MPC School

March 27-29

Wifi: IITB-Wireless | mpc.school.wifi | uMn6wC9m

Monday	11:00 - 12:30	**What is MPC?**	Manoj
	2:00 - 3:00	**Zero Knowledge**	Muthu
	3:30 - 5:00	**Garbled Circuits**	Arpita
Tuesday	9:30 - 11:00	**Randomized Encoding**	Yuval
	11:30 - 12:30	**Oblivious Transfer**	Arpita
	2:00 - 3:30	**Composition**	Muthu
	4:00 - 5:00	**MPC Complexity**	Manoj
Wednesday	9:00 - 10:30	**Honest-Majority MPC**	Vassilis
	11:00 - 12:30	**"MPC in the head"**	Yuval
	2:00 - 3:00	**Asynchronous MPC**	Vassilis

Yuval Ishai
Technion & UCLA

Muthu Venkitasubramaniam
U Rochester

Arpita Patra
IISc

Vassilis Zikas
RPI

Manoj Prabhakaran
IIT Bombay

Secure Multi-Party Computation

What is it?

Manoj Prabhakaran :: IIT Bombay

Must We Trust ebay ?

Must We Trust ebay ?

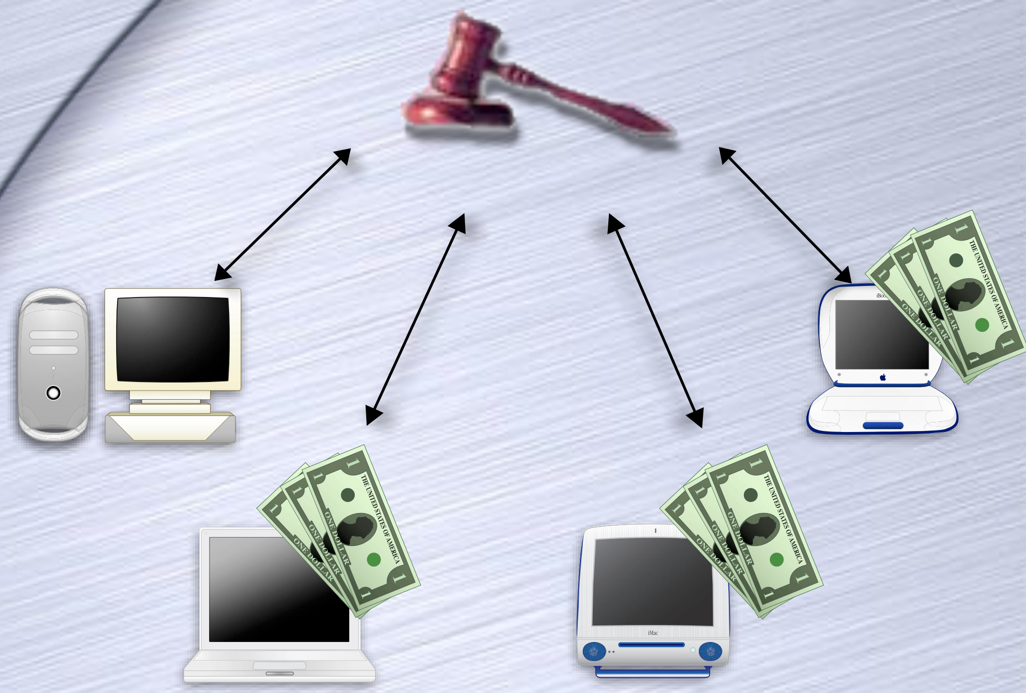
- Can we have an auction without an auctioneer?!

Must We Trust ebaY ?

- Can we have an auction without an auctioneer?!

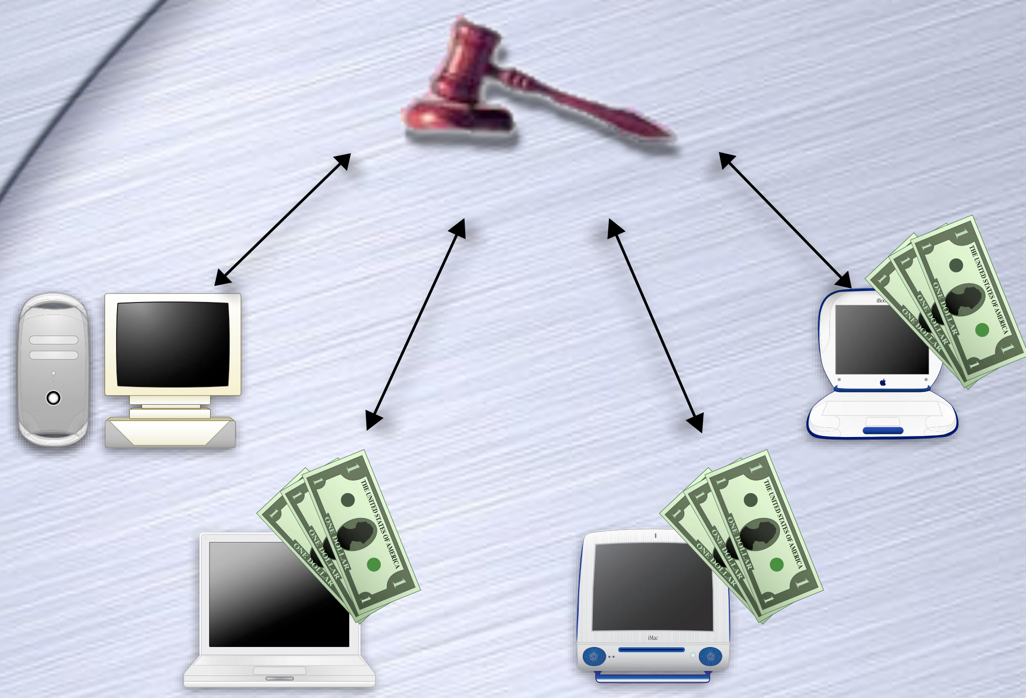
Must We Trust eBay ?

- Can we have an auction without an auctioneer?!



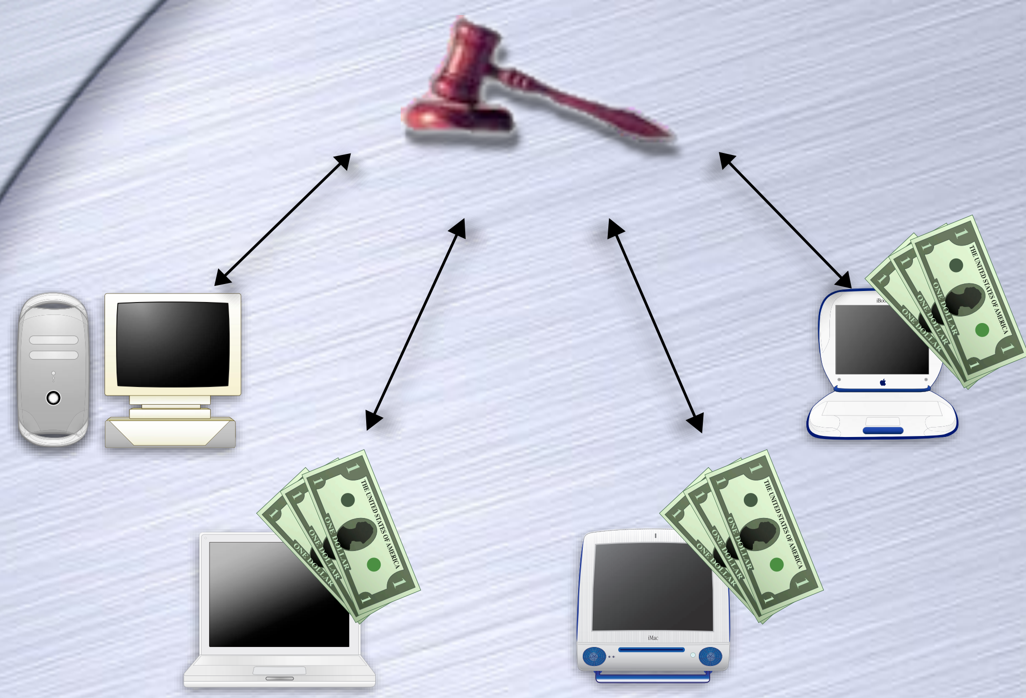
Must We Trust ebay ?

- Can we have an auction without an auctioneer?!
  - Declared winning bid should be correct



Must We Trust eBay ?

- Can we have an auction without an auctioneer?!
  - Declared winning bid should be correct
  - Only the winner and winning bid should be revealed



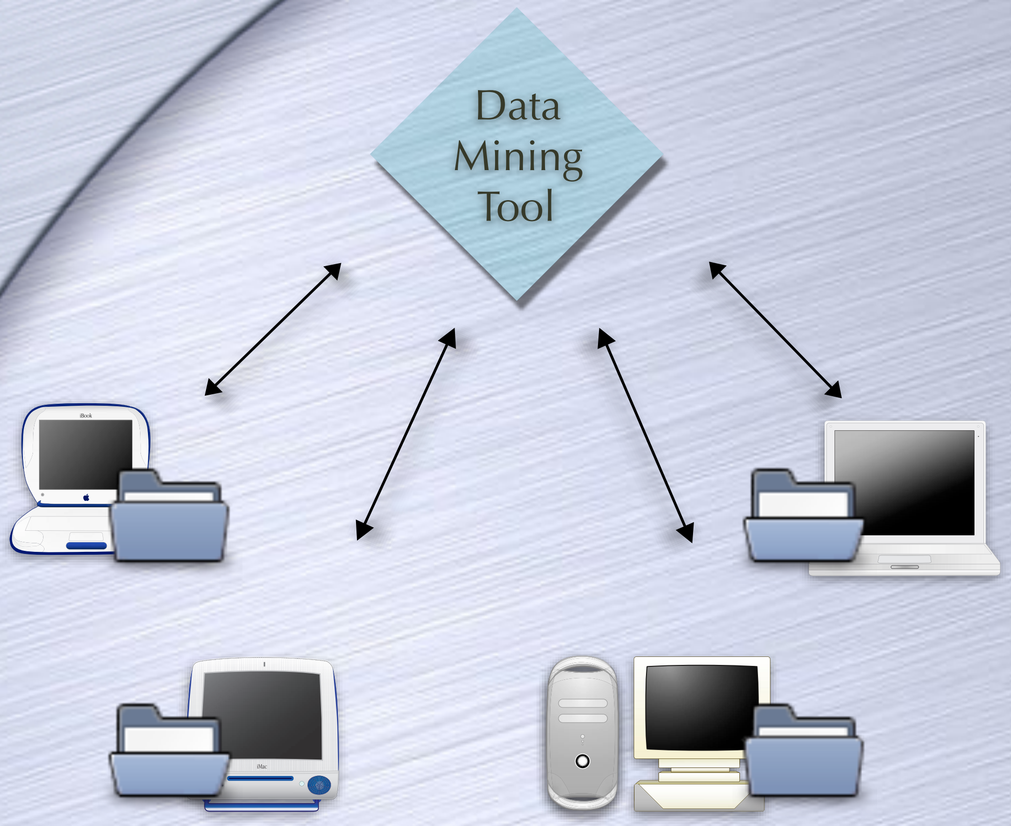
Using data without sharing?

Using data without sharing?

- Hospitals which can't share their patient records with anyone

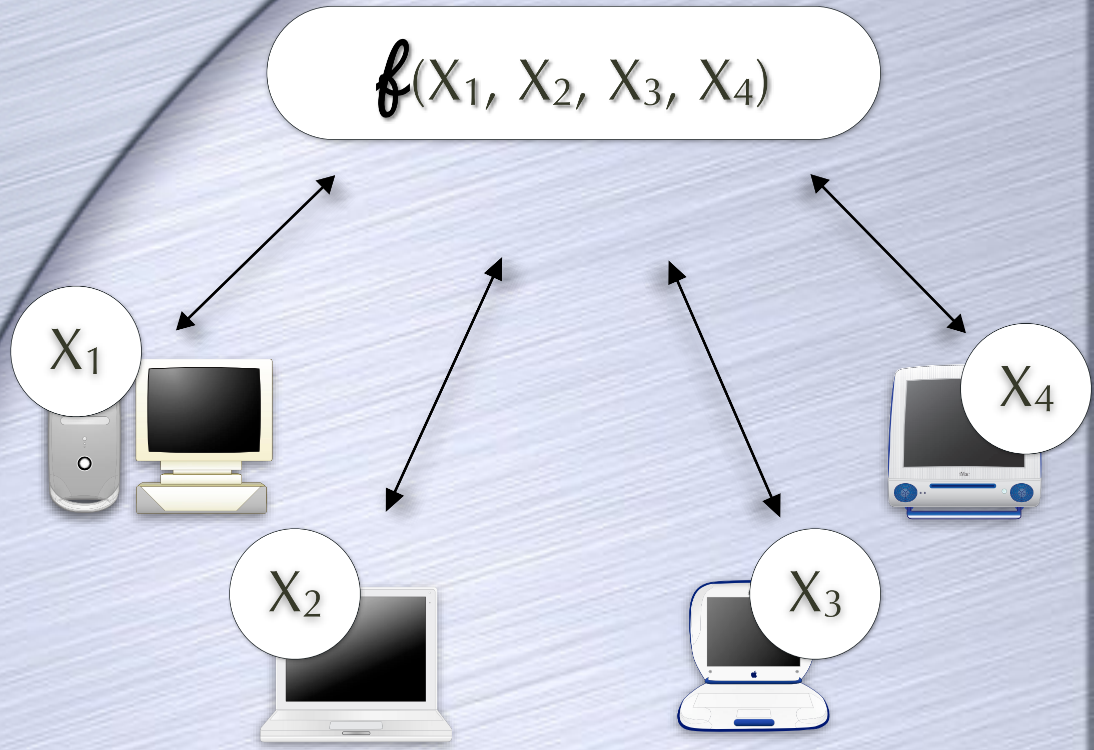
Using data without sharing?

- Hospitals which can't share their patient records with anyone
  - But want to data-mine on combined data



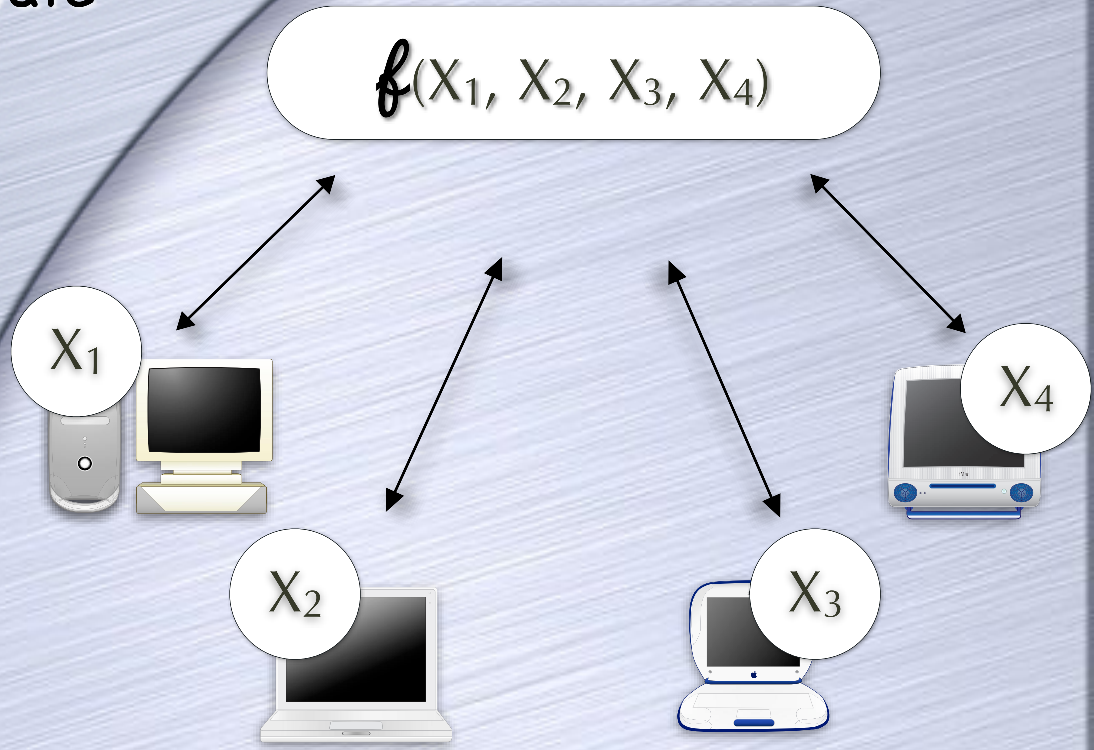
Secure Function Evaluation

- A general problem



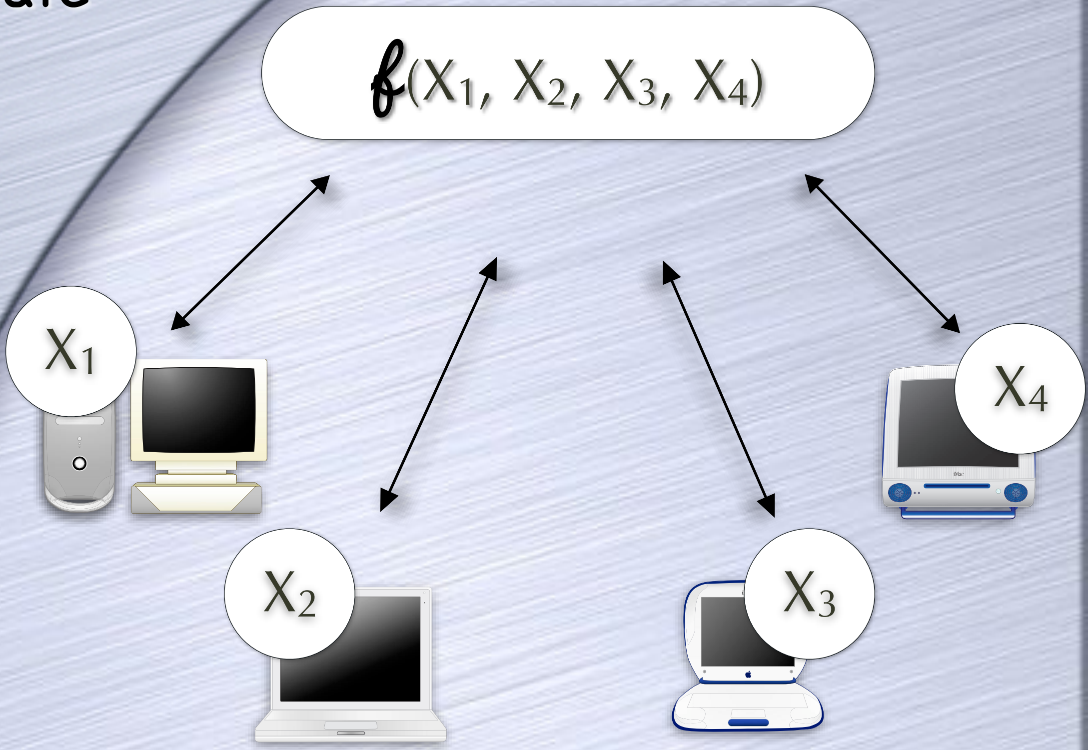
Secure Function Evaluation

- A general problem
- To compute a function of private inputs without revealing information about the inputs



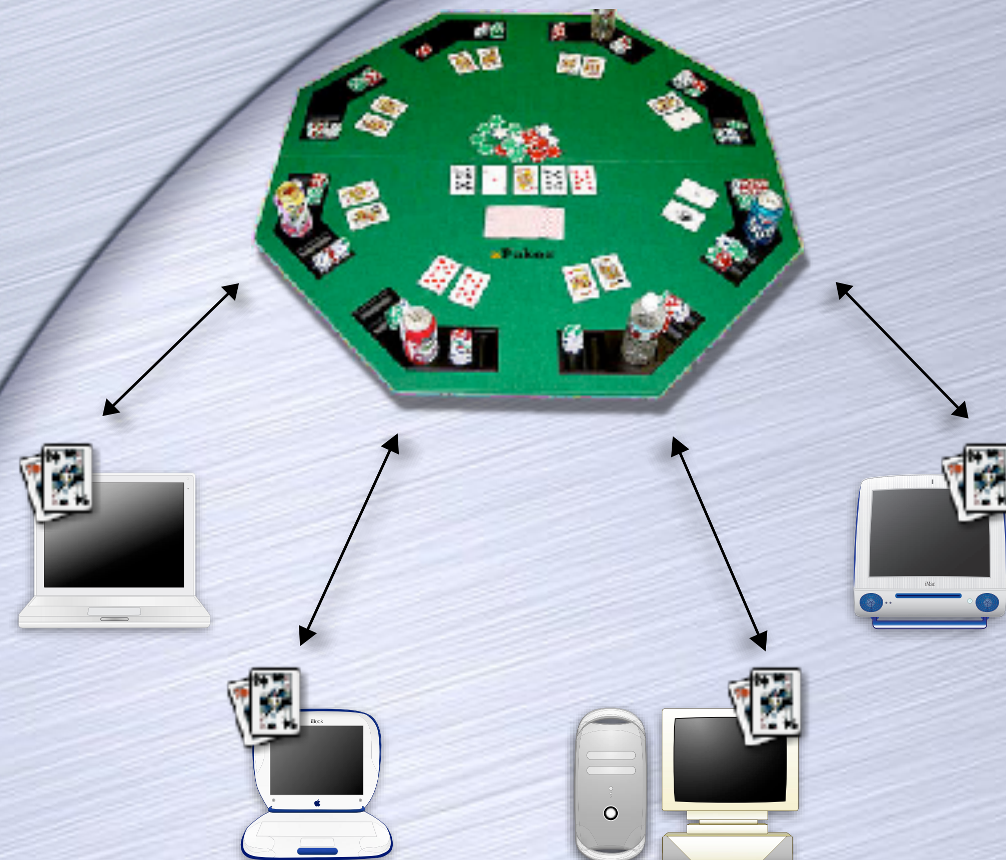
Secure Function Evaluation

- A general problem
- To compute a function of private inputs without revealing information about the inputs
  - Beyond what is revealed by the function



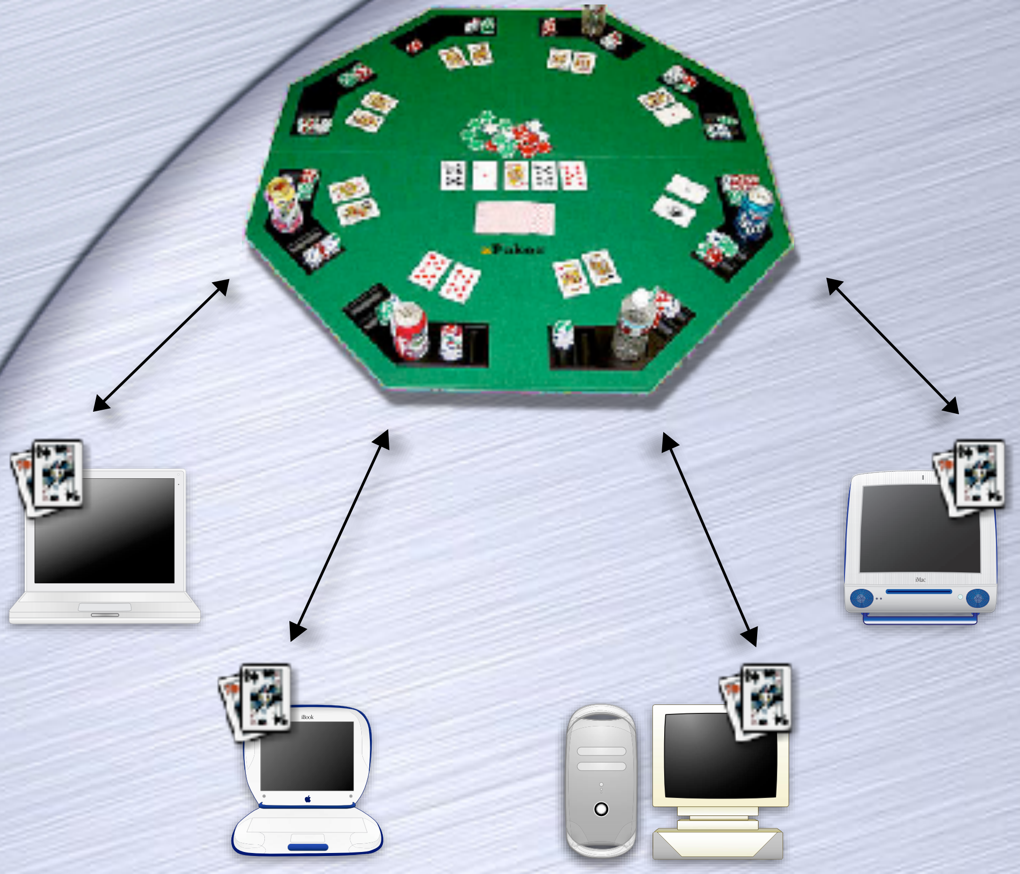
Poker With No Dealer?

