600.641 Special Topics in Theoretical Cryptography 2/27/07

Lecture 10: More on Proofs of Knowledge

Instructor: Susan Hohenberger *Scribe: Jennifer Lindsay*

1 Announcements

Next Monday, you will receive Problem Set #2, which will be due two weeks from that day. (Actually, due date extended to March 21.)

1.1 And The Turing Award Goes To...

In case you've been hibernating under a rock, this year's Turing Award winner is Frances E. Allen, the first woman ever to receive this award. Ms. Allen, who retired in 2002 after a 45-year career at IBM Research, worked with the National Security Agency on constructing codebreaking systems during the Cold War. She said that although she believed gender did not factor into the decision making process, it was high time a woman was chosen.

1.2 Problem Set #1, Question 4

Last class, we didn't comment on the 4th problem of Problem Set #1, which asked students to give a practical example of a zero-knowledge proof or proof of knowledge. Some students were apparently confused about the distinction between zero-knowledge proofs and proofs of knowledge. For the record, here is the difference:

- In a *Zero-Knowledge Proof*, the prover attempts to convince the verifier that some fact is true (e.g. $G \in$ 3COL or $(G, H) \in$ NISO) without revealing any additional information.
- In a *Proof of Knowledge*, the prover attempts to convince the verifier that it knows some secret information. For example, given the publicly known value $y = g^x$, the prover attempts to convince the verifier that it knows x.

2 Examples of Proofs of Knowledge

Let's begin by developing some intuition through a few simple examples.

Example 2.1 (Schnorr Protocol for Proving Knowledge of DL [Sch91]) *Given a group $G = < g >$ of prime order q, and a public value g^x, the prover must convince the verifier that it knows x.*

2.1 Formal Notation for Proofs of Knowledge

There is a formal symbolic notation, introduced by Camenisch and Stadler [CS97], for compactly representing statements of this proof of knowledge. For the Schnorr protocol, this notation is as follows:

$$PK\{(\alpha) : A = g^{\alpha}\}$$

Here, the PK stands for Proof of Knowledge, as you probably already guessed. The proof statement is enclosed inside curly braces. α is the secret information the prover wishes to show knowledge of, and (A, g, q) is the public knowledge available to both the prover and the verifier. In the formal notation, secret information is represented by Greek letters and public information is represented by Roman letters. However in the actual protocol, the Greek letters are usually replaced with traditional lowercase Roman letters.

The Schnorr protocol (with $A = g^x$) is as follows:

P		**V**
1. Choose $\$ \; t \in \mathbb{Z}_q$ 2. Compute $T = g^t$	$\xrightarrow{T}$	
		1. Choose $\$ \; c \in \mathbb{Z}_q$
	$\xleftarrow{c}$	
	$\xrightarrow{s = xc + t \pmod q}$	
		1. Accept if $g^s = A^c T$

2.2 Proof of Knowledgeness

This proof is obviously complete, because if the prover truly does know x, then

$$A^c T = (g^x)^c g^t = g^{xc+t} = g^s$$

so the verifier's test of acceptance is valid. But is it a proof of knowledge? Well, what is the probability of guessing a value for x and fooling the verifier into accepting the test result? Well, if the verifier is truly choosing its challenge values c at random, the probability of extraction is $\frac{1}{q}$, which is extremely small for q suitably large. If the verifier is still not convinced, she can decrease the probability that the prover doesn't know x by repeating the protocol several times.

But what if the verifier uses some other strategy for choosing its challenge values? For example, if the c is incremented by 1 each time...can the prover exploit this knowledge to its advantage? Yes, and here's how the prover does it. Suppose the prover predicts which c the verifier will use.

1. Choose $\$ \; s \in \mathbb{Z}_q$
2. Compute and send $T = g^s / A^c$ to verifier

So you see, if the prover has advance knowledge of the verifier's next c value, the protocol is neither a proof of knowledge nor sound!

