

CS 380S

0x1A Great Papers in Computer Security

Vitaly Shmatikov

<http://www.cs.utexas.edu/~shmat/courses/cs380s/>

Secure Multi-Party Computation

- ◆ General framework for describing computation between parties who do not trust each other
- ◆ Example: elections
 - N parties, each one has a “Yes” or “No” vote
 - Goal: determine whether the majority voted “Yes”, but no voter should learn how other people voted
- ◆ Example: auctions
 - Each bidder makes an offer
 - Goal: determine whose offer won without revealing losing offers

More Examples

◆ Example: distributed data mining

- Two companies want to compare their datasets without revealing them
 - For example, compute the intersection of two lists of names

◆ Example: database privacy

- Evaluate a query on the database without revealing the query to the database owner
- Evaluate a statistical query without revealing the values of individual entries

A Couple of Observations

- ◆ In all cases, we are dealing with **distributed multi-party protocols**
 - A protocol describes how parties are supposed to exchange messages on the network
- ◆ All of these tasks can be easily computed by a trusted third party
 - The goal of secure multi-party computation is to achieve the same result without involving a trusted third party

How to Define Security?

- ◆ Must be mathematically rigorous
- ◆ Must capture all realistic attacks that a malicious participant may try to stage
- ◆ Should be “abstract”
 - Based on the desired “functionality” of the protocol, not a specific protocol
 - Goal: define security for an entire class of protocols

Functionality

- ◆ K mutually distrustful parties want to jointly carry out some task
- ◆ Model this task as a functionality

$$f: (\{0,1\}^*)^K \rightarrow (\{0,1\}^*)^K$$

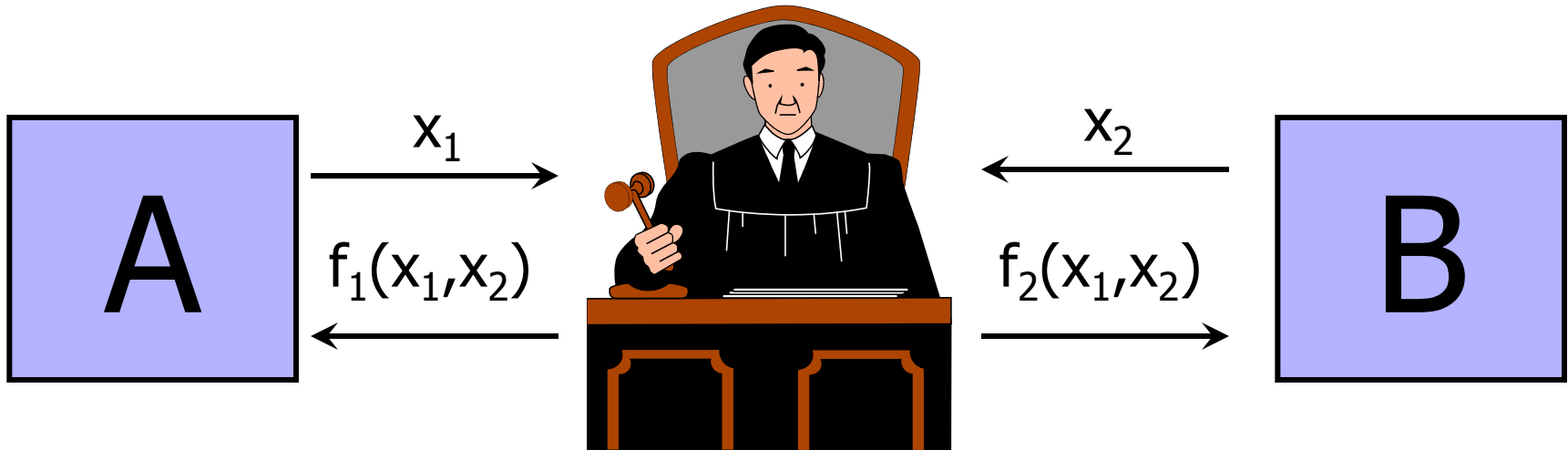
K inputs (one per party);
each input is a bitstring

K outputs

- ◆ Assume that this functionality is computable in probabilistic polynomial time

Ideal Model

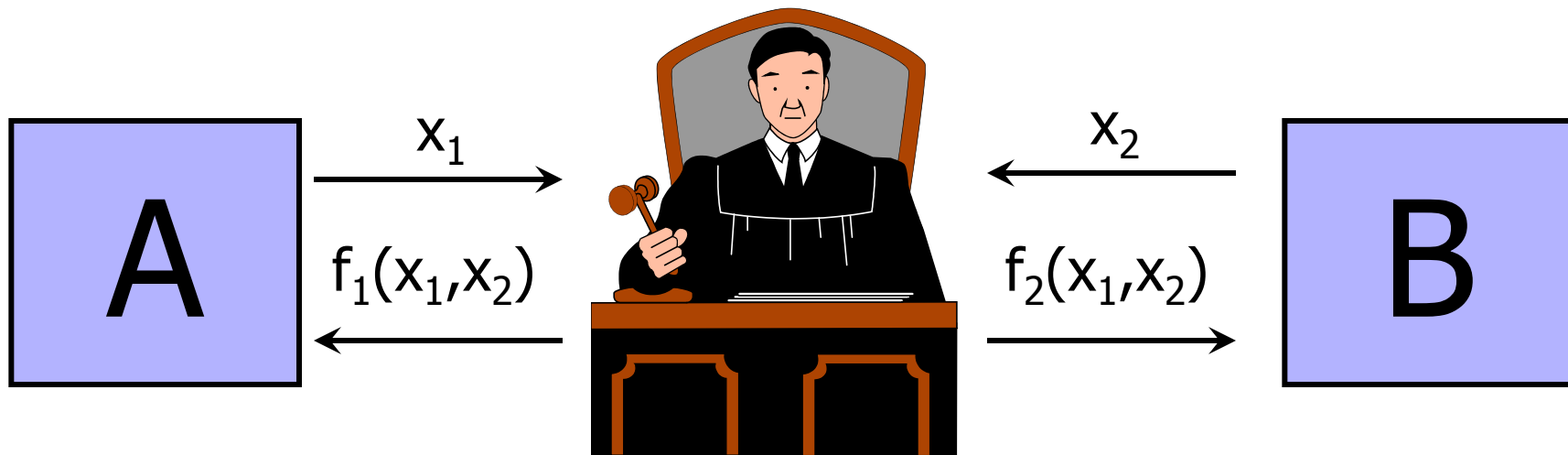
- ◆ Intuitively, we want the protocol to behave “as if” a trusted third party collected the parties’ inputs and computed the desired functionality
 - Computation in the ideal model is secure by definition!



