ILLINOIS INSTITUTE OF TECHNOLOGY

CS 528 (Fall 2021)
Data Privacy & Security

Yuan Hong
Department of Computer Science
Illinois Institute of Technology

Chapter 6
Secure Multiparty Computation

OUTLINE

- Intro to Secure Multiparty Computation
- Security Definition
- Oblivious Transfer
- Exploring Yao's Protocol with Garbled Circuits
- An Open Source Tool

THE LOVE GAME (AKA THE AND GAME)

ILLINOIS INSTITUTE OF TECHNOLOGY

Want to know if both parties are interested in each other

But…Do not want to reveal unrequited love.

Input = 1: I love you

Input = 0: I love you as a friend

Must compute F(x, y) = x AND y, giving F(x, y) to both players

Can they get the answer without revealing their inputs?

THE SPOILED CHILDREN PROBLEM (AKA YAO'S MILLIONAIRES PROBLEM)

ILLINOIS INSTITUTE OF TECHNOLOGY

Pearl wants to know whether she has more toys than Gersh
Does not want to tell Gersh anything.

Gersh is willing for Pearl to find out who has more toys,
Does not want Pearl to know how many toys he has.

Can we give Pearl the information she wants, and nothing else, without giving Gersh any information at all?

PRIVATE SET INTERSECTION

Intelligence agencies hold lists of potential terrorists

- They would like to compute the intersection
- Any other information must remain secret

Mossad

MI5

FBI

SECURE MULTIPARTY COMPUTATION

- A set of parties with private inputs
- Parties wish to jointly compute a function of their inputs so that certain security properties (like Privacy and Correctness) are preserved
- Properties must be ensured even if some of the parties maliciously attack the protocol
- Examples:
  - Secure elections
  - Auctions
  - Privacy Preserving Data Mining (Distributed Data)
  - …

EXAMPLES

- Parties do not trust each other
- Secure elections
  - N parties, each one has “Yes” or ”No” vote
  - Goal: determine whether the majority voted “Yes”, but no voter should learn how other people voted
- Auctions
  - Each bidder makes an offer
    - Offer should be committing! (cannot change it later)
  - Goal: determine whose offer won without revealing losing offers (and even the clear price)

MORE EXAMPLES

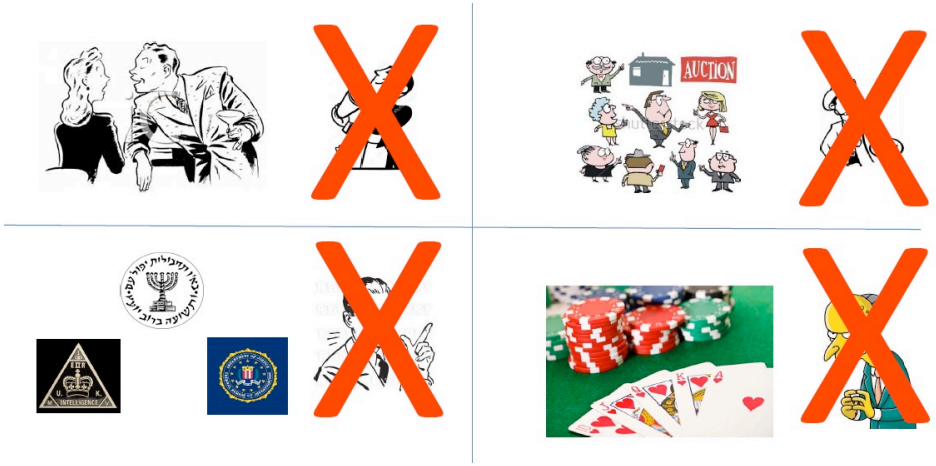
- Distributed Data Mining
  - Two companies want to compare their datasets without revealing them
    - For example, compute the intersection of two lists of names

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

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



OUTLINE

- Intro to Secure Multiparty Computation
- Security Definition
- Oblivious Transfer
- Exploring Yao's Protocol with Garbled Circuits
- An Open Source Tool

