Introduction to Secure Computation, part 3

Jonathan Katz

Outline: Hour 3

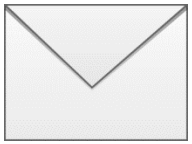
- Commitment schemes
- Zero knowledge proofs of knowledge (ZKPoK)
  - Dlog-based ZKPoKs
  - ZKPoK for NP
- Commitment protocols
- Coin tossing
- From semi-honest to malicious secure computation

Commitment schemes

- Emulates a “secure envelope”

Sender(m)

Receiver



m

Secure commitment

- Need to protect against malicious sender and malicious receiver
- “Hiding” (malicious receiver)
  - Receiver learns nothing about m in the commitment phase
- “Binding” (malicious sender)
  - Sender cannot generate commitment that it can later open two multiple messages
- Can consider information-theoretic security; for us, computational security is enough

Commitment from DDH

Sender(m)

Receiver

$G, q, g, h, g^r, h^r \cdot m$ →

- -

r, m →

Verify…

Security?

- Hiding follows from CPA-security of El Gamal encryption
  - Honest sender $\Rightarrow$ honest encryption of m
- Binding:
  - Say commitment is G, q, g, h, g_1, g_2
  - Does not matter if generated honestly (so long as G has order q; g generates G, and h, g_1, $g_2 \in$ G)
  - g, g_1 uniquely define $r = \log_g g_1$
  - $m = g_2/h^r$

ZKPoK

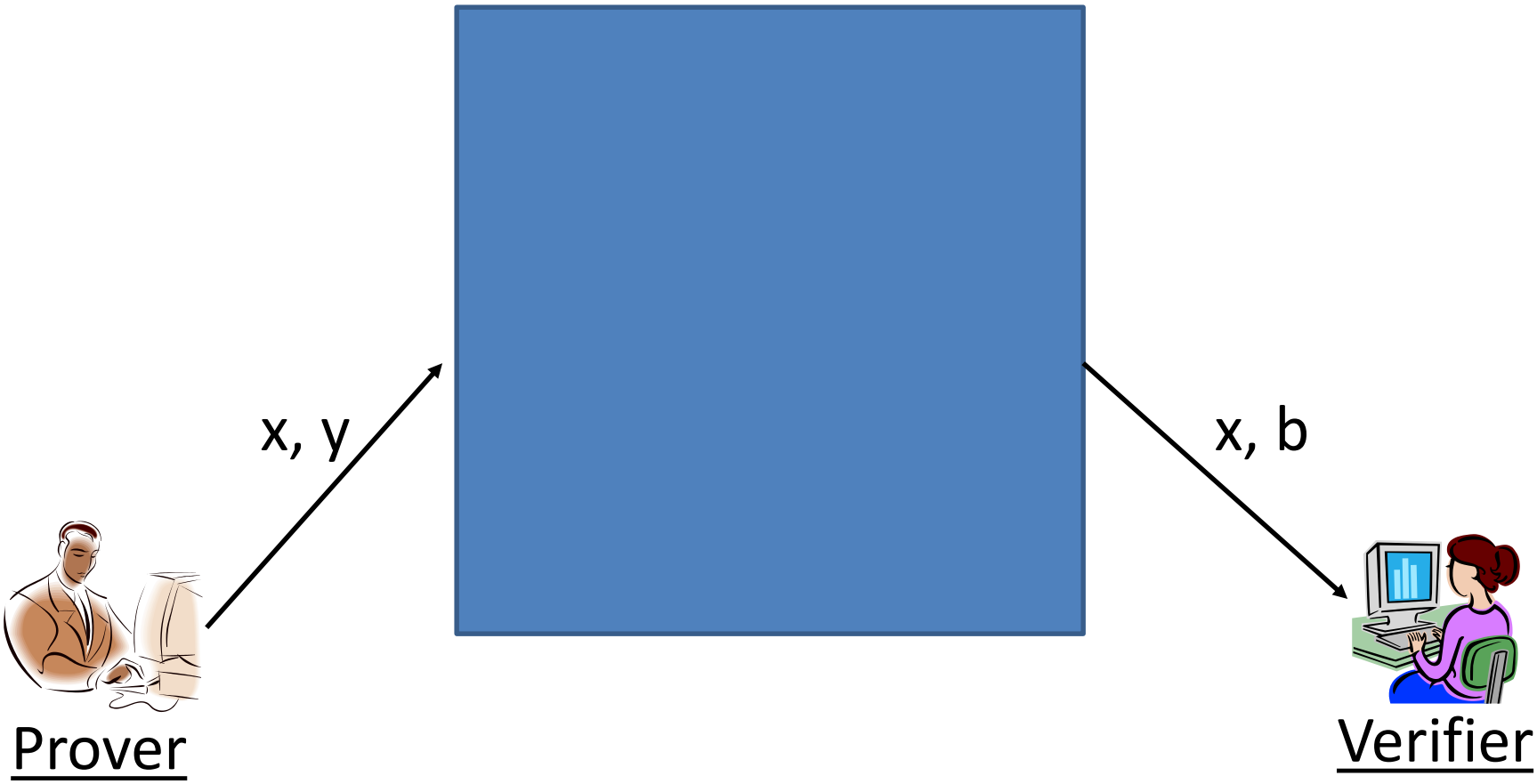
- Zero-knowledge proofs, zero-knowledge arguments, proofs of knowledge were originally defined/studied on their own

