Lecture Notes 16

38 Partial Disclosure of Information

Suppose Alice has encrypted all of her files and placed them on a public file server. She reveals one file to Bob by decrypting, perhaps because Alice runs a song library and Bob has paid her for that particular song. Bob doesn't trust Alice and wants to know for sure that the song Alice just sent him is really the same one that is stored encrypted on the file server. How does Alice convince Bob that she sent him the right song?

One solution would be for Alice to send Bob the decryption key. Then Bob could decrypt the encrypted file himself. However, assuming all of the songs are encrypted using the same key, this would result in Bob learning the decryptions of all songs in the library. The question is whether there is some way for Alice to convince Bob that she gave him the correct song without giving him any other "knowledge" other than this one fact. A way of doing this, when it is possible, is called a *zero-knowledge proof*.

Here is another example. f is a one-way permutation, and b is a hard-core predicate for f. Alice has a secret x and makes $y = f(x)$ public. She reveals the value of $b(x)$ (a single bit 0 or 1). How can she convince Bob that the bit c she sent him is really the value of $b(x)$ without also revealing additional information about x? Again, a zero-knowledge proof is desired.

Both of these problems are examples of situations in which Alice has secret information x which determines some value $v = g(x)$ that she wants to give to Bob. She wants to convince Bob that v is correct without revealing any more information about x than is implied by the fact $v = g(x)$.

39 The Classical Concept of Proof

To begin the discussion of zero knowledge proofs, we look first at the question of what it means to "convince" Bob of the truth of a statement such as $v = g(x)$.

In classical mathematics, a convincing argument is called a *proof*. It is generally formalized as a sequence of logical statements such that each statement in the sequence is a self-evident truth called an *axiom*, or it follows from one or more previous statements using a *valid rule of inference*. All statements in the proof are considered to be valid, and the last statement is called the *theorem* that the proof establishes.

One generally assumes that there are feasible algorithms for testing if a given statement is an axiom and for testing if a statement follows from one or more previous statements using a valid rule of inference. Given such algorithms, one can then verify the validity of the statements in the proof one at a time in sequence, starting from the top. If all statements check out, then the last statement, which is the theorem to be proved, is also valid. Obviously, a proof system is only useful if the given algorithms are feasible.

Let T be the set of valid statements (theorems) that one wishes to prove. A *proof system for* T is a proof system that satisfies the following conditions:

Soundness: If y is a valid proof of a statement x, then $x \in T$.

Completeness: If $x \in T$, then there exists a valid proof y of x.

That is, anything provable is a valid theorem, and every valid theorem has a proof.

The above discussion focuses completely on the problem faced by the *verifier* of a proof – how does one check that a proof is correct? But the proof comes from someone. We call the agent who produces the proof the *prover*. We can then view this classical paradigm as a two-party protocol between P (the prover) and V (the verifier). Given a statement x to be proven, P constructs a proof y and sends it to V. V verifies that y is a valid proof and checks that the last line of y is equal to x, the theorem to be proved. If so V *accepts* the proof y as a valid proof of x; otherwise V rejects y. Assuming a sound and complete proof system for T, it follows that V accepts only statements $x \in T$, and for every $x \in T$, there is a prover that will cause V to accept x.

With classical proofs, we do not discuss the problem of how the prover comes up with a proof of x. In particular, we do not assume that P is a polynomial time algorithm (or even that it is an algorithm at all). All that matters is whether a proof of x exists or not.

We remark that proofs are only interesting for statements that the verifier can't (efficiently) compute directly. For example, the veracity of the statement $1 + 2 = 3$ is easily checked by just evaluating the left and right hand sides and checking for equality, so supplying a proof of that fact does not materially ease the verifier's job.

On the other hand, let L be an $\mathcal{NP}$ language. Recall from section 3.5 that $L \in \mathcal{NP}$ iff there is a polynomial-time computable relation $R_L(x, y)$ and a polynomial $q(n)$ such that

$$L = \{x \in \{0,1\}^* \mid \exists y \in \{0,1\}^*(|y| \leq q(|x|) \wedge R_L(x, y))\}.$$

A string y is a *witness to x's membership in L* iff $|y| \leq q(|x|)$ and $R_L(x, y)$ is true.

Although there is no known feasible algorithm for deciding statements of the form $x \in L$, with the help of a witness y, it is easy to verify that $x \in L$. Namely, just check $(x, y) \in R_L$. Thus, y can be considered to be a "proof" of x, not in the classical sense described above, but in the sense of their being a feasible verification algorithm (R_L) for determining whether or not a string y is a valid proof of the statement $x \in L$.