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 function

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

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

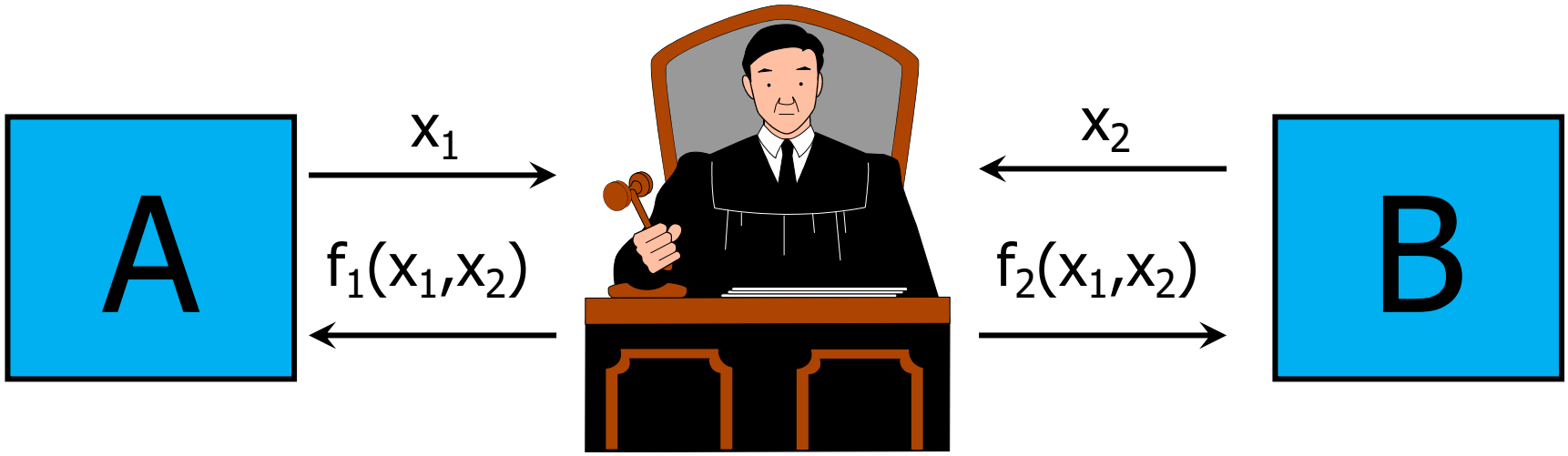
K outputs

- 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

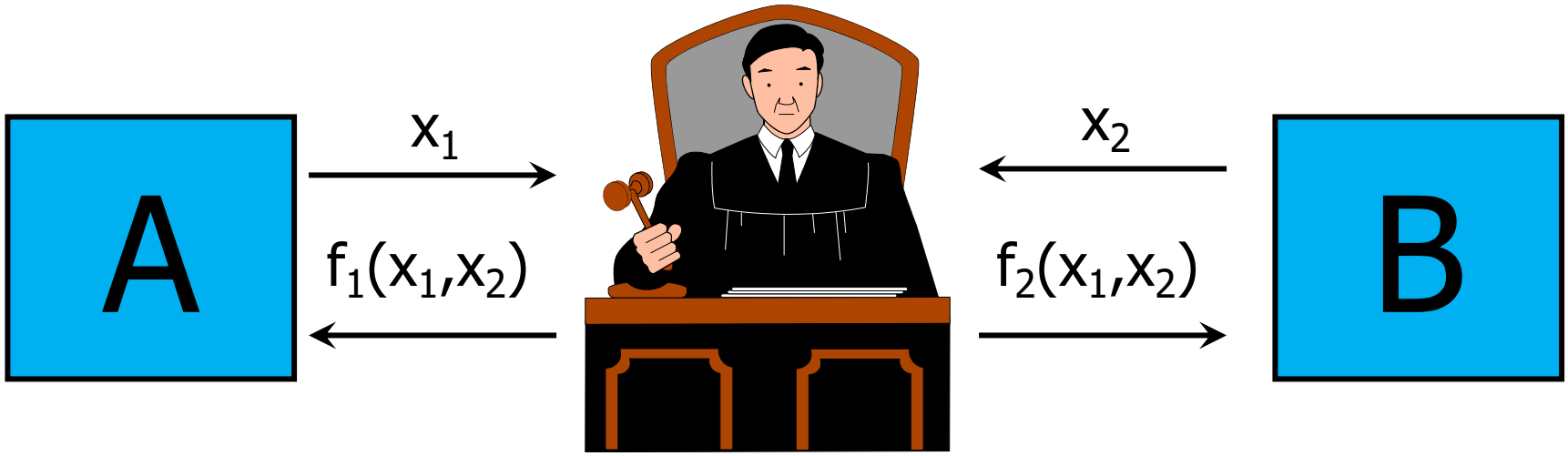
  Computation in the ideal model is secure by definition!



