



# Session 5: Sigma Protocols and Zero-Knowledge

Yehuda Lindell  
Bar-Ilan University

# 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  $x \notin L$ , for any  $P^*$ 
    - Computational soundness: only holds when  $P^*$  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 obtains  $w$  such that  $(x, w) \in R$  from any  $P^*$  with the same probability that  $P^*$  convinces  $V$
- ▶ **Equivalently:**
  - The protocol securely computes the functionality

$$f_{zk}((x, w), x) = (-, R(x, w))$$

# 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
- ▶ Zero knowledge is often avoided at significant cost

# Sigma Protocols



Bar-Ilan University  
Dept. of Computer Science

- ▶ **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  $G$  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  $\text{DLOG}_g(h)$ 
  - $P$  chooses a random  $r$  and sends  $a = g^r$  to  $V$
  - $V$  sends  $P$  a random  $e \in \{0, 1\}^t$
  - $P$  sends  $z = r + ew \pmod q$  to  $V$
  - $V$  checks that  $g^z = ah^e$
- ▶ **Completeness**
  - $g^z = g^{r+ew} = g^r(g^w)^e = ah^e$

# Schnorr's Protocol



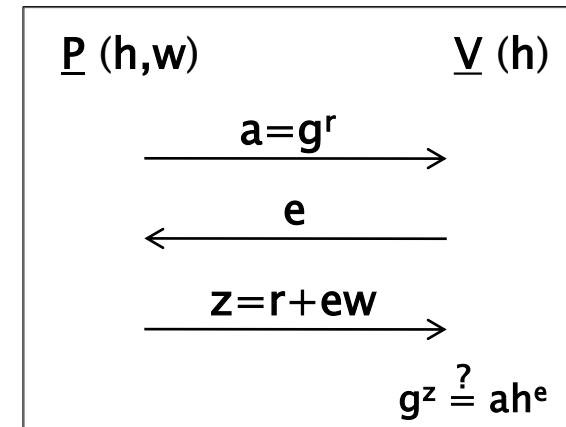
Bar-Ilan University  
Dept. of Computer Science

## ► Proof of knowledge

- Assume  $P$  can answer two queries  $e$  and  $e'$  for the same  $a$
- Then, have  $g^z = ah^e$ ,  $g^{z'} = ah^{e'}$
- Thus,  $g^z h^{-e} = g^{z'} h^{-e'}$  and  $g^{z-z'} = h^{e-e'}$
- Therefore  $h = g^{(z-z')/(e-e')}$
- That is:  $DLOG_g(h) = (z-z')/(e-e')$

## ► Conclusion:

- If  $P$  can answer with probability greater than  $1/2^t$ , then it must know the dlog



# Schnorr's Protocol

- ▶ What about zero knowledge? Seems not...
- ▶ Honest-verifier zero knowledge
  - Choose a random  $z$  and  $e$ , and compute  $a = g^z h^{-e}$
  - Clearly,  $(a, e, z)$  have same distribution, and  $g^z = ah^e$
- ▶ This is not very strong, but we will see that it yields efficient general ZK



# Definitions

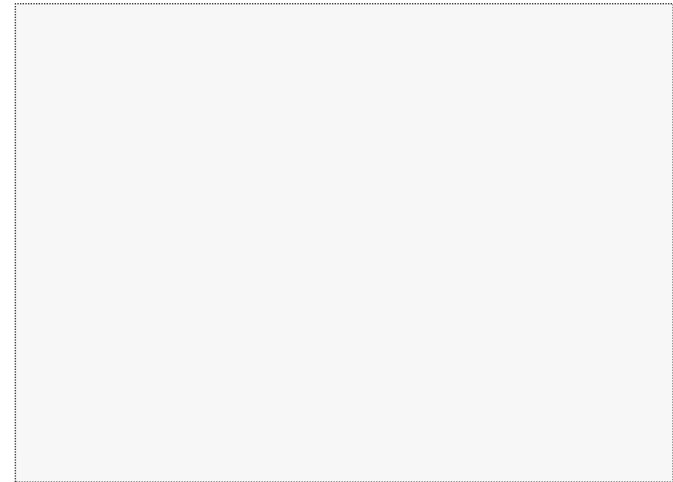


Bar-Ilan University  
Dept. of Computer Science

- ▶ **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
- ▶ **Special soundness:**
  - There exists an algorithm **A** that given any **x** and pair of transcripts  $(a, e, z), (a, e', z')$  with  $e \neq e'$  outputs **w** s.t.  $(x, w) \in R$
- ▶ **Special 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**