Poker With No Dealer?



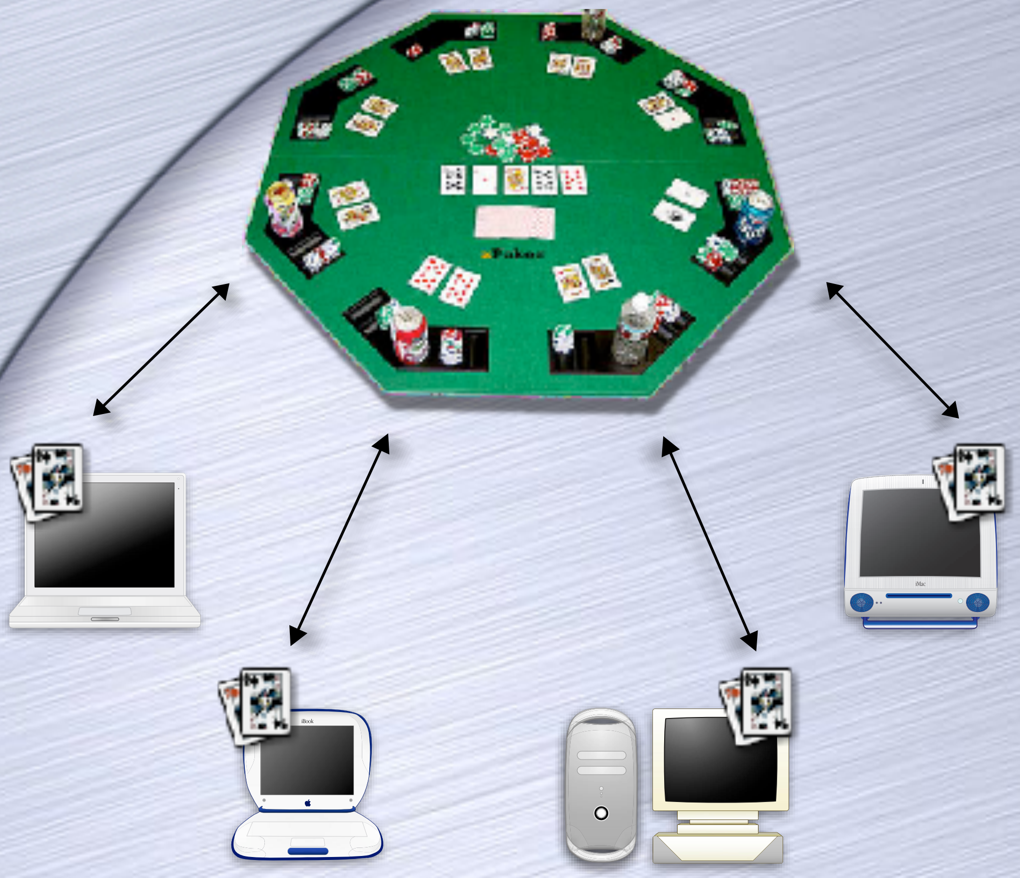
Poker With No Dealer?

- Need to ensure



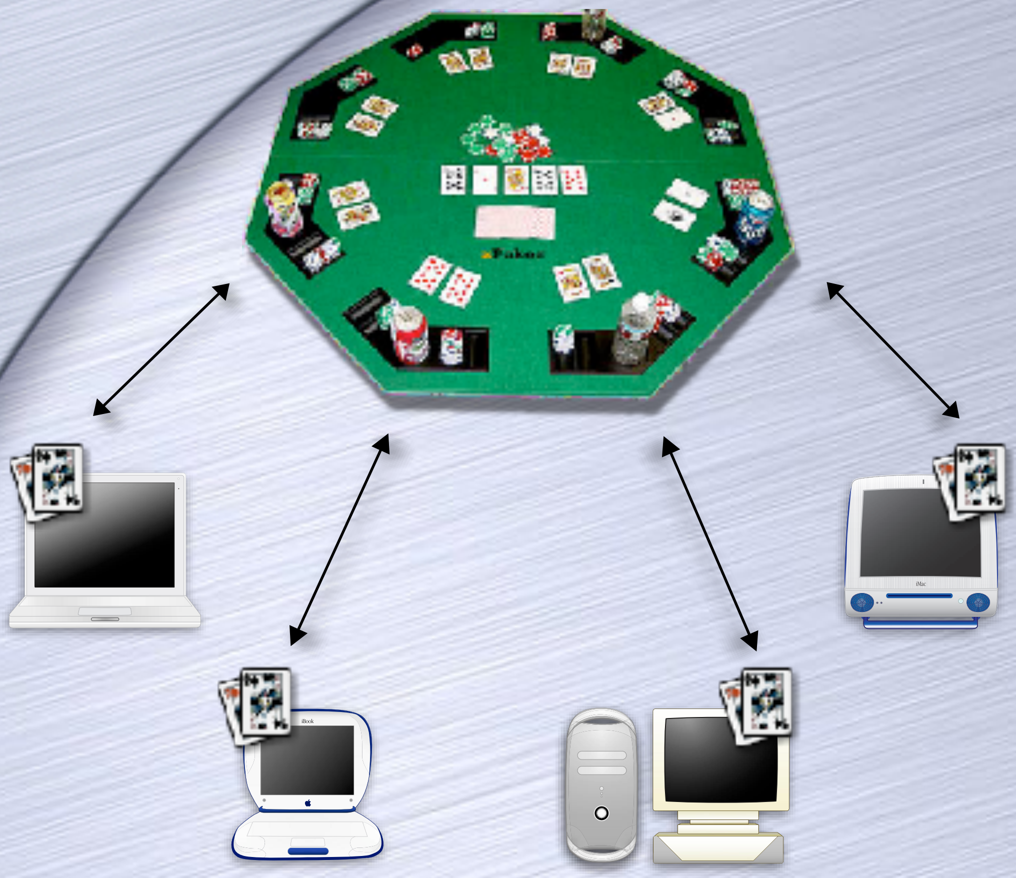
Poker With No Dealer?

- Need to ensure
  - Cards are shuffled and dealt correctly



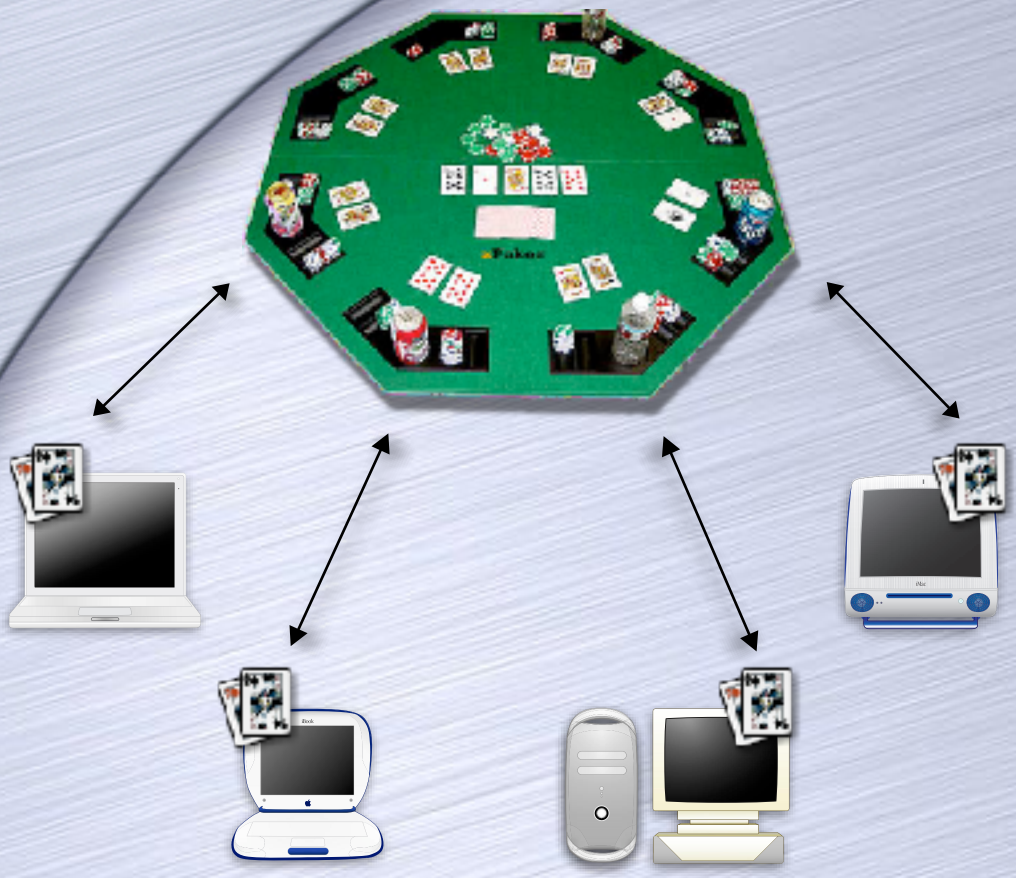
Poker With No Dealer?

- Need to ensure
  - Cards are shuffled and dealt correctly
  - Complete secrecy



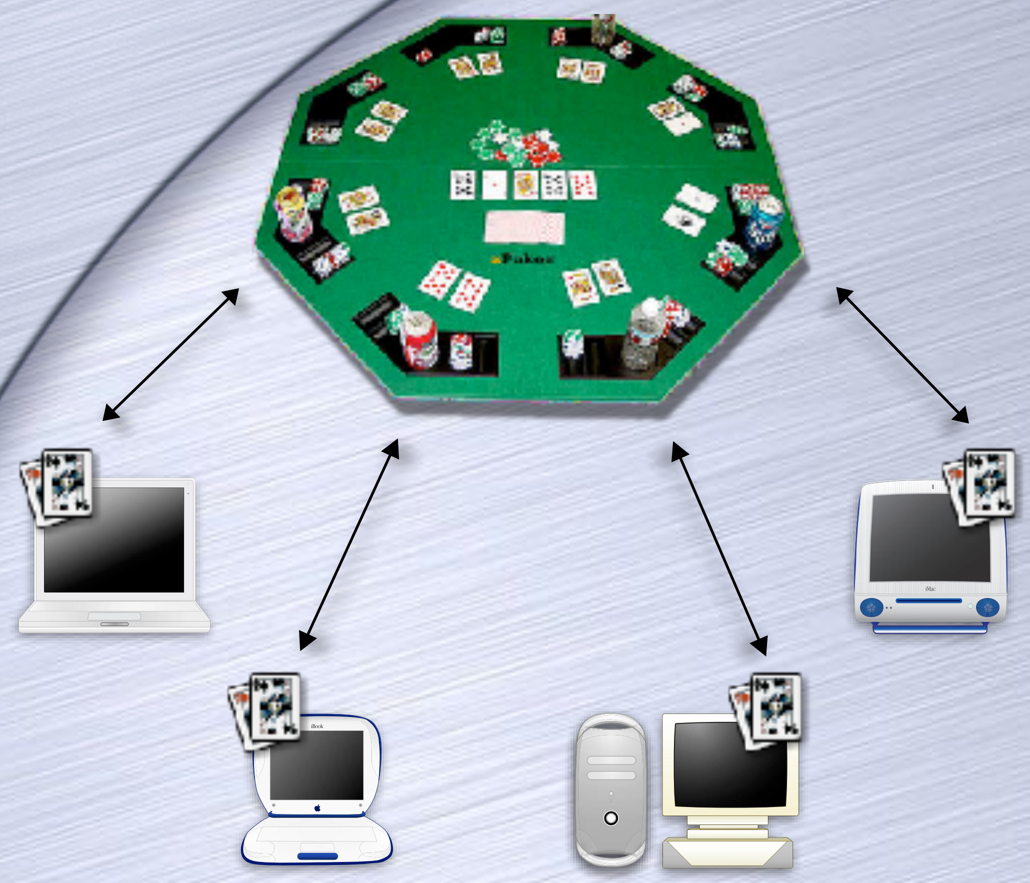
Poker With No Dealer?

- Need to ensure
  - Cards are shuffled and dealt correctly
  - Complete secrecy
  - No “cheating” by players, even if they collude



Poker With No Dealer?

- Need to ensure
  - Cards are shuffled and dealt correctly
  - Complete secrecy
  - No "cheating" by players, even if they collude
- No universally trusted dealer



The Ambitious Goal

The Ambitious Goal

- Without any trusted party, securely do
  - Distributed Data mining
  - E-commerce
  - Network Games
  - E-voting
  - Secure function evaluation
  - ....

The Ambitious Goal

- Without any trusted party, securely do
  - Distributed Data mining
  - E-commerce
  - Network Games
  - E-voting
  - Secure function evaluation
  - ....

The Ambitious Goal

- Without any trusted party, securely do
  - Distributed Data mining
  - E-commerc
  - Network G
  - E-voting
  - Secure fun
  - ....

Secure
Multi-Party Computation
(MPC)

Any task that uses a trusted party!

Emulating Trusted Computation

Emulating Trusted Computation

- Encryption/Authentication allow us to emulate a trusted channel

Emulating Trusted Computation

- Encryption/Authentication allow us to emulate a trusted channel
- Secure MPC: to emulate a source of trusted computation

Emulating Trusted Computation

- Encryption/Authentication allow us to emulate a trusted channel
- Secure MPC: to emulate a source of trusted computation
  - Trusted means it will not "leak" a party's information to others

Emulating Trusted Computation

- Encryption/Authentication allow us to emulate a trusted channel
- Secure MPC: to emulate a source of trusted computation
  - Trusted means it will not “leak” a party’s information to others
  - And it will not cheat in the computation

Emulating Trusted Computation

- Encryption/Authentication allow us to emulate a trusted channel
- Secure MPC: to emulate a source of trusted computation
  - Trusted means it will not "leak" a party's information to others
  - And it will not cheat in the computation
- A tool for mutually distrusting parties to collaborate

Is it for Real?

Is it for Real?

- Getting there! Many implementations/platforms

Is it for Real?

- Getting there! Many implementations/platforms
  - Fairplay, VIFF

Is it for Real?

- Getting there! Many implementations/platforms
  - Fairplay, VIFF
  - Sharemind

Is it for Real?

- Getting there! Many implementations/platforms
  - Fairplay, VIFF
  - Sharemind
  - SCAPI

Is it for Real?

- Getting there! Many implementations/platforms
  - Fairplay, VIFF
  - Sharemind
  - SCAPI
  - Obliv-C

Is it for Real?

- Getting there! Many implementations/platforms
  - Fairplay, VIFF
  - Sharemind
  - SCAPI
  - Obliv-C
  - JustGarble

Is it for Real?

- Getting there! Many implementations/platforms
  - Fairplay, VIFF
  - Sharemind
  - SCAPI
  - Obliv-C
  - JustGarble
  - SPDZ/MASCOT

Is it for Real?

- Getting there! Many implementations/platforms
  - Fairplay, VIFF
  - Sharemind
  - SCAPI
  - Obliv-C
  - JustGarble
  - SPDZ/MASCOT
  - ObliVM

Is it for Real?

- Getting there! Many implementations/platforms
  - Fairplay, VIFF
  - Sharemind
  - SCAPI
  - Obliv-C
  - JustGarble
  - SPDZ/MASCOT
  - ObliVM
  - ...

Is it for Real?

- And many practical systems using some form of MPC

Is it for Real?

- And many practical systems using some form of MPC
  - Danish company Partisia with real-life deployments (since 2008)

Is it for Real?

- And many practical systems using some form of MPC
  - Danish company Partisia with real-life deployments (since 2008)
    - sugar beet auction, electricity auction, spectrum auction, key management

Is it for Real?

- And many practical systems using some form of MPC
  - Danish company Partisia with real-life deployments (since 2008)
    - sugar beet auction, electricity auction, spectrum auction, key management
  - A prototype for credit rating, supported by Danish banks

Is it for Real?

- And many practical systems using some form of MPC
  - Danish company Partisia with real-life deployments (since 2008)
    - sugar beet auction, electricity auction, spectrum auction, key management
  - A prototype for credit rating, supported by Danish banks
  - A proposal to the Estonian Tax & Customs Board
  - A proposal for Satellite Collision Analysis
  - ...

Mental Poker

Adi Shamir, Ronald L. Rivest and Leonard M. Adleman

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ABSTRACT

Can two potentially dishonest players play a fair game of poker without using any cards—for example, over the phone? This paper provides the following answers:

1. No. (Rigorous mathematical proof supplied.)
2. Yes. (Correct and complete protocol given.)

This Tutorial

This Tutorial

- What does it mean to be secure?

This Tutorial

- What does it mean to be secure?
- How does one do MPC? Warm up

This Tutorial

- What does it mean to be secure?
- How does one do MPC? Warm up
    - An important, basic protocol: “Basic” GMW

This Tutorial

- What does it mean to be secure?
- How does one do MPC? Warm up
  - An important, basic protocol: "Basic" GMW
- Glimpses of various issues

What does it mean to be Secure?

Terminology

Terminology

- Protocol: Instructions to the (honest) parties on what messages to send to whom based on input/local randomness and messages received so far.
  - The next-message function

Terminology

- Protocol: Instructions to the (honest) parties on what messages to send to whom based on input/local randomness and messages received so far.
  - The next-message function
- Functionality: What we are aiming to achieve
  - Specified as the program of a trusted party

Terminology

- Protocol: Instructions to the (honest) parties on what messages to send to whom based on input/local randomness and messages received so far.
  - The next-message function
- Functionality: What we are aiming to achieve
  - Specified as the program of a trusted party

Security Issues to Consider

Security Issues to Consider

- Protocol may leak a party's secrets
  - Clearly an issue
  - Even if we trust everyone not to cheat in our protocol (i.e., honest-but-curious)
    - Also, a liability for a party if extra information reaches it (e.g., in medical data mining)

Security Issues to Consider

- Protocol may leak a party's secrets
  - Clearly an issue
  - Even if we trust everyone not to cheat in our protocol (i.e., honest-but-curious)
    - Also, a liability for a party if extra information reaches it (e.g., in medical data mining)
- Protocol may give adversary illegitimate influence on the outcome
  - Say in poker, if adversary can influence hands dealt
  - In auction, if adversary can choose its bid to just beat the others'

Defining Security

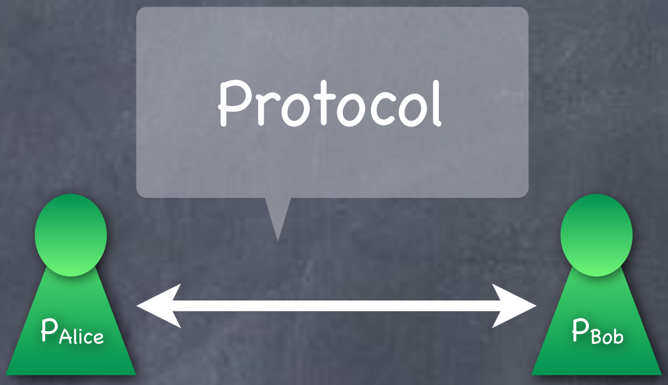
Defining Security

- REAL/IDEAL paradigm

Defining Security

- REAL/IDEAL paradigm

IDEAL



REAL

Defining Security

- REAL/IDEAL paradigm

- Security guarantee: Whatever an adversary can do in the REAL world, an adversary could have done the same in the IDEAL world

Defining Security

- REAL/IDEAL paradigm

- Security guarantee: Whatever an adversary can do in the REAL world, an adversary could have done the same in the IDEAL world
  - Can't blame the protocol for anything undesirable

Adversary

Adversary

- REAL-adversary can corrupt any set of players
  - IDEAL-adversary should corrupt the same set of players

Adversary

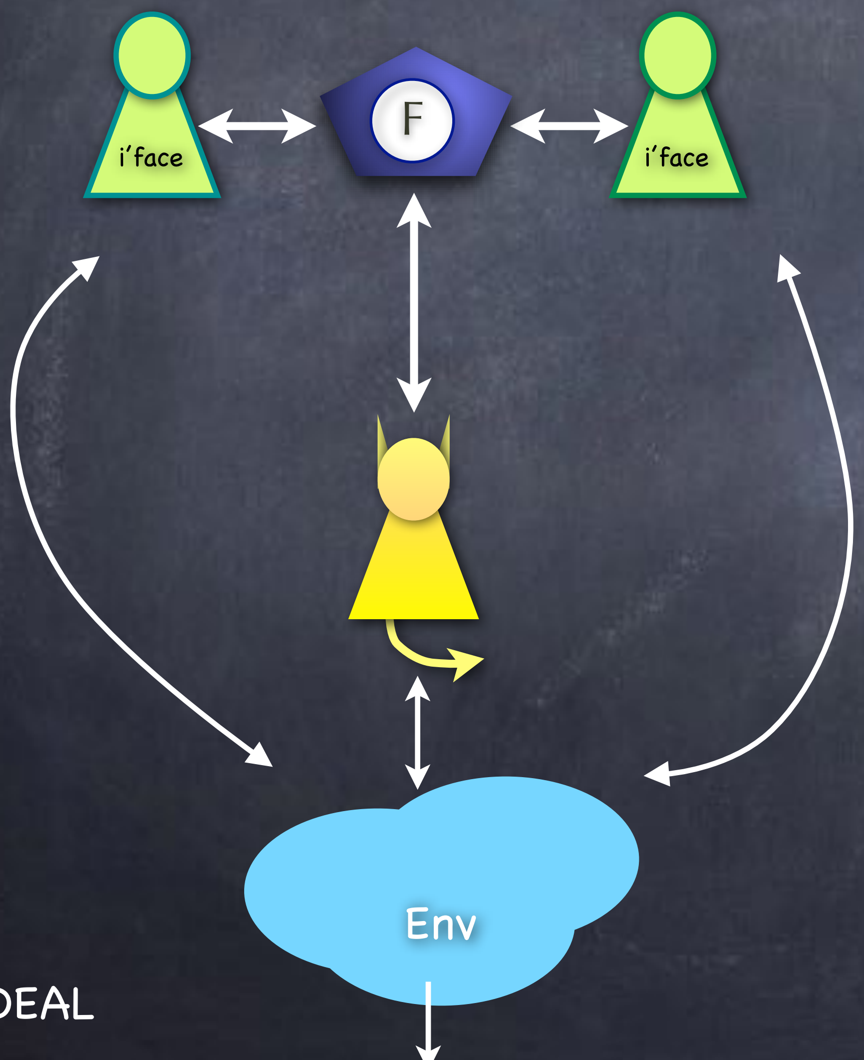
- REAL-adversary can corrupt any set of players
  - IDEAL-adversary should corrupt the same set of players
- More sophisticated notion: adaptive adversary which corrupts players dynamically during/after the execution
  - We'll stick to static adversaries

Adversary

- REAL-adversary can corrupt any set of players
  - IDEAL-adversary should corrupt the same set of players
- More sophisticated notion: adaptive adversary which corrupts players dynamically during/after the execution
  - We'll stick to static adversaries
- Passive vs. Active adversary: Passive adversary gets only read access to the internal state of the corrupted players. Active adversary overwrites their state and program.

