CS 810: Complexity Theory	4/13/2007

Lecture 27: Zero-Knowledge

Instructor: Dieter van Melkebeek   Scribe: Matthew Anderson

Last lecture we looked at one cryptographic primitive: the one-way function. Today we will see several applications of one-way functions in the construction of cryptographic protocols. All cryptographic primitives imply the existence of one-way functions, so in order to accomplish anything interesting we must assume their existence. We show how to construct a cryptographic pseudorandom generator using one-way functions, which in turn can be used to construct a protocol for confidential information transfer. We will also discuss several interactive protocols for authentication that allow a party to prove their identity but provide the verifier with no extra information, these types of protocols are known as zero-knowledge proofs.

1 Authentication Using One-Way Functions

One-way functions are functions that are computable in polynomial time in one direction but are hard to invert in the average case. The notion of hardness of some boolean function f, HI_f, was the size, s, such that all circuits of size less than s cannot compute f correctly on more than a fraction of $\frac{1}{2} + \frac{1}{s}$ of the inputs. A cryptographic pseudorandom generator (CPRG) is a standard PRG with the additional restriction that the sequence should be computable in polynomial time from the seed (rather than exponential time that is the case of standard PRGs). CPRGs maintain the property that the output sequence is not distinguishable from uniform by any polynomial time adversary (asymptotically). In our discussion we restrict ourselves to a subclass of one-way functions, namely one-way permutations. All of the results that we present here can be reformulated in the context of the more general one-way function; however, the analysis is more complex.

1.1 A Simple Flawed Authentication Protocol

Start with some one-way permutation f. Recall that for any authentication protocol to have an advantage the parties involved must have some additional knowledge that any other party (such as an eavesdropper) does not have access to. In this case both parties, A and B, know something different: A knows it's password, x, and B stores $f(x)$. In order to authenticate A's identity A will send B $f(x)$. B can easily verify that $f(x)$ matches the value stored for A. Since B stores the password as $f(x)$ instead of the plain text x and f is a one-way permutation it is difficult to invert and recover x from $f(x)$. This prevents even whoever is maintaining the list of values at B from inverting the function to determine x. Unfortunately this protocol is seriously flawed. Any eavesdropper, E, can watch A send $f(x)$ to B; then when E wants to pretend to be A, E simply repeats the message $f(x)$ to B which B has no choice but to accept.

1.2 A Better Authentication Protocol

We can apply the previous flawed protocol iteratively so that a password is used to authenticate only one time. This protocol will allow A to authenticate k times with B. A picks a password x, then computes $x \rightarrow f(x) \rightarrow f(f(x)) \rightarrow ... \rightarrow f^k(x)$. Where f^i is f composed i times with itself.

Ahead of time B stores $f^k(x)$. During its first login A provides B with $f^{k-1}(x)$. B verifies that $f(f^{k-1}(x))$ matches the value stored. B then stores $f^{k-1}(x)$. On the next login A provides $f^{k-2}(x)$, again B verifies $f(f^{k-2}(x))$ matches the value stored and updates the stored value. During the i^{th} login A sends $f^{k-i}(x)$ and B verifies $f(f^{k-i}(x))$ matches the value stored and updates the value. If x is chosen uniformly at random and f is a one-way permutation then it should be hard to compute $f^{k-i}(x)$ from $f^{k-i+1}(x)$. If the permutation is hard, then an adversary will not be able to invert a single step with more than $\frac{1}{\text{poly}}$ probability. This means the chance to invert an entire polynomial length sequence is very small if k is some polynomial in $|x| = n$.

One negative aspect of this protocol is that the number of logins, k, needs to be known *a priori*. Zero-knowledge proofs will be a way to get around this constraint. We discuss these later in this lecture.