Slightly More Formally

- ◆ A protocol is secure if it **emulates** an ideal setting where the parties hand their inputs to a “trusted party,” who locally computes the desired outputs and hands them back to the parties

[Goldreich-Micali-Wigderson 1987]



Adversary Models

- ◆ Some participants may be dishonest (corrupt)
 - If all were honest, we would not need secure multi-party computation
- ◆ **Semi-honest** (aka passive; honest-but-curious)
 - Follows protocol, but tries to learn more from received messages than he would learn in the ideal model
- ◆ **Malicious**
 - Deviates from the protocol in arbitrary ways, lies about his inputs, may quit at any point
- ◆ For now, focus on semi-honest adversaries and two-party protocols

Correctness and Security

- ◆ How do we argue that the real protocol “emulates” the ideal protocol?
- ◆ Correctness
 - All honest participants should receive the correct result of evaluating functionality f
 - Because a trusted third party would compute f correctly
- ◆ Security
 - All corrupt participants should learn no more from the protocol than what they would learn in the ideal model
 - What does corrupt participant learn in ideal model?
 - His own input and the result of evaluating f

Simulation

- ◆ Corrupt participant's **view** of the protocol = record of messages sent and received
 - In the ideal world, view consists simply of his input and the result of evaluating f
- ◆ How to argue that real protocol does not leak more useful information than ideal-world view?
- ◆ Key idea: **simulation**
 - If real-world view (i.e., messages received in the real protocol) can be simulated with access only to the ideal-world view, then real-world protocol is secure
 - Simulation must be indistinguishable from real view

Technicalities

- ◆ **Distance** between probability distributions A and B over a common set X is

$$\frac{1}{2} * \sum_x (|\Pr(A=x) - \Pr(B=x)|)$$

- ◆ **Probability ensemble** A_i is a set of discrete probability distributions

- Index i ranges over some set I

- ◆ Function $f(n)$ is **negligible** if it is asymptotically smaller than the inverse of any polynomial

$$\forall \text{ constant } c \exists m \text{ such that } |f(n)| < 1/n^c \quad \forall n > m$$

Notions of Indistinguishability

- ◆ Distribution ensembles A_i and B_i are **equal**
- ◆ Distribution ensembles A_i and B_i are **statistically close** if $\text{dist}(A_i, B_i)$ is a negligible function of i
- ◆ Distribution ensembles A_i and B_i are **computationally indistinguishable** ($A_i \approx B_i$) if, for any probabilistic polynomial-time algorithm D , $|\Pr(D(A_i)=1) - \Pr(D(B_i)=1)|$ is a negligible function of i
 - No efficient algorithm can tell the difference between A_i and B_i except with a negligible probability

SMC Definition (First Attempt)

- ◆ Protocol for computing $f(X_A, X_B)$ betw. A and B is secure if there exist efficient simulator algorithms S_A and S_B such that for all input pairs $(x_A, x_B) \dots$
- ◆ Correctness: $(Y_A, Y_B) \approx f(x_A, x_B)$
 - Intuition: outputs received by honest parties are indistinguishable from the correct result of evaluating f
- ◆ Security: $\text{view}_A(\text{real protocol}) \approx S_A(x_A, Y_A)$
 $\text{view}_B(\text{real protocol}) \approx S_B(x_B, Y_B)$
 - Intuition: a corrupt party's view of the protocol can be simulated from its input and output
- ◆ This definition does not work! Why?

Randomized Ideal Functionality

- ◆ Consider a coin flipping functionality
 - $f()=(b,-)$ where b is random bit
 - $f()$ flips a coin and tells A the result; B learns nothing
- ◆ The following protocol “implements” $f()$
 1. A chooses bit b randomly
 2. A sends b to B
 3. A outputs b
- ◆ It is obviously insecure (why?)
- ◆ Yet it is correct and simulatable according to our attempted definition (why?)

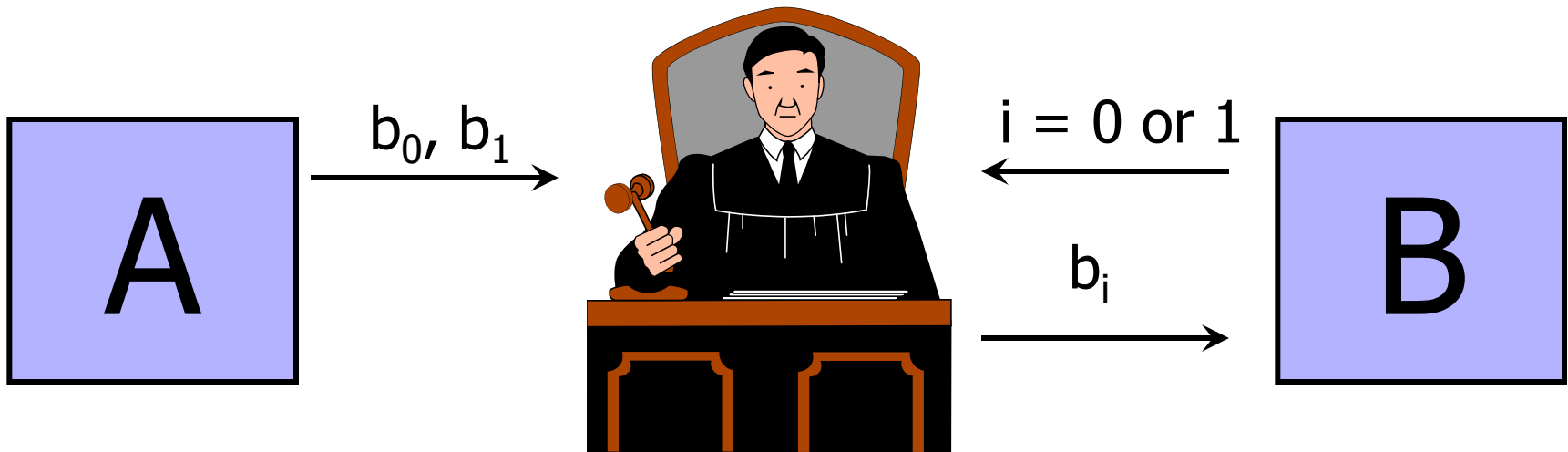
SMC Definition

- ◆ Protocol for computing $f(X_A, X_B)$ betw. A and B is secure if there exist efficient simulator algorithms S_A and S_B such that for all input pairs $(x_A, x_B) \dots$
- ◆ Correctness: $(Y_A, Y_B) \approx f(x_A, x_B)$
- ◆ Security: $(\text{view}_A(\text{real protocol}), Y_B) \approx (S_A(x_A, Y_A), Y_B)$
 $(\text{view}_B(\text{real protocol}), Y_A) \approx (S_B(x_B, Y_B), Y_A)$
 - Intuition: if a corrupt party's view of the protocol is correlated with the honest party's output, the simulator must be able to capture this correlation
- ◆ Does this fix the problem with coin-flipping f?

