15-503/859P: Cryptography **Lecturer:** Steven Rudich and Manuel Blum
Topic: Zero-Knowledge Proofs **Date:** 1 Feb 2006
Scribe: Yinmeng Zhang

5.1 Homework

5.1.1 Caveat Discipulus

If on a homework or exam you are not sure about a solution, be sure to say so. If you claim incorrectly you have a correct proof, you lose points. If you can point out the weaknesses in your proof, that is worth something. Honesty above all!

5.1.2 Vertex Cover

Recall that an instance of the vertex cover consists of a graph $G = (V, E)$ and a positive integer Bound B. We ask if there exists a set $S \subseteq V$, $|S| \leq B$ such that all edges are incident to some vertex in S. In Homework 1, we asked for a zero-knowledge protocol for Vertex Cover.

Aside: Notice that we require every edge of G to have an endpoint in S, but it is not necessary for every vertex in S to be an endpoint of an edge of G. For example, if we have an independent set (a graph with no edges), then it is vacuously true that *any* subset of its vertices is a vertex cover.

5.1.2.1 Solution from the Audience

Let m be the number of edges of G, and k be the security parameter. For $k(m+2)$ rounds, in each round :

1. Prover commits to
   (a) A random permutation of the original graph
   (b) A vertex cover on the permuted graph (by pointing arrows at the appropriate vertices)
2. The Verifier can then choose to see one of
   (a) The graph and permutation [*accept* iff the graph is isomorphic to the original G]
   (b) The arrows [*accept* iff there are B vertices]
   (c) An edge [*accept* iff Prover can exhibit an arrow pointing to one of the endpoints]

Proof of Correctness:

Observe that an honest Prover can always produce an acceptable proof that can be verified in polynomial time: thus we have *completeness*.

If a crooked Prover $\tilde{P}$ could produce a graph and permutation **AND** point arrows at B vertices such that **FOR ALL** edges there is an arrow to one of the endpoints, then $\tilde{P}$ could reverse the permutation on the B vertices and produce a vertex cover of the original graph G.

Thus, the contrapositive tells us that if a dishonest prover $\tilde{P}$ does not know a vertex cover, then in each round, *at least one* of the private pieces must be unacceptable. There are $m+2$ pieces (m edges, the permutation, and the set of B arrows), so in each round a Verifier has at least a $1/(m+2)$ probability of catching a dishonest Prover. Thus, after $k(m+2)$ rounds, a dishonest Prover has passed with probability at most $(1-1/(m+2))^{k(m+2)}$. Thus, we have *soundness*.

Finally, notice that any single private piece can easily be simulated by the Verifier alone. In particular, we can pick a random permutation of G, pick B random arrows on an unlabelled copy of G, or pick a random endpoint v of a specific unlabelled edge and put an arrow on v. Therefore the proof system is *zero-knowledge*.

■

5.1.2.2 More on the Soundness Guarantee

You may recall that

$$\lim_{n\to\infty}(1-1/n)^n = 1/e$$

In fact, for all natural $n > 1$, $(1-1/n)^n \leq 1/e$, which allows us to give a nice upper bound of $(1/e)^k$ on the probability a dishonest Prover can pass the protocol. This bound was quoted in the Trevisan notes.

In general, on an instance x, a zero-knowledge protocol should catch a dishonest Prover with probability $1/p(|x|)$ for some polynomial p. Thus, after $kp(|x|)$ rounds, which is polynomial, there probability a dishonest Prover can pass is $(1-1/p(|x|))^{kp(|x|)} \geq (1/e)^k$, which is exponential.

5.1.2.3 Solution from the Teacher

Let n be the number of vertices in the graph G, and k be the security parameter. For k rounds, in each round :

1. Prover commits to a random permutation of the original graph
2. The Verifier can then choose to see one of
   (a) The graph and permutation [*accept* iff the graph is isomorphic to the original G]
   (b) All possible edges between the $n-B$ vertices not in the cover [*accept* iff Prover can exhibit $n-B$ vertices with no edges between them]

Proof of Correctness:

Recall the "King's Crown" picture of a vertex cover: the vertices not in the cover form the points of the crown, and by definition there can be no edges between them. An honest Prover can easily construct a random permutation and knows $n-B$ vertices with no edges between them. Thus we have *completeness*.

