CS 290G (Fall 2014) – Introduction to Cryptography Dec 9th, 2014

Lecture 16: Zero Knowledge Proof System Construction

Instructor: Rachel Lin *Scribe: Asad Ismail*

1 Recap

In the previous class, we discussed *knowledge* that is conveyed within a message or an interactive conversation. We took a behavioural view and tried to define what we meant by *no knowledge being learnt.* If the behaviour of an interacting participant within the conversation does not change as a result of the conversation, we say no knowledge was learnt. Put differently, if whatever a participant can do after the conclusion of the conversation could have been done by him before the conversation as well, the conversation has not increased his capability in any way and thus, has not revealed any knowledge to him.

2 Zero Knowledge Proof System

A proof system enables a prover to convince a verifier of the validity of a statement. For example, Our statement could be $x \in L$, where L is an arbitrary NP language $\exists$ witness w which satisfies relation $R_L(x, w) = 1$. A proof system which is zero-knowledge proves this statement while revealing nothing to the verifier other than the statement validity. Proving statements for NP languages is pretty easy as all NP languages are efficiently verifiable (We just need to send the witness w over to the verifier, who can verify whether $R_L(x, w) = 1$). The tricky part is proving in a manner which is zero-knowledge. In this lecture, we will construct a ZK Proof System for any arbitrary NP-language L.

2.1 Definition

Recall: $L = \{x : \exists w, R_L(x, w) = 1\}$
Definition: For a ZK Proof System $< P, V >$, P, V are PPT interactive machines. Before we define the properties which make a ZK Proof System, let us define a few key terms for the interaction between P (the prover) and V (the verifier).

- $P(x, w)$ has both the statement and witness as an input, while $V(x)$ simply has the statement.
- $e \leftarrow [P(x, w) \leftrightarrow V(x)]$ s.t. e is the entire log of the execution of the protocol. Namely, given e, we can recreate the complete protocol execution.
- $Out_v(e)$ is the output of V in this log e

- $View_v(e)$ is everything V *sees* in an execution, namely the input x, all the messages from P, and all the random coins V tosses

Now, we want three properties from a ZK Proof System:

- **Correctness/Completeness**: Every true statement can be proven.
  $\forall x \in L, \exists w \in R_L(x)$: $Pr[e \leftarrow [P(x,w) \leftrightarrow V(x)] : Out_v(e) = 1] = 1$
- **Soundness**: Every false statement should be rejected, even if prover P^* is malicious.
  $\forall x \notin L, \forall$ n.u. PPT P^*: $Pr[e \leftarrow [P^*(x) \leftrightarrow V(x)] : Out_v(e) = 1] \leq neg(n)$
- **Zero Knowledge**: A verifier learns nothing from the prover, even if it is malicious.
  $\forall$ n.u. PPT $V^*, \exists$ Simulator $S, \forall x \in L, \exists w \in R_L(x)$:
  $\{e \leftarrow [P(x,w) \leftrightarrow V^*(x)] : View_{V^*}(e)\} \approx \{S(x)\}$

2.2 Construction

Theorem: If One-Way Permutations (OWP) exist $\rightarrow$ Every NP language has a ZK Proof System
Proof: We will prove this theorem in two steps:

- Step 1: Construct a ZK Proof System for a specific NP language
- Step 2: Extend the ZK Proof System to any arbitrary language L

We will first construct a ZK Proof System for Graph 3 Coloring, L_{3-col}, which is a NP-Complete language. We will then show how any statement and witness in an arbitrary language L can be mapped to L_{3-col} using the Cook Reduction.

For any graph $G = (V, E)$, an edge is represented by a tuple of 2 vertices (v_i, v_j). Imagine we have the color set $\{R, B, G\}$. A graph is 3-colorable if we can find a coloring or assignment for every vertice $\phi : V \rightarrow \{R, B, G\}$ which satisfies the constraint:
$\forall (v_i, v_j) \in E : \phi(v_i) \neq \phi(v_j)$.

