Rational Secret Sharing and Multiparty Computation

Joseph Halpern
Cornell University

Vanessa Teague
Stanford University

Secret Sharing

- Suppose Alice wants to share a secret among n agents
  - Any subset of m < n agents can pool their shares and reconstruct the secret
  - No subset of size < m learns anything
  - Assumption: up to n-m agents may be ``bad'', and may not reveal their shares
  - The rest of the agents are ``good'', and follow the protocol
  - The bad agents can't prevent the good agents from recovering the secret.

Shamir's Protocol

Alice chooses a degree m-1 polynomial such that f(0) is her secret

- Gives f(1) to agent 1, f(2) to agent 2, …
- Any m agents can reconstruct f and compute the secret f(0)
  - m points determine a degree m-1 polynomial
- No subset of size < m can reconstruct the polynomial or learn the secret.

Rational Agents

- The partition into ``bad'' and ``good'' is not always appropriate.
- Suppose instead agents are "rational"
  - They have preferences, and act so as to maximize their expected utility
- We assume preferences are
  - firstly, getting the secret is better than not getting it
  - secondarily, the fewer of the other agents that get it, the better

Shamir's protocol doesn't work with rational agents.

- What incentive does an agent have to reveal his share?
- An agent is always better off not revealing his share than revealing it!
- Not revealing the share weakly dominates revealing it.

- Is there a protocol for reconstructing the secret that rational agents will follow?

Results

- Theorem 1: Fixed-length protocols don't work with rational agents.
  - if the length of the protocol is common knowledge, nobody learns the secret

  All protocols in the literature have fixed length.
- Theorem 2: There's a random-length protocol that does work
  - Constant expected running time

Secure multiparty computation

- Everyone has a secret and wants to learn some function f of everyone's secret without revealing their secret:
  - Agents want to know what is the highest salary?
    - Private input: their own salary
  - ISPs want to know if there is an intruder.
    - Private input: their routing requests
  - Insurance companies want to know if there is a pattern of false claims
    - Private input: their claim histories

Solutions to Secure Multiparty Computation

General solutions [GMW 87, BGW 88...]:

If

- no more than 1/3 (sometimes $\frac{1}{2}$) of the players bad
- the players truthfully report their inputs

then the function can be computed

- Just as if there had been a trusted party to whom all the players report their secrets.

The protocols are all fixed length.

Multiparty Computation with Rational Agents

Even with a trusted central party, can't always do multiparty computation with rational agents

- Suppose everyone's input is a bit and we want to compute the exclusive or.
- If player 1 lies about his bit and everyone else tells the truth, then player 1 can compute the exclusive or;
- No one else will know it.

Shoham and Tennenholtz characterize those functions that are noncooperatively computable with a trusted central party and rational agents

Rational Multiparty Computation

- Theorem 3: There is no fixed-length protocols for rational multiparty computation without a trusted party
  - Proof similar to Theorem 1
- Theorem 4: There is a random-length protocol that does work, even without a trusted part for all functions that are non-cooperatively computable
  - Idea: add our randomized secret-reconstruction protocol to the end of GMW's multiparty computation protocol

Technical Details: Iterated deletion

What counts as a good solution?

- It must be a Nash equilibrium
- It must survive iterated deletion of weakly dominated strategies:
  - First delete all weakly dominated strategies
  - Then delete all those that are weakly dominated after the first set is deleted
  - …
  - Continue until no more weakly dominated strategies

Proof of Impossibility Result

With a fixed length protocol

- At first step, delete all strategies that send a share at the last step
- Roughly speaking, at second step, delete all strategies that send a share at second-last step,
- Continue …

Some subtleties

The actual proof is much more subtle

- Problem: why can a strategy that sends a share at the second-last step be deleted after all strategies that send a share at the last step have been deleted
  - Need to know that other strategies have not been deleted.

Idea of Randomized Protocol

- Suppose there are three agents
- Each agent can toss a fair coin and be forced to reveal its outcome
- Idea: toss a coin, send your share if you get heads
  - If you don't send your share, you will get caught

Some Problems

- What if you lie about your share?
  - Assume the shares are cryptographically signed
- What if the other two agents get heads, you get tails?
  - You learn the secret, they don't!
  - Solution: implement coin toss where it can't be the case that only two agents get heads
- What if u(only you get the secret)/4 > u(everyone gets the secret)
  - Toss a coin with a different probability

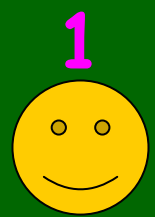
Improved Protocol

- Each agent i tosses coin b_i
- Arrange it so that each agent learns
  $b_1 \oplus b_2 \oplus b_3$
- Without learning the other agents' coins
  - To do this, use auxiliary coin
- Agent i sends his share iff $b_i = 1$ and
  $b_1 \oplus b_2 \oplus b_3 = 1$
  - Agent i won't send if 2/3 agents toss heads

