6.876/18.426: Advanced Cryptography March 5, 2003

Lecture 8: Communication Efficiency for NP Arguments

Scribed by: Chris Peikert

Today is Adam Smith's guest lecture.

The protocol we'll construct today will illustrate two important ideas that tend to recur frequently in cryptography:

1. The use of interaction to decrease the need for computation. (E.g., consider the result that IP = PSPACE: PSPACE may contain extremely hard problems, but by interacting with a prover, an efficient verifier can decide such problems.)

2. ZK for NP can be used as a general tool to make "any" protocol "private," with only a polynomial blowup in communication complexity. (The meanings of the words "any" and "private" will be formalized, in increasing generality, throughout this course.)

There are two main results we'll cover in this lecture:

1. Any language $L \in$ NP has an *argument* using only $O(k^2 \log n)$ communication (i.e., total bits sent). Here, k is the *security parameter*, so that soundness is negligible in k, while $n = |x|$ is the *length* of the theorem $x \in L$ to be proved.

2. The above argument can be transformed to be *zero-knowledge*, with total communication complexity $\text{poly}(k, \log n)$.

A few comments are in order: in the past, we have made no distinction between the security parameter k and the length n of the theorem (that is, we have set $k = n$ for simplicity). In this case, it is useful to make them independent parameters in order to optimize the communication complexity. Typically, we will set $k \ll n$. With a careful choice of intractability assumptions, we will be able to achieve negligible (in n) soundness with only polylogarithmic (in n) communication.

Note also that the protocol is merely an *argument*: that is, it is sound only against bounded cheating provers. In particular, we will assume that any cheating prover is implemented by a polynomial-sized circuit family.

1 An Argument for NP

There are two general tools (also useful in many other areas of cryptography and complexity) which we use in constructing our protocol: probabilistically checkable proofs (PCPs) and hash trees. We'll first discuss PCPs, which will motivate a (failed) attempt at a protocol. Then we'll see how hash trees can fix the problems we encounter.

1.1 PCPs

The literature on PCPs started in the 1980s, and the topic remains an active area of research today. They allow a verifier to be convinced of a statement by examining only a "few" locations of a proof.

More formally, consider the following scenario: suppose a 3CNF $\phi \in$ 3SAT is satisfiable. One way a verifier can be convinced of this fact is for it to receive a satisfying assignment w, against which the verifier checks the truth of every clause. Now suppose there is a 3CNF ϕ' for which there is a "gap": either (1) ϕ' is satisfiable, or (2) any assignment to ϕ' satisfies at most 99% of the clauses. To convince a verifier that ϕ' is satisfiable, one can write down an entire satisfying assignment w'. The verifier can then pick a clause at random and check if it is satisfied, looking at only the 3 bits of w corresponding to the literals appearing in the selected clause. If the verifier repeats this experiment, say, $100k$ times, then it achieves $\sim e^{-k}$ soundness while examining only $300k$ bits of w'.

In fact, there is a way to convert instances of 3SAT into instances of a "gap problem" like the one described above.

Theorem 1 *There exist efficiently-computable functions $f : \phi \mapsto \phi'$ and $g : (\phi, w) \mapsto (\phi', w')$ such that:*

1. $|f(\phi)| \leq |\phi|^2$.
2. *If ϕ is satisfiable, then $\phi' = f(\phi)$ is satisfiable; if ϕ is not satisfiable, then any assignment to $\phi' = f(\phi)$ satisfies at most 99% of its clauses.*
3. *If w satisfies ϕ, then $g(\phi, w) = (f(\phi), w')$, where w' satisfies $\phi' = f(\phi)$.*

1.2 A First Try

This suggests the following interactive protocol for 3SAT: on common input ϕ and private input w (which satisfies ϕ) to P, P computes $(\phi', w') = g(\phi, w)$ and V computes $\phi' = f(\phi)$. Then:

1. V chooses $100k$ clauses of ϕ' at random (with replacement) and sends their indices $i_1, \ldots, i_{100k}$ to P.
2. P sends the values of the 3 literals appearing in each of the clauses $i_1, \ldots, i_{100k}$, according to w'.

Finally, V checks that the chosen clauses are satisfied, and that the assignments to the literals are consistent (e.g., if x_3 appears in clause i_1 and $\overline{x_3}$ appears in clause i_7, then the claimed values of x_3 and $\overline{x_3}$ should be consistent). If so, V accepts, otherwise it rejects.