- Cleaner/easier to formalize via secure computation!

ZKPoK

- Let L be NP language with relation R
  - So L = {x : $\exists$y s.t. (x, y) $\in$ R}
  - Checking if (x, y) $\in$ R takes time poly(|x|)

ZKPoK



ZKPoK

- "Zero knowledge"
  - Verifier learns nothing about y (other than whether $x \in L$)
- "Proof of knowledge"
  - If b=1, then the prover knows y such that $(x, y) \in R$

Schnorr proofs

- Fix G, q, g
- $R = \{(h, w) : g^w = h\}$
  - Note that associated language is trivial!

Schnorr proofs

$h=g^w$

$r \leftarrow \mathbb{Z}_q$

$A=g^r$

$c \leftarrow \mathbb{Z}_q$

c

$s=cw+r$

Check: $h^cA = g^s$

Schnorr proofs

- Make non-interactive using Fiat-Shamir transform
- Prover:
  - $r \leftarrow \mathbb{Z}_q$
  - $A=g^r$
  - $c=H(A)$
  - $s=cw+r$
  - Proof = A, c, s
- Verifier checks as before (and checks H(A)=c)

Schnorr proofs

- Claim: The previous scheme is a ZKPoK in the random-oracle model

ZK

- Consider simulation for malicious verifier
- Interesting case is when verifier gets (h, 1)
  - Need to simulate a valid proof without knowing $\log_g h$

ZK

- Real proof ($h=g^w$)
  - $r \leftarrow \mathbb{Z}_q$
  - $A=g^r$
  - $c=H(A)$
  - $s=cw+r$
  - Output (A, c, s)
  - Check: $h^c A = g^s$
- Simulated proof
  - $c, s \leftarrow \mathbb{Z}_q$
  - $A = g^s/h^c$
  - Program $H(A)=c$
  - Output (A, c, s)

In both cases, A and c are uniform, and $s=cw + \log_g A$

$\Rightarrow$ Perfect simulation!

PoK

- Consider simulation for malicious prover
- Interesting case is when prover outputs a valid proof
  - Need to extract $w = \log_g h$
- Proof idea (sketch!)
  - Assume prover generates valid proof w.h.p. over choice of H(A)
  - Rewind prover and change H(A)=c to H(A)=c'; get valid proofs for both

PoK, continued

- Now have two valid proofs (A, c, s), (A, c', s') for h
- Note $h^c A = g^s$ and $h^{c'} A = g^{s'}$
- So can compute $\log_g h = (s-s')/(c-c')$
  - Note $c-c' \neq 0$

ZKPoK for NP

- Want to show existence of ZKPoK for any NP-relation R
- Strategy: show ZKPoK for NP-complete relation R^*
  - Any NP-relation R can be reduced to R^*
- (Unfortunately, this generally leads to poor efficiency)