2 Constructing CPRGs

We can leverage our construction in the previous section to build a CPRG. Consider looking at a prefix of the interaction sequence between A and B: $(f^k(x), f^{k-1}(x), ..., f^i(x))$. If one had a procedure that given a prefix could determine with high probability the next element of the sequence, $f^{i-1}(x)$, then one could determine the entire sequence with high probability. Last lecture we showed that PR sequences are unpredictable (otherwise one could construct a small circuit that could distinguish them from uniform). The protocol from the previous section almost gives us a CPRG with x as the seed. However, if a sequence is PR it should also be symmetric (this sequence is easy to compute in one direction). The proof of unpredictability of PR sequences depended on the fact that the sequence was a bit sequence, in this situation we do not have a bit sequence. For example, using the exponentiation one-way function we saw last time, the lowest order bit is efficiently predictable (using some properties of quadratic residues), though the highest order bit is as hard to compute as the function itself. In the squaring and RSA function all bits are difficult to compute. We can formalize this idea of hard to compute bits:

Definition 1 (Hardcore bit)**.** *A hardcore bit for f is a polynomial time boolean function h such that for any $c > 0$, all circuits C of size $\leq n^c$,*

$$\Pr_{|x|=n}[C(f(x)) = h(x)] \leq \frac{1}{2} + \frac{1}{n^c}. \tag{1}$$

Once we have a hardcore bit, h, for f we can use our protocol to construct a CPRG with the sequence $(h(f^k(x)), h(f^{k-1}(x)), ..., h(f(x)), h(x))$. The input is a seed of length n and the output is a bit sequence of length k, where k is polynomial in n.

Claim 1. *This sequence is a CPRG.*

Proof. Suppose that the sequence were predictable: that given a prefix $(h(f^k(x)), ..., h(f(x)))$ one could compute $h(x)$ using some small circuit C. Then there exists another (slightly larger) circuit C' that computes $h(x)$ from $f(x)$ with high probability. We can construct C' by building the prefix, which can be done efficiently in this direction, then applying C to the result. Therefore C' is a small circuit that computes $h(x)$ from $f(x)$ which contradicts the definition of h as a hardcore bit. $\square$

This is reasonable but do such hardcore bits h exist? We can show how to construct an appropriate h from any one-way permutation f (this also holds for one-way functions):

Lemma 1. *For all one-way permutations f, the function $g(x, y) = (f(x), y)$ with $|x| = |y|$ is a one-way permutation and $h(z = (x, y)) = \langle x, y\rangle$ is hardcore for g.*

$\langle x, y\rangle$ is the inner product between x and y modulo 2 ($\langle x, y\rangle = \sum_i x_i y_i (\mod 2)$). We can rewrite our CPRG output sequence for a uniformly chosen x and y:

$$\langle f^k(x), y\rangle, ...\langle f^2(x), y\rangle, \langle f(x), y\rangle, y. \tag{2}$$

We can add y to the end because it was selected uniformly at random.

Theorem 1. *The existence of a one-way permutation implies that for all $\epsilon > 0$ there exists a CPRG with seed length $l(n) = n^\epsilon$.*

Proof. Follows directly from Lemma 1 and Claim 1. □

Proof. (Lemma 1) As defined g is a one-way permutation. This follows because f is a one-way permutation and y is tacked on to the end.

We show that h is a hardcore bit by contradiction. Suppose there exists a circuit C, for some $\epsilon = \frac{1}{\text{poly}}$ such that:

$$\Pr_{z=(x,y)}[C(g(z)) = h(z)] \geq \frac{1}{2} + \epsilon. \tag{3}$$

Then by an averaging argument for a fraction at least ϵ of the x's,

$$\Pr_y[C(f(x), y) = \langle x, y\rangle] \geq \frac{1}{2} + \frac{\epsilon}{2}. \tag{4}$$

But $\langle x, y\rangle$ taken over all y is just the Hadamard encoding of x. This gives us a small circuit that predicts the Hadamard code. We can use our list decoding procedure, with high probability, to make a list of all possible x candidates. Since $\epsilon = \frac{1}{\text{poly}}$ the decoding algorithm runs in polynomial time. We then run f on all values in the list and pick one that works. For at least ϵ fraction of x (namely those for which (4) holds), this procedure works with high probability. This means we can invert f on a fraction at least $\frac{1}{\text{poly}}$ inputs with high probability. This contradicts the hardness of f. So h is hardcore. □

We can use this CPRG to send secret messages between two parties by first establishing a common secret seed then xoring the PR sequence with the message text as a one-time pad.