Oblivious Transfer (OT)

[Rabin 1981]

◆ Fundamental SMC primitive



- A inputs two bits, B inputs the index of one of A's bits
- B learns his chosen bit, A learns nothing
 - A does not learn which bit B has chosen; B does not learn the value of the bit that he did not choose
- Generalizes to bitstrings, M instead of 2, etc.

One-Way Trapdoor Functions

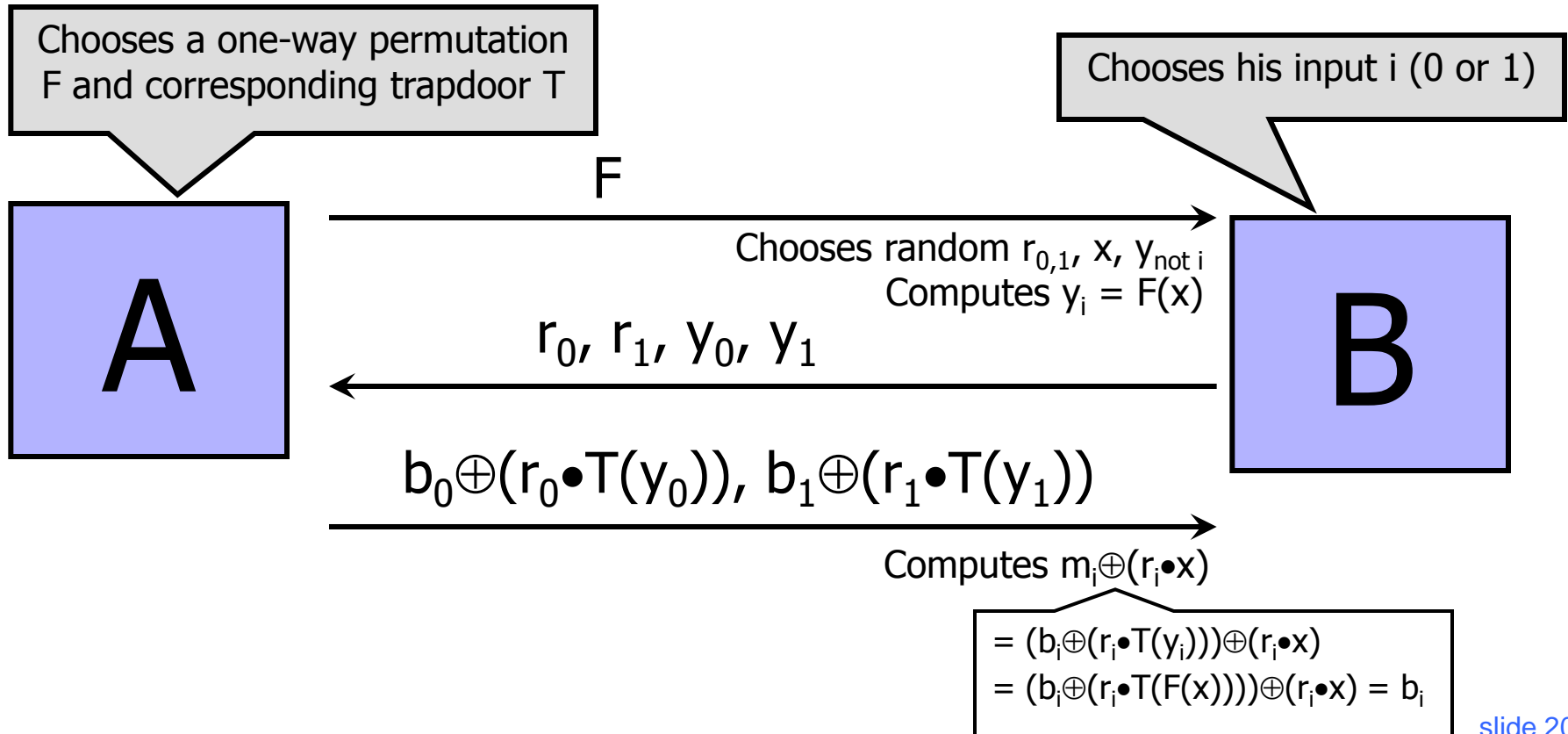
- ◆ Intuition: **one-way functions** are easy to compute, but hard to invert (skip formal definition)
 - We will be interested in one-way permutations
- ◆ Intuition: **one-way trapdoor functions** are one-way functions that are easy to invert given some extra information called the trapdoor
 - Example: if $n=pq$ where p and q are large primes and e is relatively prime to $\varphi(n)$, $f_{e,n}(m) = m^e \bmod n$ is easy to compute, but it is believed to be hard to invert
 - Given the trapdoor d s.t. $de=1 \bmod \varphi(n)$, $f_{e,n}(m)$ is easy to invert because $f_{e,n}(m)^d = (m^e)^d = m \bmod n$

