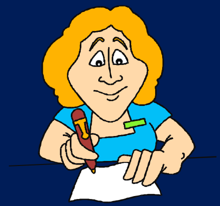
An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries

Benny Pinkas

University of Haifa

Yehuda Lindell, Bar-Ilan University

Secure Two-Party Computation

Alice 

 Bob

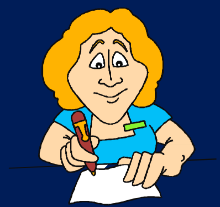
Input: x y

Output: F(x,y) *and nothing else*

E.g., the millionaires problem

F(x,y) = 1 iff x>y.

Secure Two-Party Computation: security



Input: x y

Output: F(x,y) and nothing else

As if… x y

F(x,y) F(x,y)

Wish to have similar privacy, without the aid of a TTP.

Possible scenarios

- Two parties vs. Multi-party
- Adversaries
  - Semi-honest: follow the protocol but try to learn more
  - Malicious: can do anything

  - It is easier to design solutions which are only good against semi-honest adversaries

- Yao [82,86]:
  - A generic protocol for two-party computation (of any function!) secure against semi-honest adversaries

This talk

- Securing Yao's protocol against malicious adversaries
- Using "cut-and-choose", unlike other solutions which use generic or number-theoretic ZK proofs
- Keeping it efficient
  - Similar computational overhead ☺
  - Larger communication overhead ☹

…This talk

- …And proving security in the ideal/real simulation paradigm
  – This is the main motivation:
    - Implement a functionality (efficiently!) using our protocol
    - Use it as a primitive in more complex protocols
    - Analyze in the hybrid model (i.e., assuming a trusted party computes the functionality) [C]
- Example: computing the k’th ranked element [AMP]

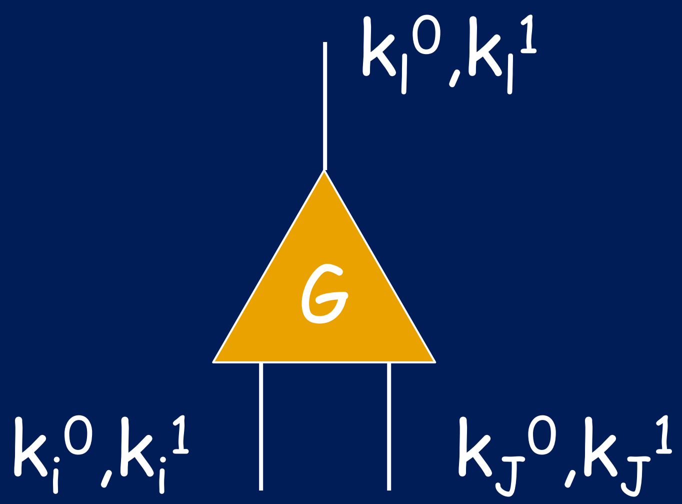
Theorem (stating our results)

- Constant-round black-box reduction of secure two-party computation (secure in the real/ideal model simulation paradigm against malicious parties) to
  - oblivious transfer
  - and perfectly hiding commitments
- Also, a black-box reduction to
  - oblivious transfer alone
  - with a number of rounds which is linear in a statistical security parameter.

Yao's Protocol for Generic secure two-party computation

- P_1 and P_2 wish to compute a function F, defined as a Binary circuit.
- P_1 (aka circuit constructor) constructs a Binary circuit computing F, and then garbles it.

- Garbled values:



k_i^0 = 0 on wire i
k_i^1 = 1 on wire i

P_2 will learn one string per wire, but not which bit it corresponds to.

Therefore doesn't learn intermediate values.