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 of protocol participants may be corrupt
  - If all were honest, 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
- We first 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 function 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 ideal model
  - What does corrupt participant learn in ideal model?
    - His input (obviously) and the result of evaluating f

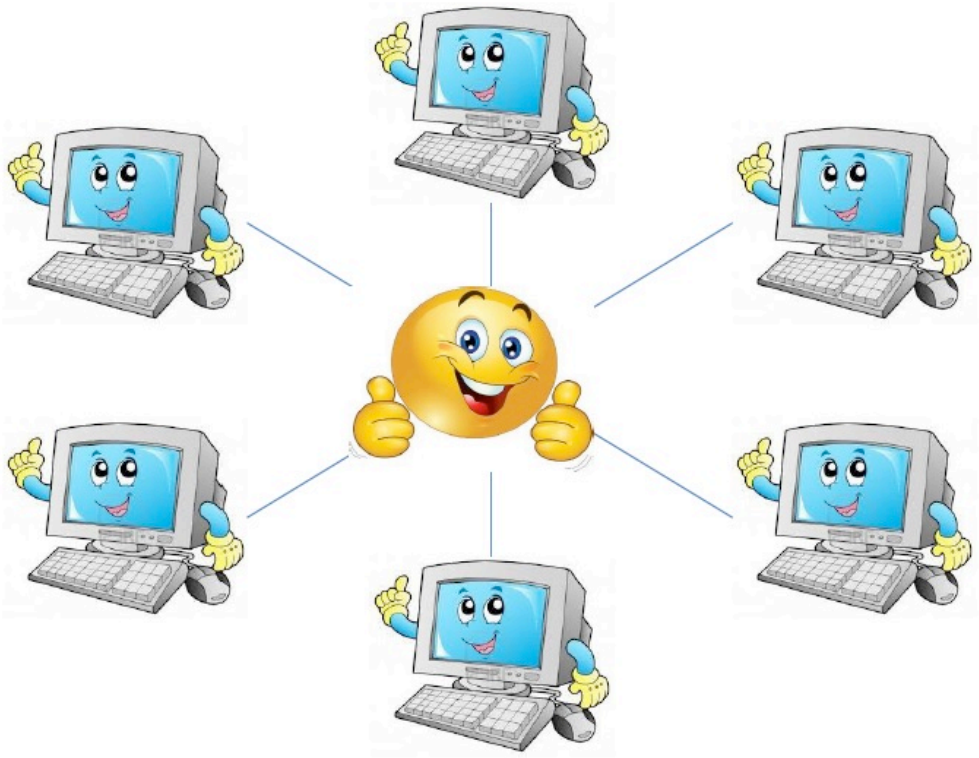
SIMULATION FOR SECURITY PROOF

ILLINOIS INSTITUTE OF TECHNOLOGY

- 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: polynomial simulation for security proof
  - 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