If a crooked Prover $\tilde{P}$ knew $n-B$ vertices with no edges between them, then all edges in G must have at least one endpoint in the other B vertices. That is, those B vertices form a vertex cover. So, if $\tilde{P}$ knew both a permutation of G and $n-B$ vertices in the permuted graph with no edges

between them, then $\tilde{P}$ could reverse the permutation on the other B vertices to produce a vertex cover of the original graph G. The contrapositive tells us that if a dishonest prover $\tilde{P}$ does not know a vertex cover, then in each round, at least one of the private pieces must be unacceptable. There are 2 pieces, so in each round a Verifier has at least a $1/2$ probability of catching a dishonest Prover. Thus, after k rounds, a dishonest Prover has passed with probability at most $(1/2)^k$. Thus, we have *soundness*.

In each round, the Verifier sees either a random permutation of G, or $n - B$ vertices with no edges between them. Clearly this can easily be simulated by the Verifier alone, so the proof is *zero-knowledge*.

■

5.1.3 For More Information

See Professor Steven Rudich

- portable close-up magician
- 412-401-4019
- office hours before class, 2PM at Starbucks (Craig & Forbes)

5.2 Formal Definition

In class we have talked a lot about "gray paint", known to scientific community as *Ideal Envelopes* or *Bit Commitments*, which we now work into our formal definition.

5.2.1 Implementation: Trusted Third Party

One way to implement gray paint is to utilize a trusted third party, say the famously trustworthy website `www.bitcommitment.com`. The website maintains the functionality of the bitcommitment: the ability to COMMIT and to REVEAL. To COMMIT, the Prover sends a value x (privately) to the website, and the website stores it. To REVEAL, the Prover sends a request to the website, and the website forwards x to the Verifier. The website guarantees that the v the Prover commits to is the same x that the Verifier sees.

This system requires that both the Prover and the Verifier trust the website, and that the Prover does not mind having to show his private bits to the website. Someday we will talk about how to port lottery tickets to the digital world more sensibly, but that day is not today.

5.3 IP with Bit Commitment

An **IP proof system with bit commitment** is a 9-tuple

$$(P, V, H, x, y, z, r_p, r_v, r_h)$$

where P is the Prover, V is the Verifier, and

- H is the mutually trusted function, which is usually some very simple machine, not a website. The H stands for honest. For proof purposes, we get to *assume that H is honest.* Notice that in all our proofs so far, we have implicitly assumed that the gray paint is honest.
- x is the common input. In our proofs so far, it is the instance of the problem we are considering. For example, in Vertex Cover, it would be the graph G and the security parameter.
- y is the Prover's private input – no one else knows y! In our proofs so far, it has been the solution that the Prover knows. For example, in Vertex Cover, it is the B vertices in the cover.
- z is the Verifier's private input. Why does the Verifier need a private input? We haven't seen a need in any of our proofs so far, but this is a formal definition; we might as well throw it in just in case.
- r_p, r_v, and r_h are the random bits of the Prover, Verifier, and Honest function respectively.

We introduce the notation $(P \leftrightarrow_H V)$, meaning a conversation between Prover P and Verifier V with mutually accessible Honest function H.

The definition of interactive proof system with trusted party H is analogous to that of the previous lecture. Let L be a language (such as Vertex Cover). An **interactive proof system for language L with trusted party H** satisfies the following two conditions:

- Completeness: For every $x \in L$,

  $$Pr[P \leftrightarrow_H V(x) = 1] \geq 2/3$$

  That is, for every instance for which there is a proof, the honest Prover can convince the Verifier of it at least 2/3 of the time. In all our proofs so far, we have done even better: the honest Prover is guaranteed to convince the Verifier, 100% of the time.

- Soundness: For every $x \notin L$ and for every possible Prover P^*,

  $$Pr[P^* \leftrightarrow_H V] \leq 1/3$$

  That is, for every instance for which there is no proof, no Prover, no matter how tricky and dishonest, can falsely convince the Verifier more than 1/3 of the time.

