Session 4:
Efficient Zero Knowledge

Yehuda Lindell

Bar-Ilan University

Proof Systems

- **Completeness**: can convince of a true statement
- **Soundness**: cannot convince for a false statement
- **Classic proofs:**
  - Written by hand; non-interactive
- **Interactive proofs:**
  - Prover and verifier interact
  - Adds a lot of power (NP vs PSPACE)

Graph Non-Isomorphism

- **P claims that and are not isomorphic**
- **Verifier step**
  - Chooses a random bit
  - Computes as a random permutation of
  - Sends to prover P
- **Prover step**
  - Find (inefficiently) the bit b such that
  - Send to V

Graph Non-Isomorphism

- **Completeness:** easy
- **Soundness:**
  - If the graphs are isomorphic, then a random permutation of G_0 has **the same distribution** as a random permutation of G_1
  - **P** cannot know which bit V started with, and so is right with probability at most ½
  - Repeating n times reduces the cheating probability to 2^{-n}

Zero Knowledge

- **Prover P, verifier V, language L, statement x**
- **P proves that $x \in L$ without revealing anything but that fact**
  - **Completeness:** as before
  - **Soundness: V** accepts with negligible probability when $x \notin L$, for any P^*
    - Computational soundness: when P^* is polynomial-time
- **Zero-knowledge:**
  - For every V^* there exists a simulator S such that $S(x)$ outputs a view indistinguishable from V^*'s view in a real execution with P

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 $\boldsymbol{K}$ that obtains $\boldsymbol{w}$ from any $\boldsymbol{P}^*$ where $(\boldsymbol{x},\boldsymbol{w}) \in \boldsymbol{R}$ with the same probability that $\boldsymbol{P}^*$ convinces $\boldsymbol{V}$
- **Equivalently:**
  - The protocol securely computes the functionality $\boldsymbol{f}_{\boldsymbol{zk}}((\boldsymbol{x},\boldsymbol{w}),\boldsymbol{x}) = (\boldsymbol{\lambda}, \boldsymbol{R}(\boldsymbol{x},\boldsymbol{w}))$

Zero Knowledge

- **An amazing concept; everything can be proven in zero knowledge**
- **Central to fundamental feasibility results of cryptography (e.g., GMW)**
- **But, can it be efficient?**
  - It seems that zero-knowledge protocols for "interesting languages" are complicated and expensive

Sigma Protocols

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

An Example – Schnorr DLOG

- Let $\mathbb{G}$ be a group of order q, with generator g
- **P** and **V** have input $h = g^w$, **P** has w
- **P** proves that to **V** that it knows w
  - **P** chooses a random $r \leftarrow \mathbb{Z}_q$ and sends $a = g^r$ to **V**
  - **V** sends **P** a random $e \in \{0,1\}^t$
  - **P** sends $z = r + ew \bmod q$ to **V**
  - **V** checks that $g^z = a \cdot h^e$
- **Completeness**
  - Follows since $g^z = g^{r+ew} = g^r \cdot (g^w)^e = a \cdot h^e$

Schnorr's Protocol

- **Proof of knowledge**
  - Assume **P** can answer two queries e_1 and e_2 for the same first message a
  - Then, we have $g^{z_1} = a \cdot h^{e_1}$ and $g^{z_2} = a \cdot h^{e_2}$
  - Thus, $a = g^{z_1} \cdot h^{-e_1} = g^{z_2} \cdot h^{-e_2}$ and so $g^{z_1 - z_2} = h^{e_1 - e_2}$
  - Therefore $DLOG_g(h) = (z_1 - z_2)(e_1 - e_2)^{-1} \bmod q$
  - Since are all known from the transcripts, this can be computed
- **Conclusion:**
  - If **P** can answer with probability greater than , then it must know the dlog

$$\begin{array}{lcr}
\underline{\mathbf{P}}\ (\boldsymbol{h}, \boldsymbol{w}) & & \underline{\mathbf{V}}\ (\boldsymbol{h}) \\
& \xrightarrow{\boldsymbol{a} = \boldsymbol{gr}} & \\
& \xleftarrow{\boldsymbol{e}} & \\
& \xrightarrow{\boldsymbol{z} = \boldsymbol{r} + \boldsymbol{ew}} & \\
& & \boldsymbol{g}^{\boldsymbol{z}} \stackrel{?}{=} \boldsymbol{ah}^{\boldsymbol{e}}
\end{array}$$

Schnorr's Protocol

- **What about zero knowledge? Seems not…**
- **Honest-verifier zero knowledge**
  - Choose a random z and e, and compute $a = g^z \cdot h^{-e}$
  - Observe that (a, e, z) chosen this way has the same distribution as when V chooses e randomly
    - In particular, $g^z = a \cdot h^e$
- **This is not very strong, 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 random **t**-bit string **e**
    - **P** sends a reply **z**
    - **V** accepts based solely on (**x,a,e,z**)

Definitions

- **Completeness: as usual**
- **<u>Special</u> 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**
- **<u>Special</u> honest-verifier ZK**
  - There exists an **M** that given **x** and **e** outputs (**a,e,z**) which is distributed exactly like a real execution where **V** sends **e**

Sigma Protocol DH Tuple

- **Relation R of Diffie-Hellman tuples**
  - ($\mathbf{g,h,u,v}$) $\in\mathbf{R}$ iff exists $\mathbf{w}$ s.t. $\mathbf{u=g^w}$ and $\mathbf{v = h^w}$
  - Useful in many protocols
- **Protocol**
  - **P** chooses a random **r** and sends $\mathbf{a=g^r}$, $\mathbf{b=h^r}$
  - **V** sends a random **e**
  - **P** sends $\mathbf{z=r+ew}$ mod **q**
  - **V** checks that $\mathbf{g^z=au^e}$, $\mathbf{h^z=bv^e}$