Bird's eye view of Yao's protocol

- P_1 defines garbled values for every wire
- P_1 constructs tables which enable to
  - compute the garbled output of a gate
  - given the garbled values of the gate's input wires
- Applying this to the entire circuit, it is possible to compute the circuit's output (and no internal value), given the garbled values of the circuit's input wires.
- It is also possible to let each player learn a different output

Running the protocol (semi-honest case)

- P_1 sends to P_2
  - Tables encoding each gate.
  - Garbled values (k's) of P_1's input values.
- For every wire i of P_2's input:
  - The parties run an oblivious transfer (OT) protocol
  - P_1's input is k_i^0, k_i^1
  - P_2's input is its input bit (*b*).
  - P_2 learns k_i^b
- Afterwards P_2 can compute the circuit by itself.
- Efficient for medium size circuits [Fairplay - NMPS]
- Full proof (after modifications) against *semi-honest* adversaries [LP06]

Security against malicious adversaries

How can parties prove that they behave correctly?

1. A zero-knowledge proof based on a reduction to an NP complete problem [GMW]
   - GMW's compiler
   - Generic, shows feasibility, non black-box, rather inefficient.
2. Prove correctness of the circuit gate-by-gate
   - Jarecki-Shmatikov (Eurocrypt '07).
   - More efficient than the reduction based approach, but still requires a ZK proof per gate.
     - ☹ instead of doing symmetric key operations per gate, we now have to do public key operations.
3. Cut-and-choose based solutions…

Malicious Behavior and Cut-and-Choose

- What can a malicious circuit constructor (P_1) do?
  - Can certainly construct a circuit which computes F' instead of F.
- Folk solution: "cut and choose"
  - P_1 constructs many circuits and commits to them.
  - P_2 asks P_1 to open a randomly chosen subset of the circuits, and checks that they are all correct.
  - The parties evaluate the remaining circuits.
- Intuition: An illegitimate circuit is identified whp.
  - But there are more problems…
- Efficiency: more copies of the circuit, but the computation does not change by much.

Cut-and-Choose based security for Yao's protocol

- Mohassel-Franklin 2006
  - Cut-and-choose based protocol against malicious adversaries.
  - Cannot be fully proven in the ideal/real model paradigm.
  - Only one party learns output; no output for the circuit constructor.
- Main issue (found by Kiraz-Schoenmakers)
  - P_1 can cheat in the OT protocol (where it is the sender): provides corrupt input to the 0 choice, and good input for 1.
  - If P_2's input is 1 all checks go well.
  - If P_2's input is 0, it must abort *(and cannot complain) !*
  - Checking the circuits does not help.
  - Therefore MF cannot be proven in the ideal/real model
  - KS suggest a solution using committing OT.

Our contributions

- Efficient protocol for malicious parties
  - A cut-and-choose based implementation of Yao's protocol.
  - Both parties can have (possibly differing) outputs.
  - Proof is complex but protocol is efficient:
    - Public key ops: only O(1) (regular) OTs per input bit.
    - Communication is multiplied by a statistical security parameter *s* (to obtain cheating probability < $2^{-O(s)}$).
- Simulation based proof
  - Proof based on the real/ideal model simulation paradigm.
  - Rather than separate proofs for privacy and correctness.
  - The protocol can therefore be called by other protocols.
- Rest of talk: discuss the problems we encountered.

Basic Protocol

n-bit inputs. Statistical security parameter *s*.

1. The parties agree to a circuit C computing F(). P_1 constructs ***s*** garbled copies of C and commits to them.
2. P_2 uses OT to learn its garbled inputs to all circuits (only *n* OTs: one per input bit for all *s* garbled circuits).
3. P_1 sends the commitments to the circuits.
4. P_2 asks P_1 to open s/2 circuits, and verifies them.
5. If P_2 is happy, P_1 sends the garbled values of P_1's inputs in the remaining s/2 circuits.
6. P_2 evaluates these circuits

But what happens if not all circuits have the same output?