2.3 Special Soundness

The Schnorr protocol we just saw is one instance of what are called *3-stage protocols*, named such because there are three interactive steps:

1. Commit Phase (Schnorr value - T)
2. Challenge Phase (Schnorr value - c)
3. Response Phase (Schnorr value - s)

Is this protocol sound? Well, suppose (as a thought experiment) our prover, when run twice on the same randomness, and given two different challenges, has a decent probability of giving a valid response. That is, (in the thought experiment only) the prover gives the same commit value T and it is able to successfully answer challenges c_1 and c_2, where $c_1 \neq c_2$ – then I claim the prover can now extract x from (T, c_1, s_1) and (T, c_2, s_2), because$s_1 = xc_1 + t$ and $s_2 = xc_2 + t$ implies $x = \frac{s_1 - s_2}{c_1 - c_2}$. Of course, in the real world the prover would not reuse an old commit phase. This notion that from two such transcripts the "knowledge" can be extracted has a special name. It is called *special soundness*, which holds if we are able to compute the secret information from two valid runs with identical commit phases.

2.4 Zero-Knowledgeness

Is the Schnorr protocol zero-knowledge? Well, let's try to construct a simulator. Note that we can't necessarily assume S knows the secret value x, so let's try to build one that doesn't require it. Suppose for the moment that our simulator only needs to simulate the VIEW of the honest prover talking to the honest verifier. (Instead of the honest prover talking to any verifier.)

Our simulator protocol is as follows:

1. Simulator sets the random tape of the verifier
2. Simulator sends T to verifier
3. Verifier responds with challenge c
4. If simulator can answer challenge, do so, then continue from step 1
5. Otherwise, rewind and calculate $\$ s \in \mathbb{Z}_q$ and $T = g^s / A^c$
6. Restart verfier with the same random tape and send the new T value to verifier
7. Verifier responds with same challenge c (this is due to the fact that the *honest* verifier uses only his random tape to choose c; we are currently only simulating for the *honest* verifier!)
8. Simulator successfully answers challenge with s

This way, it takes the simulator no more than $2k$ steps to produce a valid transcript of length k that is indistinguishable from a real conversation between the prover and the verifier. Note that in both cases, the first round is an element selected uniformly at random from G, the second round is an element selected uniformly at random from $\mathbb{Z}_q$, and the third round is a valid response. Thus, the simulated and real VIEWs are perfectly indistinguishable.

Does this simulator strategy still work if the verifier is dishonest, i.e. if the verifier does not randomly chooses the challenge c from $\mathbb{Z}_q$? Suppose the verifier bases c on the value of T it receives from the prover during the commit phase. Notice that the above simulator strategy relies on the fact that in step 6 the verifier produces the same c every time the protocol restarts with the same randomness. If c is based on T, the new T value sent during step 5 may induce the verifier to respond with a new challenge c^*, and the simulator's strategy fails.

Therefore, given this simulator, the Schnorr protocol is *honest-verifier zero-knowledge*, that is, zero-knowledge as long as we assume the verifier is honest. Our next question is, of course, can we modify the protocol to make it *fully* zero-knowledge?

Fortunately, the answer is yes – by adding a commitment scheme. If we require the verifier to initially commit to a challenge value c for the current round, then the verifier cannot base c on the T value from the simulator (because it has not received this value at the time of the commitment to c), and yet the simulator cannot determine the c value from this commitment...or can it? Remember the simulator is infinitely powerful, so we must choose our commitment scheme carefully. If we use the Pedersen scheme, the commitment to c is perfectly hiding, so even an all-powerful prover cannot extract c. The Pedersen scheme is only computationally binding, however – but this is ok, because our verifier is restricted to be probabilistically polynomial time. (On your own, think about why the GM commitments based on quadratic residues would not work ....)