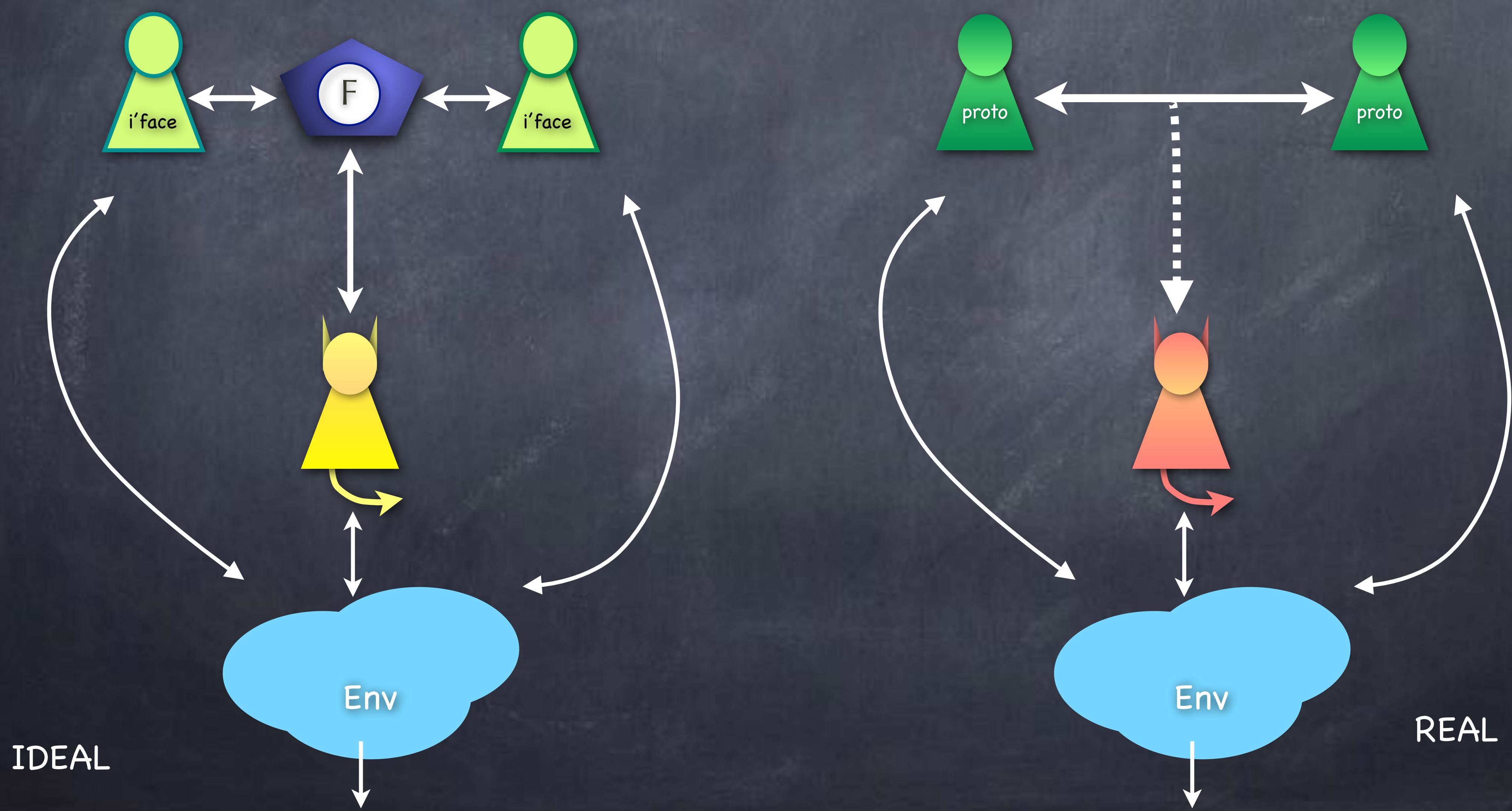
Defining Security

Defining Security

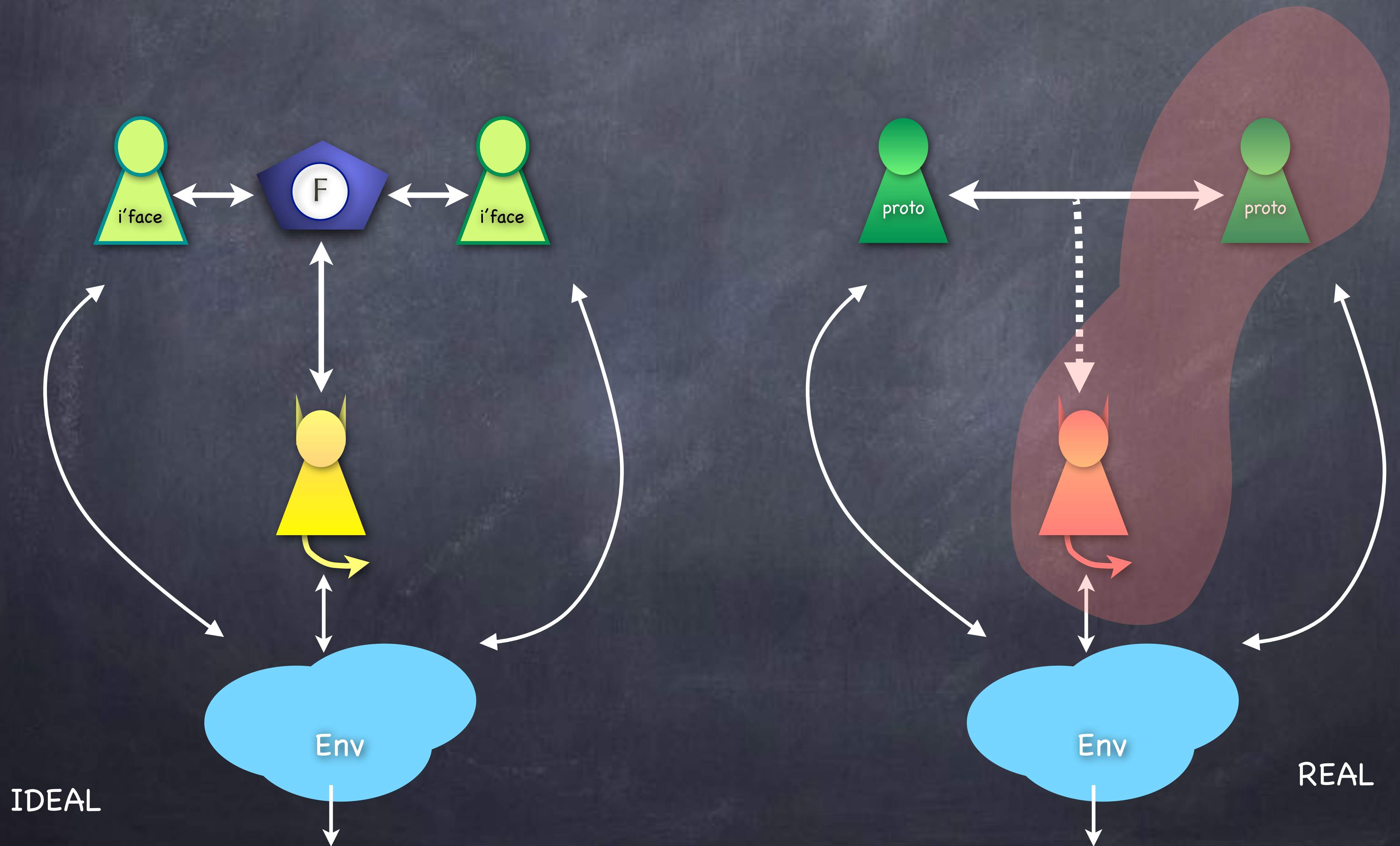


IDEAL

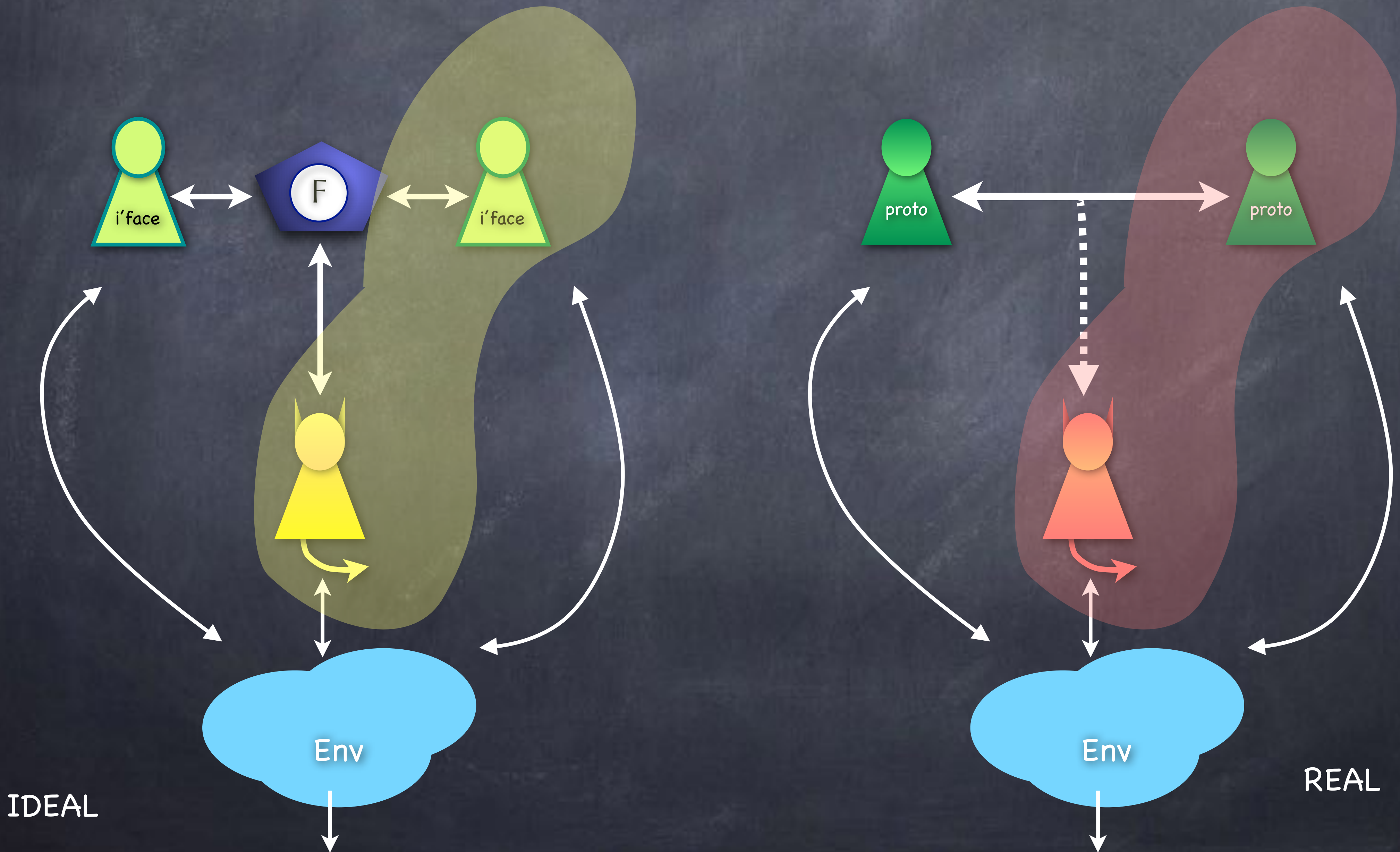
Defining Security



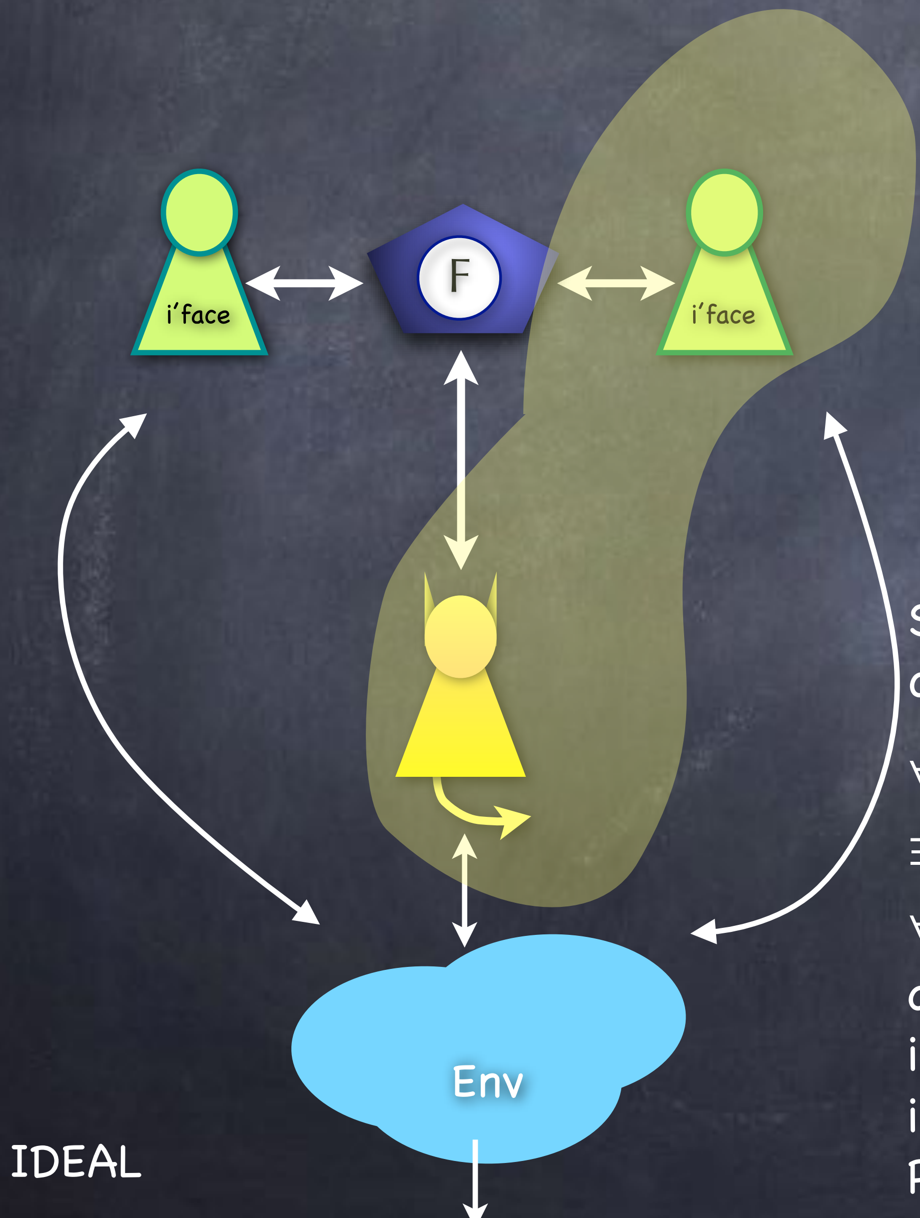
Defining Security



Defining Security



Defining Security



Secure (and correct) if:

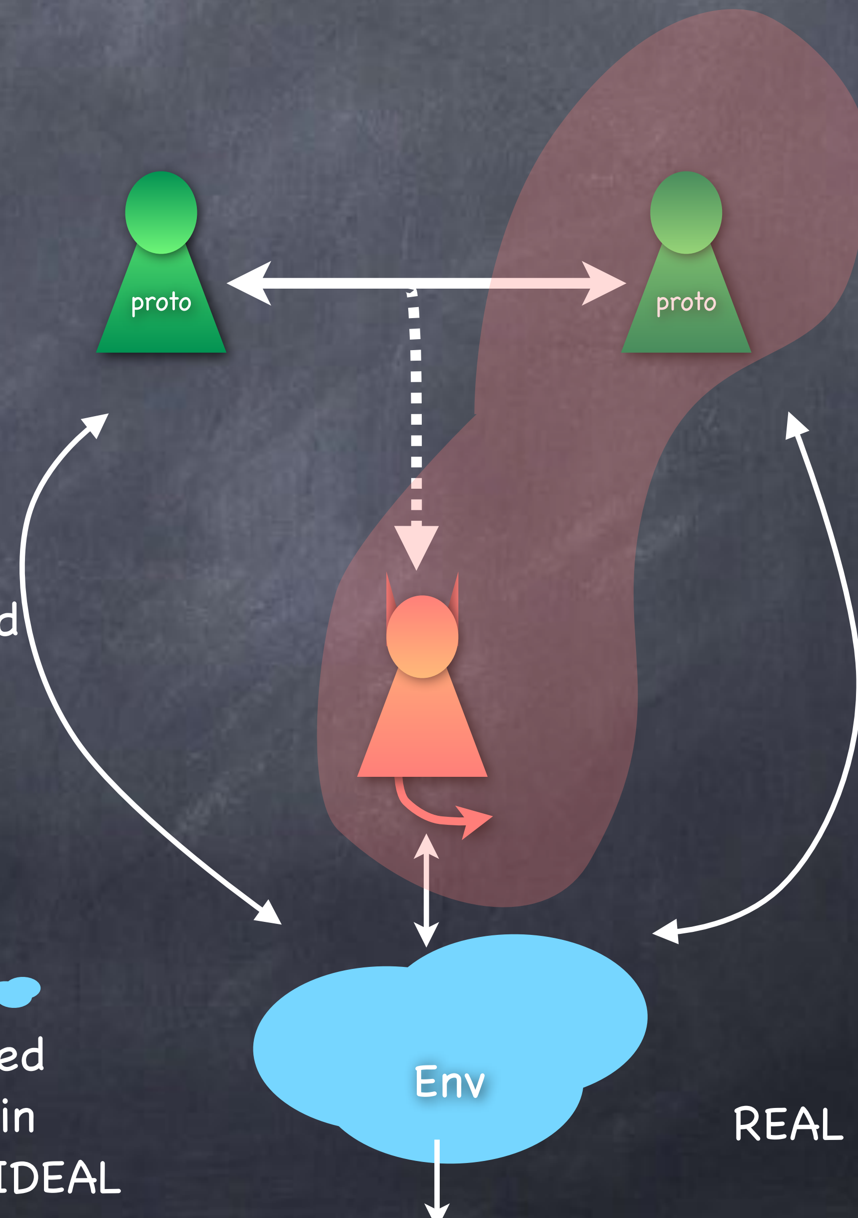
∀

∃ s.t.

∀

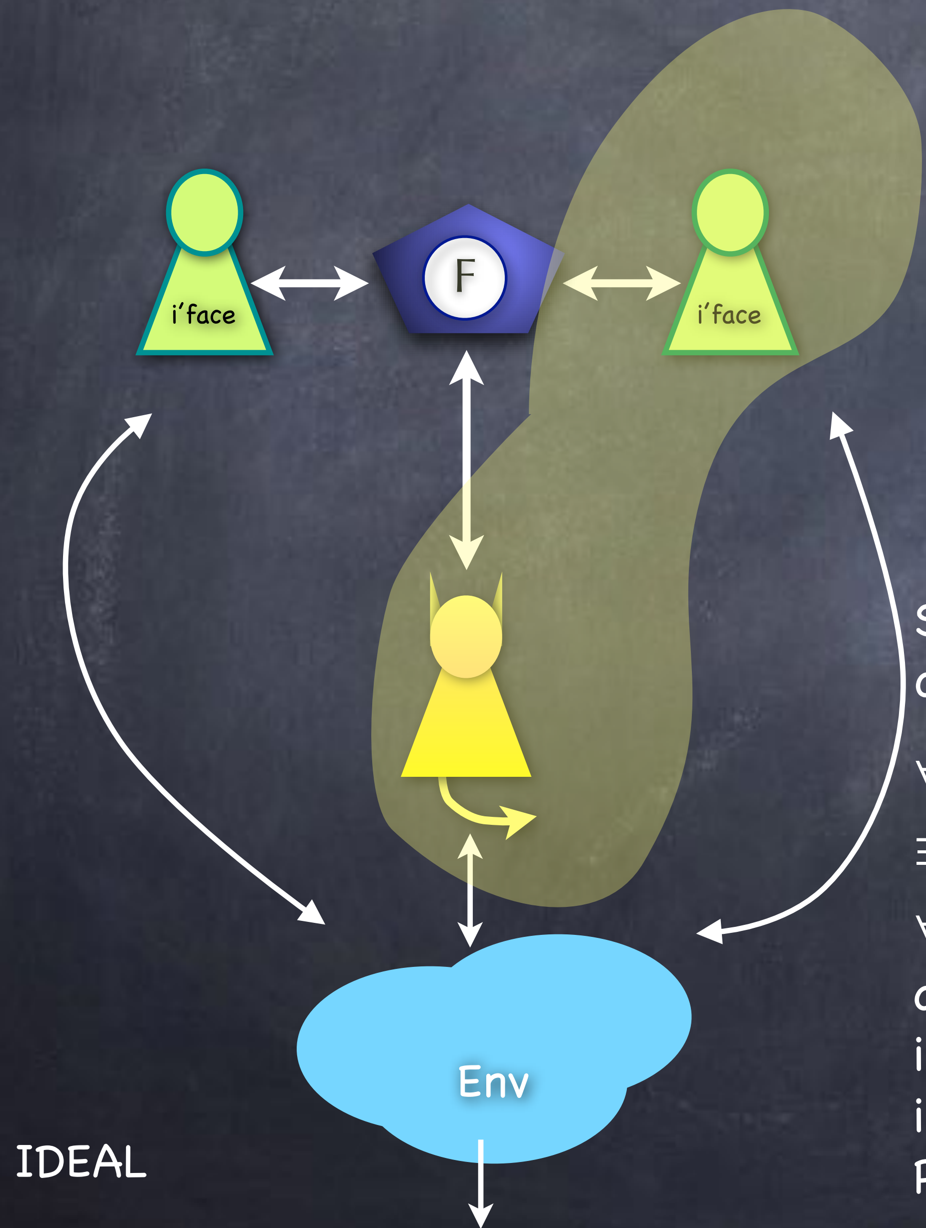
output of

is distributed identically in REAL and IDEAL



Universally Composable [Canetti'01]

Defining Security



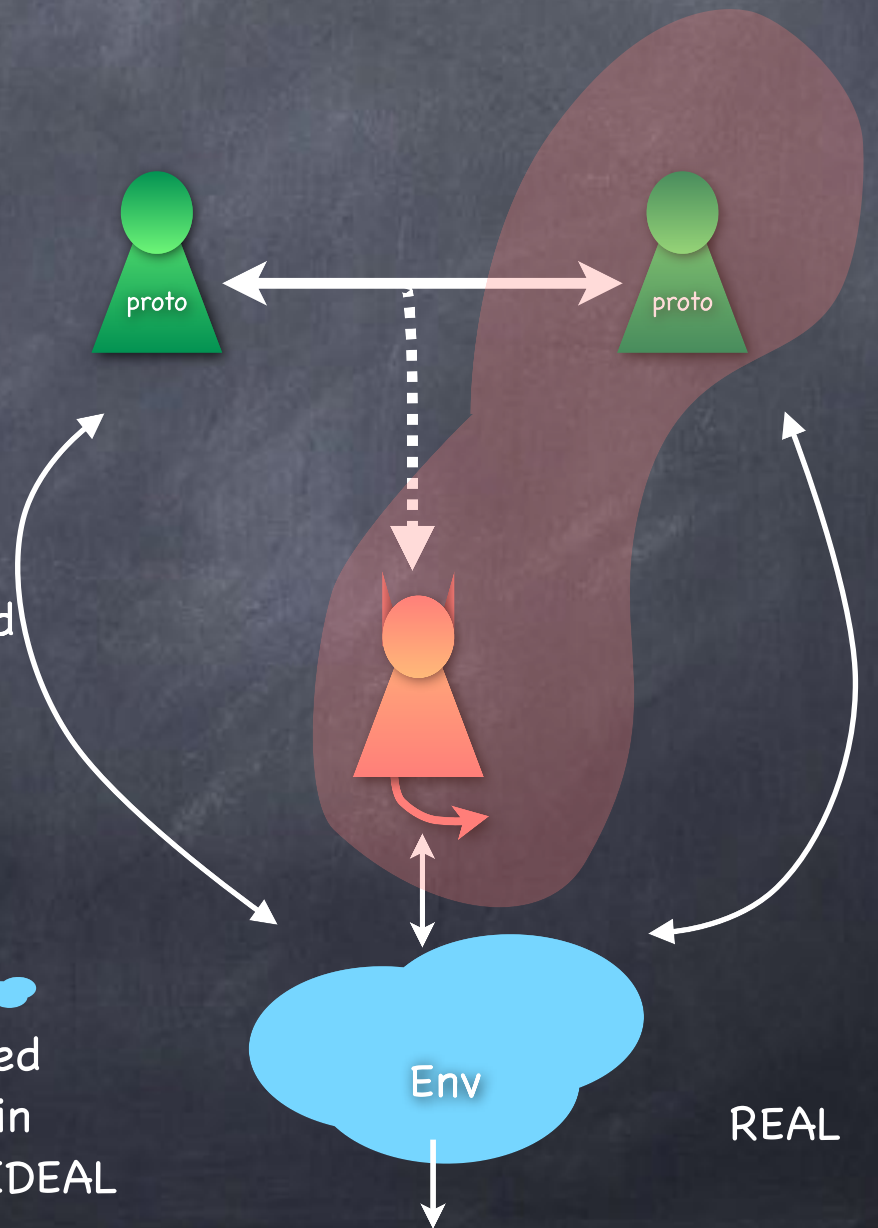
Secure (and correct) if:

∀

∃ s.t.