2.1 Bit Commitment

Another application of one-way permutations is bit commitment. Bit commitment protocols work in two phases:

1. Commit: A chooses a bit b and sends some encoding of b to B. B should not be able to determine the value of b at this stage.

2. Reveal: A sends additional information to B so that B can determine the bit b that A committed. A should not have been able to alter the value of the bit that B determines from what A committed in the first phase.

We can think of this as follows: A sends a locked box with a message inside, and later sends the key for B to open the box. We can use one-way permutations to develop a protocol for bit commitment which is successful when both parties are limited to being computationally efficient:

1. Commit: A picks a commit bit b, and $x \in_R U$. A sends B $(f(x), b \oplus h(x))$.
2. Reveal: A sends x to B. B computes $f(x)$ and verifies it matches the first value. B computes $h(x)$ and $(b \oplus h(x)) \oplus h(x) = b$.

Suppose B were able to efficiently compute b after the first phase, then B would also be able to compute $h(x)$ from $f(x)$, violating the hardness of h. A cannot change the bit that B computes in the second phase because only one choice of x maps to $f(x)$ since we are using one-way permutations. Thus this protocol has the appropriate properties.

We will see an application of this protocol in Section 3.1.

2.2 Pseudorandom Functions

A pseudorandom function generator (PRFG) is a function that generates a function $\{0,1\}^n \to \{0,1\}^n$ that appears to be a completely random function to a computationally efficient adversary. This is formalized in the following definition

Definition 2 (Pseudorandom Function Generator)**.** *A polynomial-time computable function $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is thought of taking one input to specify (i.e. output) a function, with $F(\rho, x) \to F_\rho(x)$ where ρ specifies the function and x is the actual argument. F is a PRFG iff for all $c > 0$, infinitely many n and all oracle circuits C of size at most n^c,*

$$\left| \Pr_{\rho \in_R U}[C^{F_\rho} = 1] - \Pr_{F \in_R U}[C^F = 1] \right| \leq \frac{1}{n^c}. \tag{5}$$

We can use a CPRG to construct such a PRFG.

Theorem 2. *If there exists a CPRG then there exists a PRFG.*

Proof. (Sketch) Given a polynomial time computable CPRG, $G : \{0,1\}^n \to \{0,1\}^{2n}$. We aim to construct a PRFG F. We will describe the construction but omit its proof, which follows in another hybrid argument as we have seen before.

Think of the argument x as describing a path (of length n) in a full binary tree. Consider G as (G_0, G_1) where G_0 produces the first half of the output of G and G_1 produces the second half. Then the output of $F_\rho(x) = G_{x_n} \circ G_{x_{n-1}} \circ \ldots \circ G_{x_1}(x)$ where x_i is the value of the i^{th} bit of x. This is the application of n choices of G_0 and G_1. □

Applying the same construction as above will give a regular PRG where the stretch is exponentially large for some seed ρ. The bits on the bottom level of the tree will specify the PR sequence. Each bit on the bottom level is computable in polynomial time by applying F_ρ to the appropriate x. This construction can be used to prove the following.

