Theoretical Cryptography, Lecture 4

Instructor: Steven Rudich
Scribe: Ryan Williams

January 30, 2006

1 Introduction

The official title of this lecture is

What is a Function of What?[1]

Preliminary meta-discourse: *Steven begins the lecture by saying, "Manuel verbed verbs, but nay, nary a noun." Ryan has no idea what this means, but believes in his heart that Steven is saying "Manuel was too informal."*

This lecture will formally define most of the notions that Manuel has discussed so far. Namely, we'll define conversations, interactive proof systems, and what it means for such a system to be zero-knowledge.

Here is an outline of how our train of thought will proceed. We will start with defining a "thought" of a function on an input, or a conversation a function has with itself. This notion will be extended naturally to a definition of a conversation between two functions on a common input. In turn, we'll extend this definition into one where two functions, each with their own private information, have a conversation. (This is where it will be important to understand "what is a function of what".) Then, allowing the private information of functions to be chosen at random, our discussion will move to defining a type of proof system, which we call an "interactive proof system". Finally, we'll define the class IP of languages that admit interactive proof systems, and define what it means for such a proof system to be zero-knowledge.

2 Thoughts

Typically, a conversation is understood take place between two parties. But one can also think of a conversation on just one party– a thought. Let's start in this very simple setting, and define what it means for a function to have a thought. This setting allows us to establish the notions of

[1] Or, "What is an Interactive Proof System?" – r.w.

"history" and "conversation" in an elementary way. (These notions will be greatly extended as we progress.)

Fix an alphabet Σ. Take any function f from Σ^* to Σ^*, and an $x \in \Sigma^*$. We'll begin with a definition of "f's thoughts on x", or a conversation that f has "with itself" concerning x.

Definition 2.1 *Let i be a natural number. The* **thought of f on x at time** i*, denoted as T_i, is defined inductively as*

$$T_0 \overset{def}{=} x, \quad T_i \overset{def}{=} f(T_0 \cdots T_{i-1}).$$

Thus $T_1 = f(T_0)$, $T_2 = f(T_0 f(T_0))$, and so forth. For a sequence of thoughts, we also can define a *history* of the function f's thoughts on x.

Definition 2.2 *Let i be a natural number. The* **history of thoughts of f on x up to time** i*, denoted as H_i, is defined inductively as*

$$H_0 \overset{def}{=} T_0, \quad H_i \overset{def}{=} H_{i-1} f(H_{i-1}).$$

Verify for yourself that the following proposition holds.

Proposition 1 *For all i, $H_i = T_0 \cdot \cdots \cdot T_i$.*

Note that, if f is a function that can be evaluated efficiently (*e.g.* polynomial time computable), then one can efficiently compute T_i and H_i, given the previous thoughts and history.

3 Conversations

Having given a recursive way to talk about the "thoughts" of f, wouldn't it be nice if f had another function g, with which to share its thoughts?...

...Glad you agree. Let's now extend the above to define what is a conversation between functions f and g on strings.

3.1 Definitions

Definition 3.1 *Let i be a natural number. The* **conversation between f and g on x at time** i*, denoted as C_i, is defined inductively as*

$$C_0 \overset{def}{=} x, \quad C_i \overset{def}{=} \begin{cases} f(H_{i-1}) & i \text{ odd} \\ g(H_{i-1}) & i \text{ even.} \end{cases}$$

The history of the conversation is defined analogously:

Definition 3.2 *Let i be a natural number. The* **history of f and g on x at time** i*, denoted as H_i, is defined inductively as*

$$H_0 \overset{def}{=} x, \quad H_i \overset{def}{=} H_{i-1} C_i.$$

Notice that in our definition of conversation, we let f speak first (we're nice people, after all). That is to say: at step 1, f evaluates x. We could just as easily define it so that g speaks first, *i.e.* g evaluates the history on odd inputs, and f evaluates the history on even inputs.

3.2 Accepted and Rejected Conversations

Suppose we want to define what it means for f and g to "come to a conclusion" in a conversation. The intuition here is that a conclusion arises when both parties "agree" on the same bit.

Definition 3.3 *Let f and g be functions on Σ^*, where Σ has distinguished symbols 0 and 1. The* **conversation of f and g accepts** $x \in \Sigma^*$ *iff there exists an integer i such that $C_i = C_{i+1} = 1$, and for all $j < i$, either $C_j \neq 0$ or $C_{j+1} \neq 0$.*

The definition of a *conversation rejecting* x is analogous; more precisely, it is obtained by swapping the roles of 0 and 1 in the above.

Of course, note there are functions f and g and strings x where neither acceptance nor rejection will take place. But by definition, it is *not* the case that both acceptance and rejection can take place on an input x.

4 Conversations with Privacy

In our earlier intuitive discussions about zero-knowledge, we have relied heavily on the fact that there is some information known to one party that is *not* known to the other party. (If this is not the case, then zero-knowledge is certainly impossible!)