ZKPoK for NP

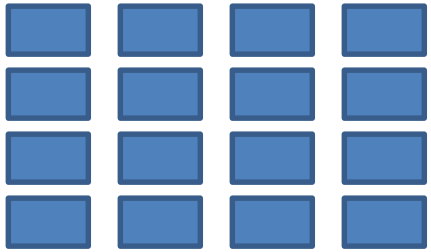
- Work with *Hamiltonian cycle*
  - Given directed graph, does there exist numbering of nodes s.t. there is a cycle $1 \rightarrow 2 \rightarrow \cdots \rightarrow n$?
  - Known to be NP-complete

ZKPoK

G, cycle

Choose random perm. π

Commit to adj. matrix of $\pi(G)$



$b \leftarrow \{0,1\}$

b

If b=0

send π

open all commitments

response

Else

open commitments

in cycle

Proof of ZK

- Simulator for malicious verifier (given G):
  1. “Guess” challenge b
  2. If b=0, commit to random permutation of G
  3. If b=1, commit to random cycle
  4. Give commitments to verifier, get b’
  5. If b’=b done; else return to step 1
  6. (abort after poly many tries)

ZK, continued

- Consider modified experiment
  1. "Guess" challenge b
  2. If b=0, commit to random permutation of G
  3. If b=1, commit to random permutation of G
  4. Give commitments to verifier, get b'
  5. If b'=b done; else return to step 1
  6. (abort after poly many tries)
- The distribution of this view is *statistically close* to the distribution of the real view

ZK continued

- Hiding of commitment implies that modified view is computationally indistinguishable from simulated view

$\Rightarrow$Simulated view is computationally indistinguishable from real view

PoK?

- As described, the protocol has soundness ½
  - Possible for prover to cheat with probability ½
- Repeat protocol k times (sequentially)
  - ZK property is preserved

PoK

- Consider simulator for malicious prover
  - Need to extract a cycle from a correct proof
- Note: correct responses for two challenges allows extraction of cycle
- Extraction done as follows:
  - Run honest execution with prover
    - If proof fails, halt
  - Rewind to each of the n executions and send other challenge
    - Hope that prover answers correctly for one of them

PoK, analysis

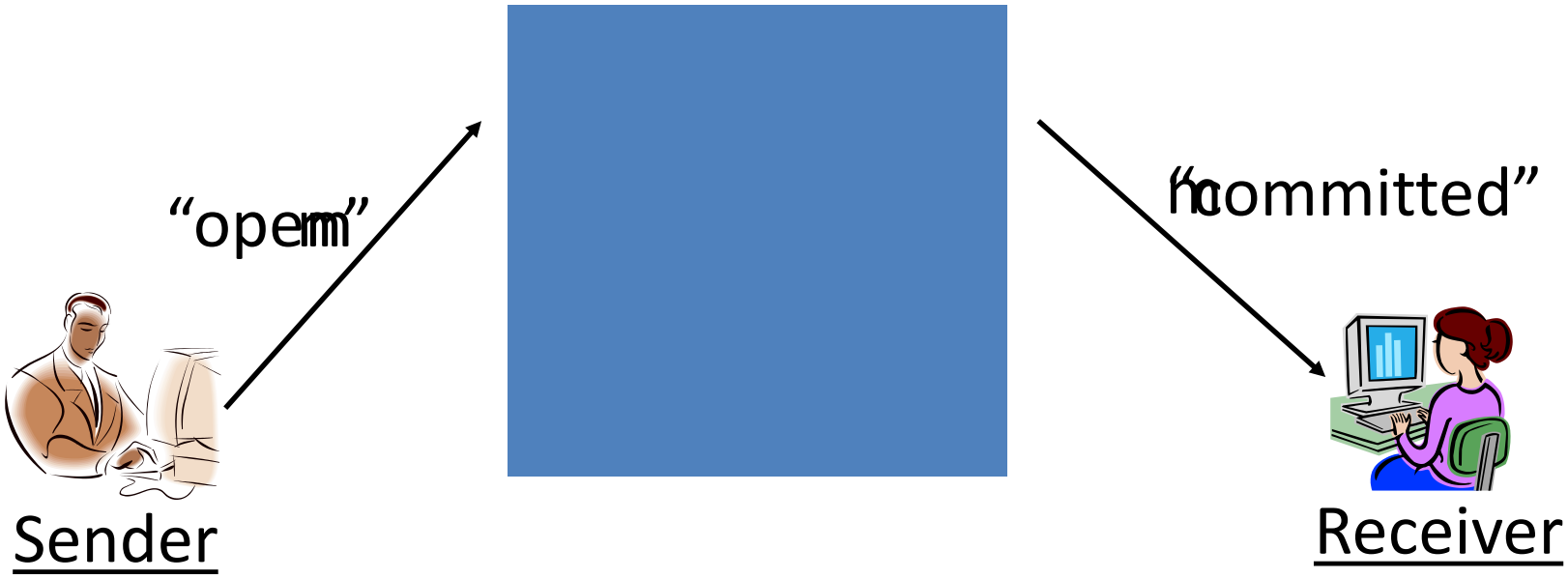
- Picture binary tree of 2^k challenges
- If Pr[prover successful] > 2^{-k} there are two accepting paths in the tree
  - Previous strategy will find the point where one path diverges from another
- Simulation can fail if Pr[prover successful] = 2^{-k}

