } (x, y) \in R\}$. Furthermore, if R is polynomial time, then $L_R \in \mathcal{NP}$. Finally, any language $L \in \mathcal{NP}$ defines a relation $R_L \stackrel{\text{def}}{=} \{(x, y) \mid x \in L \wedge y \text{ is a witness for } x\}$.

Definition 2 Let R be a polynomial-time relation, and L_R be as above. A proof system $(\mathcal{P}, \mathcal{V})$ for L_R is a *proof of knowledge for L_R with soundness error $\varepsilon(k)$* if the following holds for all x and all cheating provers $\mathcal{P}^*$: Let $\mathsf{succ}_{\mathcal{P}^*}(k) = \Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle(1^k, x) = 1]$. If $\mathsf{succ}_{\mathcal{P}^*}(k) > \varepsilon(k)$ then with probability negligibly close to $\mathsf{succ}_{\mathcal{P}^*}(k)$ one can extract a value y from $\mathcal{P}^*$ in polynomial time[2] such that $(x, y) \in R$. A *proof of knowledge for L_R* is one having negligible soundness error. $\diamondsuit$

This definition will hopefully become more clear after we show an example in the following section.

Note that the zero-knowledge (ZK) requirement is orthogonal to the proof of knowledge (PoK) requirement. The former protects the prover from a malicious verifier, while the latter protects (in some sense) a verifier from a malicious prover. On the other hand, without any additional requirements it is trivial to construct a PoK for any polynomial-time relation: on common input x, the prover simply sends y such that $(x, y) \in R$. So, we will be interested in witness indistinguishable PoKs (WI-PoKs) or ZK-PoKs.

3.1 A Proof of Knowledge for Hamiltonian Cycle

Here, we show a basic 3-round ZK PoK (with non-negligible soundness error) for the language HAM of Hamiltonian cycles (this is the set of graphs containing a Hamiltonian

[2] Paralleling the case of zero-knowledge, extraction in expected polynomial-time are often allowed.

cycle — i.e., a cycle that includes each vertex of the graph exactly once). This language is $\mathcal{NP}$-complete, so this gives[3] a PoK for all of $\mathcal{NP}$.

The protocol proceeds as follows:

$\underline{\mathcal{P}(\mathcal{G}, \text{Ham-cycle})}$		$\underline{\mathcal{V}(\mathcal{G})}$
pick a random permutation Π of the graph $\mathcal{G}$; commit to the adjacency matrix of $\Pi(\mathcal{G})$	$\left(\begin{array}{ccc} \mathsf{com}_{11} & \cdots & \mathsf{com}_{1n} \\ \vdots & \ddots & \vdots \\ \mathsf{com}_{n1} & \cdots & \mathsf{com}_{nn} \end{array}\right)$ $\longrightarrow$	
	$\longleftarrow$ c	$c \leftarrow \{0,1\}$
if $c = 0$, open all commitments and send Π if $c = 1$, open commitments on the Hamiltonian cycle		
	open according to c value $\longrightarrow$	
		verify commitments if $c = 0$ check that the matrix is equal to $\Pi(\mathcal{G})$ if $c = 1$ check that a cycle was revealed

Claim 1 *This is a proof of knowledge with soundness error* 1/2.

Proof Note that for any cheating prover and any graph $\mathcal{G}$, this prover either succeeds with probability 0, probability 1/2, or probability 1. All we need to prove (cf. the definition of proofs of knowledge) is that if $\mathcal{P}^*$ succeeds with probability 1, then we can extract from $\mathcal{P}^*$ a Hamiltonian cycle in $\mathcal{G}$ with probability negligible close to 1 (in fact, we will extract with probability 1). Succeeding with probability 1 simply means that it answers both possible challenges correctly.