Protocol for 3 players

Protocol for agent 1:

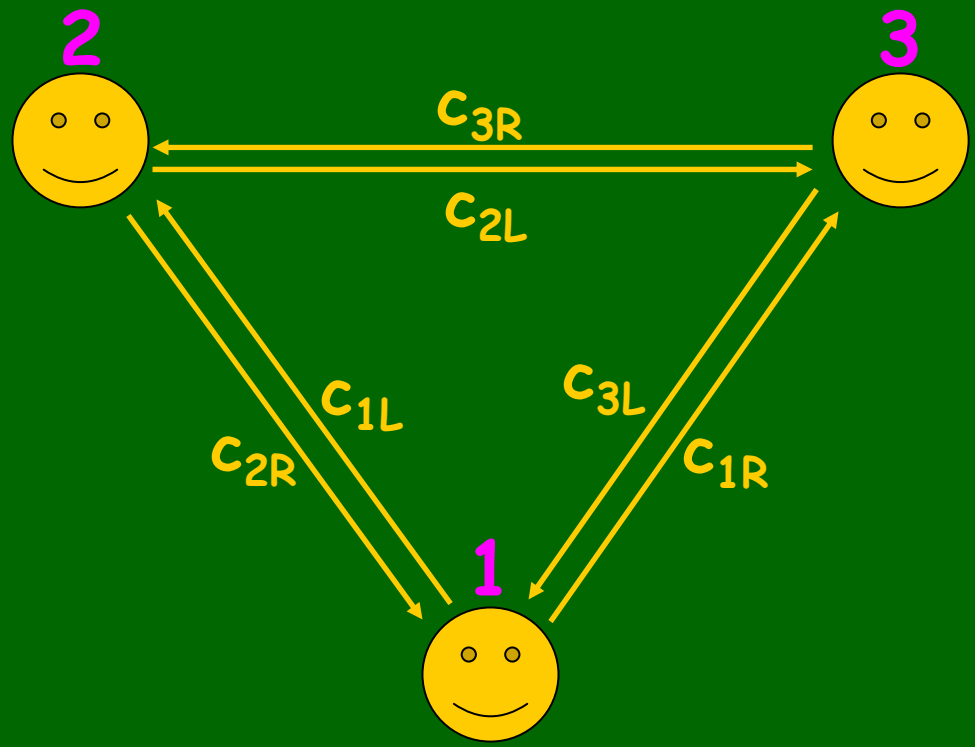
1. Toss coin b_1
2. Toss coin c_{1L}
3. Set $c_{1R} = b_1 \oplus c_{1L}$



Protocol for 3 players

Protocol for agent 1:

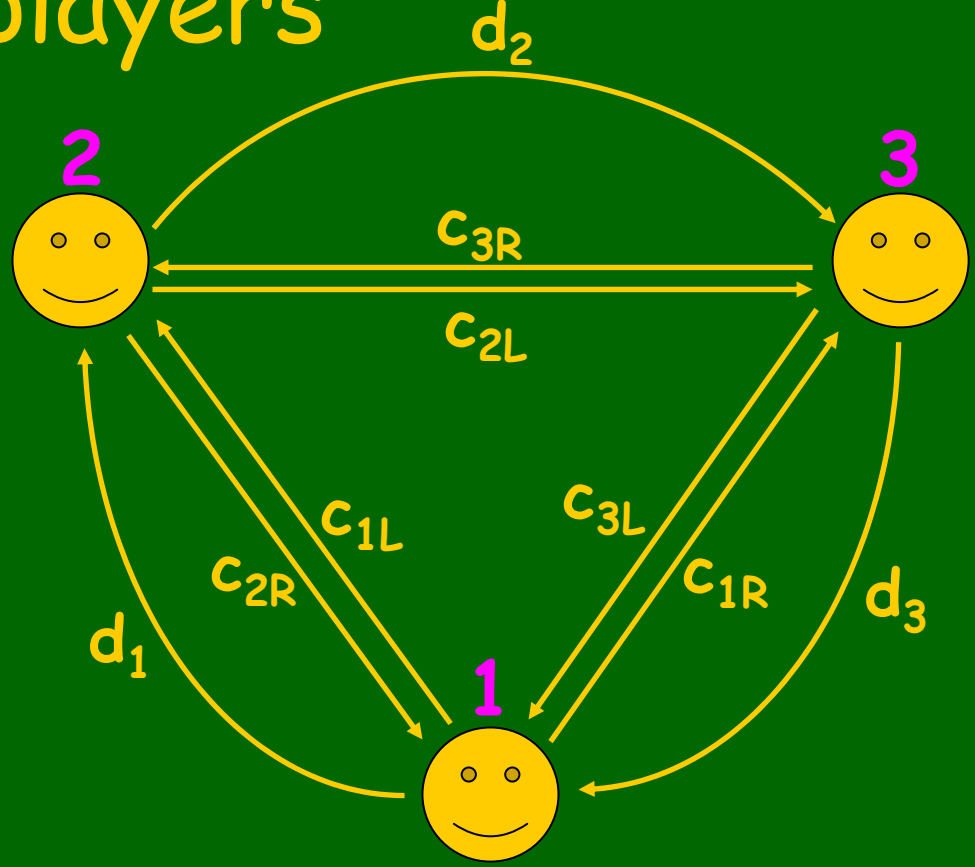
1. Toss coin b_1
2. Toss coin c_{1L}
3. Set $c_{1R} = b_1 \oplus c_{1L}$
4. Send c_{1L} left, c_{1R} right