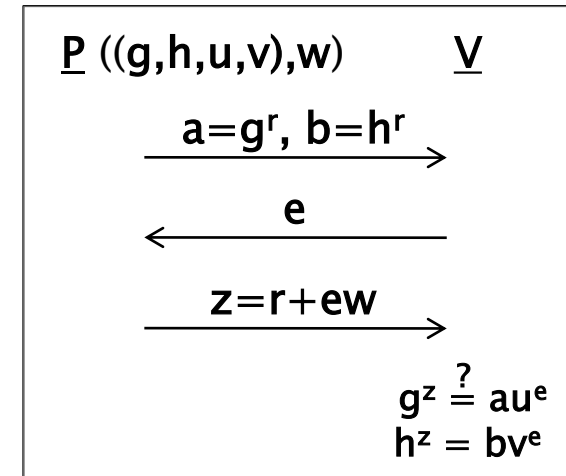


# Sigma Protocol DH Tuple

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

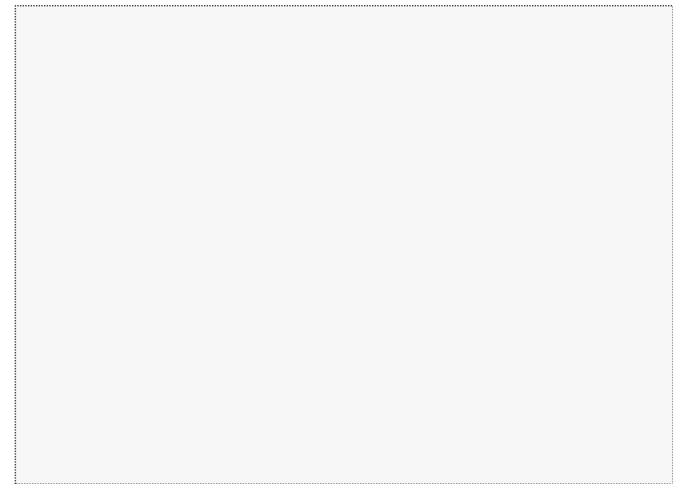
# Sigma Protocol DH Tuple

- ▶ **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
    - $a = g^z u^{-e}$
    - $b = h^z v^{-e}$



# 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  $P^*$  convinces  $V$  and the probability that  $K$  obtains a witness is at most  $2^{-t}$





# Tools for Sigma Protocols

- ▶ **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**

# AND of Sigma Protocols

- ▶ **To prove the AND of multiple statements**
  - Run all in parallel
  - Can use the same verifier challenge  $e$  in all
  
- ▶ **Sometimes it's possible to do better than this**
  - Statements can be batched
  - E.g. proving that many tuples are DDH can be done in much less time than running all
    - Batch all into one tuple and prove

# OR of Sigma Protocols



Bar-Ilan University  
Dept. of Computer Science

- ▶ **This is more complicated**
  - Given two statements and two appropriate Sigma protocols, wish to prove that at least one is true, without revealing which
- ▶ **The solution – an ingenious idea from [CDS]**
  - Using the simulator, if  $e$  is known ahead of time it is possible to cheat
  - We construct a protocol where the prover can cheat in one out of the two proofs



# OR of Sigma Protocols

- ▶ **The template for  $x_0$  or  $x_1$ :**
  - P sends two first messages  $(a_0, a_1)$
  - V sends a single challenge  $e$
  - P replies with
    - Two challenges  $e_0, e_1$  s.t.  $e_0 \oplus e_1 = e$
    - Two final messages  $z_0, z_1$
  - V accepts if  $e_0 \oplus e_1 = e$  and  $(a_0, e_0, z_0), (a_1, e_1, z_1)$  are both accepting
  