Given such a $\mathcal{G}$ and $\mathcal{P}^*$ who convinces the verifier with probability 1, we simply let the prover send its initial message; send challenge "0" and get the response; then rewind the prover and send challenge "1" and get the response. By assumption, both responses of $\mathcal{P}^*$ would cause the honest verifier to accept. So, from the $c = 0$ response, we have a graph $\mathcal{G}'$ (that $\mathcal{P}^*$ committed to in the first round) and a permutation Π such that $\Pi(\mathcal{G}) = \mathcal{G}'$. From the $c = 1$ response, we have a Hamiltonian cycle in $\mathcal{G}'$. It is now easy to recover a Hamiltonian cycle in the original graph $\mathcal{G}$. ■

[3]There are some additional subtleties here: we need it to be the case that for any $L \in \mathcal{NP}$ there exist poly-time computable functions f_1, f_2 such that: $x \in L \Leftrightarrow f_1(x) \in HAM$ and also $(f_1(x), y) \in R_{HAM} \Leftrightarrow (x, f_2(y)) \in R_L$.

The proof system is also zero knowledge. A simulator Sim can be constructed as follows:

	$\underline{\mathsf{Sim}(G)}$
Repeat k times:	
1	Guess $c' \leftarrow \{0,1\}$
$2a$	If $c' = 0$, commit to a random permutation Π of $\mathcal{G}$
$2b$	If $c' = 1$, commit to a random cycle graph
3	Send the commitments to the verifier, who responds with c
4	If $c = c'$, output a transcript including the correct response

We do not prove that this simulator "works", but leave this as an exercise for the reader.

As usual, by repeating the protocol multiple times we can decrease the soundness error. We know that by repeating the protocol sequentially we retain the zero-knowledge property (at the expense of high round complexity); what about the proof of knowledge property?

Claim 2 *Running the above protocol k times sequentially results in a proof of knowledge with soundness error $1/2^k$.*

Proof (Sketch) Assume a graph $\mathcal{G}$ and a prover $\mathcal{P}^*$ who convinces $\mathcal{V}$ with probability strictly greater than $1/2^k$. We can view the execution of $\mathcal{P}^*$ with $\mathcal{V}$ as a binary tree of height k, where the root corresponds to the beginning of the protocol and a node at level i (with the root at level 0) has two children corresponding to the two possible challenges that can be sent at round $i+1$. Call a leaf of the tree *accepting* only if the prover answers correctly to all challenges on the path from the root to this leaf. Since $\mathcal{P}^*$ convinces $\mathcal{V}$ with probability strictly greater than $1/2^k$, there are at least two accepting leaves. Intuitively, the paths from these two leaves to the root must have at least one node in common; at this node, $\mathcal{P}^*$ answers correctly for *both* possible challenges, and we can then extract as in the previous claim. We now show how to do this efficiently:

for $i = 1, \cdots, n$
 run $\mathcal{P}^*$ for round i
 by rewinding, send both $c = 0$ and $c = 1$
 if $\mathcal{P}^*$ answers correctly both times then extract a witness (as before)
 otherwise, increment i and continue along the path for which $\mathcal{P}^*$ answered correctly

(In the last step, there must be a value of the challenge for which $\mathcal{P}^*$ answers correctly since otherwise $\mathcal{P}^*$ convinces the verifier with probability 0.) Eventually, the above algorithm finds a node where two paths from the root to accepting nodes diverge (drawing a picture and following the execution of the above algorithm should convince you of this). □

If we want to construct a protocol with better round complexity, we can do so by running the basic, 3-round protocol in parallel. We know that this will not preserve the zero-knowledge property, but it will preserve the (weaker) property of witness indistinguishability. What about the proof of knowledge property?

Claim 3 *Running the basic protocol k times in parallel results in a proof of knowledge with soundness error $1/2^k$. (However, here extraction requires expected polynomial time.)*

Proof Note that we have a 3-round protocol where the prover begins by sending a vector of k commitments (to adjacency matrices); the verifier sends a k-bit challenge vector; and the prover then responds to each of the k 1-bit challenges individually, as in the basic protocol. If we can find two *different* vectors $\vec{c}, \vec{c}^*$ for which the prover responds correctly, then we can extract a witness as before: simply find an index i where $c_i \neq c_i^*$ (such an index must exist since $\vec{c} \neq \vec{c}^*$) and then extract using the i^{th} adjacency matrix sent by the prover in the first round and the i^{th} response given by the prover in the last round.