Problem 1: Inconsistent outputs

- What should P_2 do if *not* evaluated circuits yield the same output?
  - P_1 definitely cheated, but should P_2 abort?
  - If P_2 aborts, it reveals information to P_1.
- Example: suppose P_2 aborts if outputs are inconsistent.
  - P_1 constructs *s-1* circuits computing F.
  - One circuit computes F if and only if P_2's input is 0.
  - With probability ½, P_1's cheating is not detected in the first stage. Then P_2 aborts iff its input is not 0.
- Solution (providing exponential security):
  - P_2 computes the circuits, and outputs the same value as the majority of the circuits.
  - Intuition: In order to cheat, P_1 needs *s/4* corrupt circuits, and none of them should be checked by P_2.

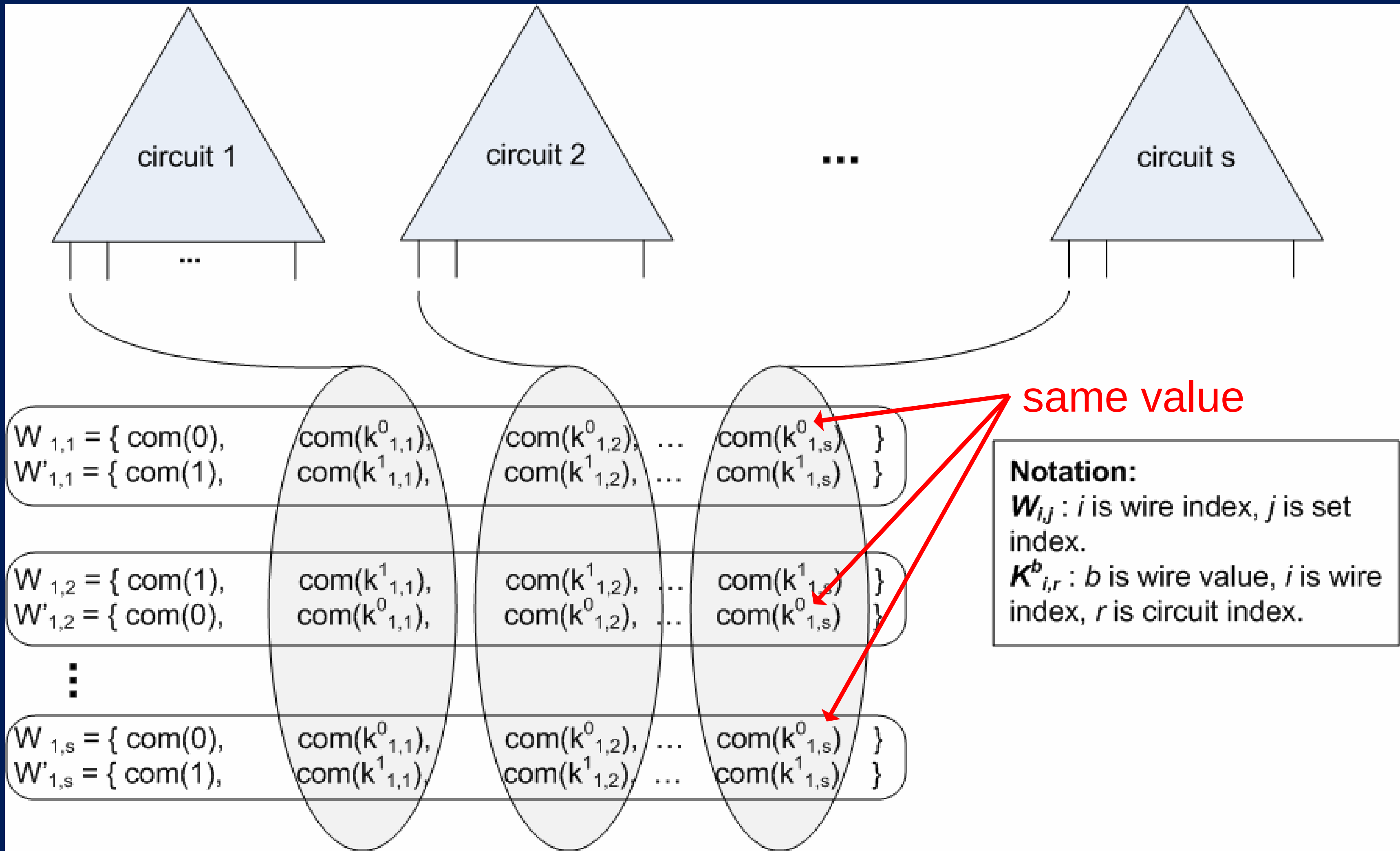
Problem 2: Input Consistency

- P_1 might provide different inputs (of P_1) to different circuits.
- Does this matter?
  - Suppose the parties compute the inner product. (Inputs are $X=x_1,\ldots,x_s$ and $Y=y_1,\ldots,y_s$, and $F(X,Y)=\sum_{i=1\ldots s} x_i \cdot y_i$.)
  - P_1 sets different inputs to different circuits: its input to the i'th circuit has $x_i=1$ and $x_j=0$ for $j\neq i$.
  - Circuit i now outputs y_i. The majority result output by P_2 is therefore 1 iff the Hamming weight of $Y > s/2$.
- Solution: must verify consistency of P_1's inputs.
- Problem 3: a simulation based proof of security (input extraction?).
- And many more issues…

Proving consistency of P_1's inputs

- We use cut-and-choose to prove consistency of *commitment sets* of P_1's inputs
  - And combine it with the cut-and-choose test used to prove consistency of circuits
  - (two "cut-and-choose"s)
- P_1 generates for each of its input wires *s* pairs of commitments sets. In each pair:
  - One set contains commitments to the garbled value of 0 for this wire, in all *s* circuits.
  - The other set contains commitments to the garbled value of 1 for this wire, in all *s* circuits.
  - The order of the pairs is random
  - P_2 receives a total of $n \cdot s \cdot s$ commitments