40 Interactive Proofs

Interactive proofs generalize the classical framework in several respects:

- Classical proofs are one-way protocols: A prover P sends a value y to a verifier V, after which V either accepts or rejects the proof. Interactive proofs permit multiple rounds of back-and-forth communication between P and V.
- Interactive proofs relax the soundness condition and permit V to accept a purported "proof" of a false theorem with small probability.
- Interactive proofs relax the completeness condition and permit P to fail to produce a valid proof of x with small probability.

These vague requirements will be made more precise below.

40.1 Interactive Turing machines

We formalize the notion of two parties P and V interacting according to a protocol. Intuitively, P and V are concurrent algorithms that can send and receive messages. However, in order to simplify

the reasoning, we assume that P and V are not truly concurrent but that they instead take turns: When it is P's turn, P runs until it is done with this phase and V waits. When P is done, the message it has constructed is transmitted to V, P is suspended, and V is activated. Now V runs until it is done with this phase, at which point it is suspended and P is reactivated with the message from V. This back-and-forth computation continues until one of the machines halts (rather than suspending), in which case the protocol stops. Note that it is possible for one of the machines to run forever during its phase, in which case the other machine remains suspended forever and never makes further progress.

We formalize the parties P and V as particular kinds of Turing machines. An *interactive Turing machine* M is a Turing machine with several tapes:

- A read-only *input tape*.
- A read-only *random tape*.
- A read/write *work tape*.
- A write-only *output tape*.
- A write-only *outgoing communication tape*.
- A read-only *incoming communication tape*.
- A read/write *switch tape* consisting of only a single cell.

Let A and B be interactive Turing machines. A *joint computation* of A and B consists of placing a common input x on the input tapes of both A and B, placing an infinite sequence of independent uniformly distributed random bits on A's and B's random tapes, and placing 0 on the switch tape. (The switch tape determines which machine is currently activated – 0 for A and 1 for B.) Control switches from A to B when A writes 1 on the switch tape. At that time, the system copies the contents of A's outgoing communication tape to B's incoming communication tape, suspends A, and resumes B. Similarly, when B changes the switch tape from 1 to 0, the system suspends B, copies its outgoing communication tape to A's incoming communication tape, and resumes A. This process continues until one of the machines halts, at which time the contents of the two output tapes is considered to be the result of the joint computation.[1]

For interactive proofs, we are generally only interested in the output of the verifier. We write $\langle A, B\rangle(x)$ to denote the output of B in the joint computation of A and B with common input x.

40.2 Time complexity

Interactive Turing machine A has time complexity $t : \mathbb{N} \to \mathbb{N}$ if for all interactive Turing machines B, for all inputs x, and for all random tapes, A never takes more than $t(|x|)$ steps.[2] A is *polynomial-time* if there exists a polynomial p such that A has time complexity p.

[1] This description of the joint computation of A and B differs slightly from that presented in the textbook. In this description, we assume an environment that copies data from one machine to the other at the point of the control switch. In the textbook, it is assumed that the tapes are merged, i.e., that there is a single input tape that is shared by the two machines, and two shared communication tapes, one for messages from A to B and one for messages from B to A. Even with these shared tapes, each machine has its own read/write head for viewing the contents of the tape.

[2] The textbook has the stronger requirement that A always halts within $t(|x|)$ steps no matter what B does. This stronger requirement seems to be difficult to satisfy if B goes into an infinite loop before A has halted and never activates A again. However, Goldreich's remarks following Definition 4.2.3 seems to suggest that he intended a definition equivalent to ours.

40.3 Interactive proof system

Definition: An *interactive proof system* for a language L is a pair of interactive Turing machines (P, V) such that V is polynomial-time and the following conditions are satisfied:

Completeness: For all $x \in L$, $\Pr[\langle P, V\rangle(x) = 1] \geq \frac{2}{3}$.

Soundness: For all $x \notin L$ and all interactive Turing machines B, $\Pr[\langle B, V\rangle(x) = 1] \leq \frac{1}{3}$.

The class $\mathcal{IP}$ consists of all languages having interactive proof systems

Note that completeness depends on both P and V, whereas soundness is a property just of V.

Clearly, $\mathcal{NP} \subseteq \mathcal{IP}$ since an interactive proof system can be easily constructed from the polynomial-time relation R_L described in section 39. Namely, if $x \in L$, P sends V a witness y to $x \in L$. V in turn checks that $|y| \leq q(|x|)$ and $(x, y) \in R_L$.[3] Note in particular that P is not required to be polynomial-time; only V is.