∀

output of is distributed identically in REAL and IDEAL



(Some) Security Models

(Some) Security Models

- Standalone security: environment is not "live": interacts with the adversary before and after (but not during) the protocol

(Some) Security Models

- Standalone security: environment is not “live”: interacts with the adversary before and after (but not during) the protocol
- Honest-majority security: adversary can corrupt only a strict minority of parties. (Not useful when only two parties involved)

(Some) Security Models

- Standalone security: environment is not "live": interacts with the adversary before and after (but not during) the protocol
- Honest-majority security: adversary can corrupt only a strict minority of parties. (Not useful when only two parties involved)
- Passive (a.k.a honest-but-curious) adversary: where corrupt parties stick to the protocol (but we don't want to trust them with information)

(Some) Security Models

- Standalone security: environment is not “live”: interacts with the adversary before and after (but not during) the protocol
- Honest-majority security: adversary can corrupt only a strict minority of parties. (Not useful when only two parties involved)
- Passive (a.k.a honest-but-curious) adversary: where corrupt parties stick to the protocol (but we don’t want to trust them with information)
- Functionality-specific non-simulation-based definitions: usually leave out subtle attacks (e.g. malleability related attacks)

(Some) Security Models

- Standalone security: environment is not "live": interacts with the adversary before and after (but not during) the protocol
- Honest-majority security: adversary can corrupt only a strict minority of parties. (Not useful when only two parties involved)
- Passive (a.k.a honest-but-curious) adversary: where corrupt parties stick to the protocol (but we don't want to trust them with information)
- Functionality-specific non-simulation-based definitions: usually leave out subtle attacks (e.g. malleability related attacks)
- Protocols using a trusted party for some basic functionality (a.k.a. set up)

(Some) Security Models

- Standalone security: environment is not "live": interacts with the adversary before and after (but not during) the protocol
- Honest-majority security: adversary can corrupt only a strict minority of parties. (Not useful when only two parties involved)
- Passive (a.k.a honest-but-curious) adversary: where corrupt parties stick to the protocol (but we don't want to trust them with information)
- Functionality-specific non-simulation-based definitions: usually leave out subtle attacks (e.g. malleability related attacks)
- Protocols using a trusted party for some basic functionality (a.k.a. set up)
- Angel-UC (UC + a helpful oracle for adversary in the ideal world)

Is MPC Possible?

Is MPC Possible?

- Can we securely realize every functionality?

Is MPC Possible?

- Can we securely realize <u>every</u> functionality?
- No & Yes!

Is MPC Possible?

- Can we securely realize <u>every</u> functionality?
- No & Yes!

<table>
<tr><th></th><th>All subsets corruptible</th><th>Honest Majority</th></tr>
<tr><td>Computationally Unbounded</td><td>No</td><td rowspan="2">Yes</td></tr>
<tr><td>Computationally Bounded (PPT)</td><td></td></tr>
</table>

Is MPC Possible?

- Can we securely realize <u>every</u> functionality?
- No & Yes!

Univ. Composable Angel-UC Standalone Passive	All subsets corruptible	Honest Majority
Computationally Unbounded	No	Yes
Computationally Bounded (PPT)	No Yes Yes Yes	

Doing MPC

A simple example

- An auction, with Alice and Bob bidding

A simple example

- An auction, with Alice and Bob bidding
- Rules:
  - A bid is an integer in the range [0,100]
  - Alice can bid only even integers and Bob odd integers
  - Person with the higher bid wins

A simple example

- An auction, with Alice and Bob bidding
- Rules:
  - A bid is an integer in the range [0,100]
  - Alice can bid only even integers and Bob odd integers
  - Person with the higher bid wins
- Goal: find out the winning bid (winner & amount) without revealing anything more about the losing bid (beyond what is revealed by the winning bid)

A simple example

- Secure protocol:
  - Count down from 100
  - At each even round Alice announces whether her bid equals the current count; at each odd round Bob does the same
  - Stop if a party says yes

A simple example

- Secure protocol:
  - Count down from 100
  - At each even round Alice announces whether her bid equals the current count; at each odd round Bob does the same
  - Stop if a party says yes
- Dutch flower auction

Oblivious Transfer

IDEAL World

Oblivious Transfer

- Pick one out of two, without revealing which

IDEAL World

Oblivious Transfer

- Pick one out of two, without revealing which
  - Intuitive property: transfer partial information "obliviously"

IDEAL World

Oblivious Transfer

- Pick one out of two, without revealing which
  - Intuitive property: transfer partial information "obliviously"

IDEAL World

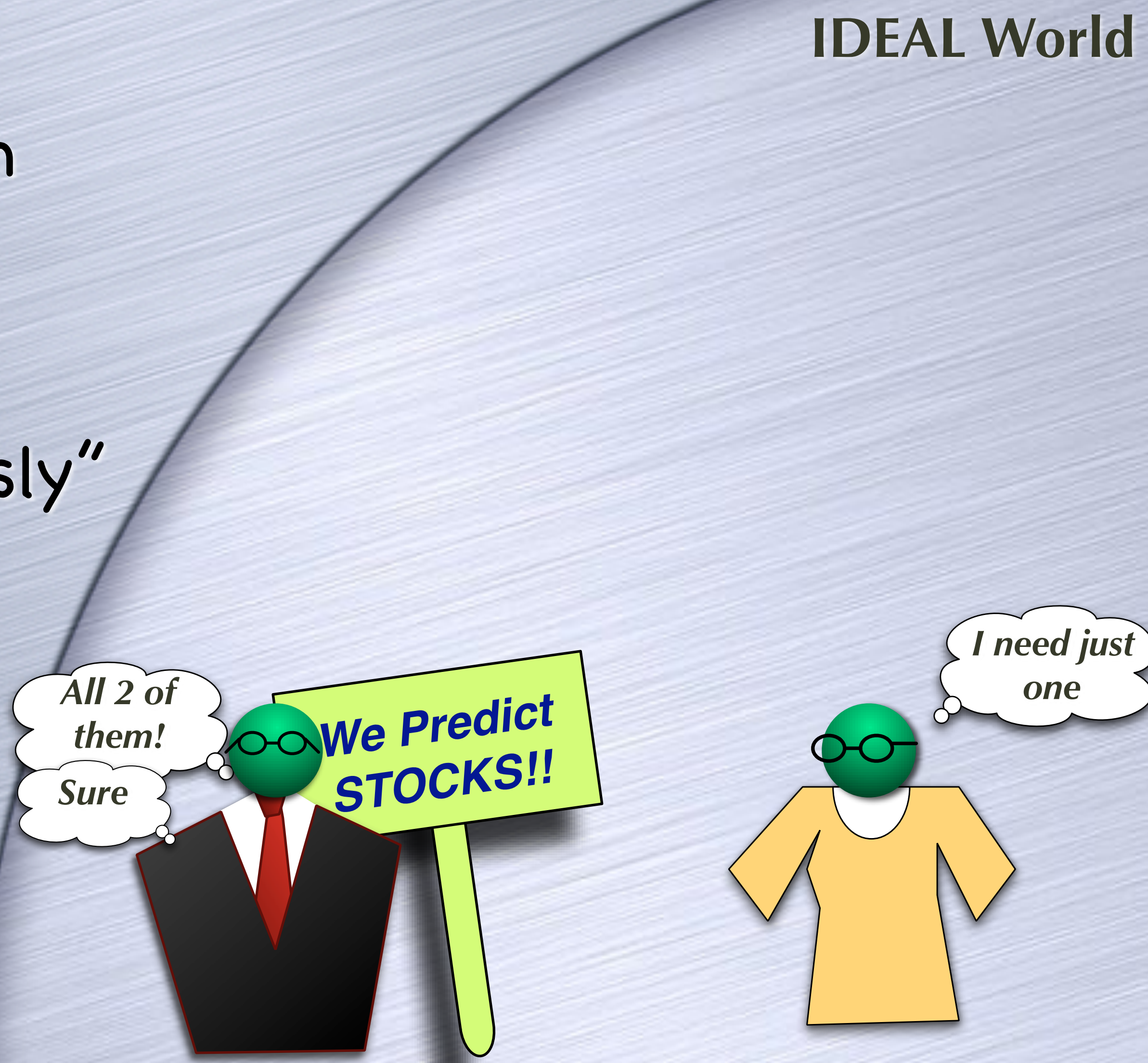
Oblivious Transfer

- Pick one out of two, without revealing which
  - Intuitive property: transfer partial information "obliviously"

IDEAL World

Oblivious Transfer

- Pick one out of two, without revealing which
  - Intuitive property: transfer partial information "obliviously"



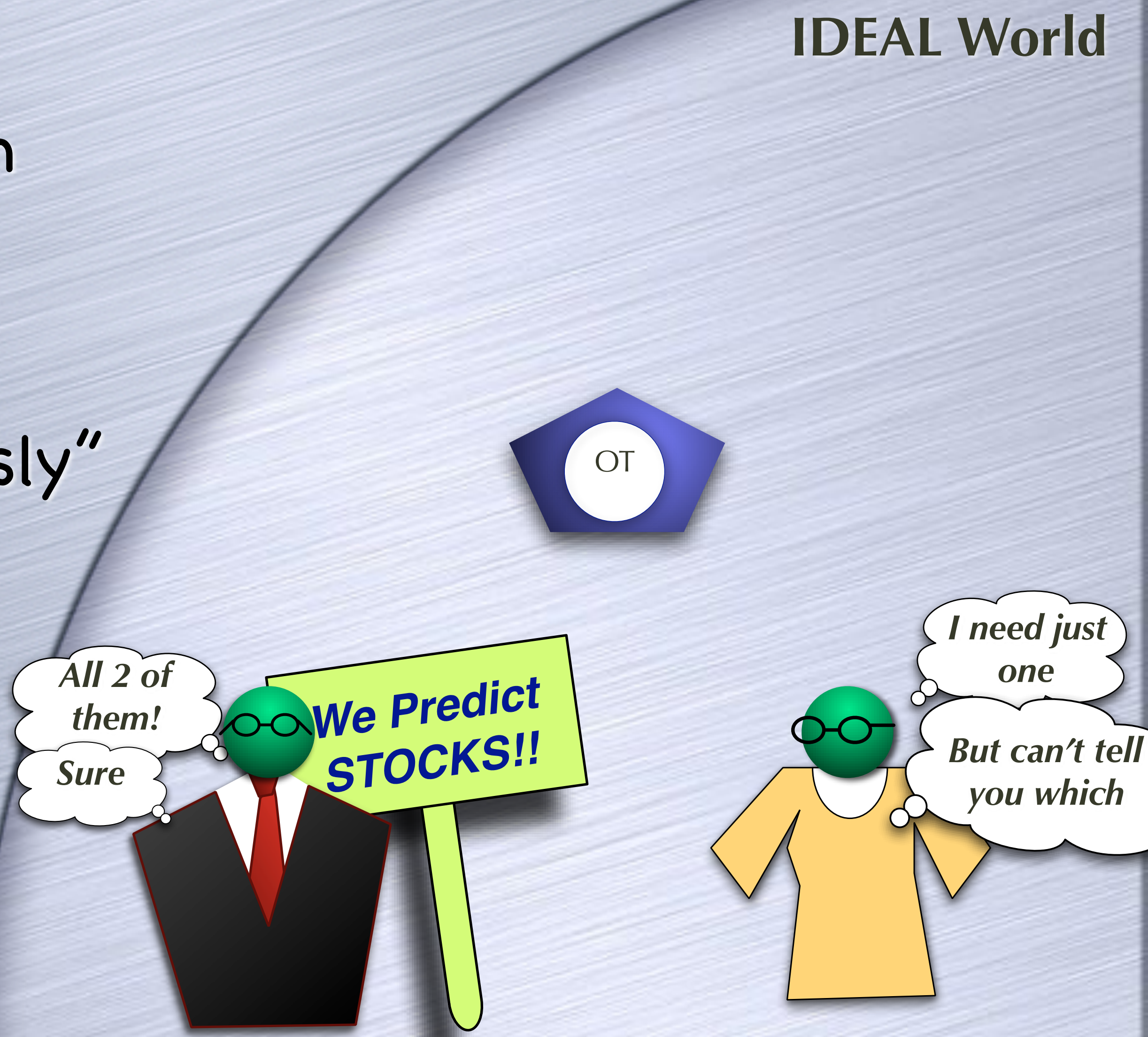
Oblivious Transfer

- Pick one out of two, without revealing which
  - Intuitive property: transfer partial information "obliviously"

IDEAL World

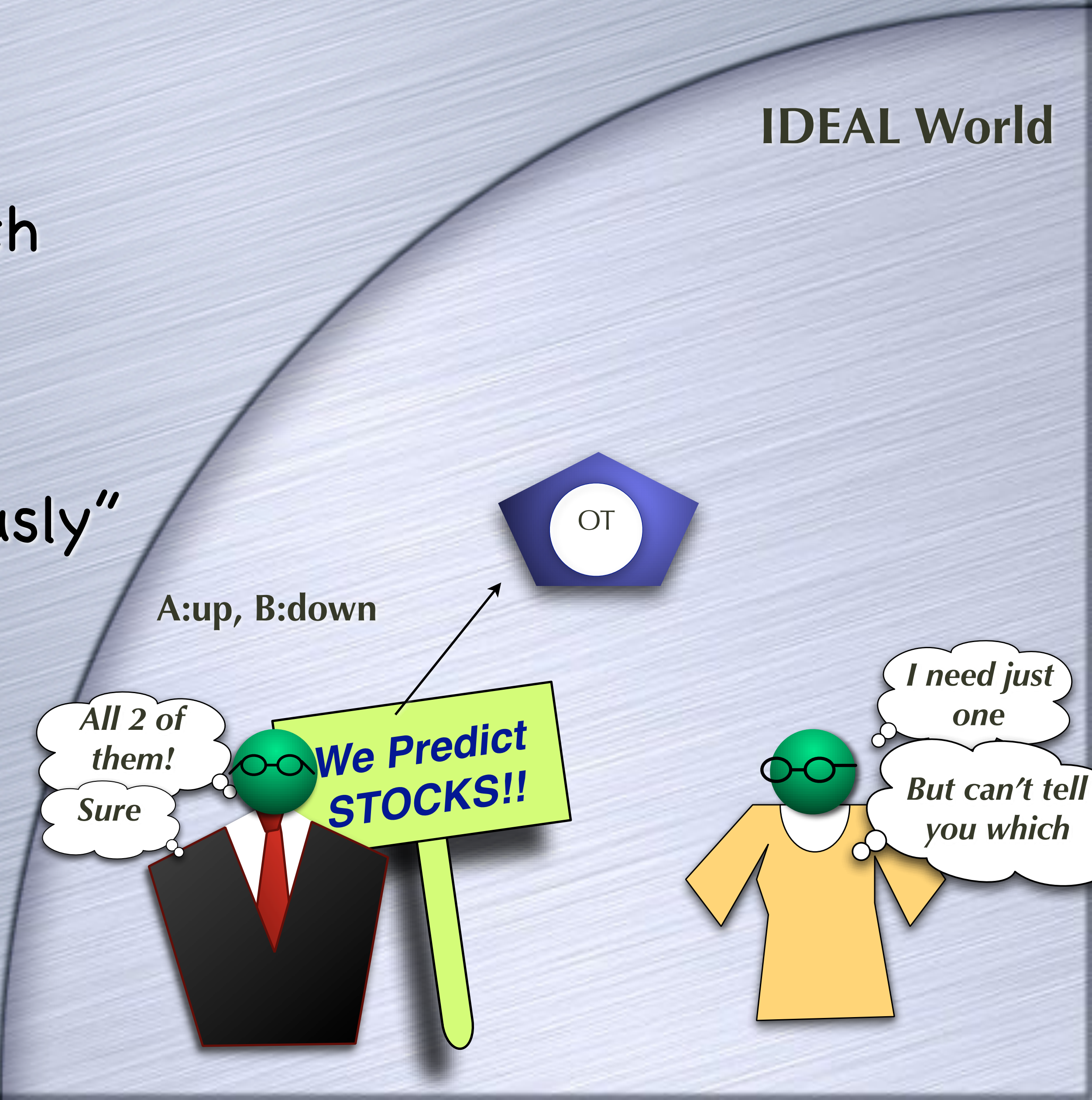
Oblivious Transfer

- Pick one out of two, without revealing which
  - Intuitive property: transfer partial information "obliviously"



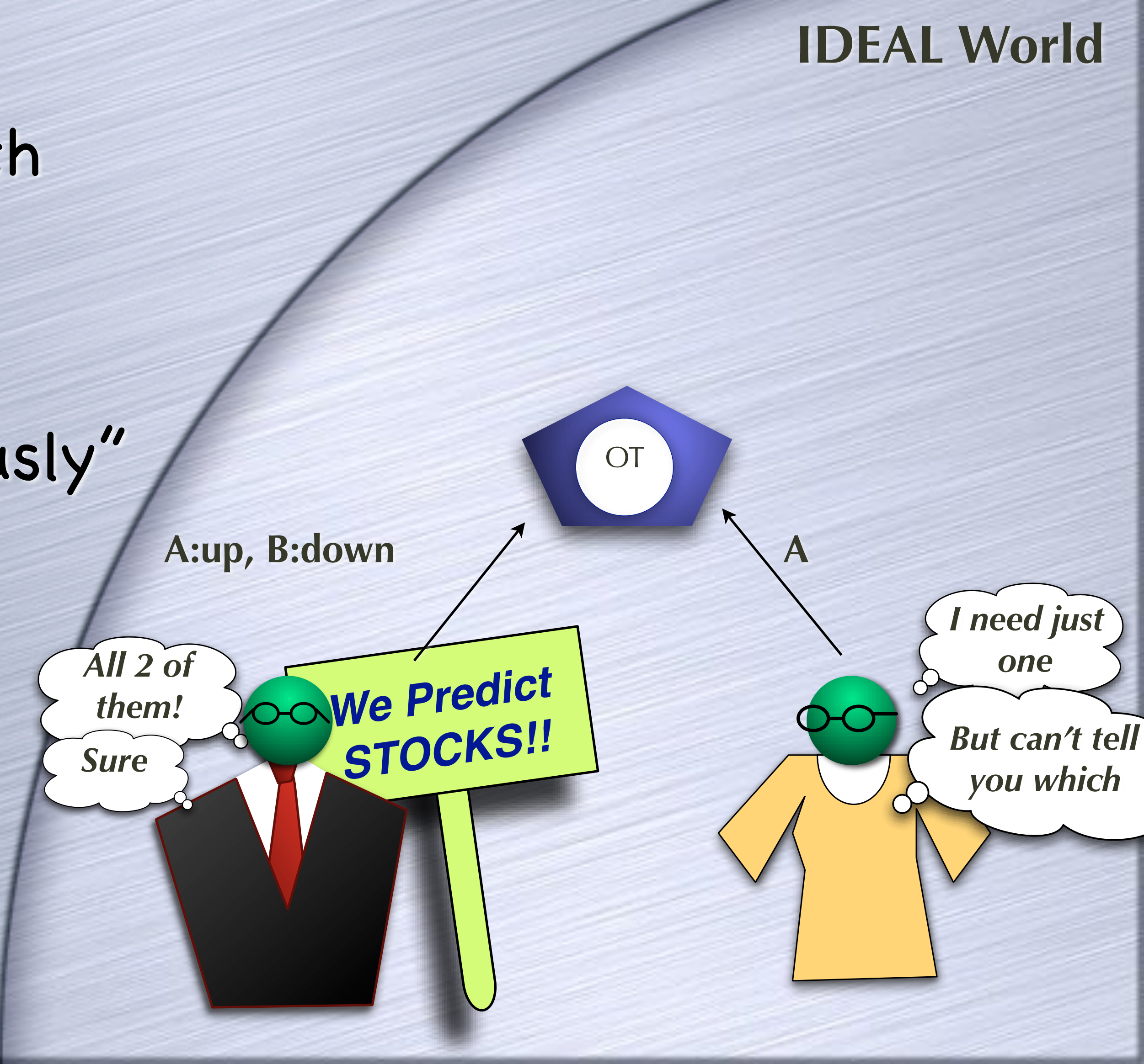
Oblivious Transfer

- Pick one out of two, without revealing which
  - Intuitive property: transfer partial information "obliviously"