Hard-Core Predicates

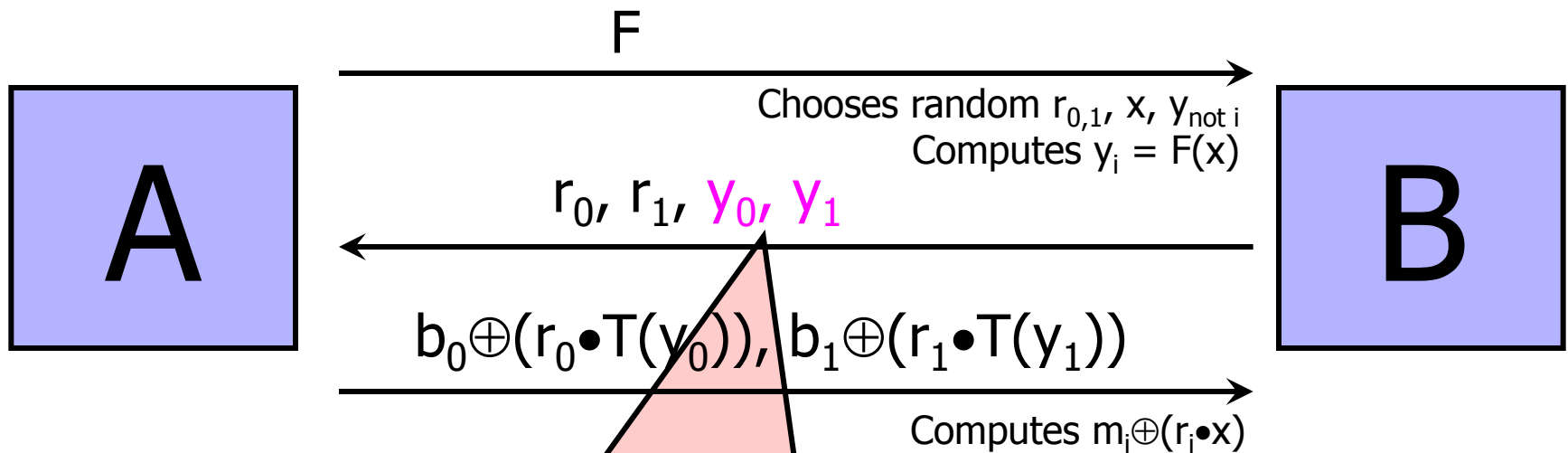
- ◆ Let $f: S \rightarrow S$ be a one-way function on some set S
- ◆ $B: S \rightarrow \{0,1\}$ is a **hard-core predicate** for f if
 - Intuition: there is a bit of information about x such that learning this bit from $f(x)$ is as hard as inverting f
 - $B(x)$ is easy to compute given $x \in S$
 - If an algorithm, given only $f(x)$, computes $B(x)$ correctly with prob $> 1/2 + \epsilon$, it can be used to invert $f(x)$ easily
 - Consequence: $B(x)$ is hard to compute given only $f(x)$
- ◆ Goldreich-Levin theorem
 - $B(x,r) = r \bullet x$ is a hard-core predicate for $g(x,r) = (f(x), r)$
 - $f(x)$ is any one-way function, $r \bullet x = (r_1 x_1) \oplus \dots \oplus (r_n x_n)$

Oblivious Transfer Protocol

- ◆ Assume the existence of some family of one-way trapdoor permutations



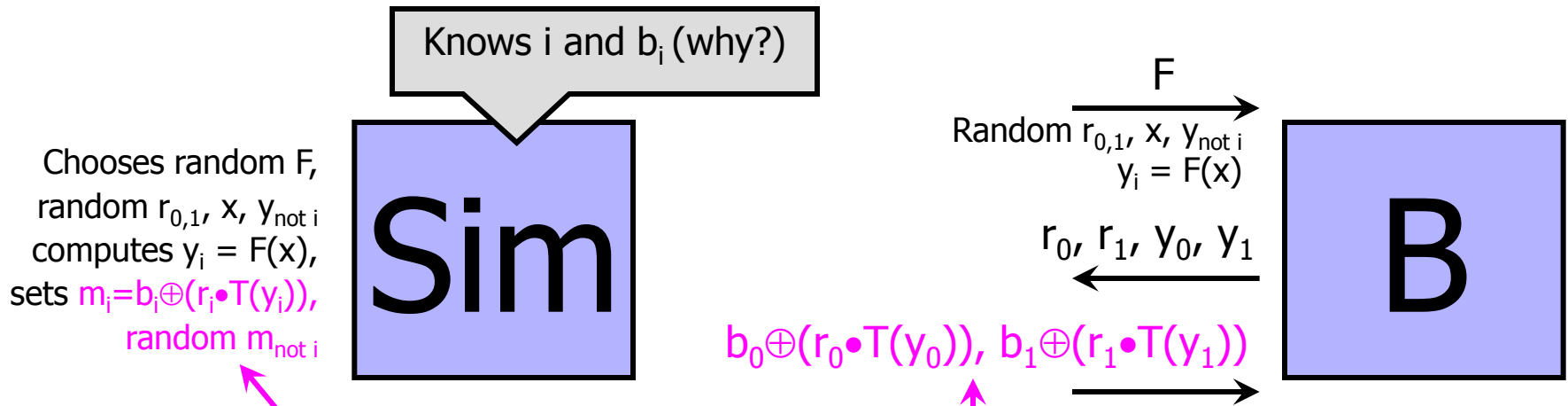
Proof of Security for B



y_0 and y_1 are uniformly random regardless of A's choice of permutation F (why?).
Therefore, A's view is independent of B's input i .

Proof of Security for A (Sketch)

- ◆ Need to build a simulator whose output is indistinguishable from B's view of the protocol



The only difference between simulation and real protocol:
 In simulation, $m_{\text{not } i}$ is random (why?)

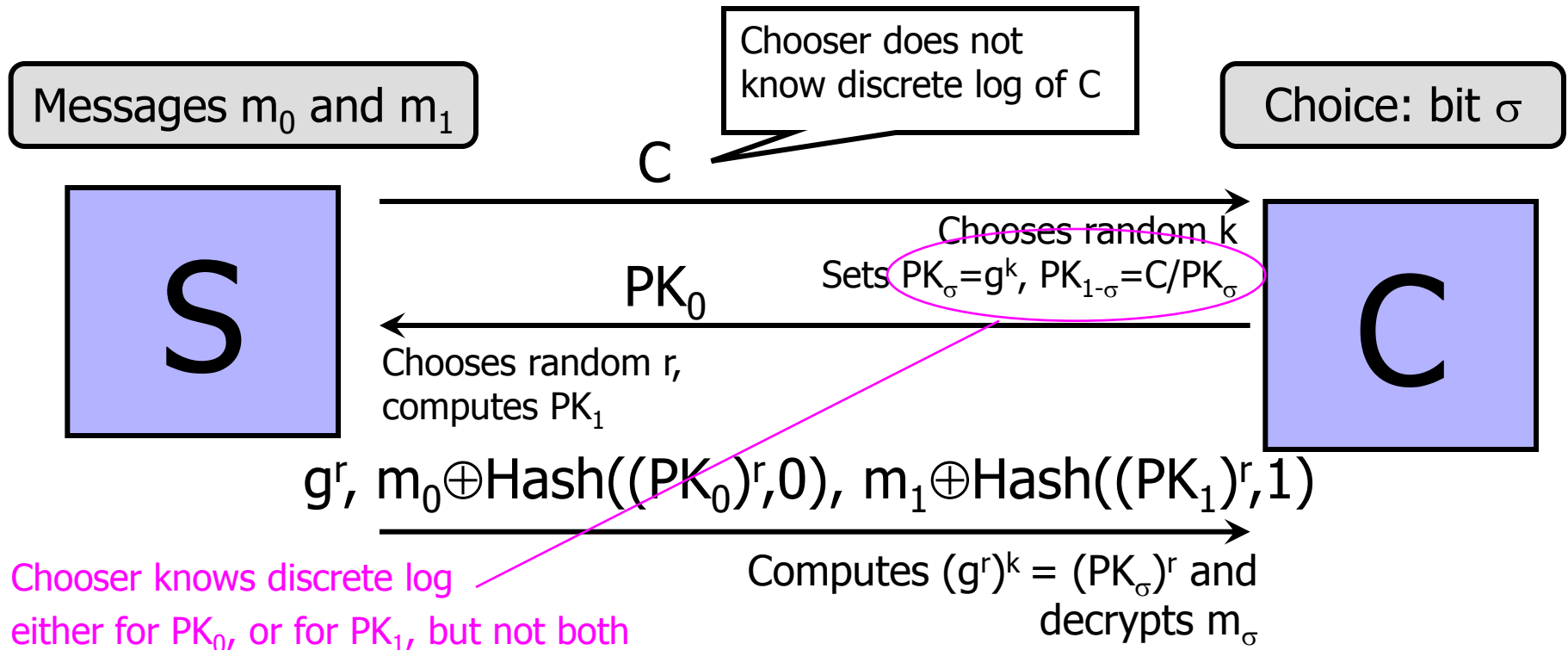
In real protocol, $m_{\text{not } i} = b_{\text{not } i} \oplus (r_{\text{not } i} \bullet T(y_{\text{not } i}))$

