Advanced Topics in Cryptography

Lecture 5

Benny Pinkas

Based on slides of Yehuda Lindell

Zero Knowledge

- Prover P, verifier V, language L
- P proves that $x \in L$ without revealing anything
  - **Completeness:** **V** always accepts when honest **P** and **V** interact
  - **Soundness:** **V** accepts with negligible probability when $\mathbf{x} \notin \mathbf{L}$, for any $\mathbf{P}^*$
    - Computational soundness: only holds when $\mathbf{P}^*$ is polynomial-time
- **Zero-knowledge:**
  - There exists a simulator **S** such that **S**(**x**) is indistinguishable from a real proof execution

ZK Proof of Knowledge

- Prover P, verifier V, relation R
- P proves that it knows a witness w for which $(x,w) \in R$ without revealing anything
  - The proof is zero knowledge as before
  - There exists an extractor **K** that can obtain from any $\mathbf{P}^*$,a **w** such that $(\mathbf{x},\mathbf{w}) \in \mathbf{R}$, with the same probability that $\mathbf{P}^*$ convinces **V.**

Sigma Protocols

- A way to obtain efficient zero knowledge
  - Many general tools
  - Many interesting languages can be proven with a sigma protocol

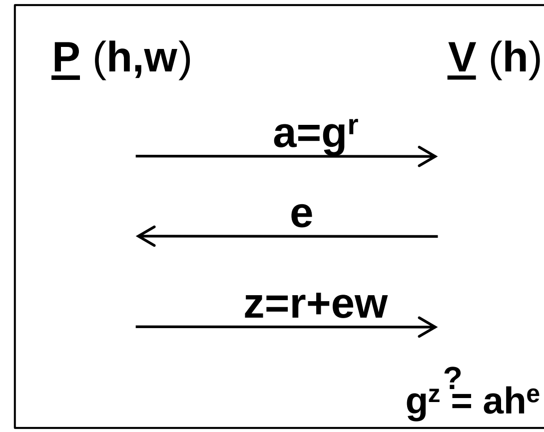
Reminder: Schnorr DLOG

- Let G be a group of order q, with generator g
- P and V have input $h \in G$. P has w such that $g^w = h$
- P proves that to V that it knows $DLOG_g(h)$
  - **P** chooses a random **r** and sends $\mathbf{a = g^r}$ to **V**
  - **V** sends **P** a random $\mathbf{e} \in \{\mathbf{0,1}\}^{\mathbf{t}}$
  - **P** sends $\mathbf{z = r + ew}$ mod **q** to **V**
  - **V** checks that $\mathbf{g^z = ah^e}$

- Completeness
  - $\mathbf{g^z = g^{r+ew} = g^r (g^w)^e = ah^e}$

Schnorr's Protocol

- Proof of knowledge
  - Assume **P** can answer two queries **e** and **e′** for the same **a**
  - Then, it holds that $\mathbf{g^z = ah^e}$, $\mathbf{g^{z'}=ah^{e'}}$
  - Thus, $\mathbf{g^z h^{-e} = g^{z'} h^{-e'}}$ and $\mathbf{g^{z-z'}=h^{e-e'}}$
  - Therefore $\mathbf{h = g^{(z-z')/(e-e')}}$
  - That is: DLOG$\mathbf{g}$($\mathbf{h}$) = ($\mathbf{z-z'}$)/($\mathbf{e-e'}$)
- Conclusion:
  - If **P** can answer with probability greater than $\mathbf{1/2^t}$, then it must know the dlog



Schnorr's Protocol

- What about zero knowledge? This does not seem easy.

- But ZK holds if the verifier sends a random challenge e
- This property is called "Honest-verifier zero knowledge"
  - The simulation:
  - Choose a random **z** and **e**, and compute $\mathbf{a = g^z h^{-e}}$
  - Clearly, (**a,e,z**) have the same distribution as in a real run, and $\mathbf{g^z = a h^e}$

- This is not a very strong guarantee, but we will see that it yields efficient general ZK.

Definitions

- Sigma protocol template
  - **Common input: P** and **V** both have **x**
  - **Private input: P** has **w** such that (**x**,**w**)$\in$**R**

  - **Protocol:**
    - **P** sends a message **a**
    - **V** sends a <u>random</u> **t**-bit string **e**
    - **P** sends a reply **z**
    - **V** accepts based solely on (**x**,**a**,**e**,**z**)

Definitions

- Completeness: as usual
- Special soundness:
  - There exists an algorithm **A** that given any **x** and pair of transcripts (**a**,**e**,**z**),(**a**,**e′**,**z′**) with **e**≠**e′** outputs **w** s.t. (**x**,**w**)∈**R**
- Special honest-verifier ZK
  - There exists an $\mathbf{M_V}$ that given any **x** and **e** outputs (**a**,**e**,**z**) which is distributed exactly like a real execution where **V** sends **e**

Tools for Sigma Protocols

- Last lecture: Prove compound statements
  - AND, OR, subset
- ZK from sigma protocols
  - Can first make a compound sigma protocol and then compile it
- ZKPOK from sigma protocols

ZK from Sigma Protocols

- A tool: commitment schemes
- Enables to commit to a chosen value while keeping it secret, with the ability to reveal the committed value later.
- A commitment has two properties:
  - Binding: After sending the commitment, it is impossible for the committing party to change the committed value.
  - Hiding: By observing the commitment, it is impossible to learn what is the committed value. (Therefore the commitment process must be probabilistic.)
- It is possible to have unconditional security for any one of these properties, but not for both.