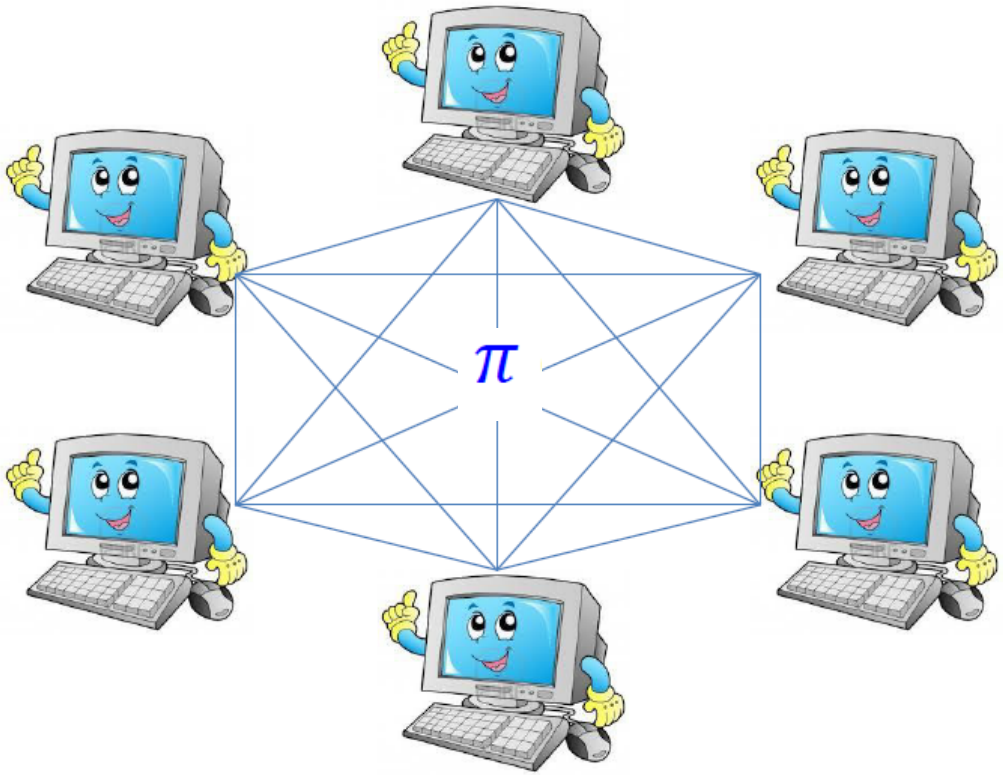
IDEAL WORLD

- Each party sends its input to the trusted party
- The trusted party computes $y=f(x_1, \ldots, x_n)$
- Trusted party sends y to each party