Proof of Security for A (Cont'd)

- ◆ Why is it computationally infeasible to distinguish random m and $m' = b \oplus (r \bullet T(y))$?
 - b is some bit, r and y are random, T is the trapdoor of a one-way trapdoor permutation
- ◆ $(r \bullet x)$ is a hard-core bit for $g(x, r) = (F(x), r)$
 - This means that $(r \bullet x)$ is hard to compute given $F(x)$
- ◆ If B can distinguish m and $m' = b \oplus (r \bullet x')$ given only $y = F(x')$, we obtain a contradiction with the fact that $(r \bullet x')$ is a hard-core bit
 - Proof omitted

Naor-Pinkas Oblivious Transfer

Setting: order- q subgroup of Z^*_p , p is prime, q divides $p-1$
 g is a generator group for which CDH assumption holds



Chooser does not know the discrete log of $PK_{1-\sigma}$, thus cannot distinguish between a random value g_z and $(PK_{1-\sigma})^r$ - **why?**

A. Yao

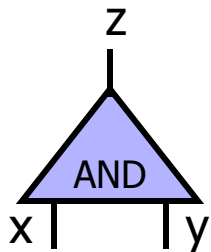
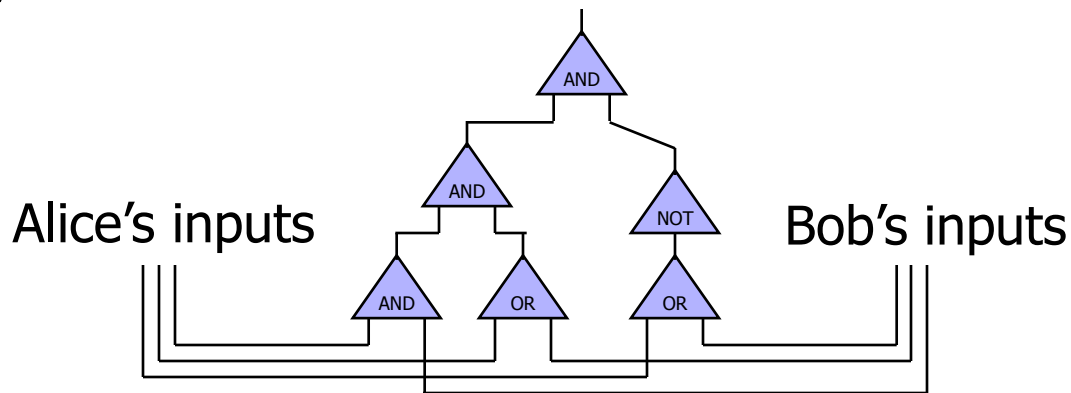
Protocols for Secure Computations

(FOCS 1982)



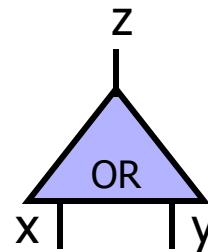
Yao's Protocol

- ◆ Compute **any** function securely
 - ... in the semi-honest model; can be extended to malicious
- ◆ First, convert the function into a **boolean circuit**



Truth table:

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

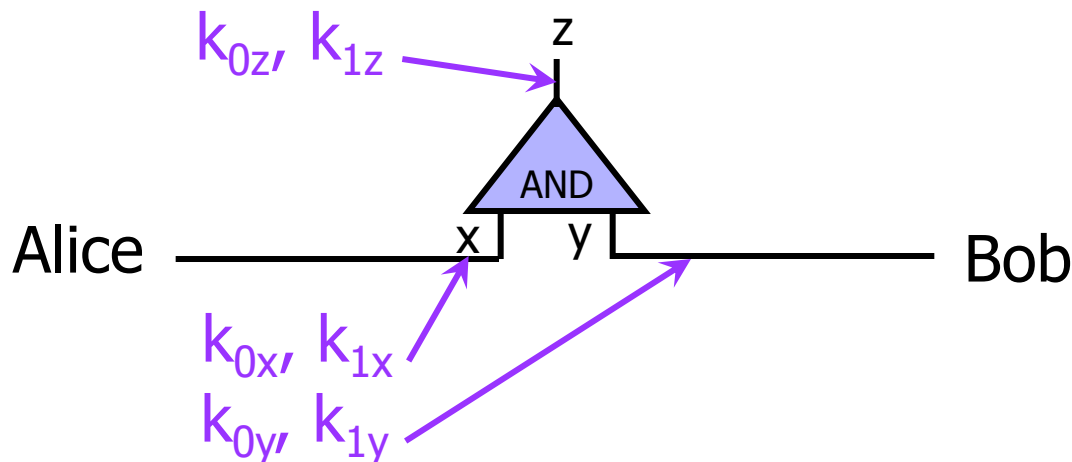


Truth table:

