Advanced Tools from Modern Cryptography

Lecture 9

MPC: Security Against Active Corruption

Handling Active Corruption

- Need to ensure that there is a well-defined input for the adversary
  - Simulator should be able to "extract" the corrupt parties' inputs
- Should make sure that the adversary cannot change the outcome
- Secrecy should hold even if the corrupt parties deviate from the protocol
  - General idea: catch deviations.
    - On catching a deviation an honest party may abort the protocol (if adversarial abort is allowed in the ideal world)
    - Or "deactivate" (potentially) corrupt players and continue the protocol. Possible when there is a large enough honest-majority
  - Note: Catching itself shouldn't reveal information about inputs

GMW Paradigm

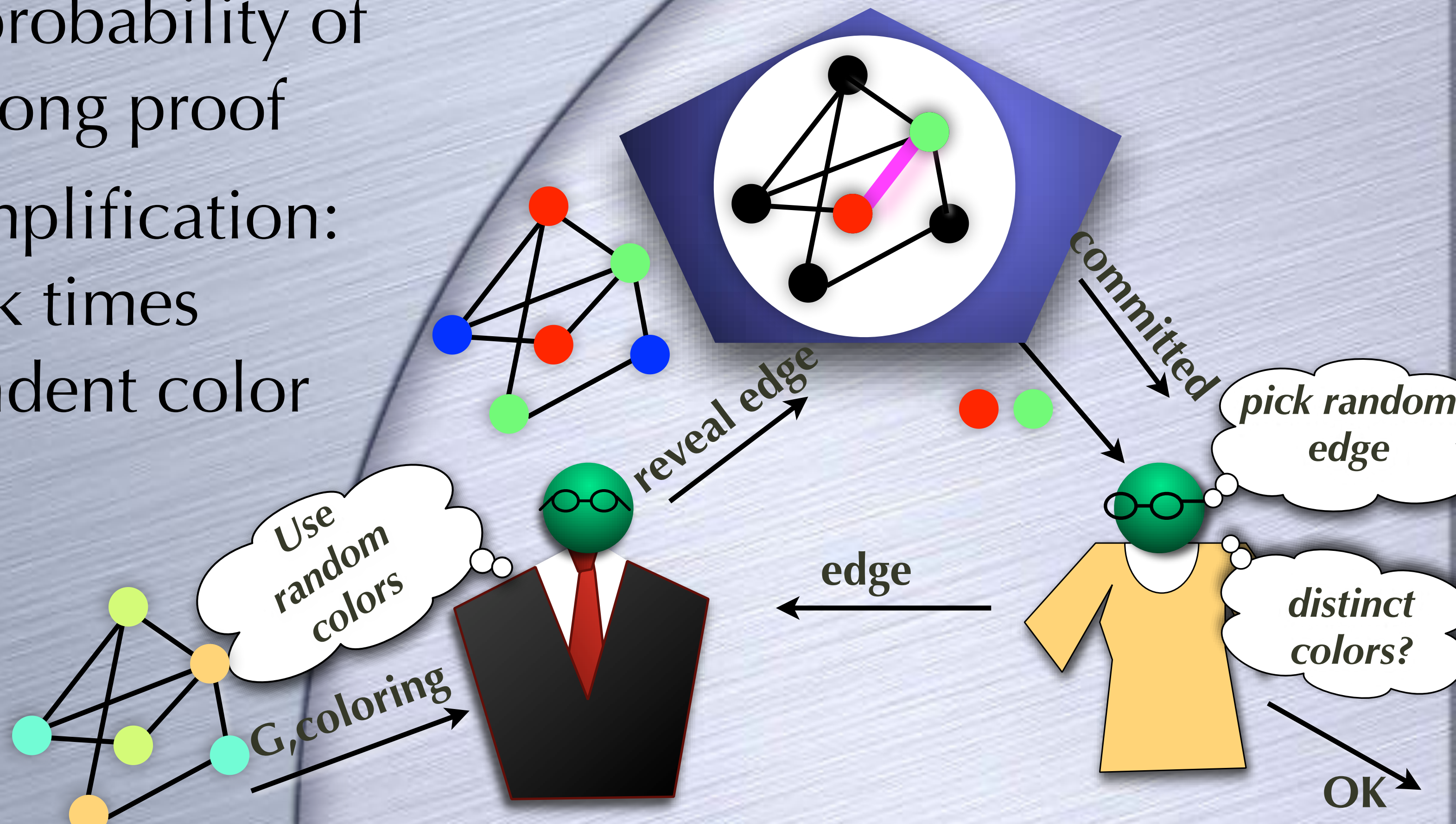
- Run a passive-secure protocol Π, but let each party "verify" that the others are following the protocol correctly
  - Correctly: pick arbitrary inputs and arbitrary randomness first, but then follow the specified program
  - Verification should not reveal information: then cannot rely on passive security of Π any more!
    - How to verify without learning any information?
    - Zero-Knowledge Proofs!