Also, $\mathcal{BPP} \subseteq \mathcal{IP}$. In this case, the prover need do nothing; the polynomial-time probabilistic Turing machine M that recognizes $L \in \mathcal{BPP}$ can be used directly as the verifier, where output 1 means "accept" and 0 means "reject". Hence, $\mathcal{NP} \cup \mathcal{BPP} \subseteq \mathcal{IP}$. Recall that it is not known whether $\mathcal{BPP} \subseteq \mathcal{NP}$.

As with the definition of $\mathcal{BPP}$, the error bounds of $2/3$ and $1/3$ in the definition can be varied considerably without changing the concept being defined.

Definition: Let $c, s : \mathbb{N} \to \mathbb{R}$ be functions satisfying $c(n) - s(n) > \frac{1}{p(n)}$ for some polynomial $p(\cdot)$. A *generalized* interactive proof system for a language L with *completeness bound* $c(\cdot)$ and *soundness bound* $s(\cdot)$ is a pair of interactive Turing machines (P, V) such that V is polynomial-time and the following conditions are satisfied:

Completeness: For all $x \in L$, $\Pr[\langle P, V\rangle(x) = 1] \geq c(|x|)$.

Soundness: For all $x \notin L$ and all interactive Turing machines B, $\Pr[\langle B, V\rangle(x) = 1] \leq s(|x|)$.

$g(n) = c(n) - s(n)$ is called the *acceptance gap* and $e(n) = \max\{1 - c(n), s(n)\}$ is called the *error probability*.

By this definition, an interactive proof system (P, V) is a generalized interactive proof system with completeness bound $c(n) = 2/3$ and soundness bound $s(n) = 1/3$. The acceptance gap and error probability are both 1/3.

We present the following non-trivial claim without proof.

Claim 1 *The following three conditions on a language L are equivalent:*

1. $L \in \mathcal{IP}$.
2. *For every polynomial $p(\cdot)$, there exists a generalized interactive proof system for L with error probability $e(n) \leq 2^{-p(n)}$.*

[3] V has to be written carefully in order to have polynomial time-complexity according to our definitions. Namely, V cannot afford to read all of y from its incoming communication tape before checking the condition that $|y| \leq q(|x|)$. Rather, it should compute $m = q(|x|)$ first and then read y symbol by symbol, stopping and rejecting when it first discovers that $|y| > m$.

3. *There exists a polynomial $p(\cdot)$ and a generalized interactive proof system for L with acceptance gap $g(n) \geq \frac{1}{p(n)}$. Furthermore, the completeness bound $c(n)$ and the soundness bound $s(n)$ for this system can be computed in time polynomial in n.*

This claim is useful in two ways. If we are given $L \in \mathcal{IP}$, we can assume the existence of an interactive proof system (P, V) with exponentially small error bound (condition 2). On the other hand, if we are tying to establish $L \in \mathcal{IP}$, it suffices to construct an interactive proof system (P, V) with polynomial-time computable completeness and soundness bounds that have a non-negligible acceptance gap.

41 Graph Non-Isomorphism is in $\mathcal{IP}$

The graph non-isomorphism language, GNI, is the set of all pairs of graphs (G_1, G_2) such that G_1 is not isomorphic[4] to G_2, written $G_1 \ncong G_2$.

An interactive proof system for GNI is based on the following idea. The verifier picks one of G_1 or G_2 at random and generates a random graph H isomorphic to it. Because $\cong$ is a transitive relation, if $G_1 \ncong G_2$, then H is isomorphic to exactly one of G_1 and G_2 but not to both. The prover determines which graph H is isomorphic to and returns the index of that graph (1 or 2), which the verifier then checks. On the other hand, if $G_1 \cong G_2$, the prover gets no information about which graph was used to generate H since H is equally likely to result whether starting from G_1 or G_2. Hence, the prover returns the correct index with probability at most 1/2, so with probability 1/2 the verifier rejects. Repeating this protocol twice lowers the error probability to below 1/3, as required by the definition of $\mathcal{IP}$. (Alternatively, one can conclude directly from Claim 1 that GNI $\in \mathcal{IP}$ since the acceptance gap of this protocol is $1/2$.)

We refer the reader to section 4.2.2 of the textbook for the detailed construction and proof. However, we make special note of Claim 4.2.9.1, which formalizes the argument presented above that the prover gets "no information" about the graph chosen by the verifier in the case that $G_1 \cong G_2$.

To give a few of the details, the verifier chooses a random value $\xi \in \{1, 2\}$ and a random $H = \Pi(G_\xi)$, where for each graph G, $\Pi(G)$ is a uniformly distributed random variable over the set $\{G' \mid G' \cong G\}$. We must establish that the random variables ξ and H so defined are statistically independent, that is, for each $\tau \in \{1, 2\}$ and each graph $G' \cong G_1 \cong G_2$,

$$\Pr[\xi = \tau \mid H(G_\xi) = G'] = \Pr[\xi = \tau] = \frac{1}{2}. \tag{1}$$

This is what says the prover's probability of correctly guessing ξ after receiving the graph H is still only $1/2$.

