600.641 Special Topics in Theoretical Cryptography 02/06/06

Lecture 6: ZK Continued and Proofs of Knowledge

Instructor: Susan Hohenberger *Scribe: Kevin Snow*

1 Review / Clarification

At the end of last class, there were some questions about how simulators work. In particular, there was a question about why a verifier who hashes the messages of the prover was NOT a problem for the simulator we described for (sequential composition) 3COL, but was a problem for the simulator we described for the parallel composition of ISO. The same logic applies to the (sequential composition) ISO and the parallel composition for ISO. Let's review these protocols and simulators.

First, lets consider the ISO protocol. Recall the steps in this protocol with a verifier who chooses "G" or "H" based on a hash of the provers first message, rather than randomness:

1. Prover sends random permutation of G, $C = \phi(G)$.
2. Verifier sends "G" if hash(C)=1 and "H" if hash(C)=0.
3. Prover sends ϕ if "G" received and $\phi * \pi^{-1}$ if "H" received. We'll call this permutation α.
4. Verifier checks that $C = \alpha(G \text{ or } H)$.

Again, this is the same as the ISO example we have repeatedly seen, except for the fact that instead of the verifier randomly choosing "G" or "H" in step two, it is chosen based on the value of C in step 1. Now lets construct the simulator for the ISO problem where the verifier uses this hashing rather than random tape:

1. Pick either graph G or H at random. (Suppose G is chosen.)
2. Choose a random permutation α
3. Compute $C = \alpha(G)$.
4. Now, V^* selects "G" or "H" based on a hash of input C.
5. If V^* responds with "G", return α and record (C, α) as the messages from the "prover".
6. If V^* responds with "H", abort, don't record anything and start over at the beginning.

Steps one through three simply generate a C that will correctly map to G, since the verifier doesn't actually know the permutation between G and H. This is analogous to the prover generating the correct mapping for C. Then, depending on the hash of C, either "G" or "H" are selected. Our example is meant to work with G since that is what was chosen in step one. So, if G is selected a correct guess is recorded. Otherwise we can rewind and try again until correct.

Let's first consider sequential composition. In step four, the correct graph would be chosen with $\frac{1}{2}$ probability assuming the hash result is normally distributed. So, when run k times, each round has $\frac{1}{2}$ probability of succeeding, yielding a total of $2k$ expected rounds to finish with k correct results. Given this prediction, and the fact that the simulator is bound by *ppt*, this is acceptable.

Next, let's consider the case of parallel composition. In step four, "G" or "H" will be selected k times in parallel. So, although there is only one round, k guesses must be correct in this single round, where each of these individual guesses is actually correct with $\frac{1}{2}$ probability. This means that the probability of ALL the guesses being correct is $(\frac{1}{2})^k$! Now, given even fairly small values for k, the simulator will have to run so many times for all guesses to be correct that it exceeds the *ppt* bound! So, it appears that this specific simulator fails under parallel composition. Although, this does not necessarily mean that *no* simulator would work for this protocol under parallel composition.

2 Classes of Zero-Knowledgeness

Thus far, we have formally defined the zero-knowledgeness property as:

$$\forall V^*_{ppt} \exists S_{ppt} \forall x \in L,\ \forall a \in \{0,1\}^{poly(|x|)}$$

$$VIEW_{P,V^*(a)}(x) = S(x,a)$$

In this section we show that this is in fact only one of several acceptable definitions of zero-knowledgeness. Each definition may be useful depending on the particular proof. In fact, the definition we are familiar with is called *Perfect ZK (PZK)* because the distributions of the view and simulator are exactly the same. More formally, for two distributions X and Y over finite domain D:

$$\forall z \in D,\ Pr[x = z] = Pr[y = z]$$

While this is the strongest definition, it is also the most difficult to construct protocols for. To make construction easier, we could relax the definition to allow the simulator to produce a distribution that is *very close* (but perhaps not identical to) the distribution produced by the real prover and real verifier. This is written as:

$$VIEW_{P,V^*(a)}(x) \cong S(x,a)$$

This second definition is called *Statistical ZK (SZK)* because the distributions are not required to be exactly the same, but only statistically close. More formally, for two distributions X and Y over finite domain D:

$$\sum z \in D \ \mid Pr[x = z] - Pr[y = z] \mid \leq negl(k)$$

This essentially says that the variance between the two distributions is negligible. A final class can be defined where the distributions of the VIEW and simulator cannot be distinguished by any p.p.t. distinguisher. This means that the two distributed may be nothing alike, but a distinguisher with limited time cannot tell them apart:

$$VIEW_{P,V^*(a)}(x) \stackrel{c}{\approx} S(x,a)$$

$$\forall T_{ppt}, \mid Pr[x \leftarrow X; T(1^k, x) = 1] - Pr[y \leftarrow Y; T(1^k, y) = 1] \mid \leq negl(k)$$

This is called *Computational ZK (CZK)* because the distributions may be completely different, but based on a hardness assumption a distinguisher can not tell them apart. An example of this might be a protocol that uses quadratic residues where the verifier is not able to tell apart one number, being a quadratic residue, from another that is not because (we assume) it cannot be done in probabilistic polynomial time.