Note that the total communication is $O(k \log n)$, because specifying each clause index requires $O(\log n)$ bits, and specifying the variable values requires only 3 bits per clause.

However, there is a huge problem with this protocol: it is not at all sound! Because $k \ll n$, there may not be *any* variable which appears in more than one of the selected clauses. This means a cheating prover can simply claim that all the relevant literals are true (*after* seeing which queries the verifier makes), without making any contradictory claims.

In order to fix this, we need a way to *bind* the prover to a *single* witness w', *before* the prover sees which clauses the verifier has selected. We have two requirements of this binding procedure: (1) $\text{Bind}(w')$ must be *short*, and (2) there must be a short way to *prove consistency* of $\text{Bind}(w')$ with one bit of w' at a time. It turns out that hash trees are the answer.

1.3 Hash Trees

Recall the definition of collision-resistant hashing:

Definition 1 *A* collision-resistant hash family *is an efficiently-computable function* $H : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^k$ *such that, for all polynomial-sized circuit families* $\{A_i\}$,

$$\Pr[\alpha \leftarrow \{0,1\}^k;\ (x,y) \leftarrow A_k(\alpha);\ H(\alpha,x) = H(\alpha,y) \text{ and } x \neq y] \leq \nu(k)$$

where ν *is some negligible function.*

This suggests letting $\text{Bind}(w') = H(\alpha, w')$, and letting V pick the key α (since a cheating V^* gains nothing from picking a "weak key"). However, we don't know how to prove consistency of $\text{Bind}(w')$ with a revealed bit of w', aside from revealing all of w' (which costs too much communication).

Instead, $\text{Bind}(w')$ will be a "tree-hash" of w' (this idea is originally due to Merkle): first, break w' into k-bit blocks, and assume wlog that there are 2^j such blocks (where $j = O(\log n)$). Imagine a complete tree of depth j with these blocks at the leaves. Every node in the tree has a unique *label*, which corresponds to the branches taken on the path from the root to that node. (For example, the leftmost leaf has label $00\cdots0$ (j zeros), the root has label ϵ (the empty string), and the right child of the root has label 1.) In addition, every node has a *value* v_ℓ, where ℓ is the label of the node. We define v_ℓ recursively:

- the value at a leaf is the corresponding k-bit block of w';
- otherwise, $v_\ell = H(\alpha, v_{\ell 0} || v_{\ell 1})$. That is, the value at a node is the hash of the values at its two children (left child first).

We define $\text{Bind}(w') = v_\epsilon$, that is, the value at the root of the tree. To "unbind" (prove consistency of) a specific bit of w', do the following: compute the label ℓ of the leaf containing the desired bit (by simply dividing the index of the desired bit by k). Then for every proper prefix p of ℓ, reveal v_{p0} and v_{p1}. That is, reveal the values of all the nodes on the path from the root to the leaf, as well as all siblings of those nodes. To verify the unbinding, first check that the desired bit of v_ℓ has the claimed value. Then, for each proper prefix p of ℓ, verify that $v_p = H(\alpha, v_{p0} || v_{p1})$. This is possible because the values v_{p0} and v_{p1} are given as part of the unbinding, and so is v_p because either $p = \epsilon$ (in which case $v_p = \text{Bind}(w')$), or $p = p'0$ or $p = p'1$ for some proper prefix p' of ℓ.

The interesting thing about tree hashing is that being able to unbind a particular bit in two different ways implies the ability to find a hash collision. For if a circuit can find some $\text{Bind}(w') = v_\epsilon = v'_\epsilon$ relative to which it can unbind two different values v_ℓ and v'_ℓ, then for every proper prefix p of ℓ it produces values $v_{p0}, v_{p1}, v'_{p0}, v'_{p1}$ that pass the

verification test. Take the shortest p such that $v_p = v'_p$, but $v_{p0} \neq v'_{p0}$ or $v_{p1} \neq v'_{p1}$ (or both). There must be such a p, because $v_\epsilon = v'_\epsilon = \text{Bind}(w')$ but $v_\ell \neq v'_\ell$. Then $H(\alpha, v_{p0}||v_{p1}) = v_p = v'_p = H(\alpha, v'_{p0}||v'_{p1})$, so $v_{p0}||v_{p1}$ and $v'_{p0}||v'_{p1}$ are distinct strings which form a hash collision.