All graphs which are 3-colorable are members of the Graph 3 Coloring language. Namely:

$L_{3-col} : \{G : \text{s.t. vertices in } G \text{ are 3-colorable}\}$

L_{3-col} is clearly an NP language as there exists a witness w (The colorings) which are efficiently checkable (Just check if the constraint holds for all edges).

It is also a NP-Complete language, where NP-Complete refers to those languages such that any arbitrary NP language can be expressed in terms of a NP-Complete language.

2.2.1 Step 1: Constructing ZK Proof System for L_{3-col}

We will first try constructing this protocol as a game played between 2 players $P(G, \phi)$ and $V(G)$ in the physical world. The protocol is described as follows:

- **Stage 1**: P picks a random permutation π of the color set $\{R, B, G\}$ and obtains a new coloring of the graph, $\pi(\phi)$ (Notice that this coloring is also a valid witness). V lays down graph G on the ground and is not allowed to see what P does next. P colors the laid graph with coloring $\pi(\phi)$ and covers up each node with an opaque cup. V is then allowed to turn back and see the graph with all nodes being covered by cups.
- **Stage 2**: V wants to verify if P has successfully met the 3-coloring constraint for the graph. V thus randomly selects an edge (v_i, v_j) from the edge set $((v_i, v_j) \leftarrow E)$ and sends his choice over to P as a challenge, asking P to open up that edge.
- **Stage 3**: P removes the cups covering node v_i and v_j, thus revealing their colors c_i and c_j. if $c_i \neq c_j$, V outputs Accept, otherwise, it outputs Reject.

Correctness and Soundness

Our construction guarantees Correctness but clearly, Soundness is not captured because a cheating prover P^* can pass the verifier's challenge, even if he doesn't have a correct witness.
After Stage 1, P^* has *committed* to his color φ. Now if $x \notin L, \exists (v_i, v_j) \in E, \varphi(v_i) = \varphi(v_j)$, the probability that V will pick this *bad* edge as his challenge is $\geq \frac{1}{|E|}$. The soundness error, or the probability that P^* can get away with cheating is $\leq 1 - \frac{1}{|E|}$.
We fix this problem by running the protocol instance multiple times so that we have V challenging P^* multiple times. Running the protocol $|E|.n$ times changes the soundness error to (Each run is independent):

$\Pr[P^* \text{ convinces } V\ |E|.n \text{ times}] \leq (1 - \frac{1}{|E|})^{|E|.n}$

and we know that $(1 - \frac{1}{|E|})^{|E|.n} \leq \frac{1}{2^n}$ when $|E|$ is sufficiently large. Thus for large enough graphs which are connected, Soundness holds when we repeat the protocol $|E|.n$ times.

Zero Knowledge

To ensure zero-knowledge, our goal is to construct a simulator, $S(G)$ that can recreate the view of the malicious verifier V^*. If we can do this for one round of the protocol, then we can just repeat the procedure $|E|.n$ times. Intuitively speaking $V^*(G)$ doesn't really learn anything by just looking at a challenge edge's coloring (especially because even the

colors themselves have been randomly permuted, so V^* just looks at two random distinct colors as the response to his challenge).
Our first observation here is that if S, acting as the prover, knows the challenge (v_i, v_j) prematurely, he can respond appropriately and we get the correct view for V^*. Our second observation is that if S randomly guesses which edge V^* will pick and colors v_i and v_j appropriately (Randomly chooses different colors for v_i and v_j), he gets it right with a probability of $\geq \frac{1}{|E|}$ for one round of the protocol. We thus have two cases:

- Case 1: S anticipates the challenge correctly, and outputs the correct view of V^*.
- Case 2: S cannot anticipate the correct challenge, and gets stuck when V^* asks a challenge S knows he will fail on.