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

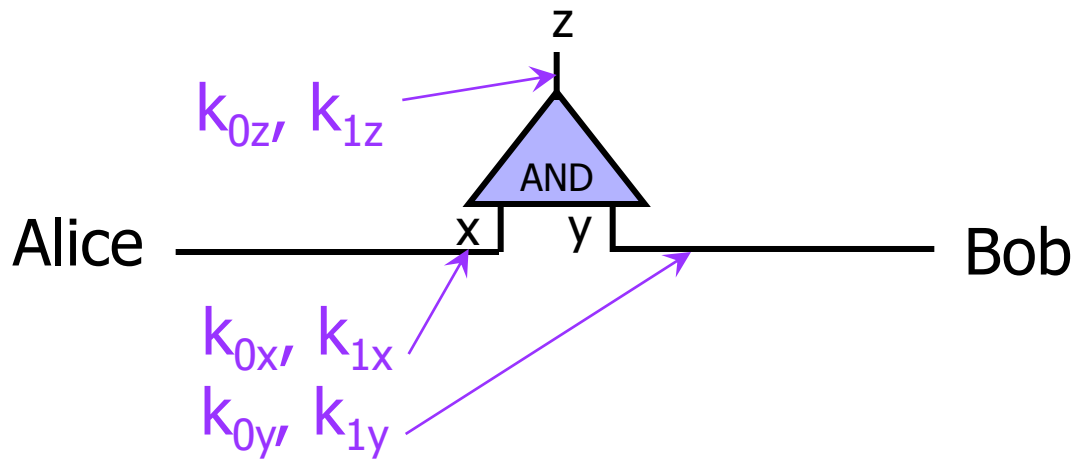
1: Pick Random Keys For Each Wire

- ◆ Evaluate one gate securely
 - Will later generalize to the entire circuit
- ◆ Alice picks two **random keys** for each wire
 - One key corresponds to "0", the other to "1"
 - 6 keys in total for a gate with 2 input wires



2: Encrypt Truth Table

- ◆ Alice encrypts each row of the truth table by encrypting the output-wire key with the corresponding pair of input-wire keys



Original truth table:

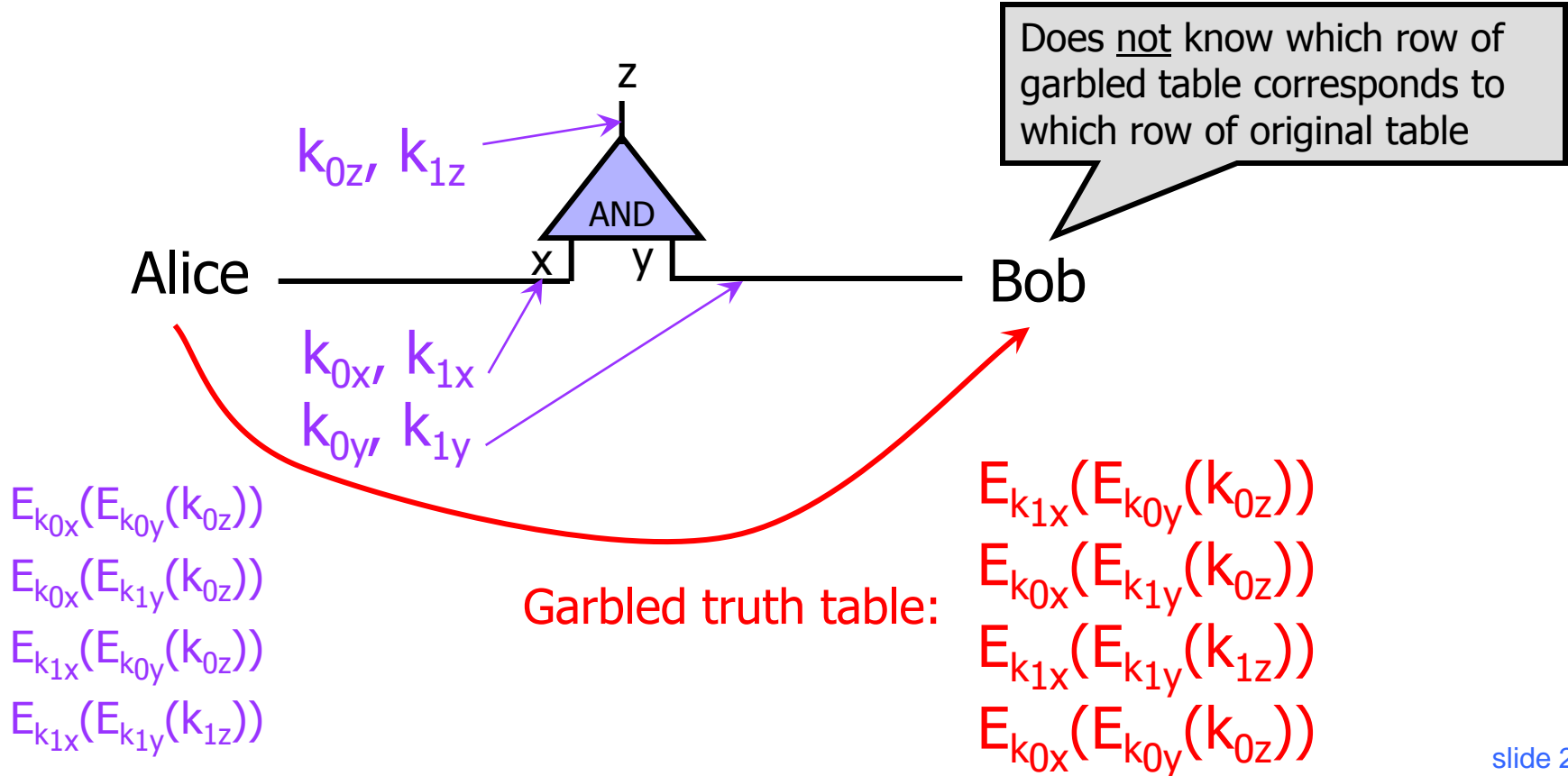
x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

Encrypted truth table:

$$\begin{aligned}
 &E_{k_{0x}}(E_{k_{0y}}(k_{0z})) \\
 &E_{k_{0x}}(E_{k_{1y}}(k_{0z})) \\
 &E_{k_{1x}}(E_{k_{0y}}(k_{0z})) \\
 &E_{k_{1x}}(E_{k_{1y}}(k_{1z}))
 \end{aligned}$$