3 What can be done in ZK?

We already know from previous lectures that there exists an interactive proof for any problem in PSPACE. In other words, IP=PSPACE. But, for what set of problems does a zero-knowledge proof exist? In 1985, Goldwasser, et al., first demonstrated that that there exists a zero-knowledge proof for quadratic non-residues [3]. Although this shows a zero-knowledge proof exists for a single problem in this space, it does not show it exists for all problems in the space.

Later, Goldreich, Micali, and Wigderson were able to show that there exists a zero-knowledge proof of membership for every language in NP [2]. They are able to achieve this by showing that there exists a zero-knowledge proof system for the 3COL language (graph 3-coloring) we have studied. Since 3COL is an NP-complete language, then we know that there exists a zero-knowledge proof for all languages in NP.

There is, however, one caveat to their findings. An assumption of the existence of one-way functions is present. We saw this in the 3COL problem as the "hats" placed over each of the colored nodes or the commitment scheme as the digital equivalent. Without these one-way functions, the zero-knowledge proof as we saw it would not be possible. Also, note that this finding refers to the computational definition of zero-knowledge because, in our 3COL protocol, the simulator we described produced views that were only computationally indistinguishable from the real vies. It is interesting to note that the finding NP $\subseteq$ CZK and the existence of one-way functions seems to imply that NP $\neq$ P, although this is still an unknown problem since the existence of one-way functions is debatable itself.

The next logical question to ask is whether any language that admits an interactive proof can also admit a zero-knowledge proof. This is a big question, but one big step towards showing ZK=IP is the finding that AM=IP [4]. The Arthur-Merlin (AM) game is a special case of the interactive proof system. The difference is that in IP, the coin toss, or randomness, is private to the verifier whereas in AM this coin toss is public.

In addition to this finding we have observed in our own study in class, a recurring format for the messages in a zero-knowledge proof:

1. The prover sends some information such as the graph permutation C in the ISO example, or commitments $C_1, C_2, ...$ in the 3COL example.

2. The verifier chooses some random value such as graph G or H for the ISO example, or a random edge for the 3COL example.

3. The prover sends more information that convinces a verifier such as the permutation π in the ISO example, or values to open a commitment in the 3COL example.

Does the verifier ever ask anything that is not random? Do they ever *need* to ask more tricky questions? The answer is No, for any ZK proof there exists an algorithm where the verifier must only ask the simple question. As we have seen in both the ISO and 3COL examples this simple question is simply a random selection out of a set of possible values. It is the provers responsibility to do the hard work. In fact, these very observations helped lead Ben-Or et al. to results that tell us if all languages with interactive proofs can also have zero-knowledge proofs [6].

The work of Ben-Or et al. found that there exists a computational zero-knowledge proof for any interactive proof, or CZK=IP. The caveat here is the assumption of the existence probabilistic encryption schemes, or schemes that use new randomness for each message so that multiple encryptions of the same message look different [5]. It turns out that this assumption is acceptable for most real-world applications.

Additionally, this work also found that under a different assumption there exists a perfect zero-knowledge proof for any interactive proof, or PZK=IP. This is only true under the assumption that you have "hats" or "envelopes for bit commitments" (i.e. the protocol is both perfectly hiding and perfectly binding). Since this is actually impossible to implement digitally, this finding is interesting, but less useful for real-world applications in the digital world.

Lets review what we have learned in this section. There exists an interactive proof for all problems in PSPACE, there exists a computational zero-knowledge proof for all problems in NP assuming the existence of one-way functions, and there exists a computational zero-knowledge proof for all problems in PSPACE assuming the existence of probabilistic encryption.

4 Proofs of Knowledge

A proof of knowledge is a proof that demonstrates knowledge of something without necessarily disclosing the actual knowledge. When no additional knowledge is released, it would be known as a zero-knowledge proof of knowledge. For example, how can I prove I know a password (that defines me) without giving out the password? For example, given a (published) public key, PK, how can I prove to someone else that I know the corresponding secret key, SK?

1. Could I sign a message m that others could verify with PK?

2. Could I decrypt a message m encrypted under PK and send it back to the verifier?

These methods do appear to demonstrate knowledge some knowledge of SK, but we would like a zero-knowledge proof of knowledge and the two methods listed will leak information. In this section we will first give an intuitive definition for proofs of knowledge, then show a few examples that will develop into a more formal definition. Lastly, we will come full circle and demonstrate a proof of knowledge for the question posed in the previous paragraph.