2.5 Proof of Knowledgeness Revisited

Ok, so now our verifier cannot act dishonestly, and this new protocol is zero-knowledge! Hurray! But, wait...is it still a proof of knowledge? In other words, *can we build an extractor with oracle access to the prover which can extract the secret value x*? (As we were able to do before we made our ZK modifications.) We have modified the original protocol by adding a new 1st step, so we cannot simply reuse our old argument. Unfortunately, we run into a problem similar to the one we saw trying to prove zero-knowledge: how can we keep the prover from changing its commit value T when we rewind the extractor? We could add another commitment step, this time for the prover, but this might break our argument for zero-knowledgeness.
At this point, we're really just chasing our tails here. Perhaps for now, we should just assume our verifier is honest and investigate if this is a useful notion.

3 Examples of Honest Verifier Zero-Knowledge Proofs of Knowledge

Example 3.1 (PK{(α,β): A=$g^\alpha h^\beta$}) *Given a group G=< g >=< h > and public knowledge A=$g^x h^y$, the prover must convince the verifier that it knows x and y.*

This protocol, which demonstrates knowledge of what a Pedersen commitment [Ped91] opens to. The proof of this protocol is as follows:

P		V
1. Choose $\$ t_1, t_2 \in \mathbb{Z}_q$ 2. Compute $T = g^{t_1} h^{t_2}$	$\xrightarrow{T}$	
	$\xleftarrow{c}$	1. Choose $\$ c \in \mathbb{Z}_q$
	$\xrightarrow{s_1 = xc + t_1,\ s_2 = yc + t_2}$	1. Accept if $g^{s_1} h^{s_2} = A^c T$

Because this example is very similar to the Schnorr scheme we saw in Section 2, our first hunch might have been to construct two Schnorr protocols both of which open to the same challenge c. However, there exist attacks against this approach.

Also, as we saw with the Schnorr protocol, if we reuse T for two different rounds, we can extract x and y. As a rule of thumb, successful protocols usually contain one commitment value per publicly known value (e.g., A - value).

Example 3.2 (PK{(α): A=g^α and B=h^α}) *Given a group G=< g >=< h > and public knowledge A=g^x and B=h^x, the prover must convince the verifier that it knows x, and also that (g,h,A,B) form a DDH-tuple (i.e., A and B contain the same exponent x).*

This protocol is due to Chaum and Pedersen [CP92] for prime order groups (and due to Fujisaki and Okamoto [FO97] for a modification to composite order groups.) Our first intuition might be to use a simple extension of the Schnorr protocol, like the one below suggested by one of your classmates:

P		V
1. Choose $\$ t \in \mathbb{Z}_q$ 2. Compute $T = g^t h^t$	$\xrightarrow{T}$	
	$\xleftarrow{c}$	1. Choose $\$ c \in \mathbb{Z}_q$
	$\xrightarrow{s = xc + t \pmod q}$	1. Accept if $(gh)^s = (AB)^c T$

Right now, alarm bells should be going off in your head, because this protocol does not follow the "one commitment per value" suggestion we proposed above. But let's see if we can construct a formal argument for why a dishonest prover can cheat this protocol. Define $\bar{B} = h^{\bar{x}}$, and suppose the prover wishes to fool the verifier into believing that $(g, h, A, \bar{B})$ is a DDH-tuple. Since the prover is infinitely powerful, it can compute $h = g^z$. T and c are chosen as outlined in the protocol above, but now the prover must choose an s value that will satisfy the verifier's test of acceptance, in other words, choose s satisfying

$$(g \cdot g^z)^s = (g^x \cdot g^{\bar{x}})^c (g \cdot g^z)^t \Leftrightarrow (1+z)s = (x+\bar{x})c + (1+z)t \Leftrightarrow s = \frac{(x+\bar{x})c + (1+z)t}{(1+z)}$$

Since $x, \bar{x}, c, z$, and t are known by the time the prover needs to send s, the prover can simply use the formula above to calculate a value of s guaranteed to satisfy the verifier's acceptance test every time! Clearly, this protocol is not guaranteeing that $(g, h, A, \bar{B})$ is a DDH-tuple!