Theorem 3. *If there exists $f \in \mathrm{P}$ with $HI_f(n) \geq 2^{n^\epsilon}$ then there exists a PRG $G : \{0,1\}^n \to \{0,1\}^{2^{n^\delta}}$ for $\delta = \Omega(\epsilon)$ such that each bit of G is computable in polynomial time and for all circuits C of size at most 2^{n^δ},*

$$\left| \Pr_{|x|=n}[C(G(x)) = 1] - \Pr_{|y|=2^{n^\delta}}[C(y) = 1] \right| \leq \frac{1}{2^{n^\delta}}. \tag{6}$$

However, for this to work a stronger hardness assumption is essential because now an exponential sized adversary circuit is appropriate since the stretch is exponential.

This type of PRG is critical to an argument of the difficulty of showing that P $\neq$ NP. Recall, if one wants to show that P $\neq$ NP a non-relativising technique must be used. Remember that the argument for the lower bound on parity for constant depth circuit does not relativize. However, it can be shown that the argument can be viewed as a "natural proof". These proofs try to prove P $\neq$ NP by showing that NP $\not\subseteq$ P/poly. That is, we would like to demonstrate an NP language L that requires superpolynomial size circuits. A proof that L has superpolynomial size circuits is called *natural* if it shows the existence of a property Π with the following properties:

1. $\Pi(\chi_{L/n}) = 1$.
2. If $\Pi(\chi_{L'/n}) = 1$ then $C_{L'}(n) = n^{\omega(1)}$.
3. Π can be computed in time $2^{O(n)}$.
4. A non trivial fraction of characteristic sequences, χ, have $\Pr[\Pi(\chi) = 1] \geq \frac{1}{2^{n^\delta}}$ for some $\delta > 0$.

All known circuit lower bound results can be characterized by these types of proofs for certain creative choices of Π. We will not show how to frame the lower bounds that we have seen in this form.

Theorem 4. *If such CPRGs exist then there are no "natural" proofs to establish super polynomial circuit lower bounds.*

Proof idea: Let G be a PRG according to Theorem 3 and interpret $G(x)$ as a characteristic sequence of some function. Each bit of $G(x)$ is computable in polynomial time so it must be the case that $\Pi(G(x)) = 0$ for all x, meaning $\Pr_x[\Pi(G(x)) = 1] = 0$. However, the last property of natural proofs says that $\Pr_\chi[\Pi(\chi) = 1] \geq \frac{1}{2^{n^\delta}}$. As Π is computable in time $2^{O(n)}$ and therefore also by a circuit of similar size, it is a distinguishing circuit contradicting the pseudorandomness of G. $\square$

As it is conjectured that such CPRG do exist, the conjecture is that natural proofs do not. We can "rule out" natural proofs of circuit lower bounds for a number of specific problems as well. For example, the worst-case hardness of the discrete log is known to imply its average-case hardness, meaning it could then be used as the basis for a CPRG. Then applying Theorem 4 we get that there are no natural proofs for proving discrete log is exponentially hard.

3 Zero-Knowledge Proofs

Our motivation in discussing zero-knowledge(ZK) proofs is to improve the behavior of the second authentication algorithm we presented so that knowing the number of logins ahead of time is unnecessary. The main idea for ZK proofs for authentication is that no party will learn anything more than that you are actually who you say you are.

Suppose you have some secret witness for some NP problem. You want to reveal that you know a witness but do not want to disclose any information other than that you know a witness. This idea can be formalized in the structure of an interactive proof system:

Definition 3 (Zero-knowledge)**.** *An IPS (P, V) for a language L is zero-knowledge if $(\forall V'$ in randomized polynomial time$)(\exists$ a simulator $S_{V'}$ in randomized polynomial time$)$ such that $(\forall x \in L)$ the distribution representing the view of V' in the interaction of $(V' \leftrightarrow P)(x)$ and $S_{V'}(x)$ are "similar".*

Definition 4 (Similarity)**.** *Similarity is a measure of how close two distributions are. There are several possible ways to define similarity:*

1. *Identical - The distribution are equal. This definition gives us perfect zero-knowledge (PZK) proofs.*

2. *Statistically Close - This is a quantitative measure of how close the distributions are with respect to the 1-norm. This leads to statistical zero-knowledge (SZK) proofs.*

3. *Computationally Indistinguishable - A polynomial time circuit cannot distinguish between the two distributions with more than $\frac{1}{\text{poly}}$ probability. This gives us computational zero-knowledge(CZK) proofs.*

The idea behind ZK proofs is that the prover is the party trying to get the verifier to believe the prover's identity, without disclosing too much to the verifier.