Sigma Protocol DH Tuple

- **Completeness**: as in DLOG
- **Special soundness:**
  - Given **(a,b,e,z)**,**(a,b,e′,z′)**, we have $g^z=au^e$, $g^{z'}=au^{e'}$, $h^z=bv^e$, $h^{z'}=bv^{e'}$ and so like in DLOG on both
    - $w = (z-z')(e-e')$
- **Special HVZK**
  - Given **(g,h,u,v)** and **e**, choose random **z** and compute
    - $a = g^z u^{-e}$
    - $b = h^z v^{-e}$

$$
\begin{array}{lcr}
\underline{P}\ ((g,h,u,v),w) & & \underline{V} \\
 & \xrightarrow{a=g^r,\ b=h^r} & \\
 & \xleftarrow{\ \ e\ \ } & \\
 & \xrightarrow{z=r+ew} & \\
 & & g^z \overset{?}{=} au^e \\
 & & h^z = bv^e
\end{array}
$$

Basic Properties

- **Any sigma protocol is an interactive proof with soundness error 2^{-t}**
- **Properties of sigma protocols are invariant under parallel composition**
- **Any sigma protocol is a proof of knowledge with error 2^{-t}**
  – The difference between the probability that $\mathbf{P}^*$ convinces **V** and the probability that **K** obtains a witness is at most $\mathbf{2^{-t}}$

Tools for Sigma Protocols

- **Prove compound statements**
  - AND, OR, subset
  - Can be done efficiently (won’t see here)
- **ZK from sigma protocols**
  - Can first make a compound sigma protocol and then compile it
- **ZKPOK from sigma protocols**

ZK from Sigma: Preliminaries

- **Commitment schemes:**
  - **Binding**: after the commitment phase, the committer cannot change the value
  - **Hiding**: the receiver does not know anything about the commitment
- **Variants**
  - Perfect and computational binding
  - Perfect and computational hiding
  - Cannot have both perfect binding and hiding

Perfectly-Binding Commitments

- **The ElGamal usage in Blum's coin tossing is a perfectly-binding commitment**
  - $\mathrm{Com}(m) = (h = g^r, u = g^s, v = h^s \cdot m)$ for $m \in \mathbb{G}$
  - **Perfect binding:** the values (h, u, v) fully define m
    - There exists a single pair (r, s) so that $h = g^r, u = g^s$ and m is fully defined by $\frac{v}{u^r}$
  - **Computational hiding:** for every $m, m' \in \mathbb{G}$, $\{\mathrm{Com}(m)\} \approx \{\mathrm{Com}(m')\}$

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 1st message α of the commit protocol
  - **V** sends a commitment c=$\mathbf{Com}_{\alpha}(\mathbf{e;r})$
  - **P** sends a message **a**
  - **V** sends (**e,r**)
  - **P** checks that c=$\mathbf{Com}_{\alpha}(\mathbf{e;r})$ and if yes 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:
    - Send **a′** generated by the simulator, for a random **e′**
    - Receiver **V**'s decommitment to **e**
    - Run the simulator again with **e**, rewind **V** and send **a**
      - Repeat until **V** decommits to **e** again
    - Conclude by sending **z**
  - Analysis…

Pedersen Commitments

- **Highly efficient perfectly-hiding commitments**
  - **Parameters:** generator $\boldsymbol{g}$, order $\boldsymbol{q}$
  - **Commit protocol** (commit to $x \in \mathbb{Z}_q$):
    - Receiver chooses random $k \leftarrow \mathbb{Z}_q$ and sends $h = g^k$
    - Sender sends $c = g^r \cdot h^x$, for a random $r \leftarrow \mathbb{Z}_q$
  - **Perfect hiding:**
    - For every $x, y \in \mathbb{Z}_q$ there exist $r, s \in \mathbb{Z}_q$ such that $r + kx = s + ky \bmod q$
  - **Computational binding:**
    - If can find $(x, r), (y, s)$ such that $g^r \cdot h^x = g^s \cdot h^y$ then can compute $k = DLOG_g(h) = {}^{r-s}/_{y-x} \bmod q$

Efficiency of ZK

- **Using Pedersen commitments, this costs only 5 additional group exponentiations**
  – This is very efficient

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**
  - The prover may choose its first message **a** differently for every commitment string, so if the extractor changes **e**, the prover changes **a**

ZKPOK from Sigma Protocols

- **Solution: use a trapdoor (equivocal) commitment scheme**
  - Given a trapdoor, it is possible to open the commitment to any value
- **Pedersen has this property, and the previous protocol can be modified only slightly to get a proof of knowledge**

ZK and Sigma Protocols

- **We typically want zero knowledge, so why bother with sigma protocols?**
  - We have 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 oundness

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

- **Database of ElGamal(K_i),E_{Ki}(T_i)**
  - Can release $\mathbf{T_i}$ without revealing anything about $\mathbf{T_j}$ for $\mathbf{j \neq I}$

Non-Interactive ZK (ROM)

- **The Fiat-Shamir paradigm**
  - To prove a statement **x**
  - Generate **a**, compute **e=H**(**a,x**), compute **z**
  - Send (**a,e,z**)
- **Properties:**
  - **Soundness:** follows from random oracle property
  - **Zero knowledge:** same
  - Can achieve simulation-soundness (non malleability) by including unique **sid** in **H**

Summary

- **Efficient zero knowledge is very important in secure computation protocols**
  - Using sigma protocols, we can get very efficient ZK
- **Sigma protocols are very useful:**
  - Efficient ZK
  - Efficient ZKPOK
  - Efficient NIZK in the random oracle model
  - Many other applications as well…