ZKPoK feasibility results

- Possible to construct *constant-round* ZKPoK protocol for all of NP

Commitment protocols

- When designing protocols, often more useful to realize the commitment **functionality**
  - This is a *reactive* functionality



Commitment protocols

- Do commitment schemes realize this functionality?
- No (in general)
  - Unclear how to extract m from commitment
  - Unclear how to simulate for receiver (need equivocation)

Commitment protocols

- From commitment schemes to commitment protocols

Commit(m; r) →

ZKPoK of m, r →

- -

m →

ZKPoK of r →

Commitment protocols

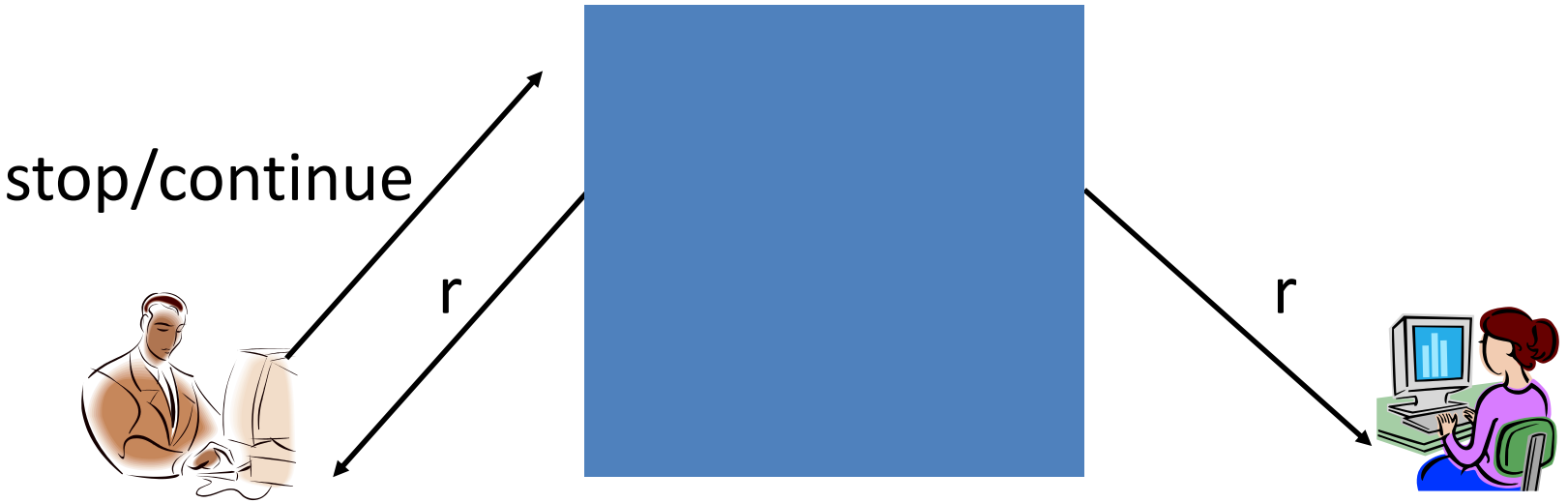
- Can also consider a variant where the functionality allows ZKPoKs about the committed value
- Alternately, have the functionality output a commitment to P_2 (with randomness chosen by P_1) in the first phase
  - Previous ideas can be adapted to realize these versions of the functionality

Security?