The commitment sets corresponding to P_1's first input wire

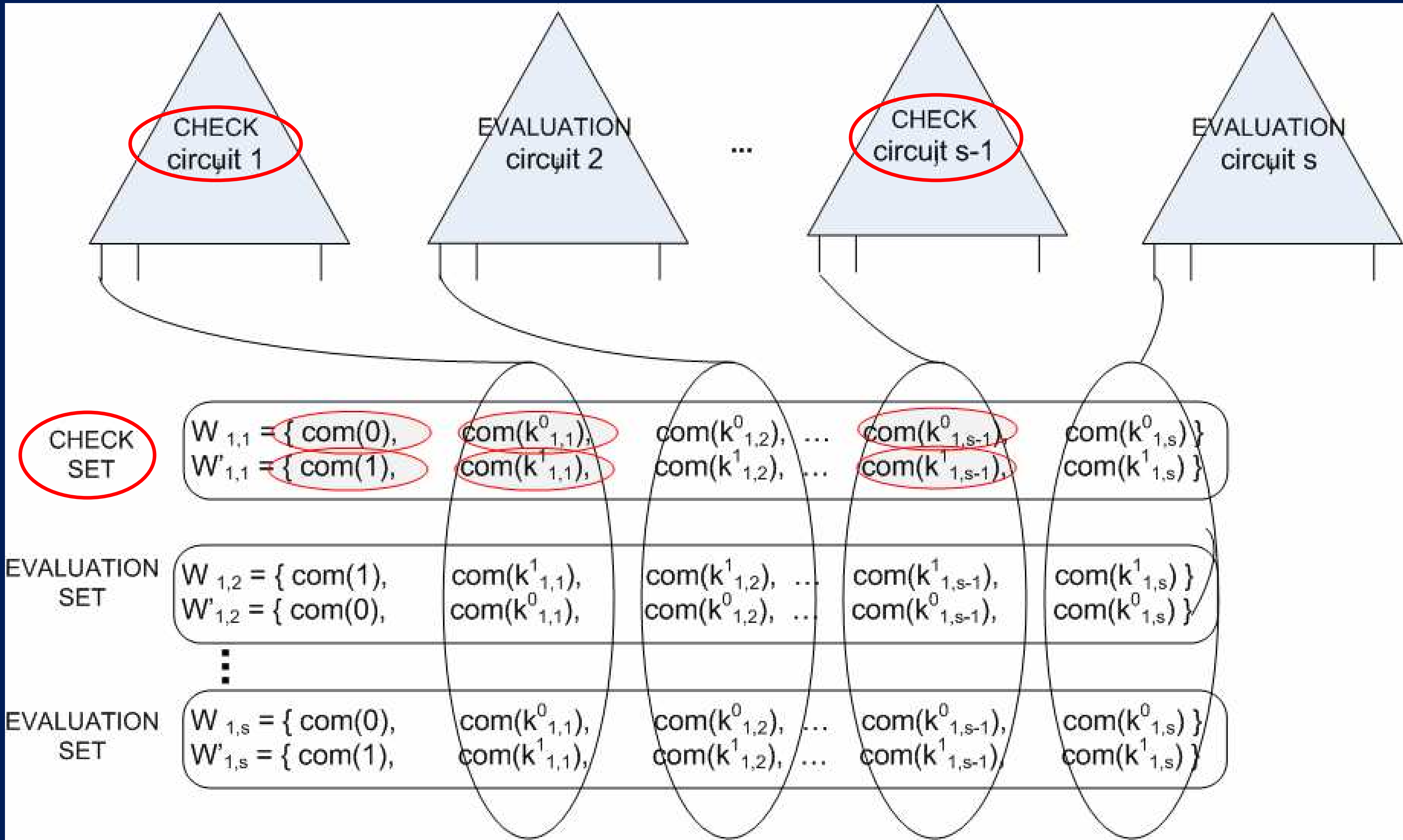


Proving consistency of P_1's inputs

- P_1 sends to P_2 the s garbled circuits and the $n \cdot s$ commitment sets

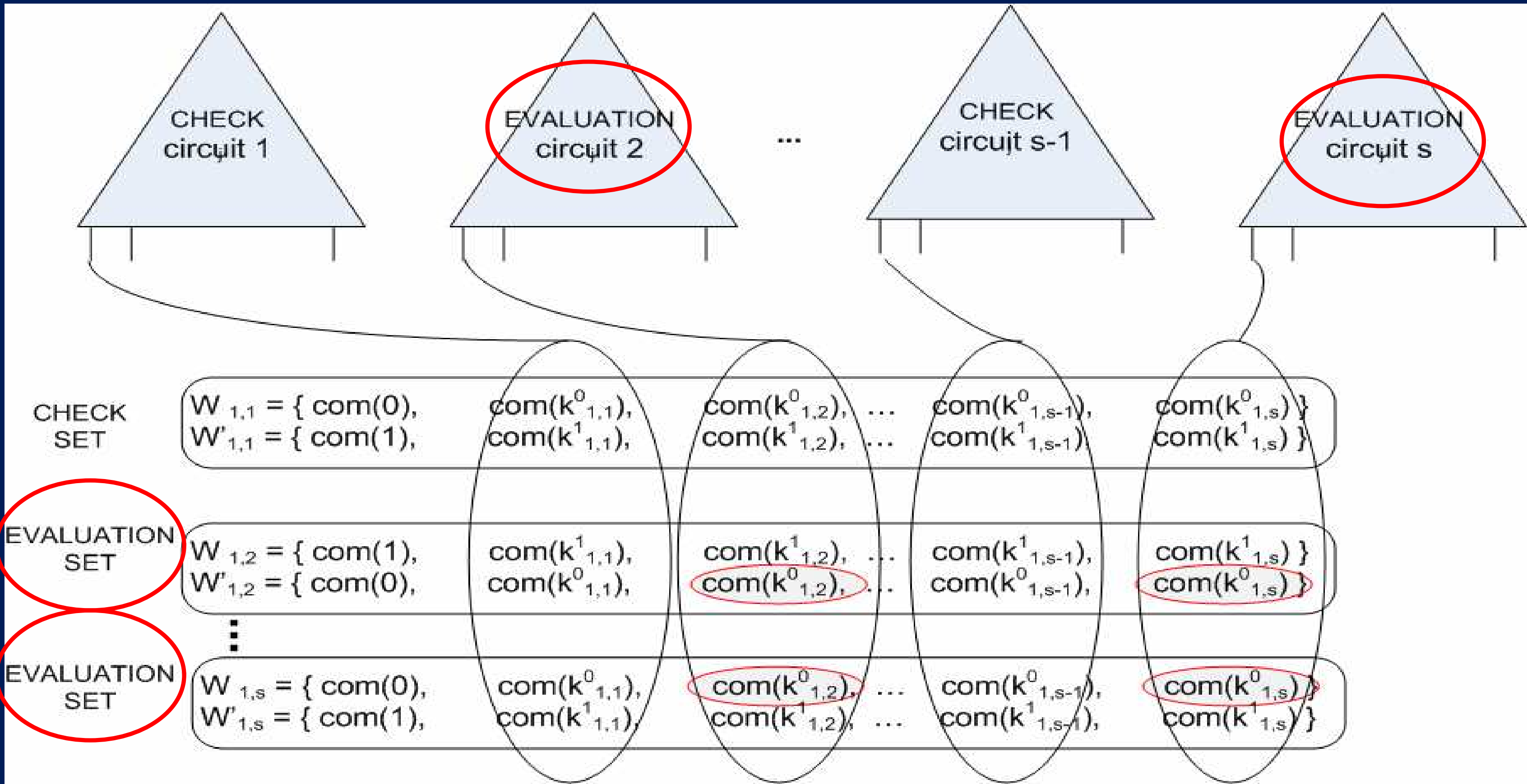
- The parties jointly pick random strings
  - $\rho \in \{0,1\}^s$ decides which circuits will be checked and which will be evaluated
  - $\rho' \in \{0,1\}^s$ decides which commitment sets will be checked and which will be evaluated

P_1 opens in *checked sets* the commitments to values in *checked circuits*



Evaluation