- ▶ **How does this work?**

# OR of Sigma Protocols

- ▶ **P sends two first messages  $(a_0, a_1)$** 
  - P has a witness for  $x_0$  (and not for  $x_1$ )
  - P chooses a random  $e_1$  and runs SIM to get  $(a_1, e_1, z_1)$
  - P sends  $(a_0, a_1)$
- ▶ **V sends a single challenge  $e$**
- ▶ **P replies with  $e_0, e_1$  s.t.  $e_0 \oplus e_1 = e$  and with  $z_0, z_1$** 
  - P already has  $z_1$  and can compute  $z_0$  using the witness
- ▶ **Soundness**
  - P doesn't know a witness for  $x_1$ , so can only answer for a single  $e_1$
  - This means that  $e$  defines a single challenge  $e_0$ , like in a regular proof

# OR of Sigma Protocols

## ► Special soundness

- Relative to first message  $(a_0, a_1)$ , and two different  $e, e'$ , at least one of  $e_0 \neq e'_0$  or  $e_1 \neq e'_1$  (because  $e_0 \oplus e_1 = e$  and  $e'_0 \oplus e'_1 = e'$ )
- Thus, we will obtain two different continuations for at least one of the statements

## ► Honest verifier ZK

- Can choose both  $e_0, e_1$ , so no problem

## ► Note: can carry out OR of different statements using different protocols

# OR of Many Statements

- ▶ **Prove  $k$  out of  $n$  statements  $x_1, \dots, x_n$** 
  - $A$  = set of indices that prover knows; others  $B$
  - Use secret sharing with threshold  $n-k$
  - Field elements  $\alpha_1, \dots, \alpha_n$ , polynomial  $f$  with free coefficient  $s$
  - Share of  $s$  for party  $P_i$ :  $f(\alpha_i)$
- ▶ **Prover**
  - For every  $i \in B$ , prover generates  $(a_i, e_i, z_i)$  using SIM
  - For every  $j \in A$ , prover generates  $a_j$  as in protocol
  - Prover sends  $(a_1, \dots, a_n)$

# OR of Many Statements

- ▶ **Prover** sent  $(a_1, \dots, a_n)$
- ▶ **Verifier** sends a random field element  $e \in \mathcal{F}$
- ▶ **Prover** finds the polynomial  $f$  of degree  $n-k$  passing through all  $(\alpha_i, e_i)$  and  $(0, e)$  (for  $i \in B$ )
  - The prover computes  $e_j = f(\alpha_j)$  for every  $j \in A$
  - The prover computes  $z_j$  as in the protocol, based on transcript  $a_j, e_j$
- ▶ Soundness follows because there are  $|\mathcal{F}|$  possible vectors and the prover can only answer one

# General Compound Statements



Bar-Ilan University  
Dept. of Computer Science

- ▶ This can be generalized to any monotone formula (meaning it contains AND/OR but no negations)
  - See Cramer, Damgård, Schoenmakers, Proofs of partial knowledge and simplified design of witness hiding protocols, CRYPTO'94.

# 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 1<sup>st</sup> message  $\alpha$  of the commit protocol
- $V$  sends a commitment  $c = \text{Com}_\alpha(e; r)$
- $P$  sends a message  $a$
- $V$  sends  $(e, r)$
- $P$  checks that  $c = \text{Com}_\alpha(e; r)$  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...



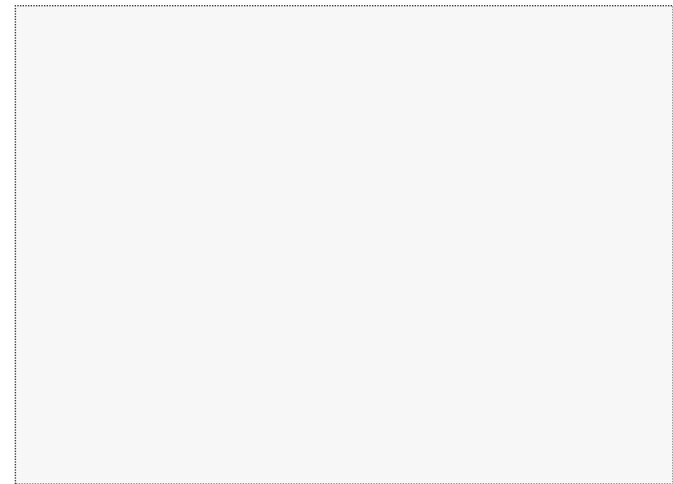
# ZK from Sigma Protocols

## ▶ Question

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

## ▶ No:

- Try to prove that cheating in the proof involves distinguishing commitments
- 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



