6.857 Computer and Network Security **Fall Term, 1997**

Lecture 12 : October 14, 1997

Lecturer: Ron Rivest *Scribe: Christine Su*

1 Topics Covered

- Halevi-Micali commitments
- Blobs (commitments with quality testing)
- Graph 3-coloring
  - with commitments
  - with blobs
- Proof of "arbitrary knowledge" (with blobs)

Reminder: Midterm will be handed out on October 30th, and the term paper proposals are due on November 6th. Term paper should be something interesting and related to computer security, there are no restrictions on how much theory or implementation need to be covered.

2 Halevi-Micali Commitment Scheme

We will start off with a commitment, $c = h(r, m)$, where h is a collision-free hash function, r is a randomly selected number, and m is the message to be committed to.

Let $h = sha1$ (yields 160 bit values)
$p = prime$ (161 bits)
$z = h(m)$
$x \in_R \{0,1\}^{160}$
pick a and b such that $ax + b = z \pmod{p}$
let $y = h(x)$
and commitment is $c = (y, a, b)$
to open this commitment, one needs to reveal x and m.

3 Graph 3-coloring with commitments

Graph 3-coloring is an NP-complete problem, which suggests that if a zero knowledge proof exists for it, then a zero knowledge proof exists for ALL problems that belong within NP. That means that for any problem which can be verified, a prover can demonstrate his knowledge of a solution without giving away the solution (so while we still can't prove we know Marilyn Monroe's middle name, we can prove we know how to factor a number, since the verifier can take the factoring, and multiply it out to check if our solution was valid).

We construct a zero knowledge proof for graph 3-coloring. A 3-coloring is shown below in Figure 1: given n vertices and m edges, such a map will color every vertex with one of three different colors, and no two adjacent vertices should have the same color. In the protocol we construct, the Prover knows a three coloring for the graph, but does not need to give it away to the verifier. In fact, any verifier can just generate a similar transcript for a conversation on his own. Therefore, this is a zero-knowledge interactive proof.

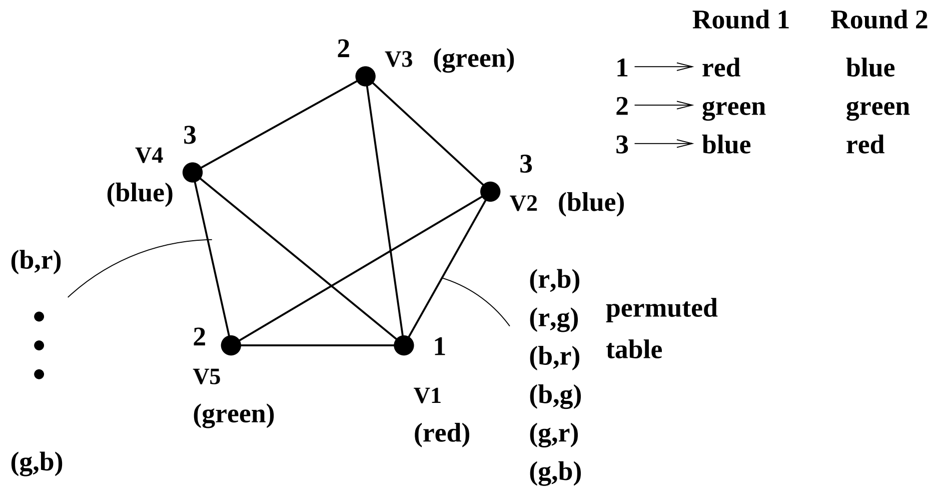


Figure 1: 3-coloring commitment

The interactive proof is performed in multiple rounds. For each round:

- The prover writes down the coloring, randomly permuting colors.
- The prover then commits to some color at each vertex. We use a secure commitment scheme, so that the verifier can't look at the commitments and learn

which ones have the same color.

- The verifier then randomly selects an edge on the graph, and requests that the prover 'reveal' the commitments for the two vertices on the edge.
- If the verifier asked for a valid edge, the prover then reveals the two requested values. At this point, the verifier just makes sure that the two revealed values do not have the same color.

Example 1 *Prover claims he knows a three-coloring for the graph in figure 1. For this example,* $C(x)$ *is a commitment of* x*, and* $R(y)$ *is the reveal operation. Note that* $C('a') \neq C('a')$ *most of the time, but that* $R(C('a')) = R(C('a'))$.