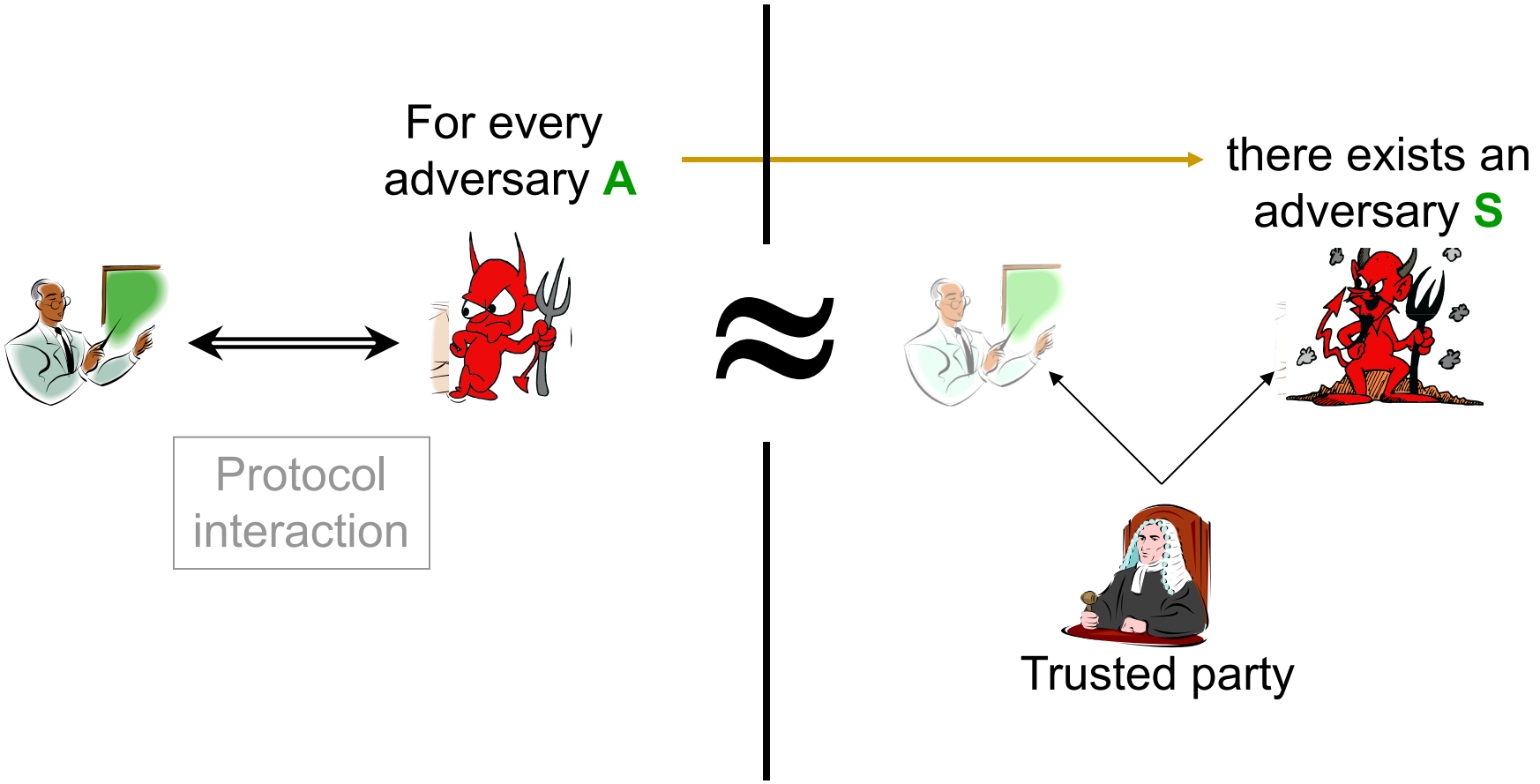


REAL WORLD

- Parties run a protocol π on inputs $(x_1, \ldots, x_n)$



SMC DEFINITION



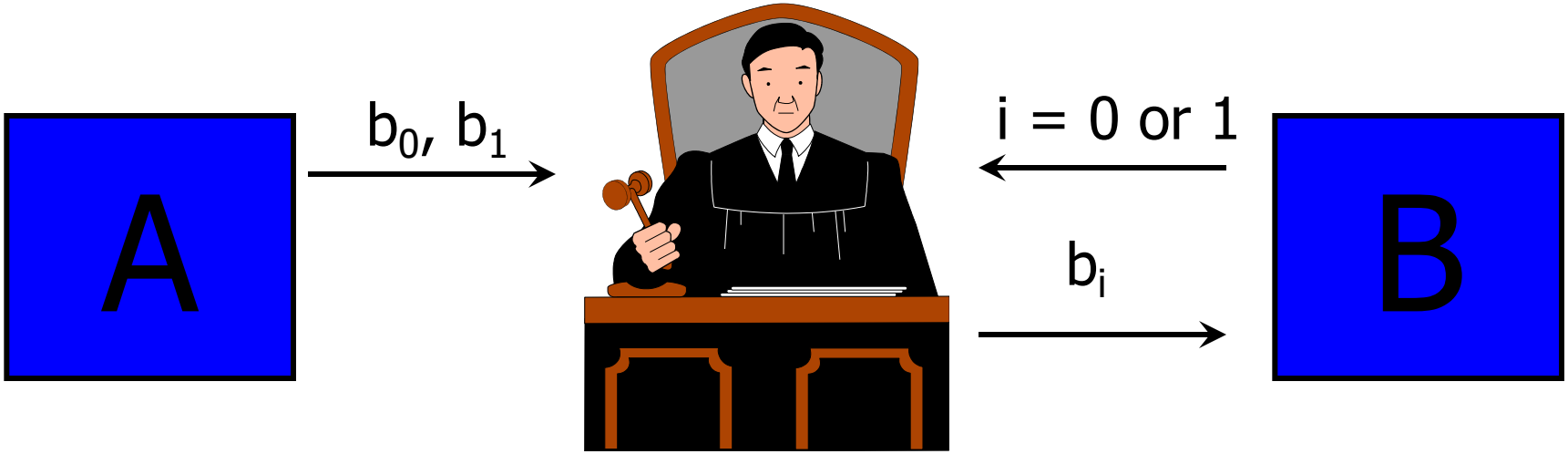
OUTLINE

- Intro to Secure Multiparty Computation
- Security Definition
- Oblivious Transfer
- Exploring Yao's Protocol with Garbled Circuits
- An open source tool

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
- Generalized to bitstrings, M instead of 2, etc.

OT - SEMI-HONEST CONSTRUCTION

1-out-of-2 Oblivious Transfer (OT)

Inputs

- Sender has two messages m_0 and m_1
- Receiver has a single bit $\sigma \in \{0,1\}$

Outputs

- Sender receives nothing
- Receiver obtains m_σ and learns nothing of $m_{1-\sigma}$

SEMI-HONEST OT

ILLINOIS INSTITUTE OF TECHNOLOGY

Let (G,E,D) be a public-key encryption scheme

- G is a key-generation algorithm $(pk, sk) \leftarrow G$
- Encryption: $c = E_{pk}(m)$
- Decryption: $m = D_{sk}(c)$

Assume that a public-key can be sampled without knowledge of its secret key:

- Oblivious key generation: $pk \leftarrow OG$
- El-Gamal encryption has this property

SEMI-HONEST OT

ILLINOIS INSTITUTE OF TECHNOLOGY

Protocol for Oblivious Transfer

Receiver (with input σ):

- Receiver chooses one key-pair (pk,sk) and one public-key pk' (obliviously of secret-key).
- Receiver sets $pk_\sigma = pk$, $pk_{1-\sigma} = pk'$
- Note: receiver can decrypt for pk_σ but not for $pk_{1-\sigma}$
- Receiver sends pk_0, pk_1 to sender

Sender (with input m_0, m_1):

- Sends $c_0 = E_{pk0}(m_0)$, $c_1 = E_{pk1}(m_1)$ to the receiver

Receiver:

- Decrypts c_σ using sk and obtains m_σ.

SECURITY PROOF

Intuition:

- Sender's view consists of only two public keys pk_0 and pk_1. Therefore, it doesn't learn anything about that value of σ.
- The receiver only knows one private key and so can only learn one message

Formally:

- Sender's view is independent of receiver's input and so can easily be simulated (just give it 2 keys)
- Receiver obtains the output m_σ (not included in its view) – indeed, $E_{pk\ \sigma}(m_\sigma)$ can be simulated with m_σ and pk_σ.

Note: this assumes semi-honest behavior. A malicious receiver can choose two keys together with their secret keys.