# Pedersen Commitments

- ▶ **Highly efficient perfectly-hiding commitments** (2 exponentiations for string commit)
  - **Parameters:** generator  $g$ , order  $q$
  - **Commit protocol** (commit to  $x$ ):
    - Receiver chooses random  $z$  and sends  $h=g^z$
    - Sender sends  $c=g^r h^x$ , for random  $r$
  - **Hiding:**
    - For every  $x,y$  there exist  $r,s$  s.t.  $r+kx = s+ky \bmod q$
  - **Binding:**
    - If can find  $(x,r),(y,s)$  s.t.  $g^r h^x = g^s h^y$ , then  $k = (r-s)/(y-x) \bmod q$

# Efficiency of ZK



Bar-Ilan University  
Dept. of Computer Science

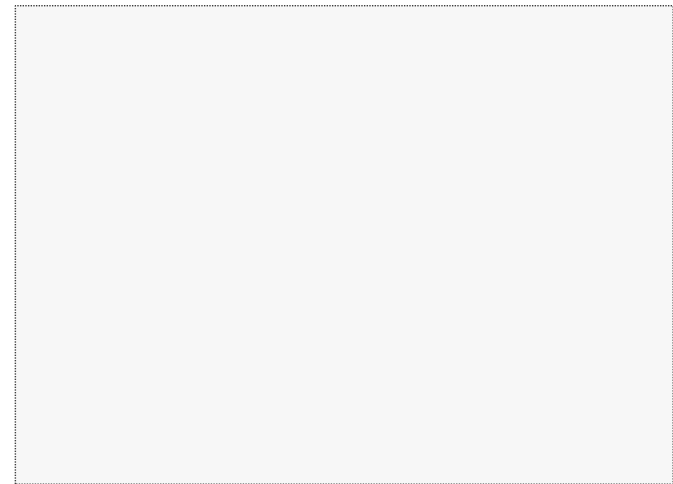
- ▶ **Using Pedersen commitments, this costs only 5 additional group exponentiations**
  - In Elliptic curve groups this is very little

# ZKPOK from Sigma Protocols



Bar-Ilan University  
Dept. of Computer Science

- ▶ 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 – given the discrete log  $k$  of  $h$ , can decommit to any value**
  - Commit to  $x$ :  $c = g^r h^x$
  - To decommit to  $y$ , find  $s$  such that  $r + kx = s + ky$
  - Compute  $s = r + k(x - y) \bmod q$

# ZKPOK from Sigma Protocols

## ▶ The basic idea

- Have  $V$  first to its challenge  $e$  using a perfectly-hiding trapdoor (equivocal) commitment