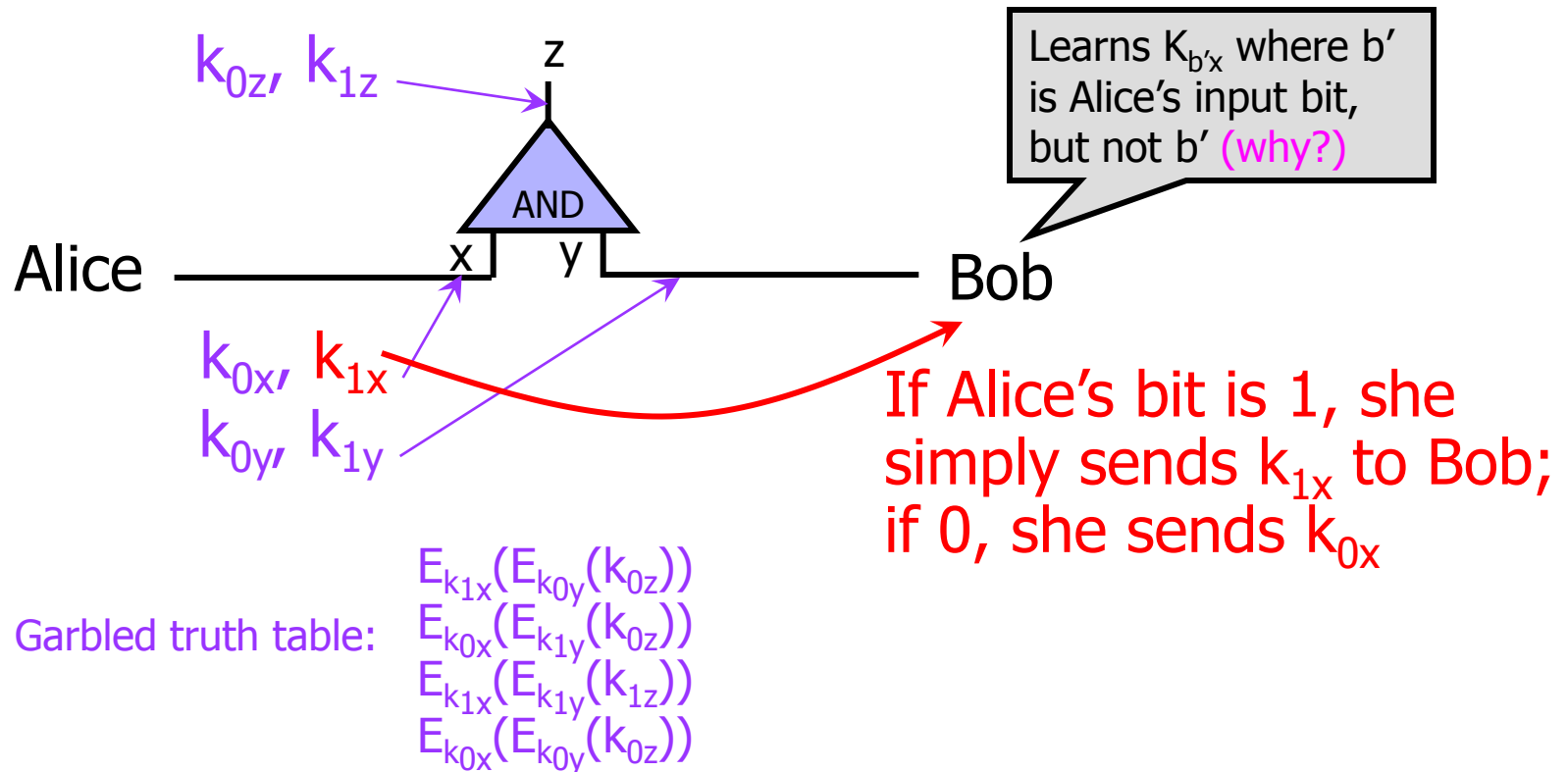
3: Send Garbled Truth Table

- ◆ Alice randomly permutes (“garbles”) encrypted truth table and sends it to Bob



4: Send Keys For Alice's Inputs

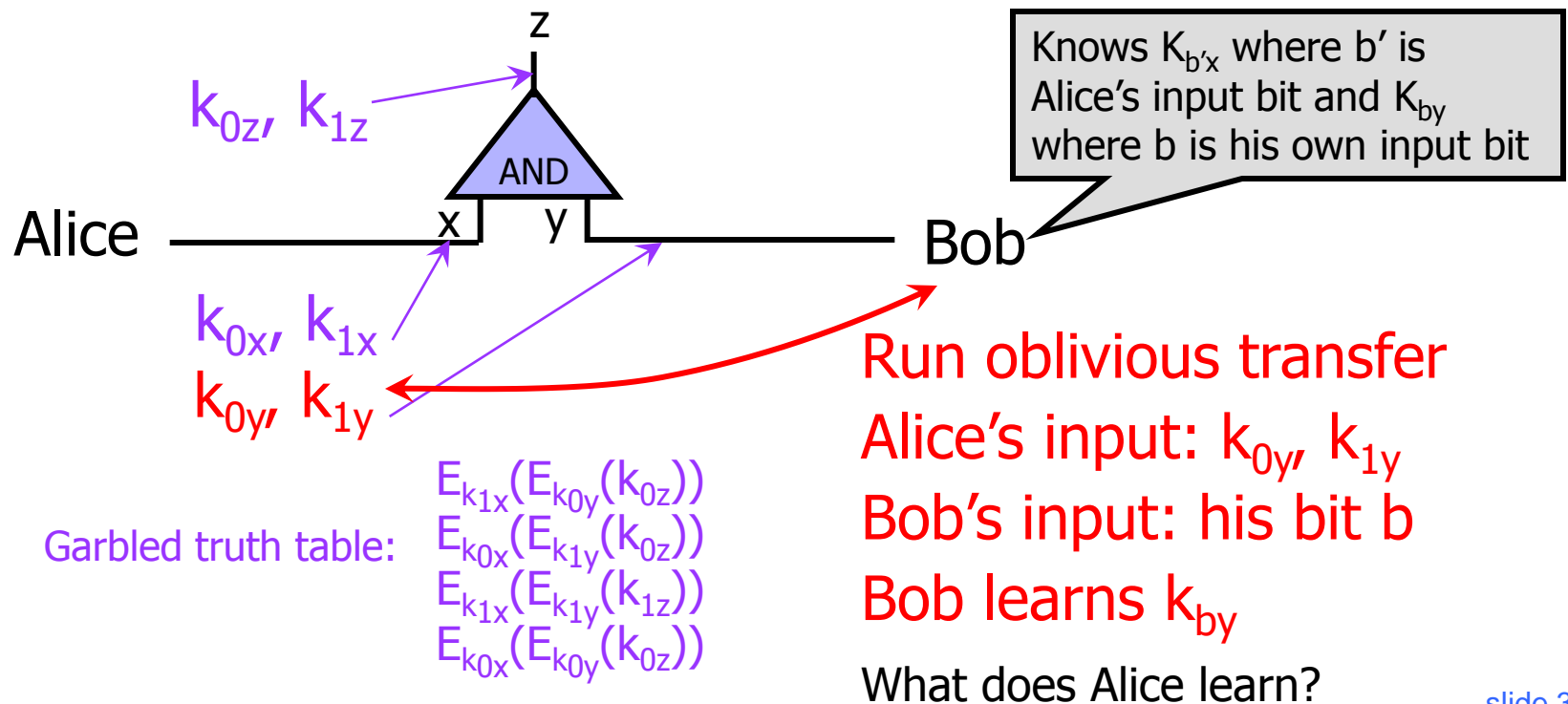
- ◆ Alice sends the key corresponding to her input bit
 - Keys are random, so Bob does not learn what this bit is