4.1 Intuitive definition

Our intuitive definition, as should not be surprising, sounds much like our definition of zero-knowledge proofs. The exception is that we now have a *validity* property rather than *soundness*.

1. Completeness: if P does know something, it succeeds in convincing the verifier.
2. Validity: if V accepts, the P really did know the value (w/ negl. acceptable error)
3. Zero Knowledgeness: V learns that P knows the value and nothing else.

4.2 Example 1

How would I prove that I know a permutation between two graphs, G and H? Well, we can use the exact same protocol we have been using all along in our ISO examples. Lets take a look at this protocol from the proof of knowledge perspective and examine some properties that will help lead to a more formalized definition of a proof of knowledge. As in the previous examples, the protocol is as follows:

P: "I know a permutation between G and H" ($\pi(G) = H$)

P (G, H, π)	**Message**	V (G, H)
rand. ϕ, $C = \phi(G)$	$C \rightarrow$	
	$\leftarrow$ "G" or "H"	rand. "G" or "H"
	if "G" send ϕ, if "H" send $\phi * \pi^{-1} \rightarrow$	Check $\phi(G) = C$

In the first step the prover sends a random permutation of G, C. Next, the verifier randomly asks to show that either "G" or "H" is isomorphic to C. The prover shows this by providing a mapping between the graph that V selects and C. The protocol is repeated until the verifier is satisfied.

Nothing new here, but lets take a closer look. First, note that no matter what the verifier says ("G" or "H"), if P knows the correct permutation it will be capable of answering. So, P is always capable of answering correctly. What is really interesting is that from two runs on the prover *with the same randomness* where the verifier makes two different selections, the permutation π can be computed. That is, given the same random graph C, if the verifier *were* allowed to ask the prover to answer for "G" and "H" then he could compute π from ϕ and $\phi * \pi^{-1}$! Of course, the verifier is not allowed to ask about both openings in the "real" world, but as a thought experiment, we see that if the prover can answer the verifier's queries with a decent probability, then the prover must know both openings (and thus, the prover must know π.)

Let's keep this in mind and take a look at one more example.

4.3 Example 2

How would I prove that I know a 3-coloring for graph G? Thats right, we can use the exact same protocol we have been using all along in our 3COL examples. Lets take a look

at this protocol one more time from the proof of knowledge perspective keeping in mind the properties we just observed in example one. As in the previous 3COL examples, the protocol is as follows:

P: "I know a 3COL for G"		
P (G, **{nodes, colors}**)	**Message**	V (G)
rand. permute coloring	commit(G) $\rightarrow$	
	$\leftarrow (i, j)$	rand. (i, j)
	$decommit(i)$, $decommit(j) \rightarrow$	Check $color(i) \neq color(j)$ and decommitments valid.

In the first step the prover, randomly permutes the valid coloring that it knows (i.e., change blue to red, red to blue, and green to green). The prover then commits to this coloring of the graph and sends the commitments to the verifier. Next, the verifier asks to reveal the color of random nodes (i, j) corresponding to a valid edge in the graph. The prover sends back the decommitments for these nodes. The verifier checks that the decommitments are valid and that the two colors of the nodes are different. If all checks pass, the verifier accepts this round; otherwise the verifier rejects. The protocol is repeated until the verifier is satisfied.

Again, note that no matter what the verifier says (for (i, j)), if P knows the correct coloring it will be capable of answering. So, P is again always capable of answering correctly. But, can the entire coloring of the graph be *extracted* as the permutation was in the ISO example? Well, given multiple runs on the same graph (with the same randomness used by prover), V can figure out the 3COL for G by guessing a different edge each time! Lets incorporate this *extraction* capability into our definition for proofs of knowledge.

4.4 Developing the Definition

Examples (1) and (2) give the intuition that the prover "knows" the information it claims to know, because she should be able to re-construct this knowledge through repeated runs of the protocol with herself. This introduces the concept of the *Extractor*. Recall that in zero-knowledge, a simulator interacted with a (potentially malicious) verifier to construct a transcript of protocol messages:

$$S^{V^*, rand.\ tape} \rightarrow transcript$$

In ZK, we argue that if the simulator succeeds, then the verifier did not learn anything.

With proofs of knowledge, a similar concept is adopted. An extractor interacts with a (potentially malicious) prover to extract the "value" that P claims to know:

$$E^{P^*, rand.tape} \rightarrow \text{“}value''$$

In a proof of knowledge, we argue that if the extractor succeeds, then the prover did know the value it claimed to know.

In the ISO example this value is a valid permutation between "G" and "H" while in the 3COL example this value is a valid 3-coloring of the G. We will see that extractors allow us to formalize the notion of proofs of knowledge, but first let's review some necessary group theory.