Pedersen Commitments

- Highly efficient perfectly-hiding commitments (two exponentiations for string commit)
  - **Parameters:** generator **g**, order **q**
  - **Commit protocol** (commit to **x**):
    - Receiver chooses random **k** and sends $\mathbf{h=g^k}$
    - Sender sends $\mathbf{c=g^r h^x}$, for random **r**
  - **Unconditionally hiding:**
    - For every **x,y** there exist **r,s** s.t. $\mathbf{r+kx = s+ky \bmod q}$
  - **Binding:**
    - If sender can open commitment in two ways, i.e. find $(\mathbf{x,r}),(\mathbf{y,s})$ s.t. $\mathbf{g^r h^x=g^s h^y}$, then $\mathbf{k = (r\text{-}s)/(y\text{-}x) \bmod q}$

ZK from Sigma Protocols

- The basic idea
  - Have **V** first commit to its challenge **e** using a perfectly-hiding commitment
- The protocol
  - **P** sends the first message α of the commit protocol, (e.g., including g,h in the case of Pedersen commitments).
  - **V** sends a commitment c=$\mathbf{Com}_{\alpha}$(**e**;**r**)
  - **P** sends a message **a**
  - **V** sends (**e**,**r**)
  - **P** checks that c=$\mathbf{Com}_{\alpha}$(**e**;**r**) and if this holds it sends a reply **z**
  - **V** accepts based on (**x**,**a**,**e**,**z**)

ZK from Sigma Protocols

- Soundness:
  - The perfectly hiding commitment reveals nothing about **e** and so soundness is preserved

- Zero knowledge
  - In order to simulate:
    - Receive a commitment from **V**.
    - Have the Sigma simulator generate **e′** and **a′**. Send a' to **V**.
    - Receive **V**'s decommitment to **e**.
    - Run Sigma protocol simulator again with **e**. Receive corresponding **a**.
    - Rewind **V** and send it **a**. If **V** does not decommit to **e** then abort.
    - Conclude by sending **z**
  - Analysis…

ZK from Sigma Protocols

- Question
  - If computational soundness suffices, can we use a computationally-hiding commitment scheme?

- No:
  - We should prove that cheating in the proof involves distinguishing between commitments to different values

  - Therefore the proof should receive a random commitment, and see if P^* can cheat
    - The reduction fails because we only know if P^* cheated after we opened the commitment

Efficiency of ZK

- Using Pedersen commitments, the entire DLOG proof costs only 5 additional group exponentiations
  - In Elliptic curve groups this is very little

ZKPOK from Sigma Protocols

- Is the previous protocol a proof of knowledge?
  - It seems not to be
  - The extractor for the Sigma protocol needs to obtain two transcripts with the same **a** and different **e**
    - Nothing prevents the prover from choosing its first message **a** differently for every commitment string.
    - In this protocol the prover sees a commitment to **e** before sending **a**.
    - Therefore if the extractor (playing the role of the verifier) changes **e**, and therefore sends a different commitment, the prover changes **a**, and extraction is impossible.

ZKPOK from Sigma Protocols

- Solution: use a trapdoor (equivocal) commitment scheme
  - That is, a commitment that given a trapdoor, it is possible to open it to any value.

- Pedersen has this property – given the discrete log k of **h**, it is possible decommit to any value
  - Suppose that you know the discrete log k of h.
  - Commit to **x**: $\mathbf{c = g^r h^x}$
  - To decommit to **y**, find **s** such that **r+kx = s+ky**
  - This is easy if **k** is known: compute **s = r+k(x-y) mod q**

ZKPOK from Sigma Protocols

- The basic idea
  - Have **V** first commit to its challenge **e** using a perfectly-hiding **trapdoor (equivocal)** commitment
- The protocol (as before, but the commitment is equivocal)
  - **P** sends the first message α of the commit protocol (which includes h in the case of Pedersen's commitment).
  - **V** sends a commitment $c=\mathbf{Com}_{\alpha}(\mathbf{e};\mathbf{r})$
  - **P** sends a message **a**
  - **V** sends (**e**,**r**)
  - **P** checks that $c=\mathbf{Com}_{\alpha}(\mathbf{e};\mathbf{r})$ and if this holds sends the **trapdoor** for the commitment and **z**
  - **V** accepts if the **trapdoor** is correct and (**x**,**a**,**e**,**z**) is accepting

ZKPOK from Sigma Protocols

P (x,w) **V** (x)

$h=g^k$, with random k →

← $c=g^r h^e$

Sigma msg **a** →

← **(e,r)**

Verify $c=g^r h^e$

(k,z) →

Verify $h=g^k$

Verify (a,e,z)

ZKPOK from Sigma Protocols

- Why does this help?
  - **Zero-knowledge** remains the same
  - **Extraction:** after verifying the proof once, the extractor obtains **k** and can rewind back to the decommitment of **c** and send any (**e′**,**r′**)

- Efficiency:
  - Just 6 exponentiations (very little)

ZK and Sigma Protocols

- We typically want zero knowledge, so why bother with sigma protocols?
  - There are many useful general transformations
    - E.g., parallel composition, compound statements
    - The ZK and ZKPOK transformations can be applied on top of the above, so obtain transformed ZK
  - It is **much harder** to prove ZK than Sigma
    - ZK – distributions and simulation
    - Sigma: only HVZK and special soundness

Using Sigma Protocols and ZK

- Prove that the El Gamal encryption (u,v) under public-key (g,h) is to the value m
  - By encryption definition $\mathbf{u=g^r}$, $\mathbf{v=h^r \cdot m}$
  - Thus (**g,h,u,v/m**) is a DH tuple
  - So, given (**g,h,u,v,m**), just prove that (**g,h,u,v/m**) is a DH tuple