- P_1 opens the commitments in *evaluation sets,* for the garbled values of P_1's input in *evaluation circuits*. P_2 verifies that these values are consistent (row wise and column wise).



Why does this prove consistency of P_1's inputs?

- Suppose that P_1 wants an input bit to be 0 in circuit C_i and 1 in C_j.
  - If C_i and C_j are evaluated circuits then all evaluation sets must contain a commitment to 0 for C_i and a commitment to 1 for C_j.
  - If C_i and C_j are checked circuits then their values must be equal in all checked sets .

- Since P_2 outputs the majority result, P_1's cheating is effective only if applied to > s/4 circuits.
  - Therefore P_1 must guess exactly which circuits and which sets will be checked $\Rightarrow$ exponentially small success probability.

What about P_2's inputs?

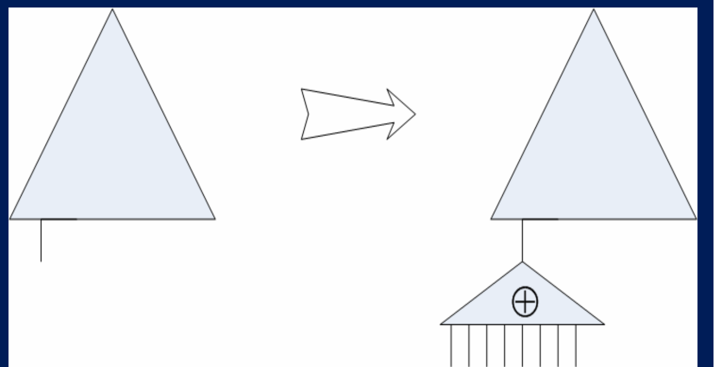
- Seems easy
  - P_2 uses OT to learn them

- But, P_1 can cheat in the OT protocol [KS]:
  - It can provide corrupt decommitment keys for the choice corresponding to a 0 input value, and good keys for 1.
  - If P_2's input is 1 all checks go well.
  - If P_2's input is 0, it cannot open the garbled values and must abort!