Finally, note that unbinding one bit reveals $O(\log n)$ values, each of which are k bits long.

1.4 The Real Protocol

We now have all the ingredients we need to build an argument for 3SAT: on common input ϕ and private input w (which satisfies ϕ) to P, P computes $(\phi', w') = g(\phi, w)$ and V computes $\phi' = f(\phi)$. Then:

1. V chooses $\alpha \leftarrow \{0,1\}^k$ at random, and sends it to P.
2. P computes $\text{Bind}(w') = v_\epsilon$ by tree hashing w' (with key α), and sends $\text{Bind}(w')$ to V.
3. V chooses $100k$ clauses of ϕ' at random (with replacement) and sends their indices $i_1, \ldots, i_{100k}$ to P.
4. P unbinds the values of the 3 literals appearing in each of the clauses $i_1, \ldots, i_{100k}$, according to w'.

Finally, V checks that (1) the chosen clauses are satisfied, (2) that the assignments to the literals are consistent, and (3) that the unbindings were valid. If so, V accepts, otherwise it rejects.

Completeness is straightforward, from our prior discussion about gap problems. Soundness arises from the fact that after the second round, there is only *one* value of each variable that a cheating (polynomially-bounded) P^* could successfully unbind. If ϕ were unsatisfiable, then at most 99% of the clauses in ϕ' would be satisfied by any particular assignment. With high probability, V would choose one of the unsatisfied clauses and reject.

The communication complexity is $2k$ bits for the first two rounds, $O(k \log n)$ for the third round, and $O(k^2 \log n)$ bits for the fourth round, for a total of $O(k^2 \log n)$.

2 Making it all Zero-Knowledge

Note that the above protocol is probably not zero-knowledge (hashing the witness w' may leak some information about it; in addition, many of the bits of w' are explicitly leaked). However, we note that it *is* a public-coin protocol. Therefore we can employ the same technique that we used to prove that IP $\subset$ ZK: conduct an "encrypted" conversation, and then prove (in zero knowledge) the NP statement that, were the conversation decrypted, the original verifier would have accepted.

Let $(\mathsf{P}_a, \mathsf{V}_a)$ be the prover/verifier from the argument above. Then, assuming the existence of a public-key cryptosystem (G, E, D) (which follows from the existence of one-way functions), we get the following zero-knowledge protocol: on common input ϕ and private input w which satisfies ϕ to P, do the following:

1. V sends the first message of $\mathsf{V}_a(\phi)$; that is, a random $\alpha \leftarrow \{0,1\}^k$.

2. P generates $(pk, sk) \leftarrow G(1^k)$, and gets the first message R of $\mathsf{P}_a(\phi, w)$ (the root of the hash tree). P sends $pk, R_e = E_{pk}(R)$ to V.

3. V sends the second message c of $\mathsf{V}_a(\phi)$; that is, the random choices of clauses.

4. P gets the second message P of $\mathsf{P}_a(\phi, w)$ (the unbindings of the variables), and sends $P_e = E_{pk}(P)$ to V.

5. P and V enter into a zero-knowledge proof of knowledge: P proves knowledge of an sk such that V_a would accept the transcript $(\alpha, D_{sk}(R_e), c, D_{sk}(P_e))$.

Completeness is again simple. Soundness follows from the fact that encryption is binding, and by the proof of knowledge of sk. Zero-knowledgeness follows from a simulator S which encrypts random strings instead of the legitimate prover messages (which we don't know how to compute). Then in the last stage, S invokes the simulator for the ZKPOK to complete the conversation. Note one subtlety here: in the last stage, the theorem is actually *false*; that is, there *does not exist* an sk such that V_a would accept. However, this false theorem *is* indistinguishable from the true theorem that is proven by P (this follows from the security of the encryption). Therefore, the output of the simulator on this false theorem is indistinguishable from the output of the simulator on the true theorem, which is indistinguishable from the conversation with P. By this hybrid argument, S produces a good transcript.

Now we analyze the communication complexity: because the theorem and witness in the final stage have length $\text{poly}(k, \log n)$, the entire protocol has complexity $\text{poly}(k, \log n)$.