Assume, then, that we have a $\mathcal{G}$ and a prover $\mathcal{P}^*$. Consider the following algorithm to extract a witness:

$\mathcal{P}^*$ sends its vector of commitments
$\vec{c} \leftarrow \{0,1\}^k$
run $\mathcal{P}^*$ using challenge $\vec{c}$
if $\mathcal{P}^*$ fails to respond correctly, halt
otherwise:
for $i = 0$ to $2^k - 1$:
 $\vec{c}^* \leftarrow \{0,1\}^k$
 run $\mathcal{P}^*$ using challenge $\vec{c}^*$
 if $\mathcal{P}^*$ responds correctly and $\vec{c} \neq \vec{c}^*$, extract a witness and halt
 run $\mathcal{P}^*$ using challenge $\langle i \rangle$
 if $\mathcal{P}^*$ responds correctly and $\vec{c} \neq \langle i \rangle$, extract a witness and halt

(In the above, $\langle i \rangle$ represents a standard k-bit binary encoding of the number i.)

Let $\varepsilon(k)$ denote the probability that $\mathcal{P}^*$ answers correctly. We need to show two things: (1) if $\varepsilon(k) > 1/2^k$, then we extract a witness with (negligibly close to) the same probability; (2) the algorithm above runs in expected polynomial time *regardless* of ε.

If $\varepsilon > 1/2^k$ then there are at least 2 difference challenges for which $\mathcal{P}^*$ answers correctly. The algorithm above enters the loop with probability exactly ε; once it enters the loop, it is guaranteed to eventually find a second, different challenge for which $\mathcal{P}^*$ answers correctly. Since it extracts a witness in this case, we have that it extracts a witness overall with probability exactly ε. (Actually, we have ignored the negligible probability with which $\mathcal{P}^*$ might be able to break the binding property of the commitment scheme. But the basic argument remains the same or we can use a commitment scheme with perfect binding.)

We also need to also argue that extraction runs in expected polynomial time (in k); this seems worrisome since the inner loop potentially counts up to $2^k - 1$. Consider the following cases: if $\varepsilon = 0$ then clearly the algorithm runs in polynomial time (since it never enters the loop). If $\varepsilon = 2^{-k}$ then we enter the loop with probability ε and then run $2^k - 1$ iterations of the inner loop. So the expected number of loop iterations is:

$$2^{-k}(2^k - 1) + (1 - 2^{-k}) \cdot 0 < 1,$$

and the algorithm runs in *expected* polynomial time (we do not extract a witness in this case, but that is ok). Finally, if $\varepsilon > 2^{-k}$ then consider what happens if $\mathcal{P}^*$ answers correctly in the initial stage for a vector $\vec{c}$. In this case, the probability of choosing $\vec{c}^* \neq \vec{c}$ for which $\mathcal{P}^*$ answers correctly is exactly $\frac{2^k\varepsilon - 1}{2^k} \geq \varepsilon/2$. So the expected number of loop iterations until such a $\vec{c}^*$ is found is at most $2/\varepsilon$. Since the probability of entering the loop in the first

place is ε, the expected number of loop iterations overall is:

$$\varepsilon \cdot \frac{2}{\varepsilon} + (1 - \varepsilon) \cdot 0 < 2,$$

and the algorithm runs in expected polynomial time. ■

References

[1] M. Bellare and O. Goldreich. *On Defining Proofs of Knowledge*. Crypto '92.

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

[3] O. Goldreich and A. Kahan. *How to Construct Constant-Round Zero-Knowledge Proof Systems for NP*. Journal of Cryptology, 1996.