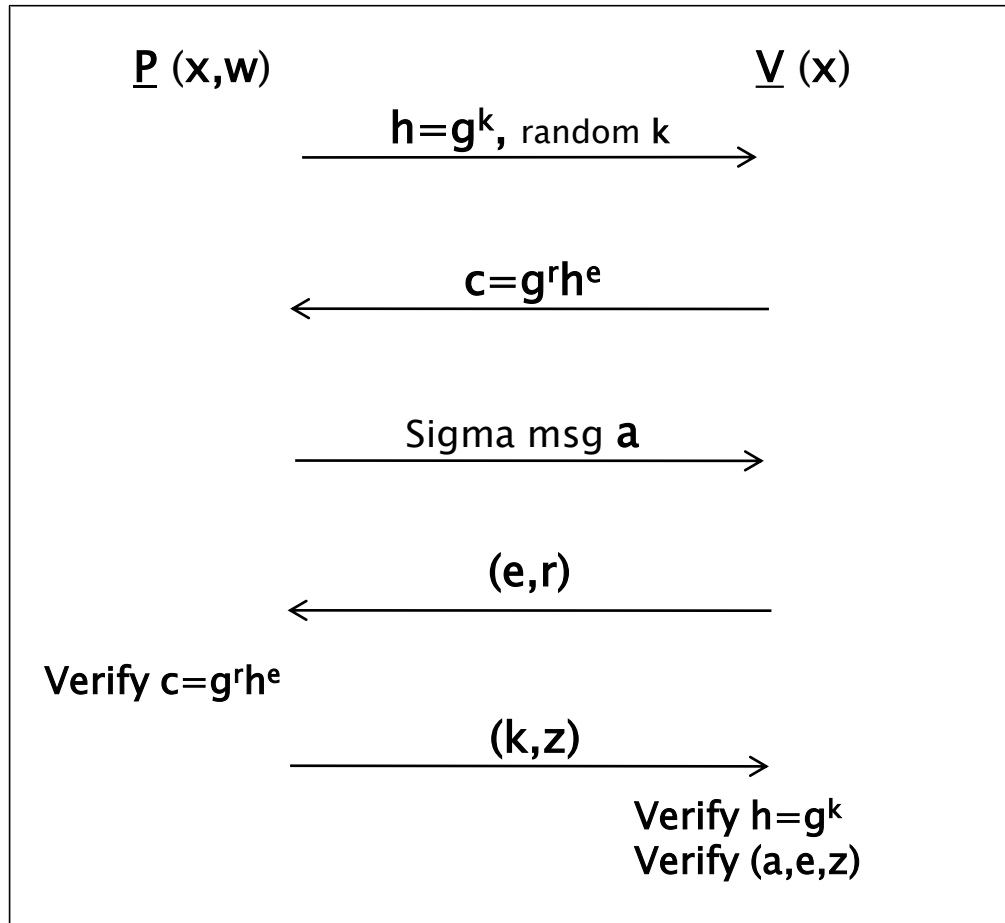
## ▶ The protocol

- $P$  sends the 1<sup>st</sup> message  $\alpha$  of the commit protocol
- $V$  sends a commitment  $c = \text{Com}_\alpha(e; r)$
- $P$  sends a message  $a$
- $V$  sends  $(e, r)$
- $P$  checks that  $c = \text{Com}_\alpha(e; r)$  and if yes sends the **trapdoor** and  $z$
- $V$  accepts if the **trapdoor** is correct and  $(x, a, e, z)$  is accepting

# ZKPOK from Sigma Protocols



Bar-Ilan University  
Dept. of Computer Science



# ZKPOK from Sigma Protocols



Bar-Ilan University  
Dept. of Computer Science

- ▶ **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



Bar-Ilan University  
Dept. of Computer Science

- ▶ 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 soundness

# Using Sigma Protocols and ZK



Bar-Ilan University  
Dept. of Computer Science

- ▶ **Prove that the El Gamal encryption  $(u,v)$  under public-key  $(g,h)$  is to the value  $m$** 
  - By encryption definition  $u=g^r$ ,  $v=h^r \cdot m$
  - 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  $(K_i), E_{K_i}(T_i)$** 
  - Can release  $T_i$  without revealing anything about  $T_j$  for  $j \neq i$

# Efficient Coin Tossing

- ▶  $P_1$  chooses a random  $x$ , sends  $(g, h, g^r, h^r x)$
- ▶  $P_1$  ZK-proves that it knows encrypted value
  - Suffices to prove that know discrete log of  $h$
- ▶  $P_2$  chooses a random  $y$  and sends to  $P_1$
- ▶  $P_1$  sends  $x$  (without decommitting)
- ▶  $P_1$  ZK-proves that encrypted value was  $x$
- ▶ Both parties output  $x+y$
  
- ▶ Cost:  $O(1)$  exponentiations

# Pedersen Commitments



Bar-Ilan University  
Dept. of Computer Science

## ► Recall the definition

- Parameters: generator  $g$ , order  $q$
- Commit protocol (commit to  $x$ ):
  - Receiver chooses random  $k$  and sends  $h=g^k$
  - Sender sends  $c=g^r h^x$ , for random  $r$

# Prove Commitment Knowledge



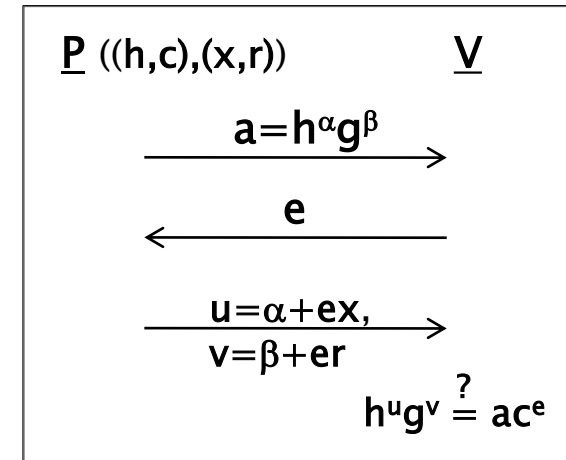
Bar-Ilan University  
Dept. of Computer Science