An Interactive Proof is **Perfect Zero-Knowledge** if it further satisfies the following condition:

- For any polytime Verifier V^* there exists a simulator M^* s.t. $(P \leftrightarrow_H V^*)(x) \equiv M^*(x, z)$

  That is, a machine M, knowing only what the Verifier knows, should be able to model P and V^*'s conversation with the correct probability distribution. This means V^* is not getting any information from P. Note that this holds for any V^* - even a crooked Verifier cannot trick the Prover into revealing anything in the conversation the Verifier could not simulate alone.

5.3.1 Note: The 2/3 Bound

There is no especial reason for using the numbers 2/3 and 1/3, except that they are nice round numbers. What is necessary is that there is a gap between the completeness bound and the soundness bound. $1/2 + \epsilon$ and $1/2 - \epsilon$ for any non-negligible (meaning at least $1/p(|x|)$ for some polynomial p) epsilon will suffice.

5.3.2 Note: Simulator vs Prover

It may bother you that the simulator can model the conversation without knowing P's secret, yet a dishonest Prover $\tilde{P}$ cannot model the conversation (thereby tricking the Verifier) without knowing P's secret.

One way to resolve this seeming paradox is to observe that a dishonest Prover would not have access to the verifier's private random bits, which *are* part of the input to the simulator. In other words, the trick is that the prover has to commit before knowing what the verifier wants to see, while the simulator commits afterwards.

5.4 Case Study: 3-SAT

Instance:

- an array of variables $v_1, \ldots, v_n$ and
- an array of clauses $c_1, \ldots, c_m$, where each c_i is a set of literals, where each literal is either a variable or the complement of a variable

Question:

- Is there a truth assignment (f : variables $\rightarrow \{T, F\}$) to the variables such that for every clause there exists a literal set to true? (Recall that the complement of a variable is true iff the variable is false.)

5.4.1 Protocol 1 - Wrong

1. The Prover randomly permutes the clauses and commits to the permutation and the permuted clauses.
2. The Prover commits to a representative true literal in each permuted clause.
3. The Verifier chooses to either see the clauses[to check that the problem is correct] or randomly chooses a clause to see that a representative true literal has been marked[to check that the clause has a literal set to true]

It is not hard to see that Protocol 1 is zero knowledge, but unfortunately being able to randomly permute the clauses and mark a true literal in each clause does not prove that the Prover knows a

solution. In fact, a dishonest Prover could simply mark all the literals to be true, and thereby pass this protocol.

We need to be able to check *consistency* - that a variable has been set to exactly one value.

5.4.2 Protocol 2 - Wrong

1. The Prover randomly permutes the clauses and commits to the permutation and the permuted clauses.
2. The Prover commits to a representative true literal in each permuted clause.
3. *The Prover to also commit to a table of literals and assignments.*
4. The Verifier chooses to either see the clauses[to check that the problem is correct] or randomly chooses a clause to see the representative true literal *and its corresponding row in the table of literal assignments*[to check that the clause has a literal set to true and that the assignment is consistent].

Unfortunately, while this protocol is not susceptible to dishonest provers, it is susceptible to dishonest verifiers. When we reveal the true literal, we also reveal whether it is a variable or its complement. Over many rounds the verifier might get a good idea of how many clauses have positive literals marked and how many clauses have negative literals marked.

5.4.3 Protocol 3 - Right!

We want to somehow permute the literals. One way to approach the problem is as follows:

The Prover randomly maps the literals $v_1, \neg v_1, \ldots, v_n, \neg v_n$ to the new names $a_1, \neg a_1, \ldots, a_n, \neg a_n$ so that if v_i maps to a_j then $\neg v_i$ maps to $\neg a_j$, and if v_i maps to $\neg a_j$, then $\neg v_i$ maps to a_j. Thus, in the new naming scheme, we have hidden whether the literals in the clauses were originally positive or negative.

1. The Prover commits to the mapping, the mapped instance, and a table of truth values for the mapped literals.
2. The Verifier can choose to see the mapping and mapped instance to check that the problem instance has not been changed. Or, the Verifier can choose a clause, and the Prover will reveal a literal in that clause and its entry in the table so that the Verifier can check that the clause contains a literal set to true.