Prover sends to Verifier :

$$\begin{aligned} &\textit{color at } v_1 = C(\textit{'red'}) \\ &\textit{color at } v_2 = C(\textit{'blue'}) \\ &\textit{color at } v_3 = C(\textit{'green'}) \\ &\textit{color at } v_4 = C(\textit{'blue'}) \\ &\textit{color at } v_5 = C(\textit{'green'}) \end{aligned}$$

Verifier sends to Prover: E_{v_2,v_3}.

Prover then reveals v_2, v_3*, which show the verifier that one is blue, and the other is green in this round, or more significantly, that the two neighbors don't have equal coloring. If so (as here), verifier accepts this round.*

3.1 Analysis of Scheme

Now, we must show that it is indeed a valid proof, AND that it is zero knowledge.

- *Completeness* requires that the valid prover be able to complete this protocol successfully. Intuitively this works, since if the prover has a valid coloring of the graph, he will always end up providing two vertices with different colors, and the verifier will always accept.
- *Soundness* requires that an invalid prover will not stand a good chance of fooling the verifier. In this case, we assume that the prover doesn't have a valid 3-coloring. This means that in the best case for the prover, he only has one edge that has vertices of the same color. At any round, since the verifier asks for an

edge randomly, there is a $1/m$ chance that the bad edge will be asked for, or in any given round, there is a $1/m$ chance that the verifier will catch the prover.

This means that there is a $(1-1/m)$ chance of the Prover fooling the Verifier in any round. If we run for m^2 rounds, this means that there is a $(1-1/m)^{m^2} = e^{-m}$ best-case chance for the Prover to fool the verifier. This is very unlikely for large m. However, it also requires that we run this MANY times. For example, a graph with one hundred edged requires ten-thousand iterations to achieve this security.

- *Zero-Knowledge* also needs to be shown. Intuitively, at any round, the only thing the verifier learns is that the two vertices are assigned a different color. But he already knew that. Since we permute the colors each round, it is impossible to draw any sort of link between the values in the different rounds.

  More sctrictly, for any verifier, the verifier can easily create a transcript with similar distribution by randomly assigning colors to the graph, challenging itself with the edge it normally would pick, and revealing the 'colors' at the edges. If this revealing would show an invalid mapping, just wind back and try again. The distribution would still be the same, since the same set of edges will be picked by the verifier whether it is talking to the prover or to itself.

4 Blob

A **blob** is a commitment with an extra function: equality testing. Every blob supports three operations:

- *commit* takes data and 'hides' it in such a way that it can later be revealed by the committer, but not changed.
- *reveal* is the reverse of *commit*. It takes a value that was run through the *commit* function and uncommits it. Often this is done by providing the message and some key, which are then fed to *reveal* and it decides whether the *commit* value corresponds to a commit of the message.
- *prove equality*. This is the extra feature of a blob. This allows a person to take two *commit* values, and check (with the help of the person who originally performed the *commit* operation) if the two are commiting the same value. Since *commit* is probabilistic, you can't just check whether the two values are equal, since they are likely not to be equal even if their contents are the same.

We have seen that in CvHP scheme

$$\begin{array}{ll} \alpha^r\beta^m \text{ (mod p)} & \text{commitment to } m \\ \alpha^{r'}\beta^{m'} \text{ (mod p)} & \text{commitment to } m' \\ \hline \alpha^{r-r'}\beta^{m-m'} = \alpha^{r-r'} & \text{if } m = m' \end{array}$$

One can demonstrate that the ratio of commitments has form of $\alpha^{r-r'} = \alpha^k$ by revealing k. So, when the prover wishes to demonstrate equality, he merely presents k (which is the discrete log of the ratios). Note that this does not let us prove inequality.

Using **blobs**, we can speed up the ZKP for 3-coloring. The scheme follows the steps below. For each round:

- for each edge:
  - Create a color table which contains each possible valid coloring for the edge, but with the entries permuted: $\begin{cases} (r,b) \\ (r,g) \\ (b,g) \\ (b,r) \\ (g,r) \\ (g,b) \end{cases}$
  - Commit to each entry in the table seperately (so, in line one, put $C(r), C(b)$ for the above table.)
- verifier picks challenge $c \in_R \{0,1\}$
- if $c = 0$: reveal all the tables. Verifier checks that the tables contain six entries randomly permuted, with no duplicates and no invalid entries.
- if $c = 1$: Prover first supplies an index for each edge which corresponds to the entry in the color table for that edge (so in figure 1, edge (v_1, v_2) would point to entry 1 in the table shown for that round). He similarly would provide one such index for every edge.