- ▶ **Relation:**  $((h,c),(x,r)) \in R$  iff  $c = g^r h^x$
- ▶ **Sigma protocol:**
  - P chooses random  $\alpha, \beta$  and sends  $a = h^\alpha g^\beta$
  - V sends a random  $e$
  - P sends  $u = \alpha + ex$ ,  $v = \beta + er$
  - V checks that  $h^u g^v = ac^e$
- ▶ **Completeness:**
  - $h^u g^v = h^{\alpha+ex} g^{\beta+er} = h^\alpha g^\beta (h^x g^r)^e = ac^e$

# Pedersen Commitment Proof

## ► Special soundness:

- Given  $(a, e, u, v), (a, e', u', v')$ , we have  
 $h^u g^v = ac^e, h^{u'} g^{v'} = ac^{e'}$   
 Thus,  $h^u g^v c^{-e} = h^{u'} g^{v'} c^{-e'}$   
 and  $h^{u-u'} g^{v-v'} = c^{e-e'}$
- Conclude:  $x = (u-u')(e-e')$  and  
 $r = (v-v')(e-e')$



## ► Special HVZK

- Given  $(g, h, h, c)$  and  $e$ , choose random  $u, v$  and compute  
 $a = h^u g^v c^{-e}$

# Proof of Pedersen Value

- ▶ Prove that the Pedersen committed value is  $x$
- ▶ Relation:  $((h, c, x), (r)) \in R$  iff  $c = g^r h^x$ 
  - Observe:  $ch^{-x} = g^r$
  - Conclusion: just prove that you know the discrete log of  $ch^{-x}$
- ▶ Application: statistical coin tossing

# Guillou-Quisquater

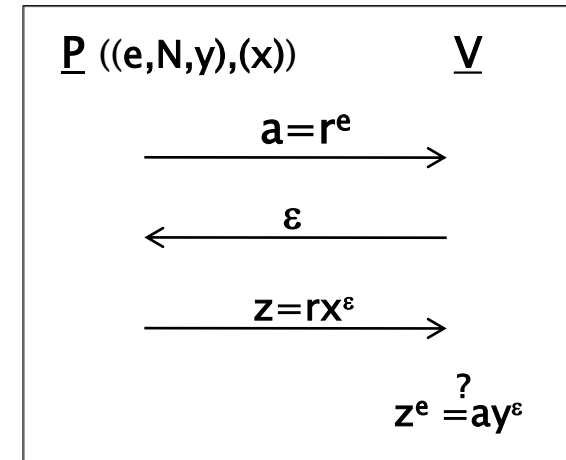
- ▶ Common input: RSA key  $(e, N)$ , value  $y$
- ▶  $P$  proves that it knows  $x$  s.t.  $x^e = y \bmod N$
- ▶ Protocol
  - $P$  chooses a random  $r$  and sends  $a = r^e \bmod N$
  - $V$  sends a random  $\varepsilon$
  - $P$  sends  $z = rx^\varepsilon \bmod N$  to  $V$
  - $V$  checks that  $z^e = ay^\varepsilon$
- ▶ Completeness:
  - $z^e = (rx^\varepsilon)^e = r^e(x^\varepsilon)^e = ay^\varepsilon$



# Guillou-Quisquater

## ► Special soundness:

- Given  $(a, \epsilon, z), (a, \epsilon', z')$ , we have  
 $z^e = ay^\epsilon, z'^e = ay^{\epsilon'}$
- Thus,  $z^e y^{-\epsilon} = z'^e y^{-\epsilon'}$   
and  $z^e z'^{-e} = y^{\epsilon-\epsilon'}$   
and  $y = (zz'^{-1})^{e/(\epsilon-\epsilon')}$
- Conclude:  $x = (zz'^{-1})^{1/(\epsilon-\epsilon')}$