5: Use OT on Keys for Bob's Input

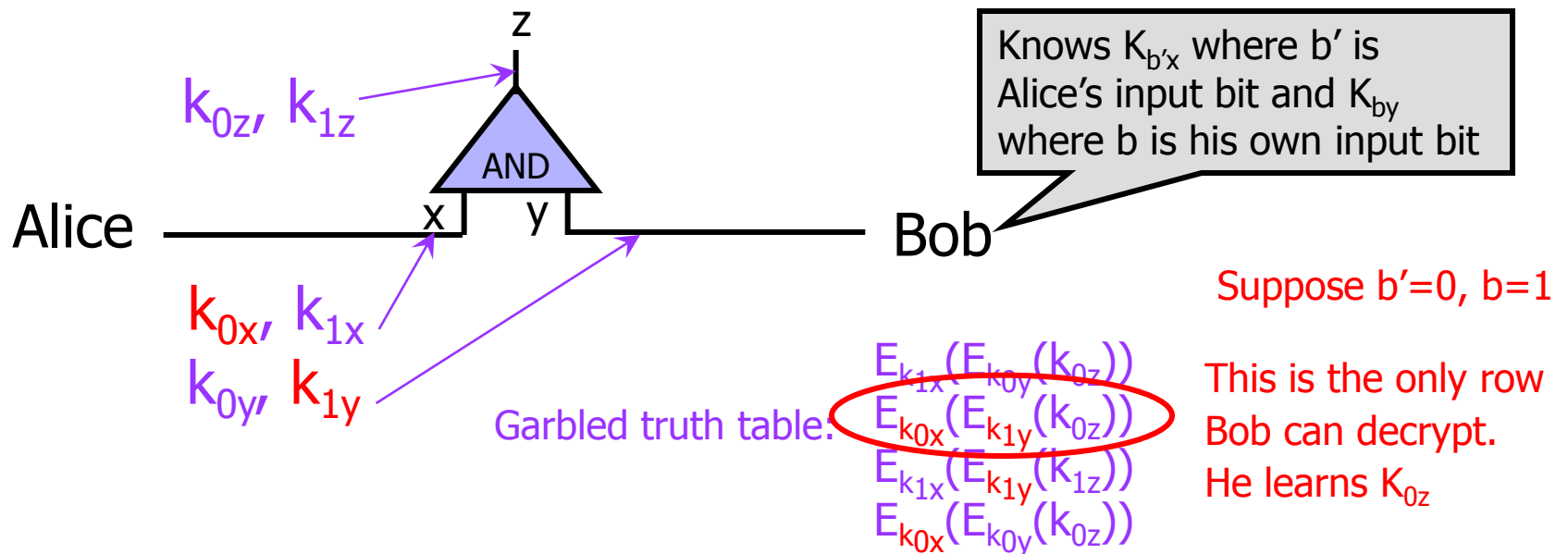
◆ Alice and Bob run oblivious transfer protocol

- Alice's input is the two keys corresponding to Bob's wire
- Bob's input into OT is simply his 1-bit input on that wire



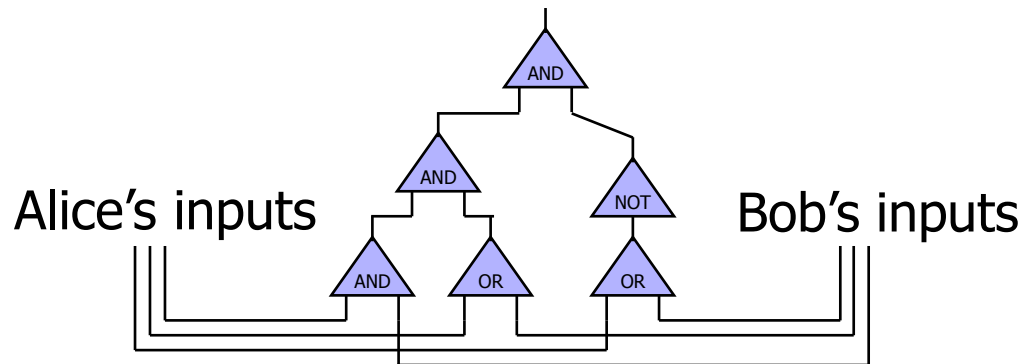
6: Evaluate Garbled Gate

- ◆ Using the two keys that he learned, Bob decrypts exactly one of the output-wire keys
 - Bob does not learn if this key corresponds to 0 or 1
 - Why is this important?



7: Evaluate Entire Circuit

- ◆ In this way, Bob evaluates entire garbled circuit
 - For each wire in the circuit, Bob learns only one key
 - It corresponds to 0 or 1 (Bob does not know which)
 - Therefore, Bob does not learn intermediate values (why?)



- ◆ Bob tells Alice the key for the final output wire and she tells him if it corresponds to 0 or 1
 - Bob does not tell her intermediate wire keys (why?)

Brief Discussion of Yao's Protocol

- ◆ Function must be converted into a circuit
 - For many functions, circuit will be huge
- ◆ If m gates in the circuit and n inputs, then need $4m$ encryptions and n oblivious transfers
 - Oblivious transfers for all inputs can be done in parallel
- ◆ Yao's construction gives a constant-round protocol for secure computation of any function in the semi-honest model
 - Number of rounds does not depend on the number of inputs or the size of the circuit!