  The above demonstrate that the assignment of colors are from a table where no connected vertices have the same colors. Unfortunately, this doesn't require any consistency (for example, the prover can supply edge (v_1, v_2) with (r, g), and $v_2, v_3)$ with (b, r). Obviously, if we could look inside the commits, we would learn that the vertex is both green and blue, which is impossible. However, we can't, so instead, we use the equality testing to make sure that even though we don't know the coloring, we do know that they are equal. So, the prover must also supply, for every vertex, a proof that all the edges that connect to the vertex have the same choice of color on each edge.

In each pass of this protocol, we have a 50% chance of catching the prover if he's invalid, so our security is $1/2^k$ where k is the number of rounds we go through.

5 Proof of Knowledge for Circuits

Satisfiability is another well-known NP-complete problem, and one which is simpler to map to many problems in computing. Given a logic circuit, what (if any) set of inputs lead to an output of 'true'?

We can imagine representing most problems as such circuits (after all, that is how microchips work). We now present a zero-knowledge proof for circuits.

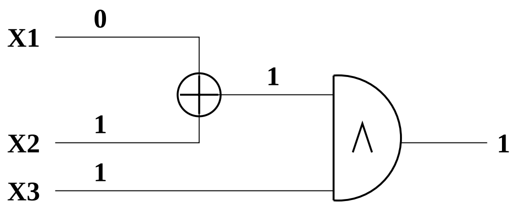


Figure 2: Logic circuitry

For example, I know an x that makes $f(x) = 1$, where f is a boolean circuit we had agreed to use. More generally, one can prove knowledge of arbitrary secret x that satisfies some conditions. (e.g. "x is a factor of n", "x is way to 3-color nodes of graph", or "x is isomorphism between G_1 and G_2")

Condition $\equiv f(x) = 1$, x is a bit vector, and f is a circuit (e.g. "is divisor of n")

Example 2 $x_3(x_1 \oplus x_2) = 1$

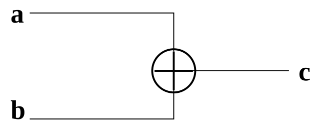
How to prove?

- use commitments with equality-testing (blobs)

$$\left.\begin{array}{l} u = \alpha^r \beta^m \\ v = \alpha^{r'} \beta^{m'} \end{array}\right\} m = m' \text{ if one can solve } log_\alpha(u/v)$$

this reveals no information about m or m', other than the equality it is trying to prove.

1. Prover generates a truth table for each gate, but randomly permutes the rows so that they are not the same each round. He then commits to every entry in the table seperately.



XOR

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

permute rows

a	b	c
1	1	0
0	1	1
1	0	1
0	0	0

Figure 3: Logic gate truth table

Prover also commits the value in every wire (including the input and output wires).

2. Verifier flips a coin $c \in_R \{0,1\}$, and gives the challenge to prover

3. Prover:

   - opens output commitment to show that the output is one.
   - if $c = 0$: opens all permuted truth tables. Verifier needs to check that all the tables make sense (contain every possible entry, no duplicates, and no invalid entries).
   - if $c = 1$: for each gate, prove values on wires (a,b,c) equal values in some row of the permuted truth table. Make sure that those values are consistent (equal) across gates where the same wire is the output of one gate and the input of another.

Repeat the process t times, accept if all checks pass.

5.1 Analysis

Again we need to show the three properties of ZKP:

- *Completeness* holds, since the entire protocol can be done by a valid prover with no difficulty.
- *Soundness* is more difficult to demonstrate. For an invalid Prover to succeed, he must either predict the challenge (we assume he can't), or else must be able to construct things that will let him pass the challenges without knowing them in advance. Since this is an invalid prover, he doesn't have a way of assigning correct values in all wires that lead to an output of 'true', so somewhere along the line, he's going to have to either change the value of a wire, or the values in a truth table.

  Since the challenge is unpredictable, he only has a 50% chance of changing the values in the truth tables and not be caught, and a 50% chance of changing the values on the wires and not be caught (this is basically having the output of one gate be a 0, and the input to the next gate be a 1).

  This means that at any pass, an invalid prover will get away with 1/2 probability, and after k passes, will only successfully pass with $1/2^k$ probability.
- *Zero Knowledge* can be shown by similar technique to the proof for the 3-col system. Again, any verifier can generate an indistinguishable transcript by having a conversation with himself.

So we now have a ZKP for satisfiability, and therefore one for every problem in NP. In the homework we will create a ZKP for the same system but without relying on blobs.