4.5 Group Theory Primer

Basic knowledge of some of terms in group theory will be useful for the next example, and in upcoming lectures. Let's start by picking apart a statement with lots of $fancy$ group theory words. We'll see that it's not to hard to read once we see what is meant by each piece:

"Consider a finite cyclic group G of prime order q with generator g. Let $p = 2q+1$ where p, q are large primes."

A finite group can simply be considered a finite set of numbers (together with a group operator, such as multiplication, and an identity element, such as 1.) Moreover, if g is a generator for group G, then this means that all of the numbers in this group can be found using g (see how below). A prime order q simply means that the group has q elements and q is prime. Note that G is often a subgroup of the multiplicative group Z_p^* of prime order q because this is easy to work with. So, let's see an example of a group by defining the variables in the given phrase with actual numbers:

1. $q = 3$, $p = 2(3) + 1 = 7$
2. $Z_p^* = \{1, 2, 3, 4, 5, 6\}$

A generator gets every element in a group from g^i, $0 \leq i < q$. So, let's use 2 for our generator and see how our group, G, turns out:

1. $2^0 = 1$
2. $2^1 = 2$
3. $2^2 = 4$
4. $2^3 = 1$
5. $2^4 = 2$
6. ...

So, 2 is a generator of the subgroup $\{1, 2, 4\}$. We can see that the group cycles after these three elements. Also, note that any other member of this subgroup can also be its generator (except for one).

Also, the Discrete Logarithm problem deserves a mention here. In addition to factoring, the DL problem is considered hard, i.e. not solvable in *ppt*. Lets define this problem. Given a finite cyclic group G of prime order q with generator g and $t \overset{rand.}{\leftarrow} G$, it is hard to compute $x \in Z_q \; s.t. \; g^x = t$. So, $g, h \in G$, $g^x = h$ is hard to find. This will be used as a hardness assumption for future examples.

4.6 Example 3

We now return to the original question of how to prove knowledge of a password or secret key. We use Schnorr to construct the protocol [7].

P: "I know secret key SK" ($PK = g^x$, $SK = x$)

P (x, g, q, G, PK)	**Message**	V (g, q, G, PK)
rand $r \in Z_q^*$, $t = g^r \in G$	$t \rightarrow$	
	$\leftarrow c$	rand. $c \in Z_q$
	$s = x * c + r (mod\, q) \rightarrow$	Check $g^s = (PK)^c * t$

In the first step, the prover generates a random element, t in the group G using g^r, where r is random. Recall that due to the DL problem, finding r from t is hard. The verifier then chooses a random c. Next the prover replies with $s = xc + r$. If the prover knows x and acted honestly, then the equation $g^s = (PK)^c * t = g^{xc} * g^r = g^{xc+r}$ should check out, demonstrating completeness.

Now, is it possible that the prover can convince the verifier to accept (with a decent probability) and yet the prover does not know x? Given our intuition from previous examples, can we *extract* x from a prover that often succeeds in this protocol? (Of course, the extractor will be given special black-box access to the prover that a real verifier does not have!)

Can an extractor can be constructed for this protocol? Yes, fairly easily. This extractor can run the prover twice on the same randomness. Thus, the prover will use the same value $t = g^r$ both times. The extractor then replies with two different values $c \neq c'$ and obtains the two responses:

$$s = xc + r$$

$$x' = xc' + r$$

Suppose both of these protocol executions would have convinced a real verifier to accept (i.e., they are both correct.) Given these two equations, the "value", $SK = x$ can be derived:

$$\frac{S - S'}{c - c'} = x = SK$$

But, is this protocol also zero-knowledge? Discussion to continue next class...

References

[1] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design. In FOCS 86, pages 174-187, 1986.

[2] O. Goldreich, S. Micali, and A. Wigderson. How to prove all NP statements in zeroknowledge and a methodology of cryptographic protocol design. In A. M. Odlyzko, editor, Advances in Cryptology CRYPTO 86, volume 263 of Lecture Notes in Computer Science, pages 171185. Springer-Verlag, 1987.

[3] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. In STOC 85, pages 291-304, 1985.

[4] Goldwasser, S. and Sipser, M. Private Coins versus Public Coins in Interactive Proof Systems. Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC'86), Berkeley CA, May, pages 59-86, 1986.

[5] Goldwasser, S. and Micali, S. Probabilistic Encryption. Special issue of Journal of Computer and Systems Sciences, Vol. 28, No. 2, pages 270-299, April 1984.

[6] Ben Or M., O. Goldreich , S. Goldwasser, J. Hastad, J. Kilian, S. Micali, and P. Rogaway, "Everything Provable is Provable in Zero-Knowledge", Proc. of CRYPTO '88, Santa Barabara, 1988.

[7] C.P. Schnorr, Efficient identification and signatures for smart cards, in G. Brassard, ed. Advances in Cryptology – Crypto '89, 239-252, Springer-Verlag, 1990.