A correct protocol for this statement is as follows:

P		**V**
1. Choose $\$ \; t \in \mathbb{Z}_q$ 2. Compute $T_1 = g^t$ and $T_2 = h^t$		
	$\xrightarrow{T_1, T_2}$	
		1. Choose $\$ \; c \in \mathbb{Z}_q$
	$\xleftarrow{c}$	
	$\xrightarrow{s = xc + t \pmod{q}}$	
		1. Accept if $g^s = A^c T_1$ and $h^s = B^c T_2$

This protocol certainly proves that P knows x, but does it also show that (g, h, A, B) form a DDH-tuple? Let's see: suppose again we are attempting to dupe the verifier into accepting $\bar{B} = h^{\bar{x}}$. Select $T_1 = g^t$ as before and $T_2 = h^{\bar{t}}$. All other values remain the same. During the acceptance test step, we have

$$A^c T_1 = g^{xc} g^t \text{ and } \bar{B}^c T_2 = g^{\bar{x}c} g^{\bar{t}}$$

which only holds if

$$xc + t = \bar{x}c + \bar{t} \Leftrightarrow c = \frac{t - \bar{t}}{\bar{x} - x}$$

By this step, values for $t, \bar{t}, x$, and $\bar{x}$ have been fixed, so the only way the verifier will accept is if the value of c it chooses to send to the prover satisfies the equation above. The probability of choosing this specific value is negligible, so the chance that the prover can cheat the DDH-tuple portion of this protocol is also negligible. Therefore, this protocol is simultaneously a proof of knowledge of x and a proof that (g, h, A, B) is a DDH-tuple.

Example 3.3 (PK{(α,β): A=g^α or B=h^β}) *Given a group $G=< g >=< h >$ and public knowledge A and B, the prover must convince the verifier that either it knows x and that $A=g^x$, or it knows y and that $B=h^y$, but the verifier does not learn which.*

This protocol is due to Cramer, Damgård and Schoenmakers [CrS94]. Without loss of generality, we will assume in the following proof that $A = g^x$ and the prover knows x.

P		**V**
1. Construct "fake" proof for y: choose $\$ c_2, s_2 \in \mathbb{Z}_q$ and compute $T_2 = h^{s_2}/B^{c_2}$ 2. Choose $\$ t_1 \in \mathbb{Z}_q$ 3. Compute $T_1 = g^{t_1}$	$\xrightarrow{T_1, T_2}$	
		1. Choose $\$ c \in \mathbb{Z}_q$
	$\xleftarrow{c}$	
1. Compute $c_1 = c \oplus c_2$		
	$\xrightarrow{c_1, s_1 = xc_1 + t_1, c_2, s_2}$	
		1. Accept if $g^{s_1} = A^{c_1}T_1$, $h^{s_2} = B^{c_2}T_2$ and $c = c_1 \oplus c_2$

When the verifier performs these three checks, she will have no idea which of the two proofs is the true one. However, at least one of these proofs must be true, because without prior knowledge of the challenge c the prover cannot fake both. We can generalize this idea to prove k out of n statements.

This type of proof is called a *proof of partial knowledge*, because the verifier gains knowledge about only part of the proof system. We will see more of these next lecture.

3.1 More Proof of Knowledge Examples

If you haven't had your fill of proofs of knowledge yet, here are two more examples from the literature you can peruse on your own:

- Proof that a commitment opens to the product of two other committed values [CM99]
- Proof that a committed value lies in a given integer interval [Bou00]

References

[Bou00] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT '00*, pages 431–444, 2000.

[CM99] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number n is the product of two safe primes. In *EUROCRYPT '99*, pages 107–122, 1999.

[CP92] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *CRYPTO '92*, volume 740 of LNCS, pages 89–105, 1992.

[CrS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO '94*, pages 174–187, 1994.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *CRYPTO '97*, pages 410–424, 1997.

[FO97] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO '97*, pages 16–30, 1997.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91*, pages 129–140, 1991.

[Sch91] Claus Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.