One example of a ZK proof is an IPS for graph isomorphism. $L = \text{GI}$ and the input is (G_0, G_1). Since $\text{GI} \in \text{NP}$ there is a trivial IPS for GI which produces a permutation π, such that $\pi(G_0) = G_1$. However, this is clearly not ZK because the prover gives away the witness. One protocol which is ZK is as follows:

1. P: Pick $a \in_R \{0, 1\}, \pi \in_R S_n$. Send $H = \pi(G_a)$ to V.

2. V: Pick $b \in_R \{0, 1\}$. Send b to P.

3. P: Send to V a σ such that $H = \sigma(G_b)$.

4. V: Verify that $H = \sigma(G_b)$.

Completeness of this system is clear because if $G_0 \cong G_1$, the prover can aways prove it. If $G_0 \ncong G_1$, V catches the cheating prover with probability one half. Thus this is a IP for GI.

This IPS is also ZK since we can construct a simulator, $S_{V'}$, that produces a distribution which is indistinguishable from the actual distribution. $S_{V'}$ wishes to output (H, b, σ) with the same distribution as V' sees during the protocol with the actual prover.

- Pick a, π, H as in the original protocol.

- Run $V'(H)$ to determine b.

- If $(a = b)$ output (H, b, π) otherwise retry.

If $G_0 \cong G_1$, the probability that $a = b$ is exactly $\frac{1}{2}$, since the distribution of H is independent of a, so b must be chosen independently of a as well. This algorithm runs in expected polynomial time (exactly two runs are expected). PZK assumes expected polynomial time just like ZPP, since the simulator may fail many times to output anything. We can get a SZK proof system by stopping the simulator after a polynomial number of retries.

3.1 A ZK Protocol for 3-Coloring

If GI $\in$ P this protocol is useless since the verifier could solve the problem on its own. We would rather use an NP-complete problem than an NP-intermediate problem as the basis for the protocol since our belief of hardness is stronger for these complete problems.

In a similar fashion we can construct a ZK protocol using graph 3-coloring as the underlying problem. In this case the prover P knows some valid 3-coloring of the vertices, $\gamma : \textit{vertices} \rightarrow \{R, G, B\}$.

1. P: Pick a random permutation π for the colors $\{R, G, B\}$. For each vertex v, put in a lockable box the value $\pi(\gamma(v))$. Send all n boxes to V.
2. V: Pick an edge $e = (u, v) \in_R E$, send e to P.
3. P: If $e = (u, v) \in E$ send the keys for u and v to V, otherwise abort.
4. V: Accept if the colors in the two boxes are different.

This is an IPS because if P knows a 3-coloring, V will always verify that on any edge the endpoints are colored differently. If G is not 3-colorable, no matter what P does there will be at least one edge that is wrong giving V a $\frac{1}{\text{poly}}$ chance to catch a cheating prover.

This proof is ZK. The boxes that the verifier sees are indistinguishable without any keys. Thus the choice a V' makes for e is independent of what P put in the boxes. The distribution of the colors that V' sees after unlocking two boxes is uniform. So V' can simulate this interaction himself if G is 3-colorable. This scheme is PZK, however it is not clear that the lock boxes can be implemented computationally. We can translate these lock boxes into the computational realm by applying our bit commitment protocol. Using the bit commitment protocol gives a SZK protocol. This shows if one-way permutations exist there is a SZK protocol for all problems in NP (reduce each problem to 3-coloring).

4 Next Time

Next lecture we will discuss computational learning theory.