To accommodate this, we extend the previous definition of a conversation between f and g, so that it has *two* more inputs than just x. One of these inputs is information that is private to f, and the other is information that is private to g. Here we must be careful about *what is a function of what.*

Definition 4.1 *Let i be a natural number. The* **conversation between f and g with private bits r_f and r_g on x at time** i *is defined inductively as*

$$C_0^p \overset{def}{=} x, \quad C_i^p \overset{def}{=} \begin{cases} f(H_{i-1}, r_f) & i \text{ odd} \\ g(H_{i-1}, r_g) & i \text{ even.} \end{cases}$$

The definition of history with private bits is the same as the above definition of history *without* private bits, except that C_i is replaced with C_i^p.

5 Random Private Bits and Persuasive Provers

The private bits r_f and r_g will generally be chosen at random. To this end, we define the uniform distribution

$$U_n \overset{\text{def}}{=} \text{uniform distribution on } n\text{-bit strings.}$$

The notation

$$r \in_R \mathcal{D}$$

denotes that the string r is randomly chosen according to the distribution $\mathcal{D}$.

Fix $\ell_1, \ell_2 : \mathbb{N} \to \mathbb{N}$ to be two easy-to-compute functions; almost always, ℓ_1 and ℓ_2 are polynomials.

We now define the *random variable* $(f \leftrightarrow g)(x)$ to be the probability distribution over the random conversations of f and g with private r_f, r_g on x, where $r_f \in_R U_{\ell_1(|x|)}, r_g \in_R U_{\ell_2(|x|)}$.

Hence, an expression like

$$\Pr_{r_f \in_R U_{\ell_1(|x|)},\ r_g \in_R U_{\ell_2(|x|)}} [(f \leftrightarrow g)(x) \text{ accepts}]$$

is well-defined. In the following, we will drop the explicit reference to r_f and r_g in probability expressions.

Let's now think of the two functions f and g in a specific sense, as a prover function P and a verifier function V, respectively. In our informal setting of the previous lectures, the prover and verifier had conversations where the prover tried to convince the verifier of something.

Given a verifier function V, suppose we would like to convince V that input x should be accepted. We want to pick the "most persuasive" prover P possible. What does this mean? We want to maximize the probability that V accepts x. That is, we want to pick a P^* such that

$$\Pr[(P^* \leftrightarrow V)(x) \text{ accepts}]$$

is maximized, over all choices of P^*.

6 Interactive Proof Systems

We are finally in position to define an interactive proof system. We examine two possible definitions, which differ in how the success and failure probabilities are defined.

6.1 One-Sided Error Case

The first definition is that which we have been informally using so far.

Definition 6.1 *A* **(one-sided) interactive proof system** *for a language L is a pair (P, V), where $P : \Sigma^* \times \Sigma^* \to \Sigma^*$ is arbitrary, and $V : \Sigma^* \times \Sigma^* \to \Sigma^*$ is implementable by a Turing machine running in $O(n^k)$ time, for some $k > 1$. The pair has two properties:*

1. **(Completeness)** $x \in L \Longrightarrow \Pr[(P \leftrightarrow V)(x) \textit{accepts}] \geq 1$.
   This means that, with prover P, the verifier will always accept x.
2. **(Soundness)** $x \notin L \Longrightarrow$ $(\forall$ *functions* $\tilde{P} : \Sigma^* \times \Sigma^* \to \Sigma^*)$ $\Pr[(P \leftrightarrow V)(x) \textit{accepts}] \leq 1/2$.
   This means that, if x is not in the language, then no prover at all can fool the verifier into accepting, with probability at most $1/2$.

The above interactive proof system has *one-sided* error, because a non-zero probability of error only occurs in one of the two cases.

For an example of an interactive proof system under this definition, consider the protocol (*i.e.* proof system) for Hamilton Cycle from the first lecture. In that protocol, if there is a Hamilton cycle, then the prover can always convince the verifier of this. (Thus, the completeness criterion is satisfied.) On the other hand, if there is no such cycle, the verifier will catch a crooked prover with probability at least 1/2. (Thus, the soundness criterion is satisfied.)

6.2 Two-Sided Error Case

Sometimes, we will also look at the case where there is a non-zero probability of error on *both sides*, *i.e.* error in both the completeness and soundness cases.

Definition 6.2 *A* **(two-sided) interactive proof system** *for a language L is a pair (P, V), $P : \Sigma^* \times \Sigma^* \to \Sigma^*$ is arbitrary, and $V : \Sigma^* \times \Sigma^* \to \Sigma^*$ is implementable by a Turing machine running in $O(|x|^k)$ time (where x is the input), for some $k > 1$. The pair has two properties:*

1. **(Completeness)** $x \in L \Longrightarrow \Pr[(P \leftrightarrow V)(x)\,accepts] \geq 2/3$.
   This means that, with prover P, the verifier will accept x with decent probability.
2. **(Soundness)** $x \notin L \Longrightarrow$ ($\forall$ *functions* $\tilde{P} : \Sigma^* \times \Sigma^* \to \Sigma^*$) $\Pr[(P \leftrightarrow V)(x)\,accepts] \leq 1/3$.
   This means that, if x is not in the language, then no prover at all can fool the verifier into accepting, with good probability.