Oblivious Transfer

- Pick one out of two, without revealing which
  - Intuitive property: transfer partial information "obliviously"



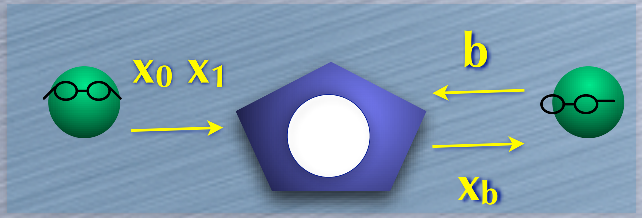
Oblivious Transfer

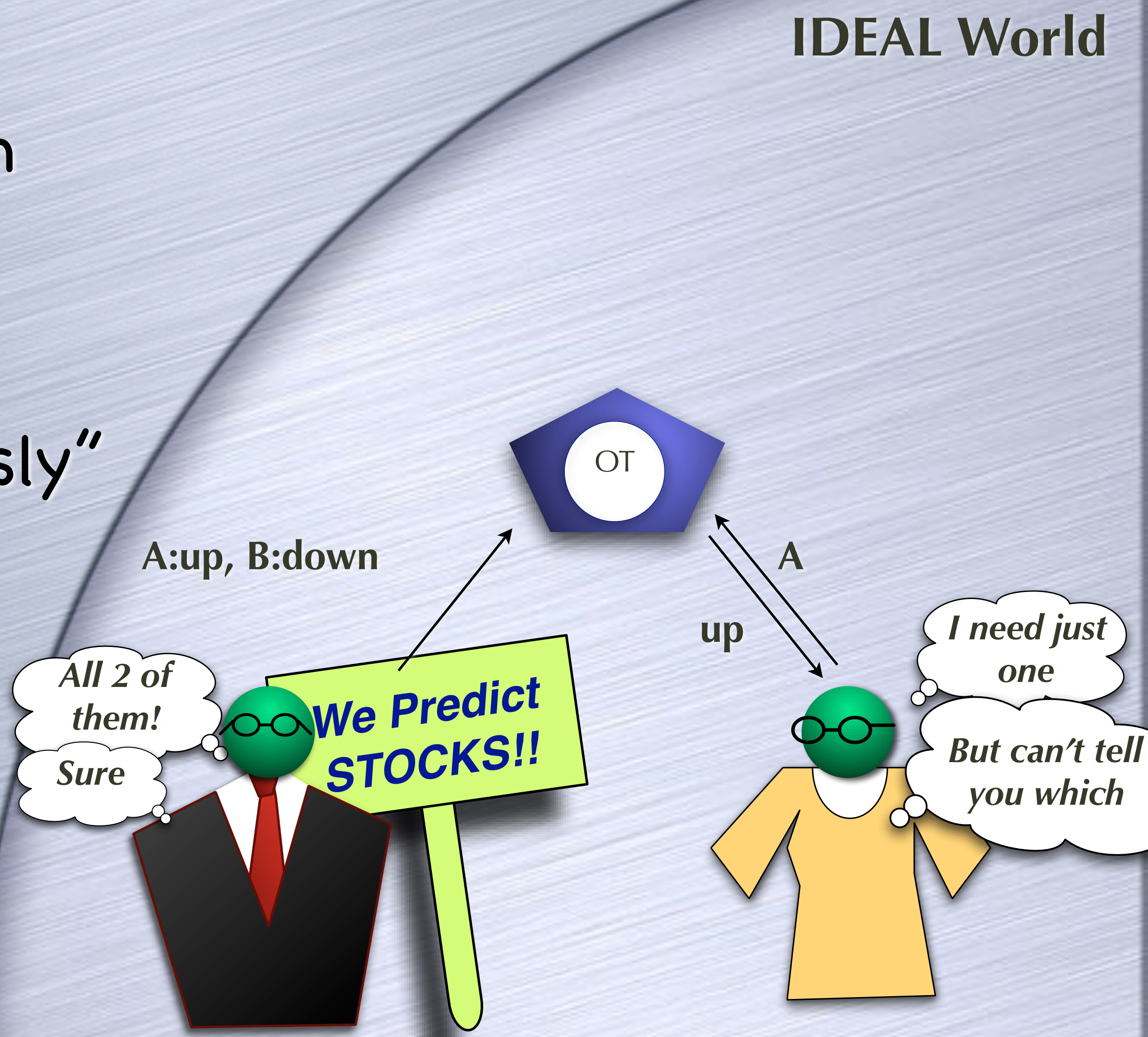
- Pick one out of two, without revealing which
  - Intuitive property: transfer partial information "obliviously"

IDEAL World

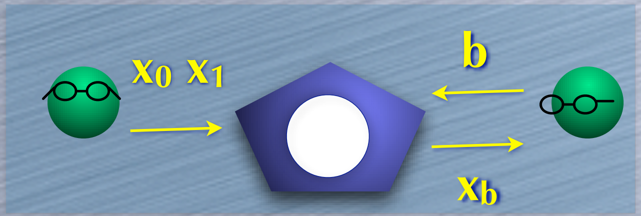
Oblivious Transfer

- Pick one out of two, without revealing which
    - Intuitive property: transfer partial information "obliviously"



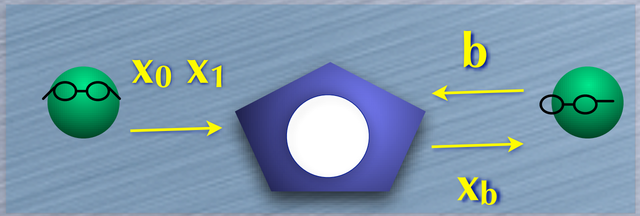


An OT Protocol (passive corruption)



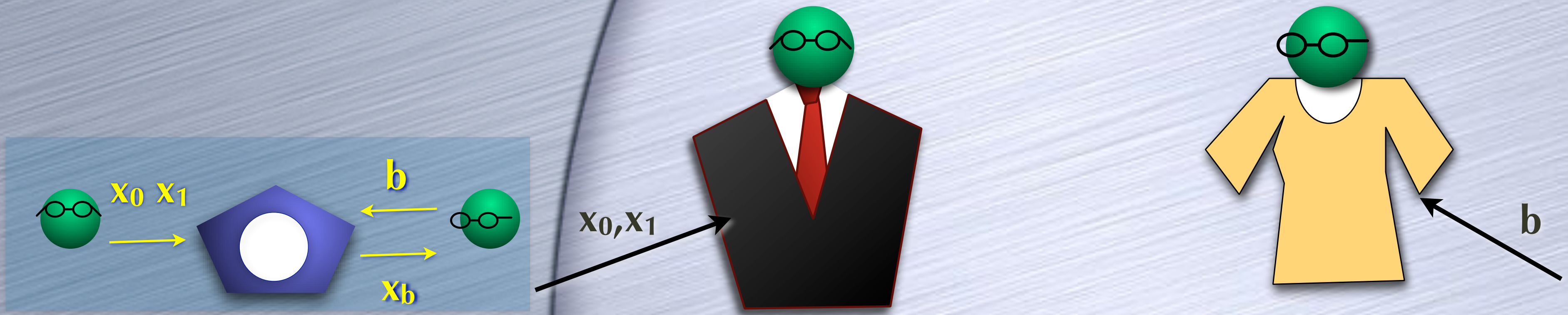
An OT Protocol (passive corruption)

- Using a (special) encryption



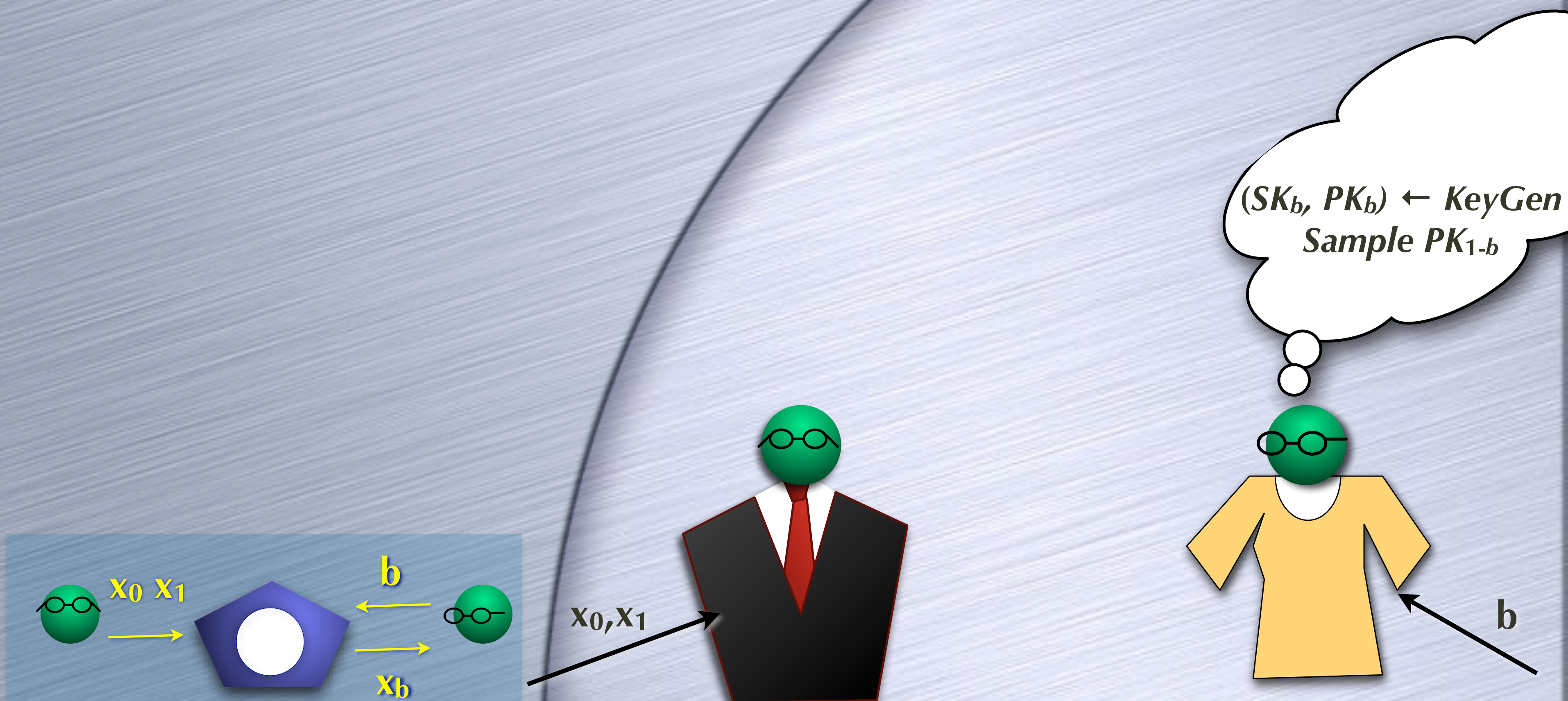
An OT Protocol (passive corruption)

- Using a (special) encryption



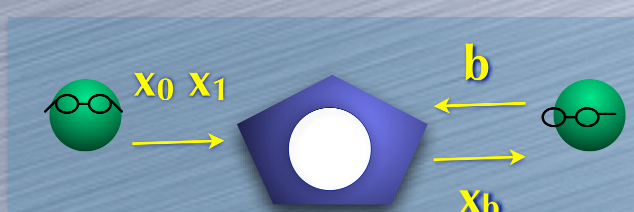
An OT Protocol (passive corruption)

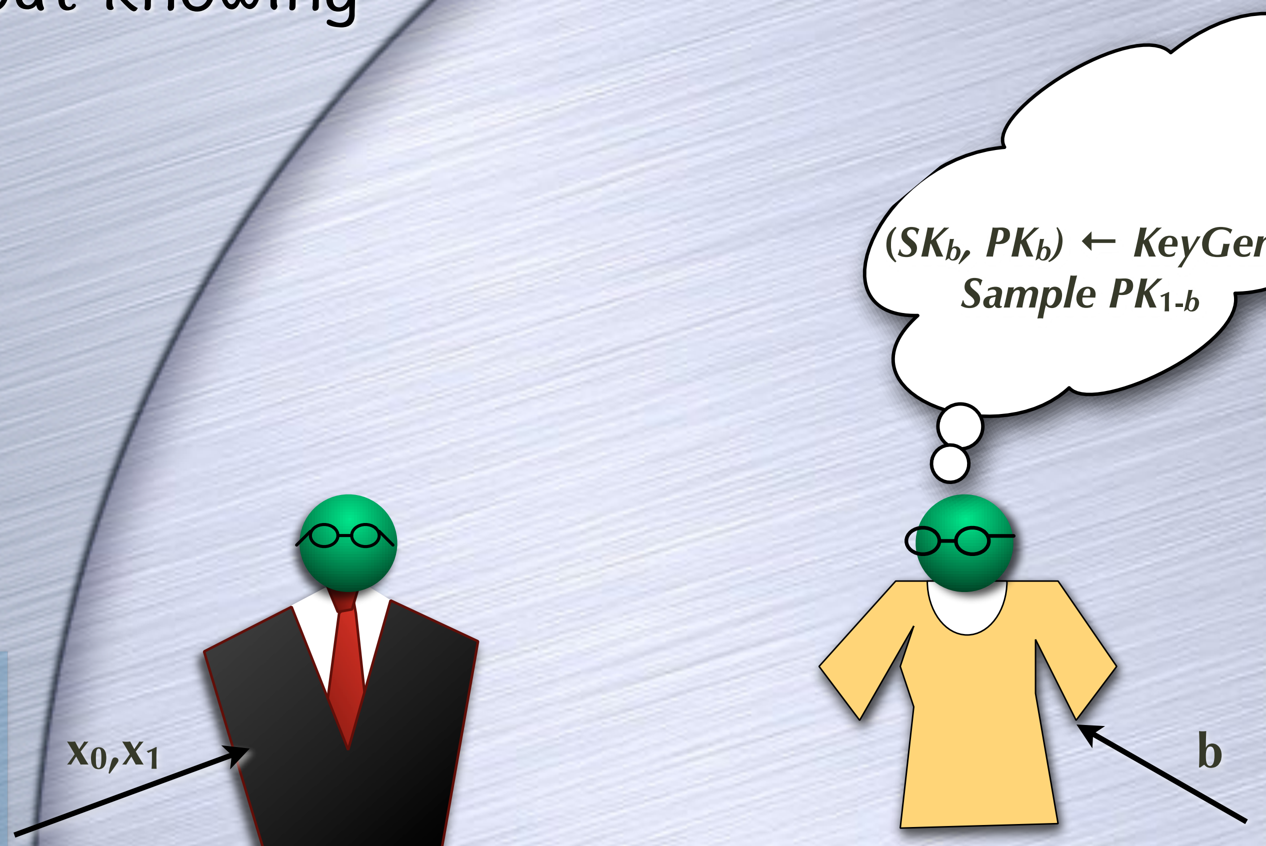
- Using a (special) encryption



An OT Protocol (passive corruption)

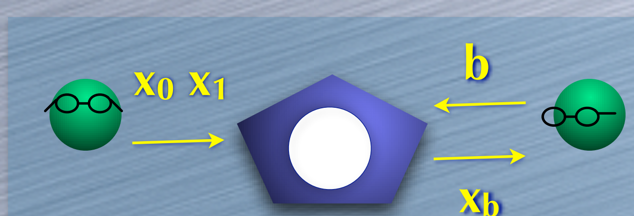
- Using a (special) encryption
  - PKE in which one can sample a public-key without knowing secret-key

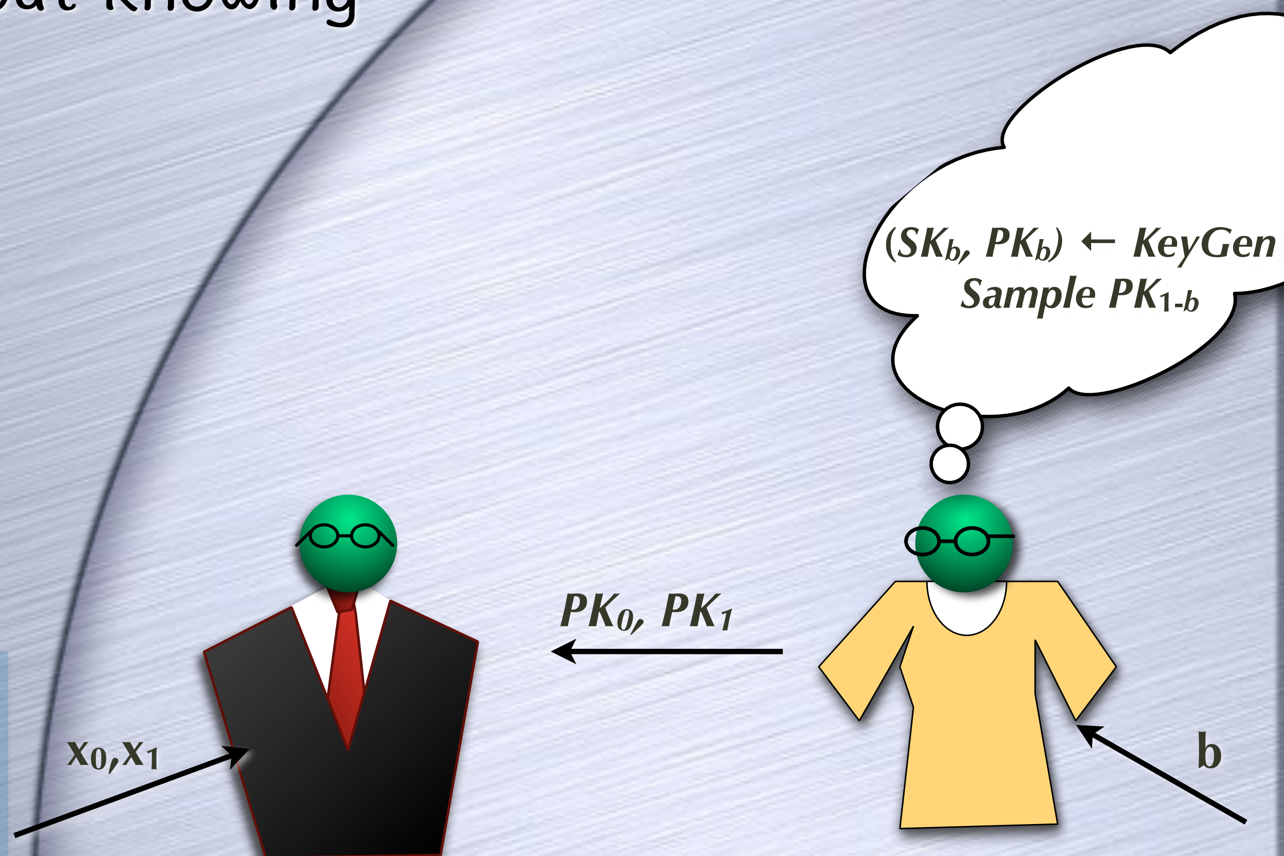




An OT Protocol (passive corruption)

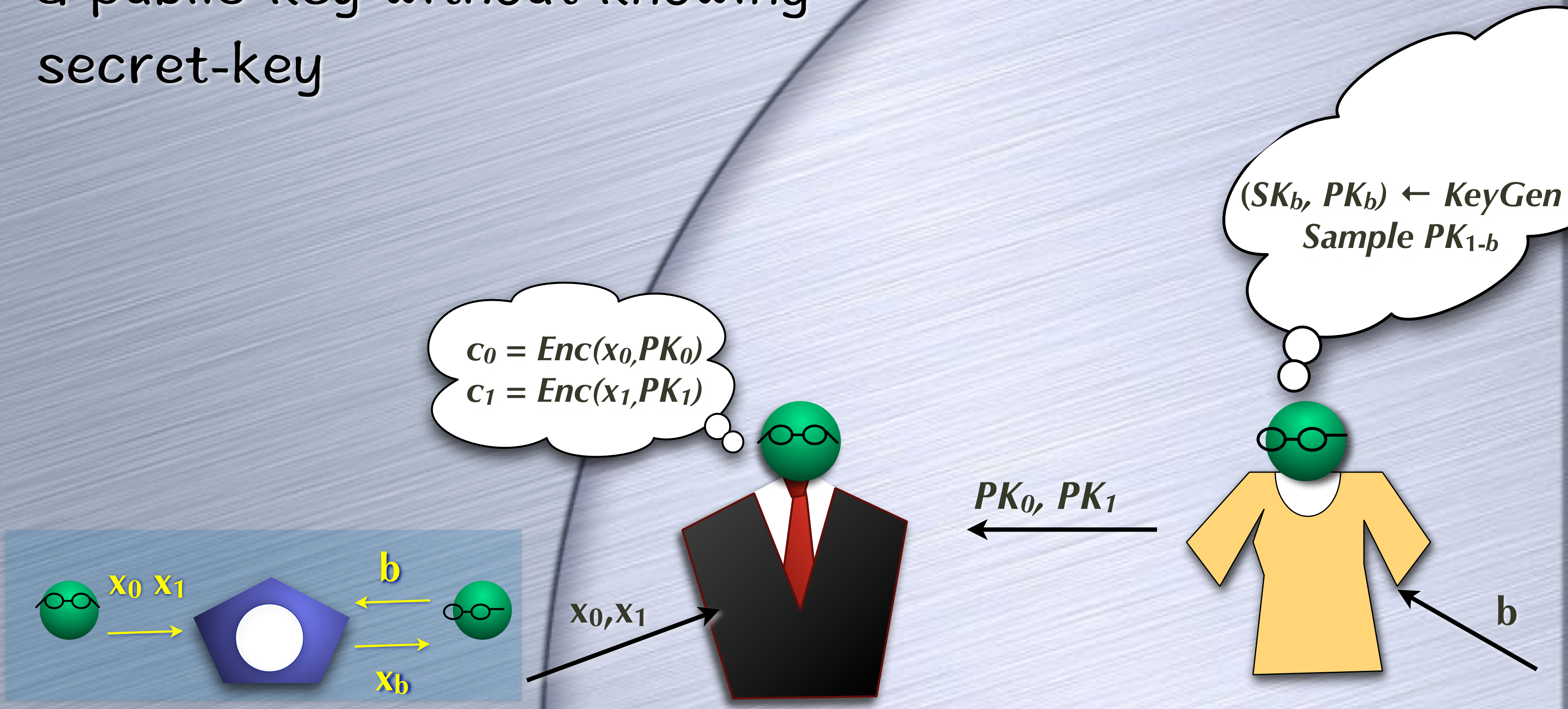
- Using a (special) encryption
  - PKE in which one can sample a public-key without knowing secret-key





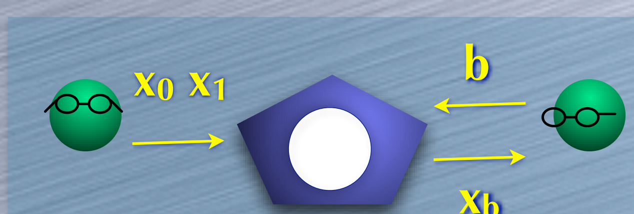
An OT Protocol (passive corruption)