## ► Special HVZK

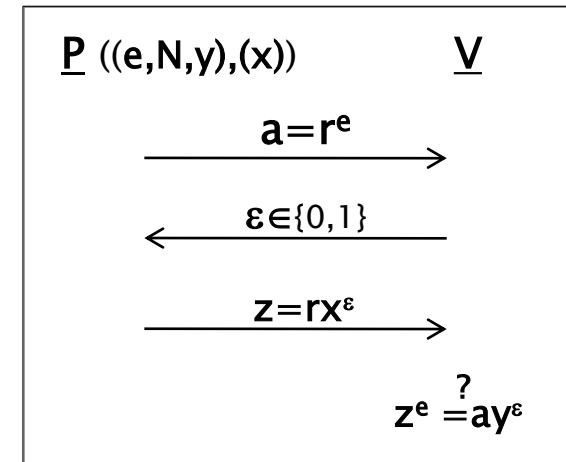
- Given  $(e, N, y)$  and  $\epsilon$ , choose random  $z$  and compute  $a = z^e y^{-\epsilon}$

# Not so Fast...

- ▶ **To run special-soundness algorithm, need to compute  $(zz'^{-1})^{1/(\varepsilon-\varepsilon')}$** 
  - This involves computing the inverse of  $(\varepsilon-\varepsilon')$  in the exponent
  - This requires knowing the order of the group
  - In RSA, this is  $\mathcal{O}(N)$  and is hard to compute!
- ▶ **Likewise, the simulation requires computing  $a = z^e y^{-\varepsilon}$** 
  - This involves computing the inverse of  $\varepsilon$  which is hard

# The Solution

- ▶ Choose  $\varepsilon$  randomly as 0 or 1
- ▶ Special soundness
  - Given  $(a, 0, z), (a, 1, z')$ , we have  $z^e = a$ ,  $z'^e = ay$  and so  $z^e = z'^e/y$  implying that  $y = z'^e/z^e$
- ▶ Zero knowledge
  - Given  $(e, N, y)$  and 0, choose random  $z$  and compute  $a = z^e$
  - Given  $(e, N, y)$  and 1, choose random  $z$  and compute  $a = z^e/y$



# A Fundamental Difference

- ▶ **This protocol has soundness of only  $\frac{1}{2}$** 
  - To get low soundness error (say  $2^{-40}$ ) need to repeat 40 times
  - These proofs are significantly more expensive
- ▶ **In TCC 2010, it was shown that there are inherent difficulties going below soundness  $\frac{1}{2}$  in groups of hidden order**

# 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  $H$

# Commitments from Sigma



Bar-Ilan University  
Dept. of Computer Science

## ▶ Hard relation $R$

- A generator  $G$  outputs  $(x, w) \in R$  s.t. for every PPT algorithm  $A$ ,  $\Pr[A(x) \in R]$  is negligible

## ▶ Example

- Output  $h = g^r$  for a random  $r$  (dlog relation)

# Commitment Scheme

- ▶ **Commitment to a string  $e \in \{0,1\}^t$** 
  - Receiver samples a hard  $(x,w)$ , and sends  $x$
  - Committer runs the sigma protocol simulator on  $(x,e)$  to get  $(a,e,z)$  and sends  $a$
- ▶ **Decommitment:**
  - Committer sends  $(a,e,z)$
  - Decommitter verifies that is accepting proof for  $x$
- ▶ **Hiding:**
  - By HVZK,  $a$  is independent of  $e$
- ▶ **Binding:**
  - Decommitting to two  $e,e'$  for the same  $a$  means finding  $w$

# Trapdoor Commitment



Bar-Ilan University  
Dept. of Computer Science

- ▶ **The scheme is actually a trapdoor commitment scheme**
  - Given  $w$ , can decommit to any value by running the real prover and not the simulator



# Hash Functions

## ▶ Need “strong” HVZK

- Simulator is given  $(e, z)$  and needs to find  $a$
- This holds for many sigma protocols

## ▶ Key for hash function

- A hard instance  $x$  of a hard relation

## ▶ Hash function

- Upon input  $(e, z)$ , let  $H(e, z) = a$  be the output of  $S(e, z)$

## ▶ Collision resistance

- Find  $(e, z), (e', z')$  s.t.  $H(e, z) = H(e', z')$
- This gives  $(a, e, z), (a, e', z')$

# Summary



Bar-Ilan University  
Dept. of Computer Science

- ▶ **Don't be afraid of using zero knowledge**
  - 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
  - Commitments and trapdoor commitments
  - Hash functions
  - More... (e.g., witness hiding)