Our proof will make use of Bayes' theorem, which relates the a priori probability of an event A to the a posteriori probability of A given that event B has occurred.

Theorem 1 (Bayes' Theorem) *Let A and B be events with non-zero probability. Then*

$$\Pr[A \mid B] = \Pr[B \mid A] \left(\frac{\Pr[A]}{\Pr[B]} \right)$$

Proof: Conditional probability is defined by

$$\Pr[A \mid B] \stackrel{\text{df}}{=} \frac{\Pr[A \cap B]}{\Pr[B]}. \tag{2}$$

[4]Graph $G_1 = (V_1, E_1)$ is isomorphic to graph $G_2 = (V_2, E_2)$ if there is an isomorphism $\pi : V_1 \to V_2$ (a 1-1 and onto function) such that $(\forall u, v \in V_1)((u, v) \in E_1 \Leftrightarrow (\pi(u), \pi(v)) \in E_2)$.

from which it follows that

$$\Pr[A \mid B] \cdot \Pr[B] = \Pr[A \cap B]. \tag{3}$$

Using equation 3 twice (once with the roles of A and B reversed), we obtain

$$\Pr[A \mid B] \cdot \Pr[B] = \Pr[A \cap B] = \Pr[B \mid A] \cdot \Pr[A],$$

from which the theorem easily follows. ■

For a graph G', the isomorphism class of G' is the set $[G'] = \{G \mid G \cong G'\}$. (This is the set of all graphs isomorphic to G'.) Since $\cong$ is an equivalence relation and $G_1 \cong G_2$, we have that $[G_1] = [G_2]$. Call that set S and let $\alpha = 1/|S|$. Let $G', G'' \in S$. The random variable $\Pi(G'')$ is uniformly distributed over S, so $\Pr[\Pi(G'') = G'] = \alpha$, independent of the choice of G' and G''.

By the definition of conditional probability, we have

$$\begin{aligned}
\Pr[\Pi(G_\xi) = G' \mid \xi = 1] &= \Pr[\Pi(G_1) = G'] \\
&= \alpha \\
&= \Pr[\Pi(G_2) = G'] \\
&= \Pr[\Pi(G_\xi) = G' \mid \xi = 2].
\end{aligned}$$

Applying Bayes' theorem and the fact that $\Pr[\xi = 1] = \Pr[\xi = 2] = 1/2$, we get

$$\begin{aligned}
\Pr[\xi = 1 \mid \Pi(G_\xi) = G'] &= \Pr[\Pi(G_\xi) = G' \mid \xi = 1] \cdot \left(\frac{\Pr[\xi = 1]}{\Pr[\Pi(G_\xi) = G']} \right) \\
&= \Pr[\Pi(G_\xi) = G' \mid \xi = 2] \cdot \left(\frac{\Pr[\xi = 2]}{\Pr[\Pi(G_\xi) = G']} \right) \\
&= \Pr[\xi = 2 \mid \Pi(G_\xi) = G'].
\end{aligned}$$

Since also

$$\Pr[\xi = 1 \mid \Pi(G_\xi) = G'] + \Pr[\xi = 2 \mid \Pi(G_\xi) = G'] = 1,$$

equation 1 follows.

42 Interactive Proof Systems with Auxiliary Inputs

We mention briefly another generalization of interactive proof systems that will be needed later. Namely, it is sometimes useful to permit an interactive Turing machine to have an additional private input tape. We write $\langle P(y), V(z) \rangle(x)$ to denote V's output on a joint computation with common input x, private input y for P, and private input z for V. The time complexity is still measured in terms of the length of the common input x, so a machine with time complexity t must never use more than $t(|x|)$ steps, regardless of its private input. In particular, such a machine might not be able to read all of its private input and still stay within the allowed time bound.

We extend the completeness and soundness conditions from section 40.3 to account for the additional inputs as follows:

Completeness: For all $x \in L$, there exists $y \in \{0,1\}^*$ such that for all $z \in \{0,1\}^*$

$$\Pr[\langle P(y), V(z) \rangle(x) = 1] \geq \frac{2}{3}.$$

Soundness: For all $x \notin L$, for all interactive Turing machines B, and for all $y, z \in \{0,1\}^*$,

$$\Pr[\langle B(y), V(z) \rangle(x) = 1] \leq \frac{1}{3}.$$