Protocol for 3 players

Protocol for agent 1:

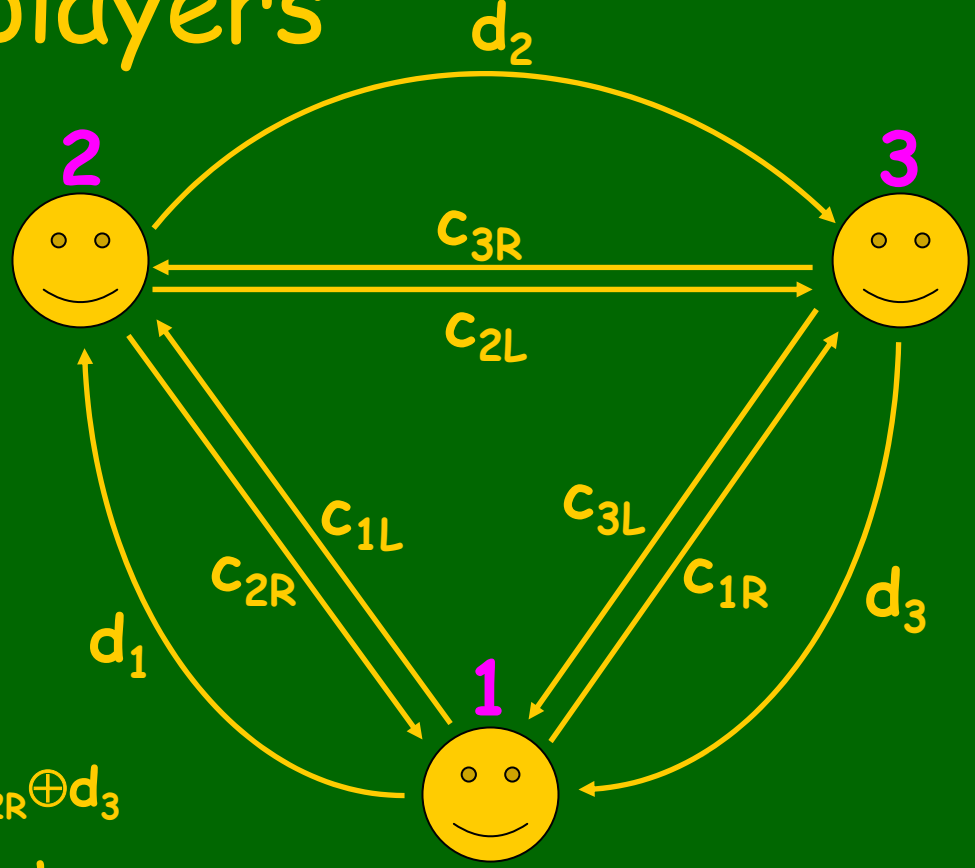
1. Toss coin b_1
2. Toss coin c_{1L}
3. Set $c_{1R} = b_1 \oplus c_{1L}$
4. Send c_{1L} left, c_{1R} right
5. Send $d_1 = b_1 \oplus c_{3L}$ left



Protocol for 3 players

Protocol for agent 1:

1. Toss coin b_1
2. Toss coin c_{1L}
3. Set $c_{1R} = b_1 \oplus c_{1L}$
4. Send c_{1L} left, c_{1R} right
5. Send $d_1 = b_1 \oplus c_{3L}$ left
6. Compute $b_1 \oplus b_2 \oplus b_3 = b_1 \oplus c_{2R} \oplus d_3$
7. If $b_1 = b_1 \oplus b_2 \oplus b_3 = 1$, send share.
8. If received share or detected cheating, quit. Else restart protocol with new share.



More than 3 players

- Partition players into 3 groups
- Each group's leader collects (some of) the group's shares, then they run the 3-player protocol
- 2-out-of-n secret sharing is possible if $n > 2$; 2-out-of-2 is not

Conclusion

- The traditional approach to secret sharing/multiparty computation assumes ``good guys'' and ``bad guys''
- For many applications, it may make more sense to assume that all agents are rational, and try to maximize utility
- This assumption has a nontrivial impact!
- We need randomization to get a protocol for these problems