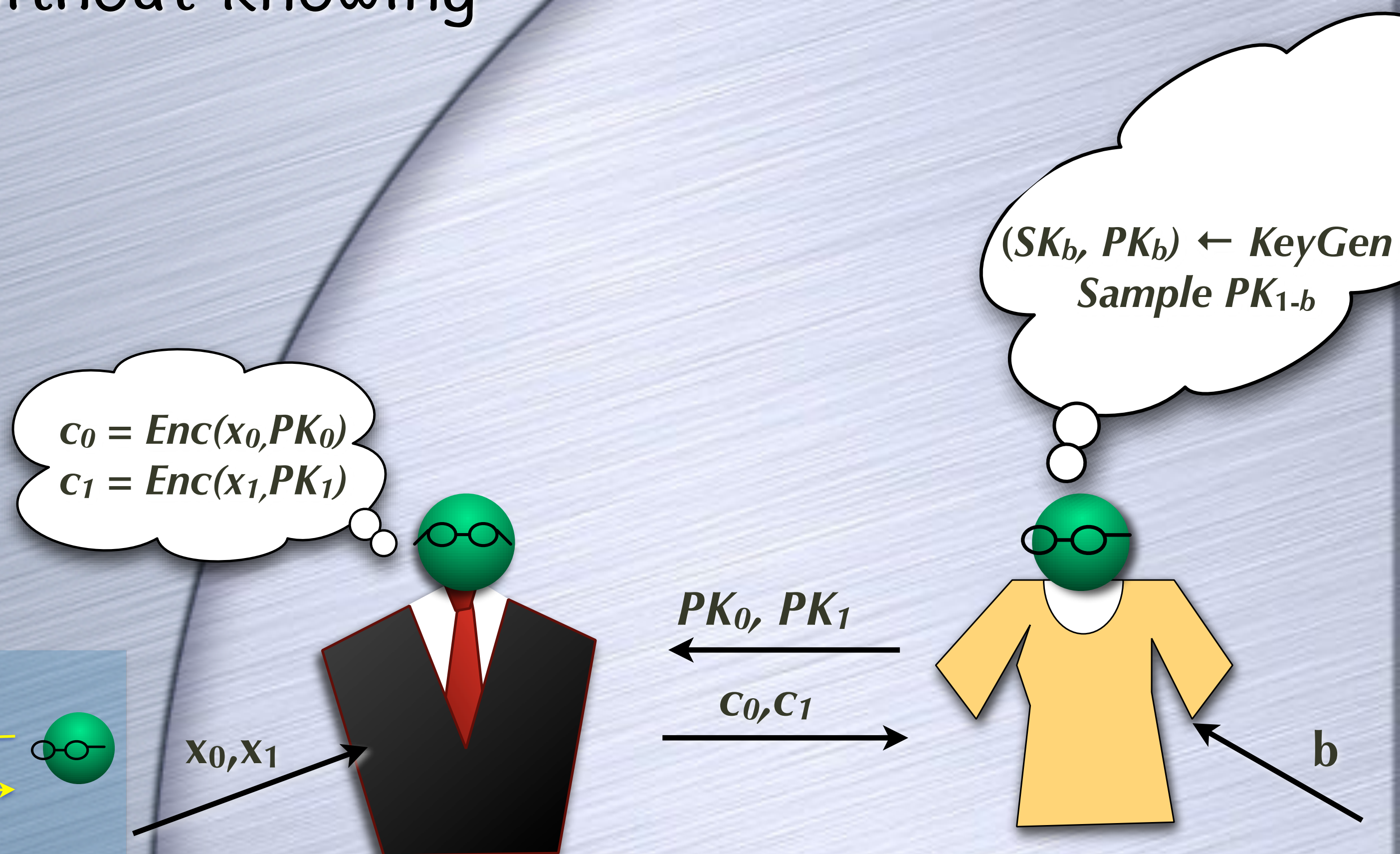
- Using a (special) encryption
  - PKE in which one can sample a public-key without knowing secret-key



An OT Protocol
(passive corruption)

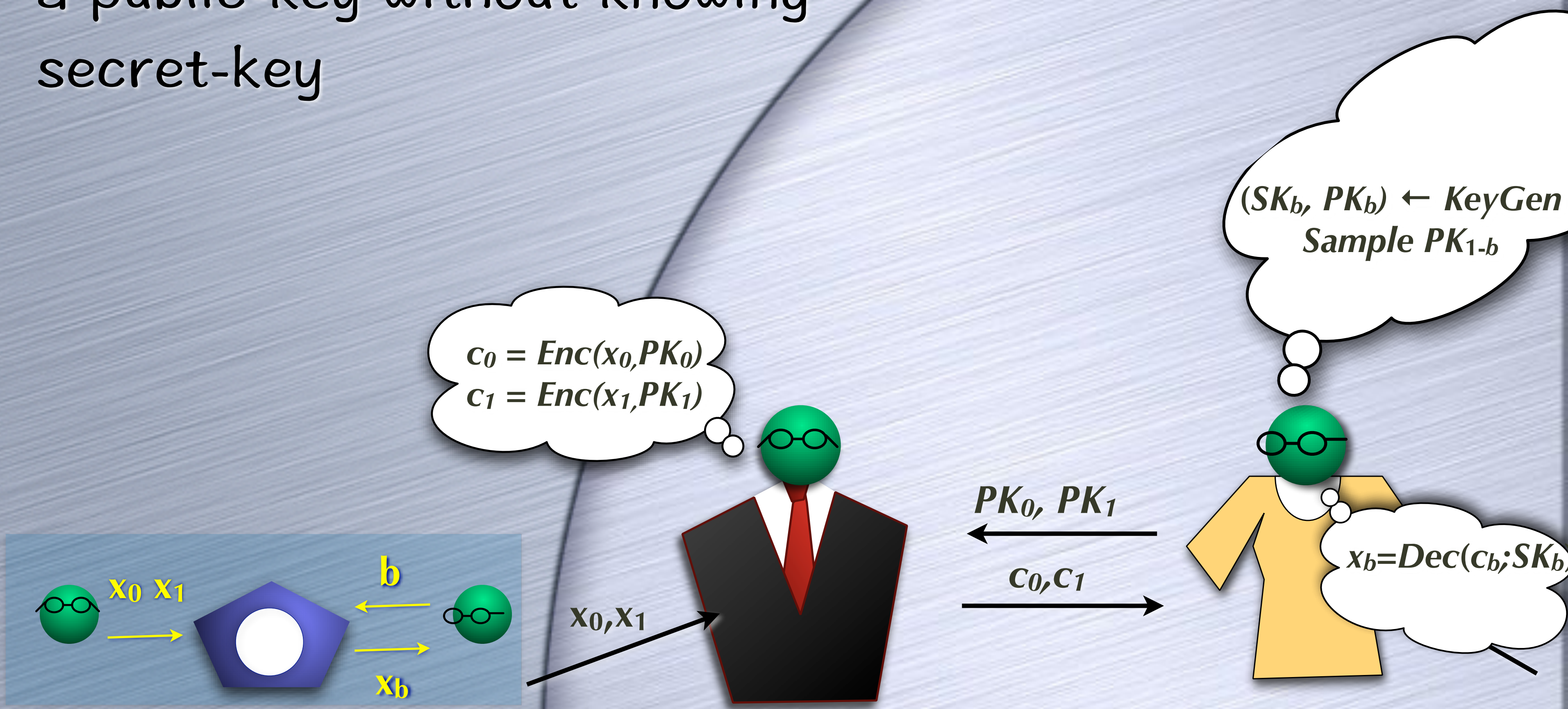
- Using a (special) encryption
  - PKE in which one can sample a public-key without knowing secret-key





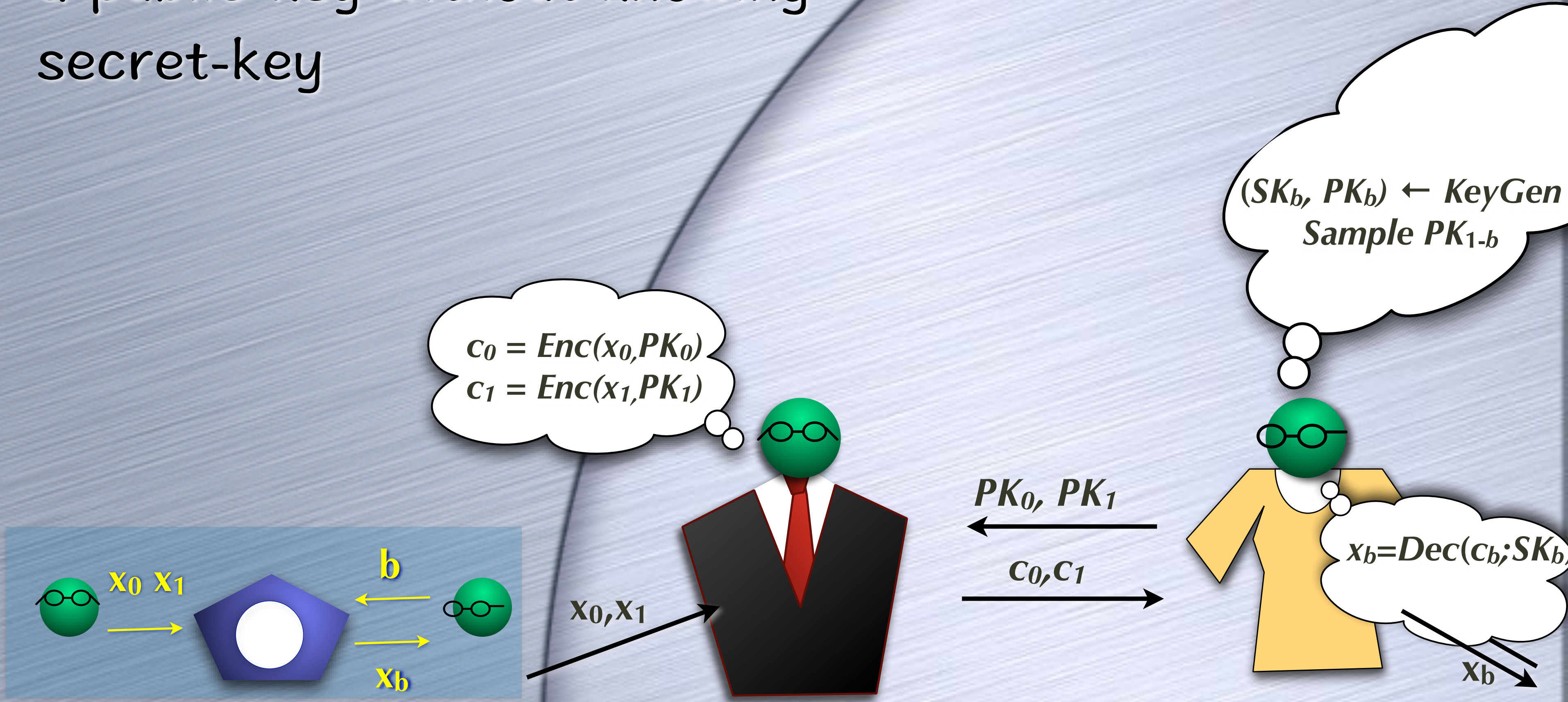
An OT Protocol (passive corruption)

- Using a (special) encryption
  - PKE in which one can sample a public-key without knowing secret-key



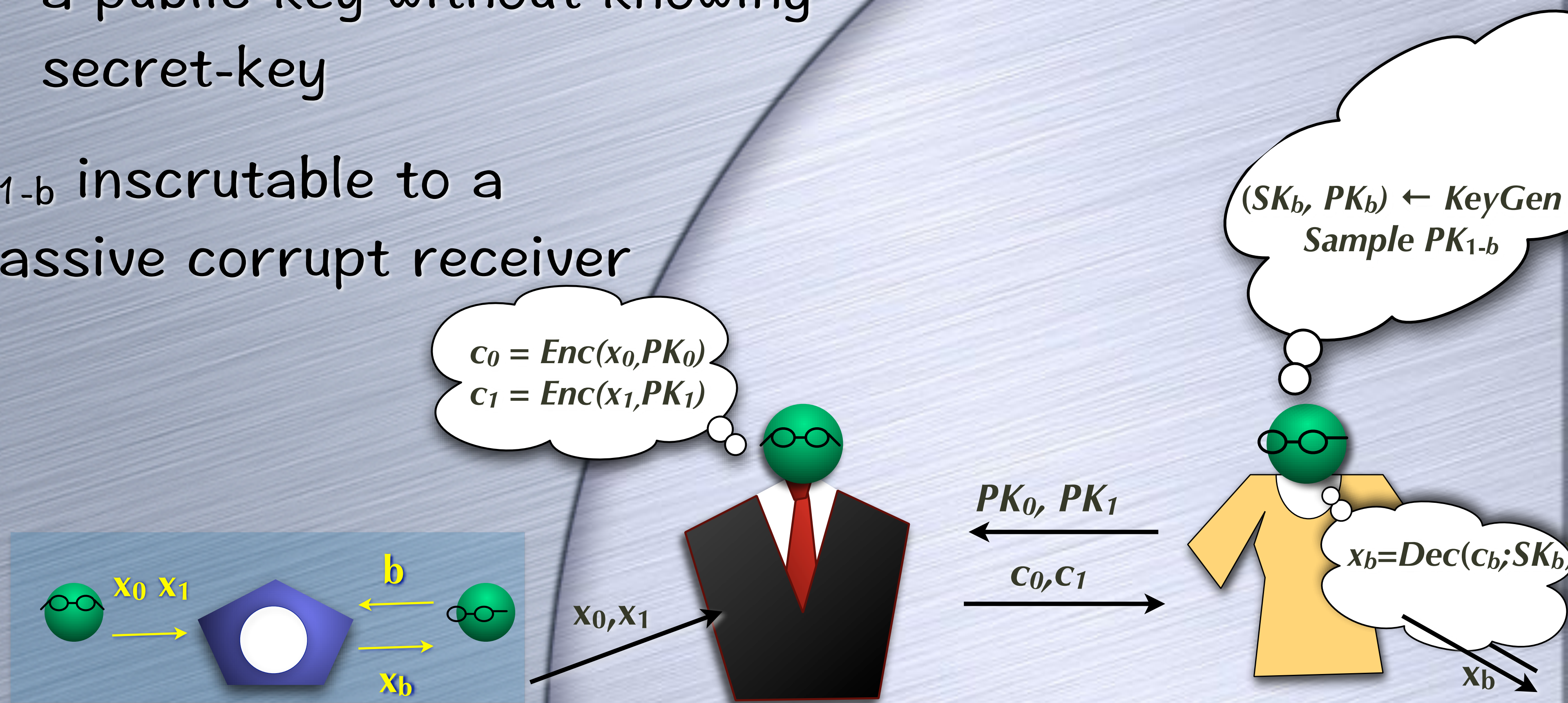
An OT Protocol (passive corruption)

- Using a (special) encryption
  - PKE in which one can sample a public-key without knowing secret-key



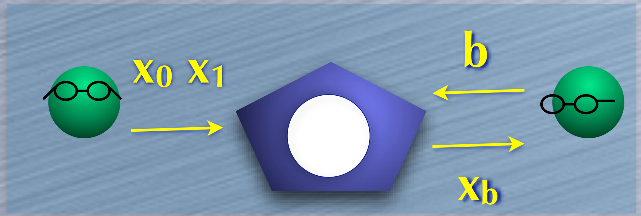
An OT Protocol (passive corruption)

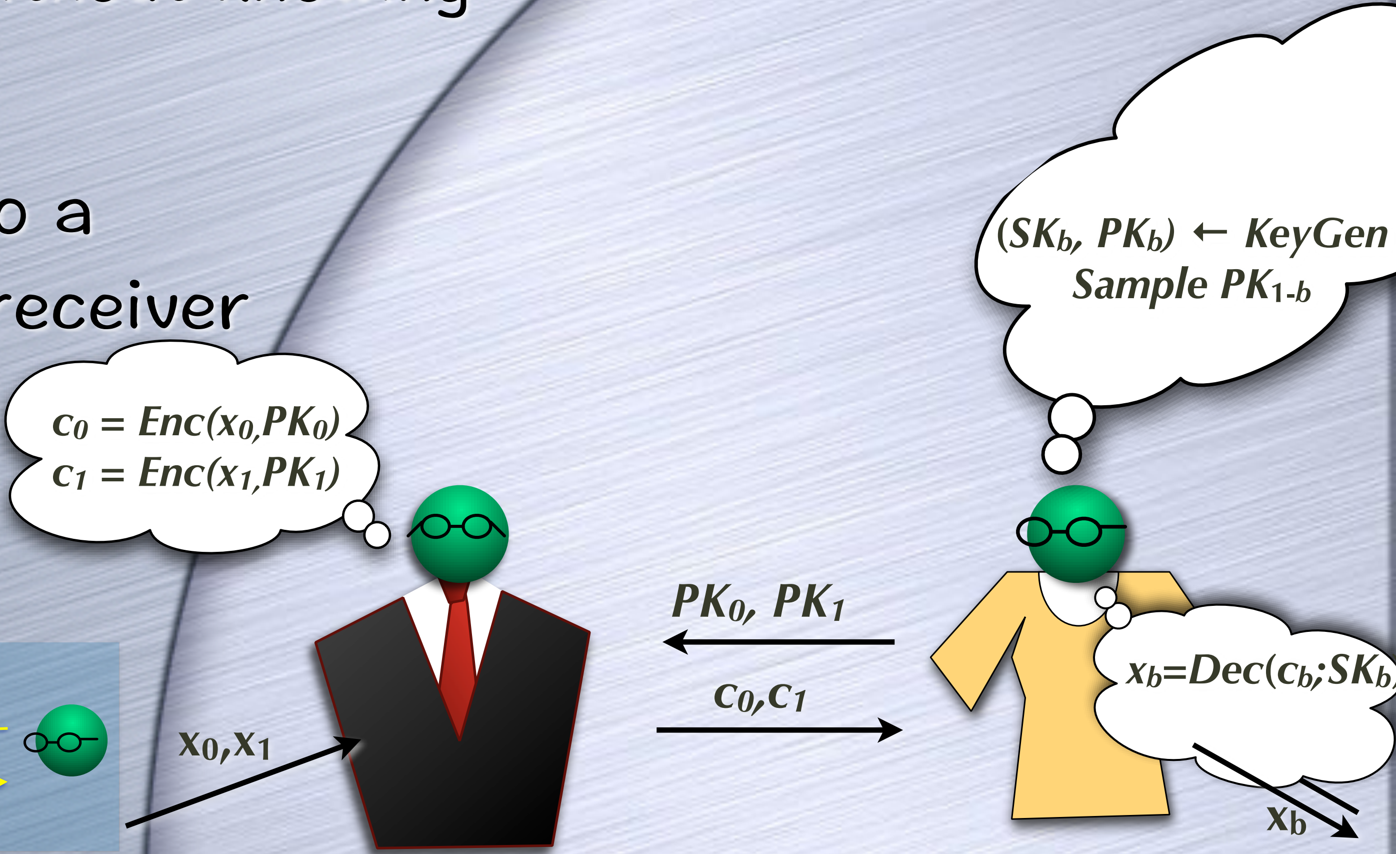
- Using a (special) encryption
  - PKE in which one can sample a public-key without knowing secret-key
- c_{1-b} inscrutable to a passive corrupt receiver



An OT Protocol (passive corruption)

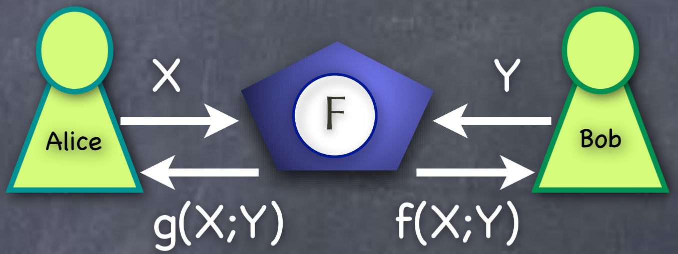
- Using a (special) encryption
  - PKE in which one can sample a public-key without knowing secret-key
- c_{1-b} inscrutable to a passive corrupt receiver
- Sender learns nothing about b





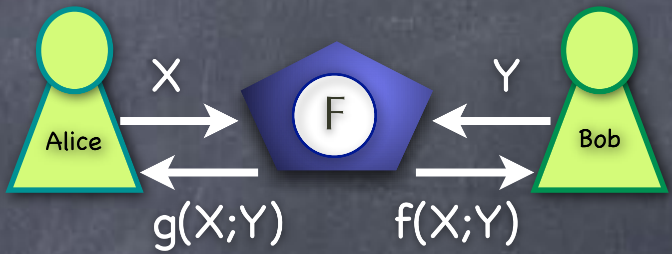
2-Party SFE

- Secure Function Evaluation (SFE) IDEAL:
  - Trusted party takes (X;Y). Outputs g(X;Y) to Alice, f(X;Y) to Bob



2-Party SFE

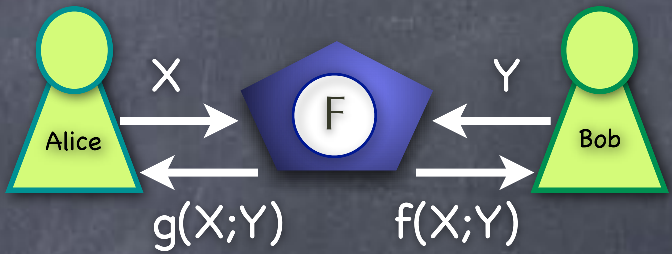
- Secure Function Evaluation (SFE) IDEAL:
  - Trusted party takes (X;Y). Outputs g(X;Y) to Alice, f(X;Y) to Bob



    - Randomized Functions: g(X;Y;r) and f(X;Y;r) s.t. neither party knows r (beyond what is revealed by output)

2-Party SFE

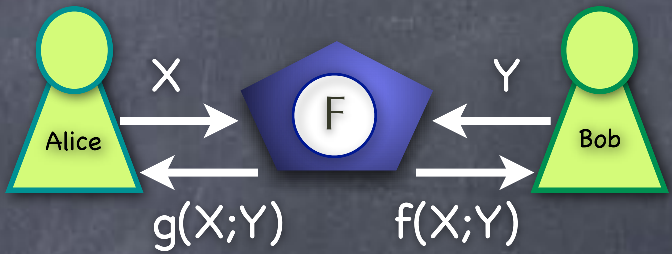
- Secure Function Evaluation (SFE) IDEAL:
  - Trusted party takes (X;Y). Outputs g(X;Y) to Alice, f(X;Y) to Bob



    - Randomized Functions: g(X;Y;r) and f(X;Y;r) s.t. neither party knows r (beyond what is revealed by output)
- OT is an instance of a (deterministic) 2-party SFE

2-Party SFE

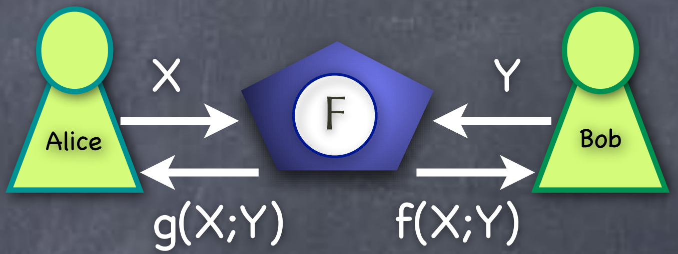
- Secure Function Evaluation (SFE) IDEAL:
  - Trusted party takes (X;Y). Outputs g(X;Y) to Alice, f(X;Y) to Bob