The above definition is the one we shall use for the remainder of this lecture. The introduction of the fractions $2/3$ and $1/3$ may look arbitrary: why not choose $3/4$ and $1/4$, or something else for that matter? You will investigate this possibility on a homework question. It turns out that the above definition is equivalent to having a completeness "success" probability of at least $1 - 1/2^{p(|x|)}$ and a soundness "failure" probability of at most $1/2^{p(|x|)}$, for any polynomial $p(n) \geq n$.

6.3 The class IP

Given the above definition, we can immediately make some simple observations concerning properties of interactive proof systems, just from the fact that V must be implemented in $O(|x|^k)$ time. Since this is the case, it must be that for all rounds i, the history H_i and the private random bits r_v must have at most polynomial length in $|x|$. (Recall V is a function that takes these two strings as input.) Otherwise, V could not even read its input within polynomial time. Moreover, the total number of messages transmitted is at most polynomial.

Now armed with a new computational model for recognizing languages, it is natural to define a complexity class for it.

Definition 6.3 IP *is the class of languages for which there exists a (two-sided) interactive proof system.*

Note that IP does not emphasize or immediately address how the prover does what it does; it is only required that the prover *exists*. For example, consider the famous example of a problem in IP:

$$\text{Graph Non-Isomorphism} \overset{\text{def}}{=} \{(G, G')|\ G, G' \text{ are non-isomorphic graphs.}$$

It is a major open question as to whether or not Graph Non-Isomorphism is solvable in polynomial time. (In fact, we do not even know if it is in NP.) However, we *can* show that the problem has an interactive proof system. So while we do not know the complexity of a prover who can solve graph non-isomorphism, we do know the complexity of *convincing a verifier* that two graphs are non-isomorphic. Here is the specification of $V(G, G', r_v)$:

$V(G, G', r_v)$:

```
        Use $r_v$ to pick a random permutation $\pi$ of the node labels.
        Use $r_v$ to pick an additional random bit $b$.
        If $b = 0$, send $(\pi(G), \pi(G'))$ to prover $P$. If $P$ returns $G$ then accept, else reject.
        If $b = 1$, send $(\pi(G'), \pi(G))$ to prover $P$. If $P$ returns $G'$ then accept, else reject.
```

What is going on in the above? V is randomly permuting the vertices of G and G', and randomly swapping their positions in the pair. Then, V challenges P to tell her *the first graph* in the pair that she sent.

If the graphs are isomorphic, then any prover has at most a $1/2$ chance of answering correctly; *e.g.* the graph $\pi(G)$ could have come from either G or G'. However, if G and G' are not isomorphic, then a prover that can solve Graph Non-Isomorphism can always answer correctly. Therefore there is a one-sided interactive proof system for this problem.

Of course, in all of our discussion, the notion of zero-knowledge has not yet come up. This will be (finally) defined in the next section.

7 Defining Zero-Knowledge

We can generalize the notion of interactive proof system to allow *private auxiliary inputs* to P and V as well, beyond their private random bits. We will let y denote P's private input and z denote V's private input in the following. These inputs represent particular knowledge that one party has, but the other does not, *e.g.*, y could be a proof that the graph denoted by x has a Hamilton cycle.

The intuitive idea of zero-knowledge, as was stressed in previous lectures, is that *everything the verifier learns from the prover, she could have created for herself.* To formalize this, we will require that the probability distribution that arises from the verifier's interactions with the prover is *identical* to some distribution that a polynomial time algorithm could generate.

Definition 7.1 *An interactive proof system (P, V) for a language L with private inputs y, z is* **perfect zero-knowledge** *iff*

$\forall$ polytime verifiers $V^*(x, z, r_v)$, where r_v is a random string of length polynomial in $|x|$,

$\exists$ polytime $M^*(x, z, r_v)$ such that

$$(P \leftrightarrow V^*)(x, y, z) \equiv M^*(x, z, r_v).$$

That is, the distribution of conversations between P and V^* is identical *to the distribution of outputs of $M^*(x, z, r_v)$, as we vary over all possible r_v.*

Notice that zero-knowledge is a requirement not on provers, but on the power of *all possible verifiers* that interact with the prover. It says that no matter what the verifier is, if it is polytime, then all of its interactions with the prover can be simulated by another polytime machine that works without the prover. For this reason, M^* is called a *simulator.*

We remark that the adjective "perfect" in the definition of perfect zero-knowledge comes from the requirement that the two distributions are identical.