Preventing the OT attack

- An easy fix: Replace each of P_2's input bits with the xor of *s* new input bits of P_2.
  - P_2 assigns to the new bits random values whose xor is the original input bit.
  - P_2 aborts if the decommitment keys to any bit are corrupt
  - P_2's abort probability is almost independent of its input:
    - If P_1 corrupts < s new bits, the probability of P_2 aborting is *independent* of whether its original input is 0 or 1.
    - P_1 must corrupt s bits and gains advantage of $2^{-(s-1)}$ in guessing P_2's input
- Caveat: Number of OTs multiplied by s.

  Solution: Use coding to replace n original bits with only 4n new ones.



Security definition

- Simulation of a real execution in the ideal model
  - Any admissible adversary in the real model can be simulated by an adversary in the ideal model
  - And therefore cannot learn more than is leaked in the ideal model.
  - The exact definition is more complex [C,G]

- Security is proved in the hybrid model, where the OT is implemented by a trusted party [C,G].

Choosing which circuits/sets to open

- This is done in order to check that P_1 is not cheating, so naturally P_2 can choose which commitments to open.
  - This is sufficient in order to handle a malicious P_1.
- However, in this case we don't know how to prove simulation in in case of a malicious P_2 … (the proof requires to cheat P_2 in the simulation)
- The parties therefore run a joint coin-tossing protocol:
  - P_2 commits to a random ρ_2
  - P_1 commits to a random ρ_1
  - P_2 decommits and reveals ρ_2
  - P_1 decommits and reveals ρ_1
  - $\rho = \rho_1 \oplus \rho_2$ is used to decide which circuits to open

OTs are done before the circuits commitments are sent to P_2

- This is done in order to enable us to prove security against a malicious P_2
  - In the simulation, we extract P_2's input to F from its inputs to the OT.
  - We can send this value to the trusted party and learn the resulting output
  - Then, construct s/2 circuits which always output this value
  - And cheat in the joint coin flipping to ensure P_2 evaluates only these circuits

- This is the essence of the proof for the case P_2 is corrupt.

Security against a corrupt P_1

- Construct a simulator which gets access to the corrupt P_1 and to the trusted party, and emulates the behavior of a corrupt P_1 in a real execution:
  - Receive the circuit commitments from P_1
  - Run the protocol to obtain a random ρ (deciding which circuits are opened). Perform P_2's checks.
  - Rewind, and run again with a different ρ^*.
  - Whp, there are many (> s/8) circuits which are checked in the first run and chosen to be evaluated in the second.
  - We can learn P_1's input to these circuits. Since the checks before went well, whp this is the input of sufficiently many circuits.
  - We send this input X of P_1 to the TTP and learn F(X,Y).

Conclusions

- Security in the ideal/real simulation paradigm for Yao's protocol
  - The basic protocol structure is kept. More copies are sent, to perform cut-and-choose. Several tweaks needed for the proof to go through.
  - The same number (order) of public key operations.
  - The proof is complicated but the protocol is efficient
    - O(1) public key operations per input bit, O(1) rounds.
    - $O(|C| \cdot s + n \cdot s^2)$ communication.
    - Woodruff shows how to use expanders to achieve $O(|C| \cdot s)$ communication for [MF]. Can probably also be applied here.
- THM: Constant-round black-box reduction of secure two-party computation to oblivious transfer and perfectly hiding commitments.
- Implementation?