    - Randomized Functions: g(X;Y;r) and f(X;Y;r) s.t. neither party knows r (beyond what is revealed by output)
- OT is an instance of a (deterministic) 2-party SFE
  - $g(x_0,x_1;b) = \text{none}$; $f(x_0,x_1;b) = x_b$

2-Party SFE

- Secure Function Evaluation (SFE) IDEAL:
  - Trusted party takes (X;Y). Outputs g(X;Y) to Alice, f(X;Y) to Bob



    - Randomized Functions: g(X;Y;r) and f(X;Y;r) s.t. neither party knows r (beyond what is revealed by output)
- OT is an instance of a (deterministic) 2-party SFE
  - $g(x_0,x_1;b)$ = none; $f(x_0,x_1;b) = x_b$
  - <u>Single-Output SFE</u>: only one party gets any output

2-Party SFE

- Can reduce any SFE (even randomized) to a single-output deterministic SFE
  - $f'(X, M, r_1; Y, r_2) = (\ g(X; Y; r_1 \oplus r_2) \oplus M,\ f(X; Y; r_1 \oplus r_2)\ )$. Compute $f'(X, M, r_1; Y, r_2)$ with random M, r_1, r_2
  - Bob sends $g(X, Y; r_1 \oplus r_2) \oplus M$ to Alice

2-Party SFE

- Can reduce any SFE (even randomized) to a single-output deterministic SFE
  - $f'(X, M, r_1; Y, r_2) = (\ g(X; Y; r_1 \oplus r_2) \oplus M,\ f(X; Y; r_1 \oplus r_2)\ )$. Compute $f'(X, M, r_1; Y, r_2)$ with random M, r_1, r_2
  - Bob sends $g(X, Y; r_1 \oplus r_2) \oplus M$ to Alice
  - Passive secure

2-Party SFE

- Can reduce any SFE (even randomized) to a single-output deterministic SFE
  - $f'(X, M, r_1; Y, r_2) = (g(X; Y; r_1 \oplus r_2) \oplus M, f(X; Y; r_1 \oplus r_2))$. Compute $f'(X, M, r_1; Y, r_2)$ with random M, r_1, r_2
  - Bob sends $g(X, Y; r_1 \oplus r_2) \oplus M$ to Alice
  - Passive secure
  - Generalizes to active security and more than 2 parties

2-Party SFE

- Can reduce any SFE (even randomized) to a single-output deterministic SFE
  - $f'(X, M, r_1; Y, r_2) = (\ g(X; Y; r_1 \oplus r_2) \oplus M,\ f(X; Y; r_1 \oplus r_2)\ )$. Compute $f'(X, M, r_1; Y, r_2)$ with random M, r_1, r_2
  - Bob sends $g(X, Y; r_1 \oplus r_2) \oplus M$ to Alice
  - Passive secure
  - Generalizes to active security and more than 2 parties
- **Can reduce any single-output deterministic SFE to OT!**

“Completeness” of OT

"Completeness" of OT

- **Can reduce any single-output deterministic SFE to OT!**

“Completeness” of OT

- **Can reduce any single-output deterministic SFE to OT!**
- For passive security

“Completeness” of OT

- **Can reduce any single-output deterministic SFE to OT!**
- For passive security
  - Proof of concept for 2 parties: An inefficient reduction

"Completeness" of OT

- **Can reduce any single-output deterministic SFE to OT!**
- For passive security
  - Proof of concept for 2 parties: An inefficient reduction
  - Yao's garbled circuit for 2 parties (later today)

“Completeness” of OT

- **Can reduce any single-output deterministic SFE to OT!**
- For passive security
  - Proof of concept for 2 parties: An inefficient reduction
  - Yao’s garbled circuit for 2 parties (later today)
  - “Basic GMW”: Information-theoretic reduction to OT

“Completeness” of OT

- **Can reduce any single-output deterministic SFE to OT!**
- For passive security
  - Proof of concept for 2 parties: An inefficient reduction
  - Yao’s garbled circuit for 2 parties (later today)
  - “Basic GMW”: Information-theoretic reduction to OT
- In fact, OT is complete even for active security

"Completeness" of OT: Proof of Concept

- Single-output 2-party function f
- Alice (who knows x, but not y) prepares a table for $f(x,\cdot)$ with $N = 2^{|y|}$ entries (one for each y)
- Bob uses y to decide which entry in the table to pick up using 1-out-of-N OT (without learning the other entries)

"Completeness" of OT: Proof of Concept

- Single-output 2-party function f
- Alice (who knows x, but not y) prepares a table for $f(x,\cdot)$ with $N = 2^{|y|}$ entries (one for each y)
- Bob uses y to decide which entry in the table to pick up using 1-out-of-N OT (without learning the other entries)
- Bob learns only f(x,y) (in addition to y). Alice learns nothing beyond x.

"Completeness" of OT: Proof of Concept

- Single-output 2-party function f
- Alice (who knows x, but not y) prepares a table for $f(x,\cdot)$ with $N = 2^{|y|}$ entries (one for each y)
- Bob uses y to decide which entry in the table to pick up using 1-out-of-N OT (without learning the other entries)
- Bob learns only f(x,y) (in addition to y). Alice learns nothing beyond x.
- Problem: N is exponentially large in |y|

1-out-of-N OT

1-out-of-N OT

- $f(\ (x_1,\ldots,x_N);\ i\ ) = (\perp;\ x_i)$

1-out-of-N OT

- $f((x_1,\ldots,x_N); i) = (\perp; x_i)$
- For passive security: simply run N copies of 1-out-of-2 OT, with inputs for j^{th} instance being $(0,x_j; b_j)$ where $b_j = 1$ iff $j=i$

1-out-of-N OT

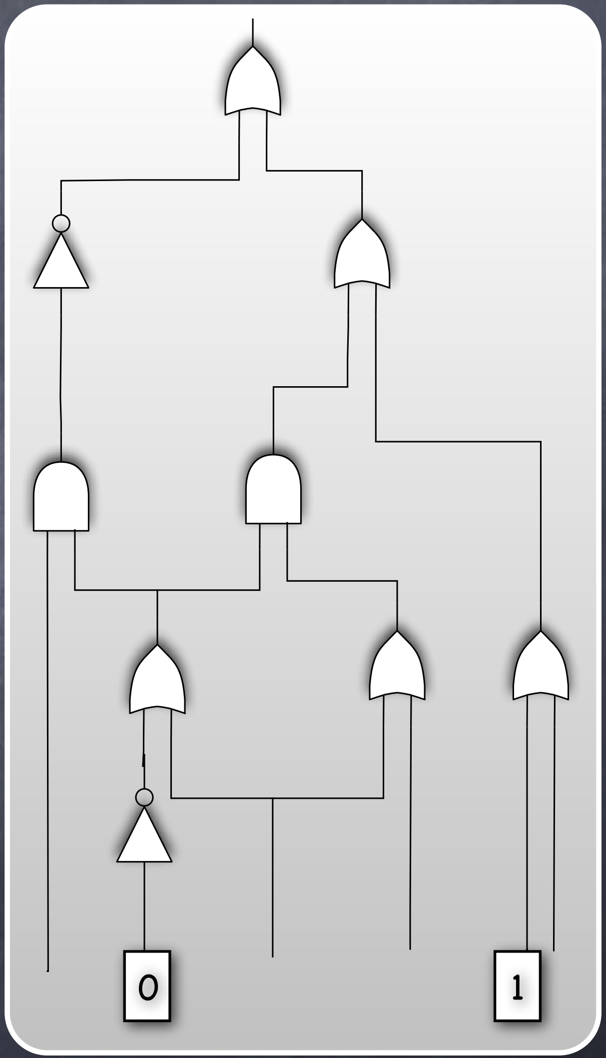
- $f((x_1,\ldots,x_N); i) = (\perp; x_i)$

- For passive security: simply run N copies of 1-out-of-2 OT, with inputs for j^{th} instance being $(0,x_j; b_j)$ where $b_j = 1$ iff $j=i$

- Aside: active security easily achievable too using a randomized protocol using N-1 copies of 1-out-of-2 OT

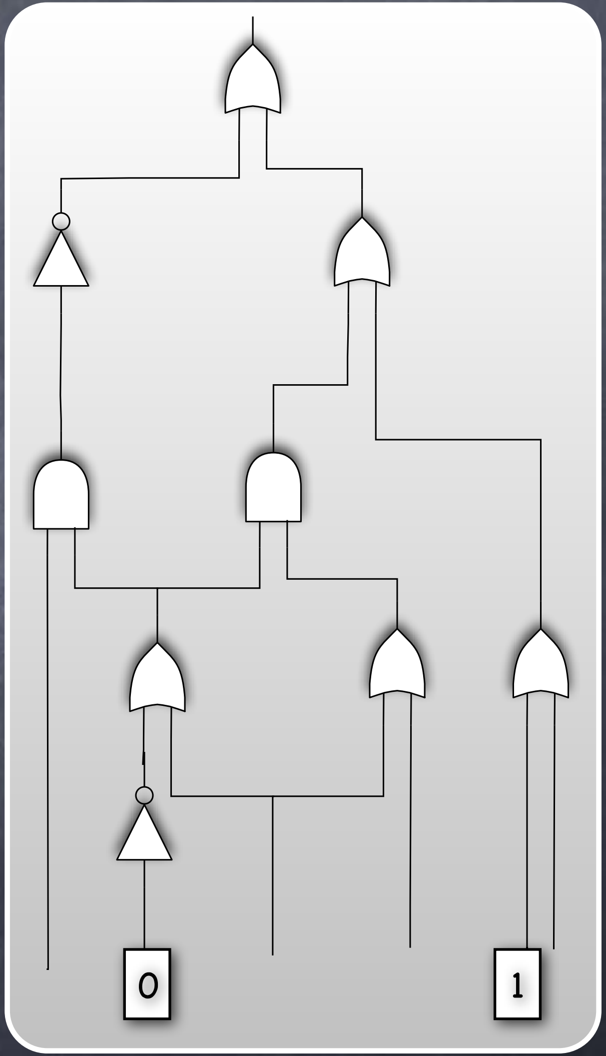
Functions as Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - Each wire comes out of a unique gate, but a wire might fan-out



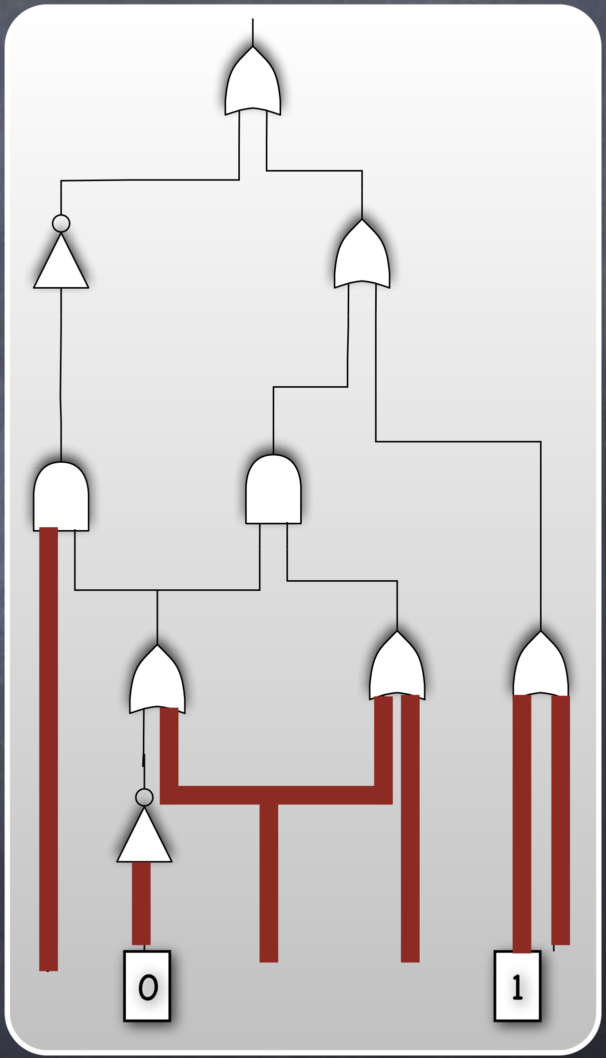
Functions as Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - Each wire comes out of a unique gate, but a wire might fan-out
  - Can evaluate wires according to a topologically sorted order of gates they come out of



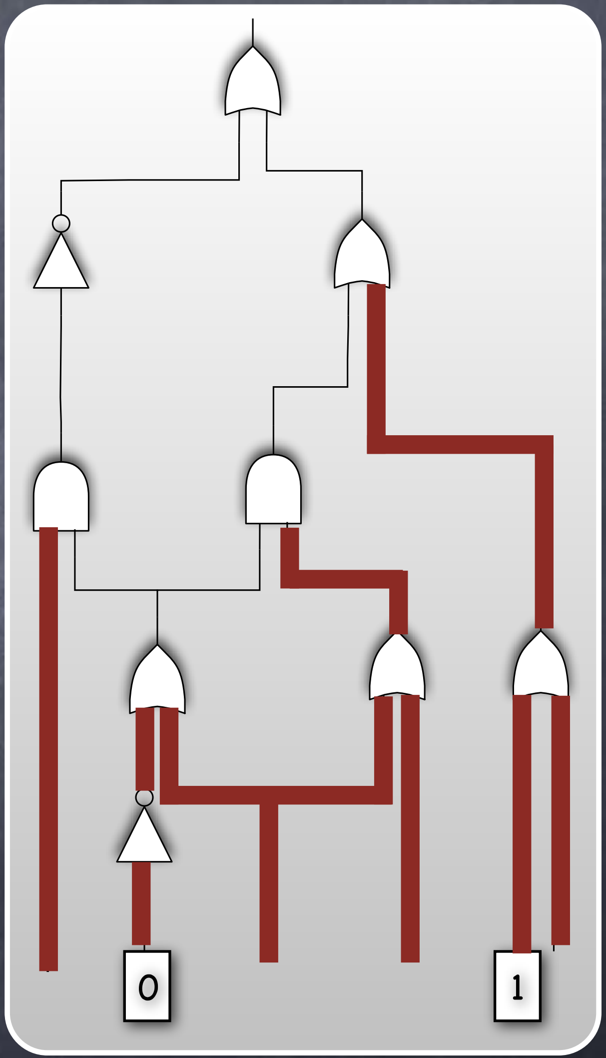
Functions as Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - Each wire comes out of a unique gate, but a wire might fan-out
  - Can evaluate wires according to a topologically sorted order of gates they come out of



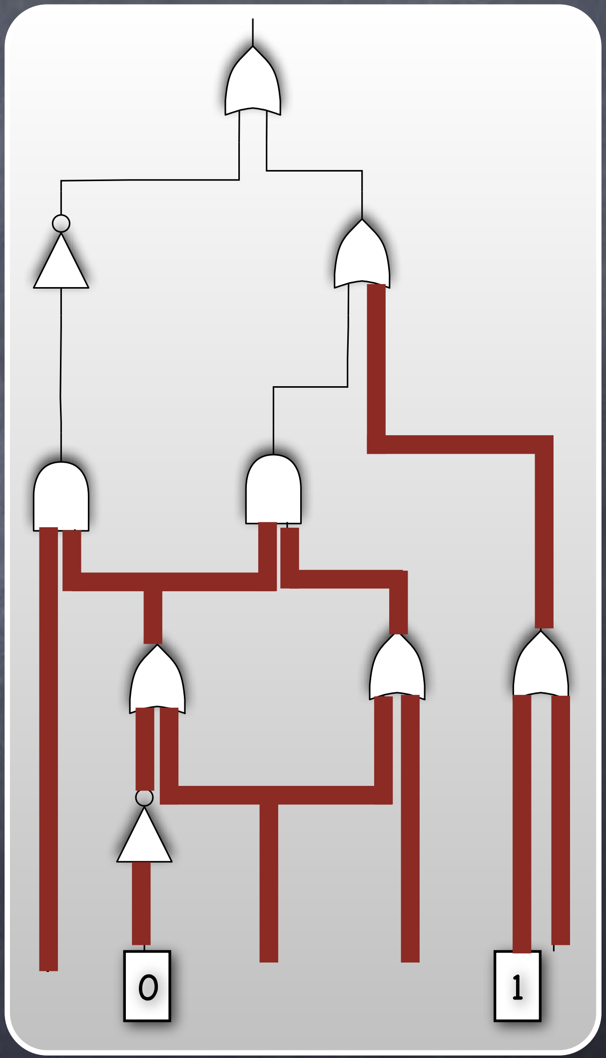
Functions as Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - Each wire comes out of a unique gate, but a wire might fan-out
  - Can evaluate wires according to a topologically sorted order of gates they come out of



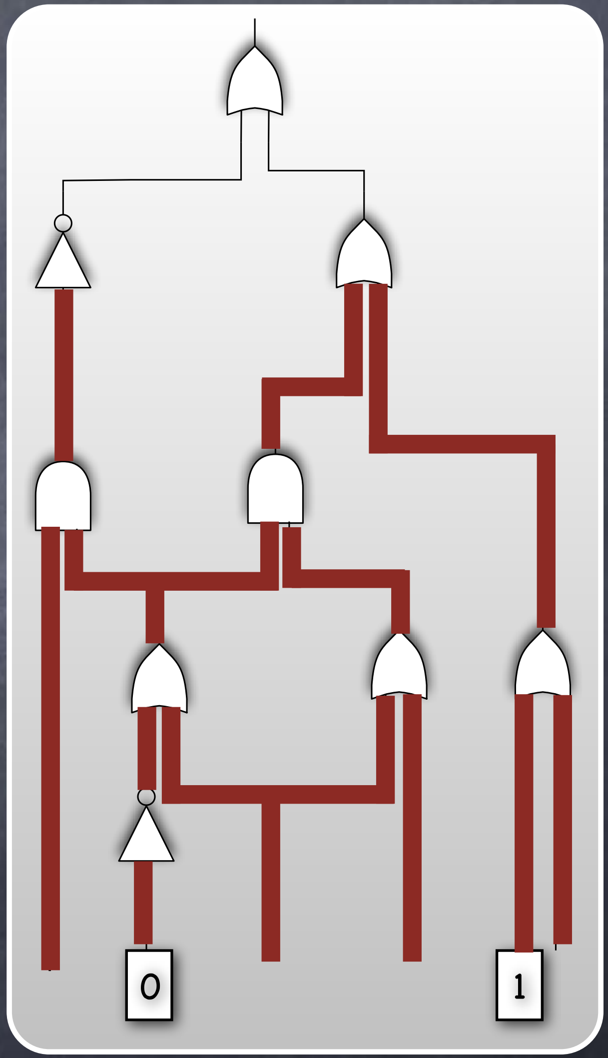
Functions as Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - Each wire comes out of a unique gate, but a wire might fan-out
  - Can evaluate wires according to a topologically sorted order of gates they come out of



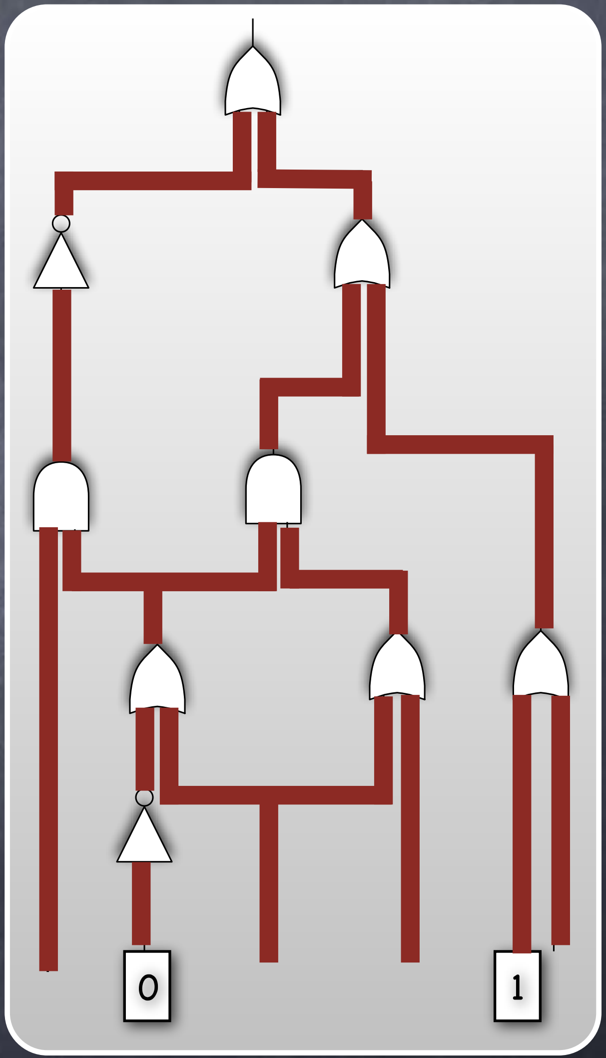
Functions as Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - Each wire comes out of a unique gate, but a wire might fan-out
  - Can evaluate wires according to a topologically sorted order of gates they come out of



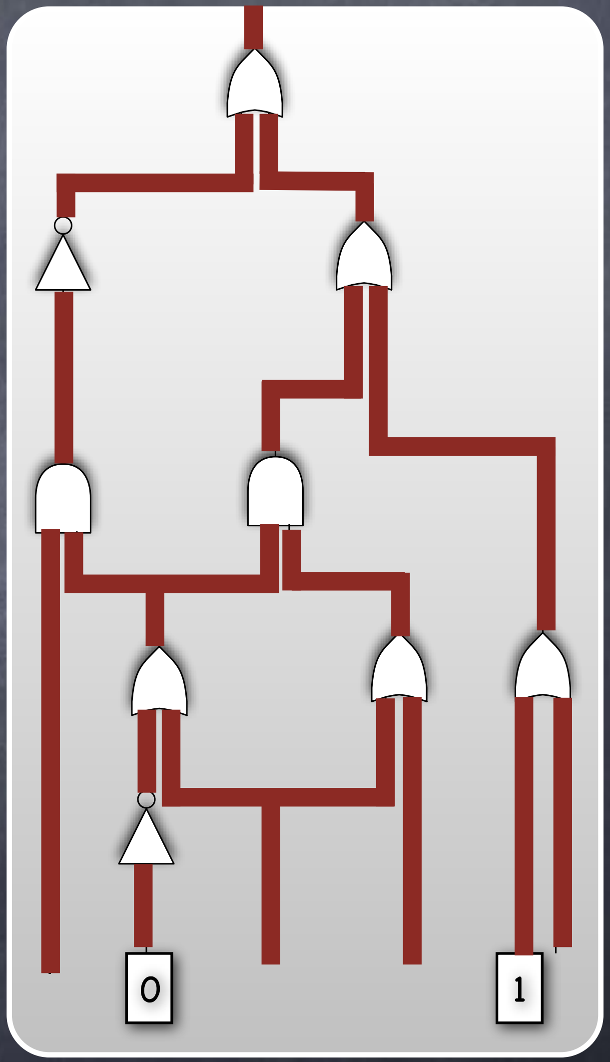
Functions as Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - Each wire comes out of a unique gate, but a wire might fan-out
  - Can evaluate wires according to a topologically sorted order of gates they come out of



Functions as Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - Each wire comes out of a unique gate, but a wire might fan-out
  - Can evaluate wires according to a topologically sorted order of gates they come out of



Functions as Circuits

Functions as Circuits

- e.g.: OR (single gate, 2 input bits, 1 bit output)

Functions as Circuits

- e.g.: OR (single gate, 2 input bits, 1 bit output)
- e.g.: X > Y for two bit inputs $X=x_1x_0$, $Y=y_1y_0$:
  $(x_1 \wedge \neg y_1) \vee (\neg(x_1 \oplus y_1) \wedge (x_0 \wedge \neg y_0)$

	00	01	10	11
00	0	0	0	0
01	1	0	0	0
10	1	1	0	0
11	1	1	1	0

Functions as Circuits

- e.g.: OR (single gate, 2 input bits, 1 bit output)
- e.g.: $X > Y$ for two bit inputs $X=x_1x_0$, $Y=y_1y_0$: $(x_1 \wedge \neg y_1) \vee (\neg(x_1 \oplus y_1) \wedge (x_0 \wedge \neg y_0)$
- Can directly convert a truth-table into a circuit, but circuit size exponential in input size

	00	01	10	11
00	0	0	0	0
01	1	0	0	0
10	1	1	0	0
11	1	1	1	0

Functions as Circuits

- e.g.: OR (single gate, 2 input bits, 1 bit output)
- e.g.: X > Y for two bit inputs $X=x_1x_0$, $Y=y_1y_0$: $(x_1 \wedge \neg y_1) \vee (\neg(x_1 \oplus y_1) \wedge (x_0 \wedge \neg y_0))$
- Can directly convert a truth-table into a circuit, but circuit size exponential in input size
- Can convert any ("efficient") program into a ("small") circuit

	00	01	10	11
00	0	0	0	0
01	1	0	0	0
10	1	1	0	0
11	1	1	1	0

Functions as Circuits

- e.g.: OR (single gate, 2 input bits, 1 bit output)
- e.g.: X > Y for two bit inputs $X=x_1x_0$, $Y=y_1y_0$:
  $(x_1 \wedge \neg y_1) \vee (\neg(x_1 \oplus y_1) \wedge (x_0 \wedge \neg y_0))$
- Can directly convert a truth-table into a circuit, but circuit size exponential in input size
- Can convert any ("efficient") program into a ("small") circuit
- Interesting problems already given as succinct programs/circuits

	00	01	10	11
00	0	0	0	0
01	1	0	0	0
10	1	1	0	0
11	1	1	1	0

Basic GMW

Basic GMW

- Adapted from the famous Goldreich-Micali-Wigderson (1987) protocol (due to Goldreich-Vainish, Haber-Micali,...)