Zero-Knowledge Proofs

- Suppose Alice wants to convince Bob that a boolean formula in n-variables $f(x_1,\ldots,x_n)$ is satisfiable
  - i.e., $\exists$ values $(v_1,\ldots,v_n)$ such that $f(v_1,\ldots,v_n) = 1$
  - But doesn't want to reveal any "knowledge" about the solution to Bob (even if solution fully determined by f)
- Zero-Knowledge Proof functionality: F_{ZK}
  - Alice sends $(f, (v_1,\ldots,v_n))$ to F_{ZK}, which sends $(f, f(v_1,\ldots,v_n))$ to Bob
- Zero-Knowledge protocol: a 2-party secure computation protocol for the functionality F_{ZK}
  - Not interesting for passive corruption (of prover)

A ZK Proof for Graph Colorability

- Uses a commitment protocol as a subroutine
- At least 1/m probability of catching a wrong proof
- Soundness amplification: Repeat say mk times (with independent color permutations)



Zero-Knowledge Proofs

- Traditional definition of ZK proofs is somewhat different
  - Simulation-based security for actively corrupt (standalone) verifier only
  - Security against prover: Soundness
    - Allows computationally unbounded corrupt provers
    - A corrupt prover should have negligible probability of getting the honest verifier to accept a false statement
- Our definition of ZK proofs corresponds to "Proof/Argument of Knowledge"
  - Argument: Soundness only against PPT prover
  - Knowledge: Prover "knows" v s.t. f(v)=1

Zero Knowledge Proofs

From Passive, Honest-Majority MPC "in the head"

- Consider an honest-majority, passive-secure MPC protocol Π for the ZK functionality, with n servers (in addition to one input client and one output client)
- Alice carries out the execution of a session of the MPC protocol with her inputs (f,v) as the input of the input client
- Alice sends the view of the output client View(out) to Bob and commits to the view of the i^{th} server, View(i), for every i, to Bob
- Bob sends a random subset $S \subseteq [n]$, $|S| < n/2$ to Alice. Alice opens View(i) for all $i \in S$.
- Bob accepts the proof (outputs) f, if every pair of views it got is consistent, and View(out) has the output f

Zero Knowledge Proofs

From Passive, Honest-Majority MPC "in the head"

- Security against corrupt Bob: Bob's view consists solely of View(out) and View(i) for $i \in S$ where S is chosen by Bob (after seeing View(out))
  - Since $|S|<n/2$, can be simulated just based on f, by the passive (adaptive) security of Π
- Security against corrupt Alice: Simulator can see what Alice commits to, but these views may not be consistent
  - If there is a vertex cover of $< n/2$ server views covering "inconsistent edges", then execution corresponds to one with $< n/2$ corrupt parties. Simulator for Π can extract v from the view of the honest parties.
  - If no such vertex cover, too many inconsistent edges and S will contain at least one such pair except with negligible probability

GMW Paradigm

- Run a passive-secure protocol Π, but let each party "verify" that the others are following the protocol correctly
  - Correctly: pick arbitrary inputs and arbitrary randomness first, but then follow the specified program
- Need to prove that each message was correctly computed, right when it is sent
  - If proof required only at the end, too late!
- Proving $\exists$ input, rand, s.t. next-message$_\Pi$ (input,rand,messages) equals the message being sent
  - Should use the same input and randomness through out!

Commit & Prove

- Proving $\exists v\ f(v)=1$ is by itself not enough for the GMW transformation
  - Multiple statements to prove with the same v
- Commit-and-Prove functionality: F_{CaP}
  - Alice sends v to F_{CaP}, which sends "committed" to Bob
  - Subsequently, for i=1,2,... Alice sends a function f_i (represented as a circuit) to F_{CaP}, which sends $(f_i,f_i(v))$ to Bob
    - More generally, Alice sends (f_i,w_i) and F_{CaP} sends $(f_i,f_i(v,w_i))$ to Bob
  - Note: same v used in all rounds

GMW Paradigm

- Run a passive-secure protocol Π, but let each party "verify" that the others are following the protocol correctly
  - Correctly: pick arbitrary inputs and arbitrary randomness first, but then follow the specified program
- Each party proves using F_{CaP} that each message was correctly computed, for the same committed inputs and randomness
  - Could "securely implement" F_{CaP} using a "plain" commitment of v (i.e., not using F_{com}), and proving statements about it using F_{ZK}
    - (Or can use F_{OT} instead of F_{Com} in the protocol for F_{ZK})
  - f_i defined so that $f_i(v) = 1$ iff Π produces message m_i on input/randomness v for the proving party, given the transcript so far
    - All communication in Π assumed to be over public channels

Composition

- We built an active-secure protocol using access to ideal F_{CaP} functionality
  - Is it OK to "replace" it by a protocol for F_{CaP}?
  - Depends on the exact definition of security
  - Hint: OK for Universally Composable security
  - Later