It's important to notice that till this point, the simulator isn't doing anything special; i.e. it is as powerful as any arbitrary prover. What makes the simulator more powerful/special is the power of being able to restart V^*. We think of V^* as a program S runs which it can arbitrarily restart it.
Thus, given this power. Our Simulator S runs as follows:

- S runs V^*
- while Case 2 keeps happening, keep restarting V^*
- If Case 1 happens, the simulator succeeds
- Outputs the $View_{V^*}$ from the last successful execution

When S runs as described above, we are sure to correctly output the view of V^*. The only thing left to check is whether S can be run *efficiently*. We need to bound the expected running time of S and make sure it is polynomial. In the algorithm described above, the only thing that is unknown is the number of times Case 2 happens until we succeed. Thus we need to calculate the expectatiton of Case 1 happening:

$E[Case1] = \frac{1}{Pr[Case1]} = \frac{1}{\frac{1}{|E|}} = |E|$

Thus, the expected running time is bounded and S can be run efficiently.

2.2.2 Step 2: Constructing ZK Proof System for arbitrary NP language L

Now that we've constructed a valid ZK Proof System for L_{3-col}, we will proceed to make one for any arbitrary NP language L. Given $L, x \in L, \exists w \in R_L(x)$, we proceed in two phases:

- Phase 1: P and V independently run the Cook Reduction to convert the problem from language L to L_{3-col}. P converts $(x, w) \in R_L \to (x', w') \in R_{L_{3-col}}$. V, on the other hand, converts $x \in L \to x' \in L_{3-col}$. As the Cook Reduction is deterministic, P and V can apply it individually without any interaction.

- Phase 2: Now the problem is simply one in L_{3-col}. P and V simply run the protocol constructed in Step 1

The security properties of the overall protocol are the same as the one in Step 1, because all communication between the overall protocol is the same as the Step 1 protocol.

3 Realizing the Physical World ZK Proof System

Now that we have a protocol in the physical world, we need to realize it in the digital domain. We notice that the only thing that needs an equivalent in the digital world are the opaque hiding cups we used. Everything else can be directly translated.
We identify two properties of the cups that we need to realize:

- Hiding: When a node is covered by a cup, it is completely hidden.
- Binding: Once a node has been covered, the prover is unable to change the color of the node under it later.

These two properties can be realized by building a Commitment Scheme.

3.1 Commitment Scheme

A Commitment scheme consists of two phases:

- Commitment Phase: The Committer C commits to a message m by using com with some random coins r that goes into com ($c = com(m; r)$). C then sends over the commitment c over to the receiver R
- Decommitment Phase: C sends over (m, r) to R and R verifies that $c = com(m; r)$.

Definition: We say a PPT com is a commitment scheme if $\exists$ poly l s.t. we have these two properties:

- Binding: $\forall n \leftarrow N$ and $\forall v_0, v_1 \in \{0,1\}^n, \forall r_0, r_1 \in \{0,1\}^{l(n)}, (v_0, r_0) \neq (v_1, r_1)$
  $com(v_0, r_0) \neq com(v_1, r_1)$
- Hiding: $\forall$ two sequences $\{v_{0n}\}, \{v_{1n}\}$, $v_{0n}, v_{1n} \in \{0,1\}^n$
  $\{r \leftarrow U_{l(n)} : com(v_{0n}; r)\} \approx \{r \leftarrow U_{l(n)} : com(v_{1n}; r)\}$

3.2 Commitment Scheme Usage and Construction

We commit to each node color under π. i.e. $\forall i \in E, com(\pi(\phi(i)); r_i)$

When challenged with (v_i, v_j) by V, we decommit $(\pi(\phi(v_i)), r_i)$ and $(\pi(\phi(v_j)), r_j)$.

Construction of 1-bit commitment *com* from OWP f

$com(v; r)$: $f(r), h(r) \oplus v$

In this construction, our hiding property is ensured by the PRG property (A OWP and a hardcore predicate ensure pseudo-randomness). Our binding property is ensured by the fact that $f(r)$ is only possible when applied on r, and $h(r) \oplus v$ is only possible when applied on v (with r having already been determined by f).