- Prove security in the ZK-hybrid model
  - Proof left as an exercise…

Coin tossing

- Want a way for two or more parties to generate a random string
  - No party should be able to bias (other than abort)



Coin tossing

- Design protocol in the F_{Com}-hybrid model

r_1

F_{Com}

"committed"

r_2

"open"

F_{Com}

r_1

$r = r_1 \oplus r_2$

$r = r_1 \oplus r_2$

Coin tossing

- Can also consider a variant where P_1 learns the value of r, and P_2 gets a commitment to r
  - Previous ideas can be adapted to realize this version of the functionality
- Can also consider a multi-party version of these functionalities

Malicious secure computation

The GMW compiler

- The GMW compiler provides a way to compile any protocol with **semi-honest** security into a protocol with **malicious** security
- Main idea: enforce correct behavior
  - Check that messages correspond to running the protocol…
  - …with consistent state…
  - …using "real" randomness
    - I.e., outside party's control
- Assume a broadcast channel for simplicity

Importance of randomness…

Input: m_0, m_1

Input: b

$r \leftarrow \mathbb{Z}_q$

$h_b = g^r$

$h_{1-b} \leftarrow G$

h_0, h_1 ←

$Enc_{h_0}(m_0),\ Enc_{h_1}(m_1)$ →

Decrypt…

Designing for efficiency

Input: m_0, m_1

$h \leftarrow G$

Input: b

$r \leftarrow \mathbb{Z}_q$

$h_b = g^r$

$h_{1-b} = h/h_b$

h →

← h_0, h_1

$Enc_{h_0}(m_0), Enc_{h_1}(m_1)$ →

Decrypt…

The GMW compiler

1. Each P_i commits to its input x_i (using F_{ZKCom})
2. Each party P_i does coin tossing so it gets randomness r_i and all other parties get a commitment to r_i
3. Run semi-honest protocol, and at each step **prove honest behavior**
   - I.e., give ZKPoK of the following NP-statement:
     $\exists$ x_i, r_i, decommitments s.t. (1) these values match steps 1 and 2; and (2) running protocol using these values (and other parties' messages) gives the next message sent

Security?

- (Sketch)
- Simulator for malicious P_i:
  - Extract x_i from step 1; simulate other commitments
  - Send x_i to functionality, get output y_i
  - Run simulator for *semi-honest* protocol; in particular, this gives some randomness r_i
  - Simulate coin tossing with output r_i (also simulate coin tossing for other parties)
  - P_i now "forced" to match simulated transcript

Advanced note

- Only ZK (but not PoK) needed in step 3

Corollary

- We can realize two-party commitment, coin-tossing, semi-honest secure computation, and ZKPoK in constant rounds
  - So we get a *constant-round* secure 2PC protocol with malicious security
- Multi-party case is trickier
  - Need constant-round coin tossing
  - Easy with random oracles; more difficult otherwise

Other feasibility results

- Information-theoretic security
  - Assuming secure channels and broadcast
  - Possible for all functions iff $t < n/2$
- Fairness
  - Assuming broadcast
  - Possible for all functions iff $t < n/2$

Research frontiers

- Efficiency!
  - How efficient can we make generic MPC?
  - MPC in other computational models
  - Efficient protocols for specific functions

- Secure computation and privacy/correctness
  - Secure MPC guarantees that real-world execution is “as secure as” ideal-world execution
    - How secure is ideal-world execution?

Research frontiers

- MPC in practice
  - Asynchrony, no broadcast
  - Parties not all online at the same time
  - Other practical concerns

- Understanding general composition
  - UC is complex, possibly overkill, needs setup
  - Are other approaches possible?
  - Composition with random oracles

References

- Goldreich, “Foundations of Cryptography, volumes 1 and 2”

- Katz lecture notes, http://www.cs.umd.edu/~jkatz/gradcrypto2

- Lindell, “Parallel Coin Tossing and Constant-Round Secure Two-Party Computation,” https://eprint.iacr.org/2001/107

Thank you!