Basic GMW

- Adapted from the famous Goldreich-Micali-Wigderson (1987) protocol (due to Goldreich-Vainish, Haber-Micali,...)
- Efficient passive secure MPC based on OT, without any other computational assumptions

Basic GMW

- Adapted from the famous Goldreich-Micali-Wigderson (1987) protocol (due to Goldreich-Vainish, Haber-Micali,...)
- Efficient passive secure MPC based on OT, without any other computational assumptions
- Idea: Computing on **secret-shared values**

Computing on Shares

Computing on Shares

- Fix any "secret" s. Let a, b be random conditioned on s = a + b. (All elements from a finite field, e.g. GF(2))

Computing on Shares

- Fix any "secret" s. Let a, b be random conditioned on s = a + b. (All elements from a finite field, e.g. GF(2))
- Each of a, b by itself carries no information about s. (e.g., can pick a at random, set b = s - a.)

Computing on Shares

- Fix any "secret" s. Let a, b be random conditioned on s = a + b. (All elements from a finite field, e.g. GF(2))
- Each of a, b by itself carries no information about s. (e.g., can pick a at random, set b = s - a.)
- Will write $[s]_1$ and $[s]_2$ to denote shares of s

Computing on Shares

Computing on Shares

- Let gates be + & $\times$ (XOR & AND for Boolean circuits)

Computing on Shares

- Let gates be + & × (XOR & AND for Boolean circuits)
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.

Computing on Shares

- Let gates be + & × (XOR & AND for Boolean circuits)
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.

Computing on Shares

- Let gates be + & × (XOR & AND for Boolean circuits)
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.

Computing on Shares

- Let gates be + & × (XOR & AND for Boolean circuits)
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.

Computing on Shares

- Let gates be + & × (XOR & AND for Boolean circuits)
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.
- $w = u + v$: Each one locally computes $[w]_i = [u]_i + [v]_i$

Computing on Shares

- Let gates be + & × (XOR & AND for Boolean circuits)
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.
- w = u + v : Each one locally computes $[w]_i = [u]_i + [v]_i$

Computing on Shares

- What about $w = u \times v$?

Computing on Shares

- What about $w = u \times v$?
  - $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$

Computing on Shares

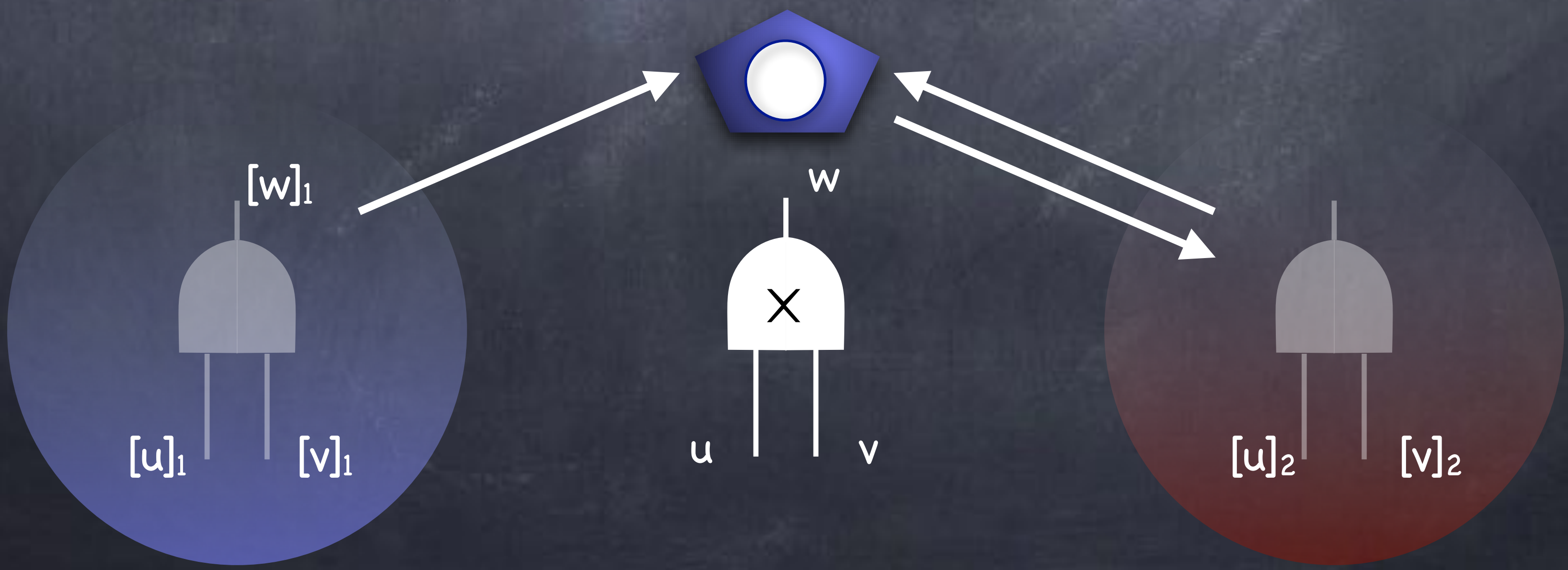
- What about $w = u \times v$?
  - $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
  - Alice picks $[w]_1$. Can let Bob compute $[w]_2$ using the naive (proof-of-concept) protocol

Computing on Shares

- What about $w = u \times v$?
  - $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
  - Alice picks $[w]_1$. Can let Bob compute $[w]_2$ using the naive (proof-of-concept) protocol

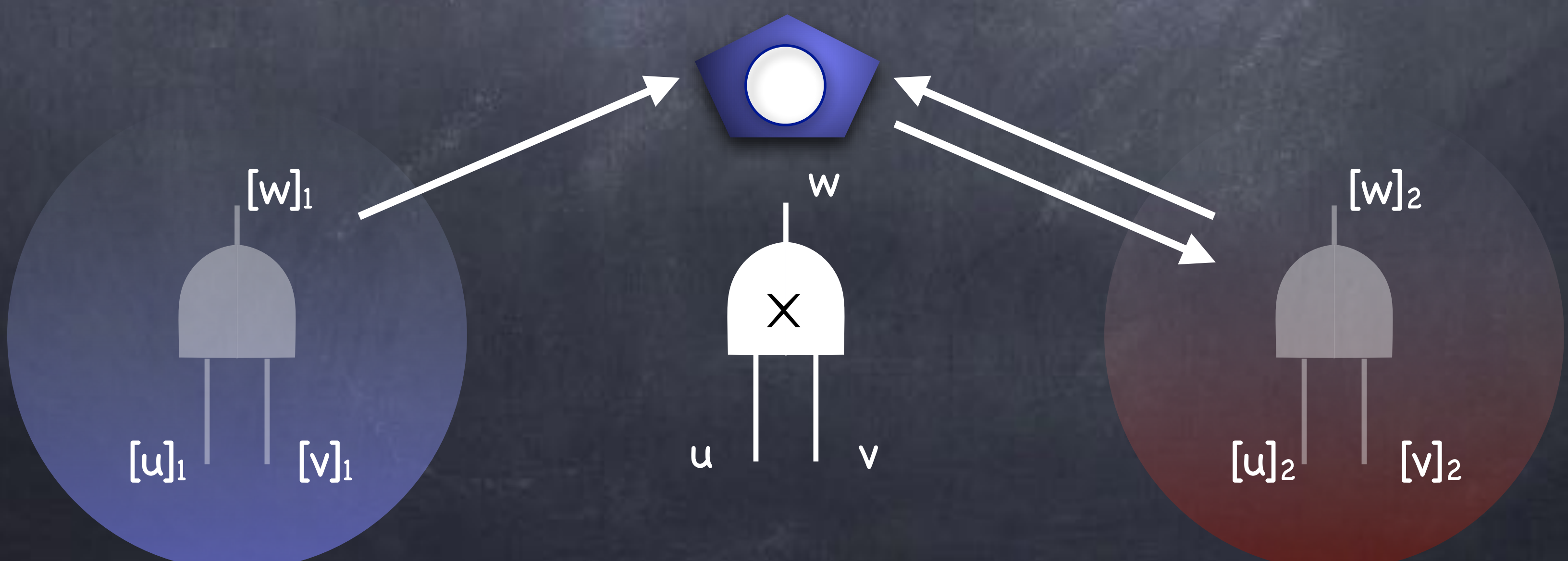
Computing on Shares

- What about $w = u \times v$?
  - $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
  - Alice picks $[w]_1$. Can let Bob compute $[w]_2$ using the naive (proof-of-concept) protocol



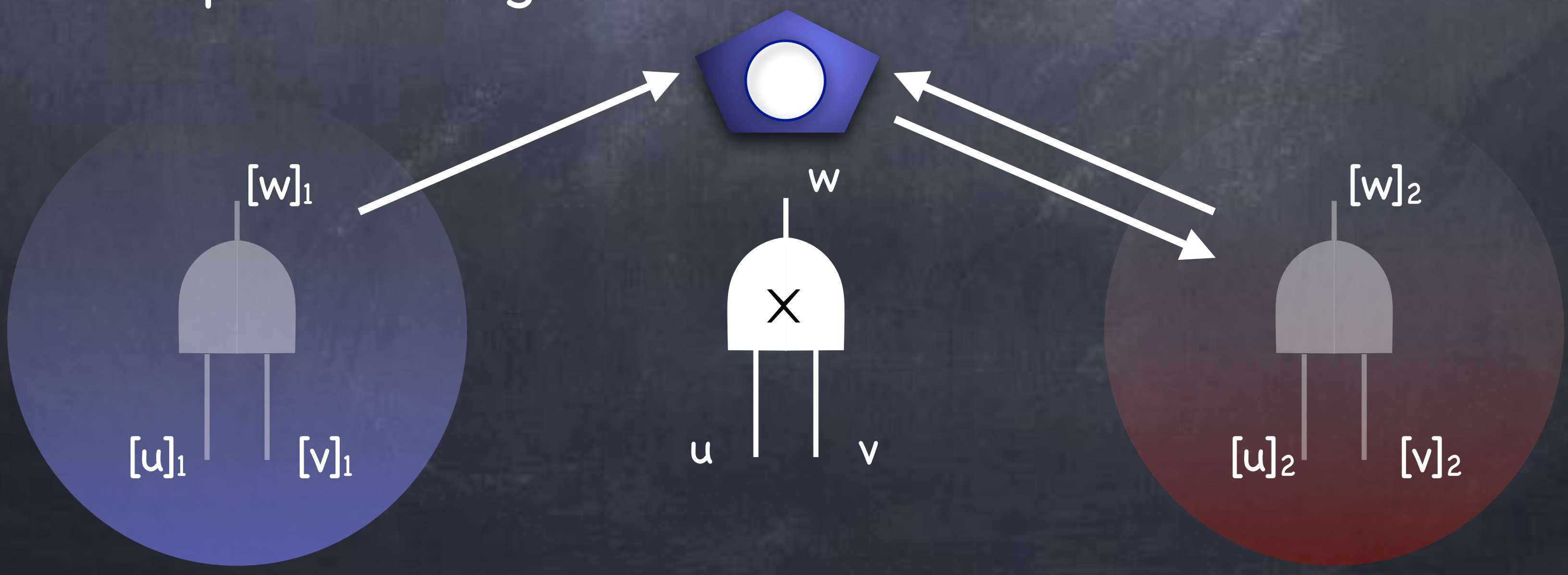
Computing on Shares

- What about $w = u \times v$?
  - $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
  - Alice picks $[w]_1$. Can let Bob compute $[w]_2$ using the naive (proof-of-concept) protocol



Computing on Shares

- What about $w = u \times v$?
  - $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
  - Alice picks $[w]_1$. Can let Bob compute $[w]_2$ using the naive (proof-of-concept) protocol
    - Note: Bob's input is $([u]_2,[v]_2)$. Over the binary field, this requires a single 1-out-of-4 OT.



GMW: many parties

GMW: many parties

- m-way sharing: $s = [s]_1 + \ldots + [s]_m$

Allows security against arbitrary number of corruptions

GMW: many parties

- m-way sharing: $s = [s]_1 + \ldots + [s]_m$
- Addition, local as before

Allows security against arbitrary number of corruptions

GMW: many parties

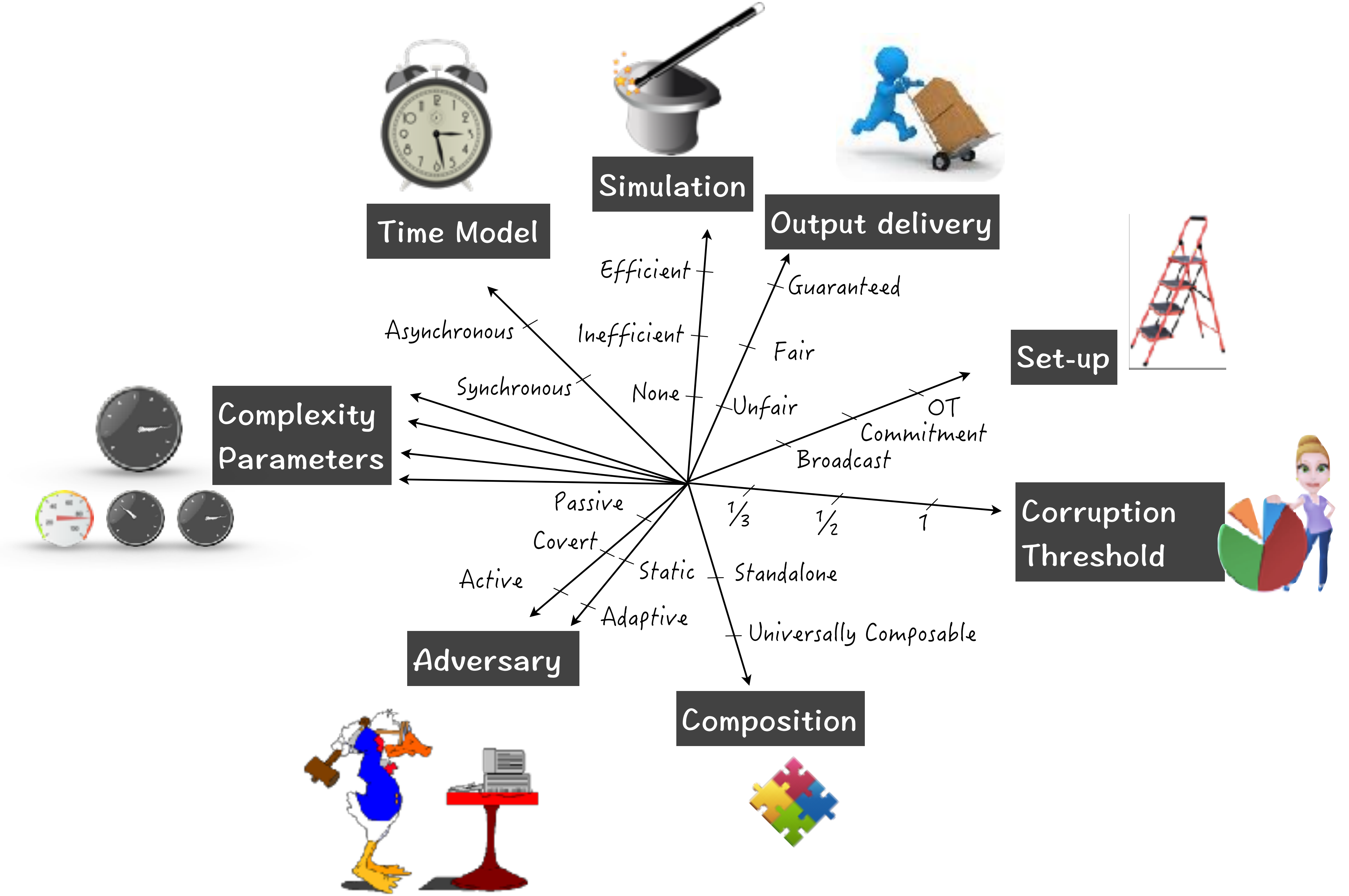
- m-way sharing: $s = [s]_1 + \ldots + [s]_m$

Allows security against arbitrary number of corruptions

- Addition, local as before
- Multiplication: For $w = u \times v$
  $[w]_1 + .. + [w]_m = ([u]_1 + .. + [u]_m) \times ([v]_1 + .. + [v]_m)$
  - Party i computes $[u]_i[v]_i$
  - For every pair (i,j), i≠j, Party i picks random a_{ij} and lets Party j securely compute b_{ij} s.t. $a_{ij} + b_{ij} = [u]_i[v]_j$ using the naive protocol (a single 1-out-of-2 OT)
  - Party i sets $[w]_i = [u]_i[v]_i + \Sigma_j (a_{ij} + b_{ji})$

MPC Dimensions

MPC Dimensions



Tutorials Outline

Today

Tutorials Outline

Today

Zero Knowledge proofs [GMR'86,...]

Tutorials Outline

Today

Zero Knowledge proofs [GMR’86,...]

- A special case of MPC, leading to many key concepts in crypto/complexity

Tutorials Outline

Today

Zero Knowledge proofs [GMR'86,...]

- A special case of MPC, leading to many key concepts in crypto/complexity
- Key ingredient in going from passive to active security

Tutorials Outline

Today

Zero Knowledge proofs [GMR'86,...]

- A special case of MPC, leading to many key concepts in crypto/complexity
- Key ingredient in going from passive to active security

Garbled Circuit [Yao'86,...]

Tutorials Outline

Today

Zero Knowledge proofs [GMR'86,...]

- A special case of MPC, leading to many key concepts in crypto/complexity
- Key ingredient in going from passive to active security

Garbled Circuit [Yao'86,...]

- First general purpose MPC (2-party, passive-security, using OT and symmetric-key encryption

Tutorials Outline

Tomorrow

Tutorials Outline

Tomorrow

Randomized Encoding

Tutorials Outline

Tomorrow

Randomized Encoding

- A general concept with applications to many crypto constructions

Tutorials Outline

Tomorrow

Randomized Encoding

- A general concept with applications to many crypto constructions
- Yao's Grabled Circuit is an instance of this

Tutorials Outline

Tomorrow

Randomized Encoding

- A general concept with applications to many crypto constructions
- Yao's Grabled Circuit is an instance of this

Oblivious Transfer

Tutorials Outline

Tomorrow

Randomized Encoding

- A general concept with applications to many crypto constructions
- Yao's Grabled Circuit is an instance of this

Oblivious Transfer

- And OT extension

Tutorials Outline

Tomorrow

- Randomized Encoding
  - A general concept with applications to many crypto constructions
  - Yao's Grabled Circuit is an instance of this

- Oblivious Transfer
  - And OT extension

- Composition issues

Tutorials Outline

Tomorrow

Randomized Encoding

- A general concept with applications to many crypto constructions
- Yao’s Grabled Circuit is an instance of this

Oblivious Transfer

- And OT extension

Composition issues

- Running two instances of a secure protocol needn’t be secure any more!

Tutorials Outline

Tomorrow

Randomized Encoding

- A general concept with applications to many crypto constructions
- Yao's Grabled Circuit is an instance of this

Oblivious Transfer

- And OT extension

Composition issues

- Running two instances of a secure protocol needn't be secure any more!

MPC Complexity

Tutorials Outline

Tomorrow

Randomized Encoding

- A general concept with applications to many crypto constructions
- Yao's Grabled Circuit is an instance of this

Oblivious Transfer

- And OT extension

Composition issues

- Running two instances of a secure protocol needn't be secure any more!

MPC Complexity

- "Cryptographic Complexity" of functionalities

Tutorials Outline
Wednesday

Tutorials Outline
Wednesday


Honest-Majority MPC

Tutorials Outline

Wednesday

Honest-Majority MPC

- When very strong security and output guarantees are possible

Tutorials Outline
Wednesday

Honest-Majority MPC

- When very strong security and output guarantees are possible
- Also useful as an encoding of computation

Tutorials Outline
Wednesday

Honest-Majority MPC

- When very strong security and output guarantees are possible
- Also useful as an encoding of computation

"MPC in the Head"

Tutorials Outline

Wednesday

Honest-Majority MPC

- When very strong security and output guarantees are possible
- Also useful as an encoding of computation

"MPC in the Head"

- A versatile technique for creating (non-honest-majority) MPC protocol from Honest-Majority MPC

Tutorials Outline
Wednesday

Honest-Majority MPC

- When very strong security and output guarantees are possible
- Also useful as an encoding of computation

"MPC in the Head"

- A versatile technique for creating (non-honest-majority) MPC protocol from Honest-Majority MPC

Asynchronous MPC

Tutorials Outline
Wednesday

Honest-Majority MPC

- When very strong security and output guarantees are possible
- Also useful as an encoding of computation

"MPC in the Head"

- A versatile technique for creating (non-honest-majority) MPC protocol from Honest-Majority MPC

Asynchronous MPC

- Everything till this point assumes a "synchronous" network