GENERALIZATION

Can define 1-out-of-k oblivious transfer

Protocol remains the same:

- Choose k-1 public keys for which the secret key is unknown
- Choose 1 public-key and secret-key pair

OUTLINE

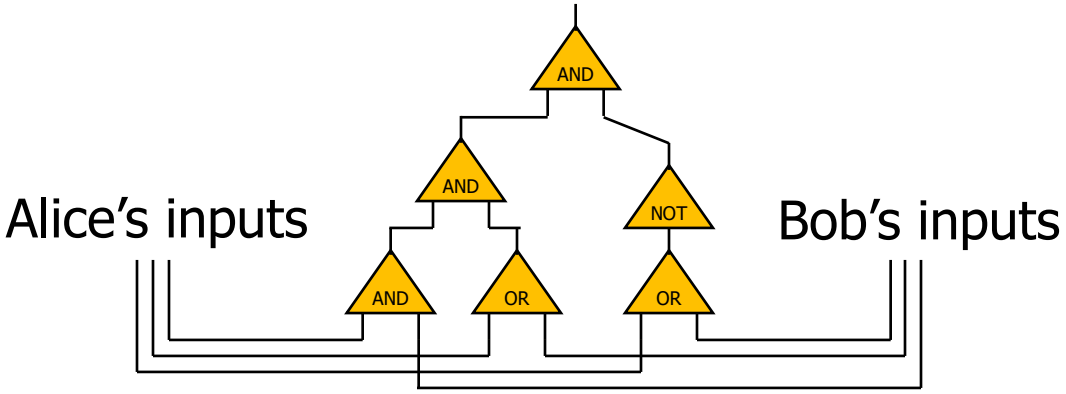
- Intro to Secure Multiparty Computation
- Security Definition
- Oblivious Transfer
- Exploring Yao's Protocol with Garbled Circuits
- An Open Source Tool

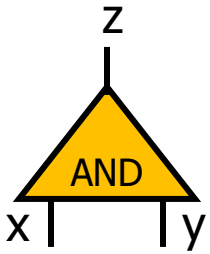
YAO'S PROTOCOL

Compute any function securely

- … in the semi-honest model

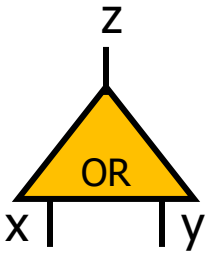
First, convert the function into a Boolean circuit (see an example on Slide 38)





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

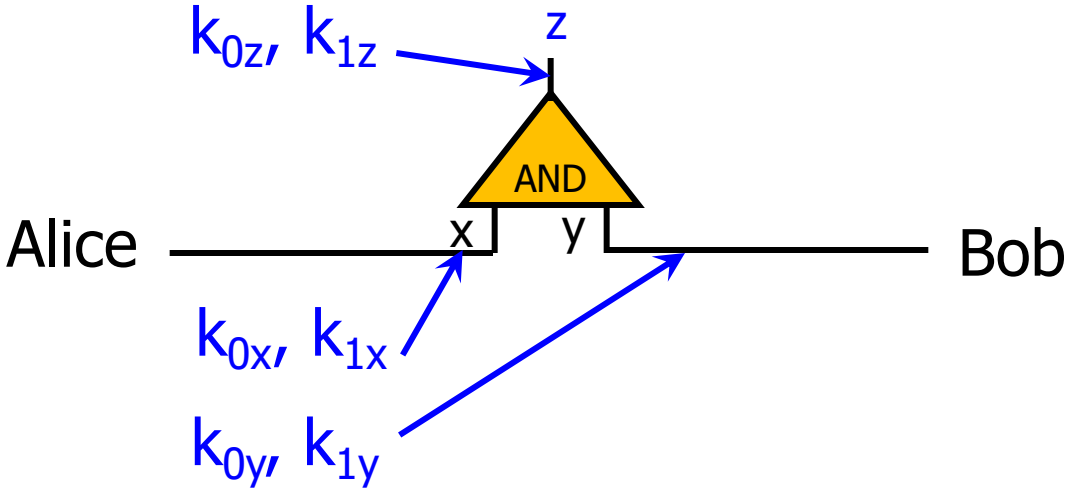
ILLINOIS INSTITUTE OF TECHNOLOGY

Next, evaluate <u>one gate</u> securely

- Later, generalize to the entire circuit

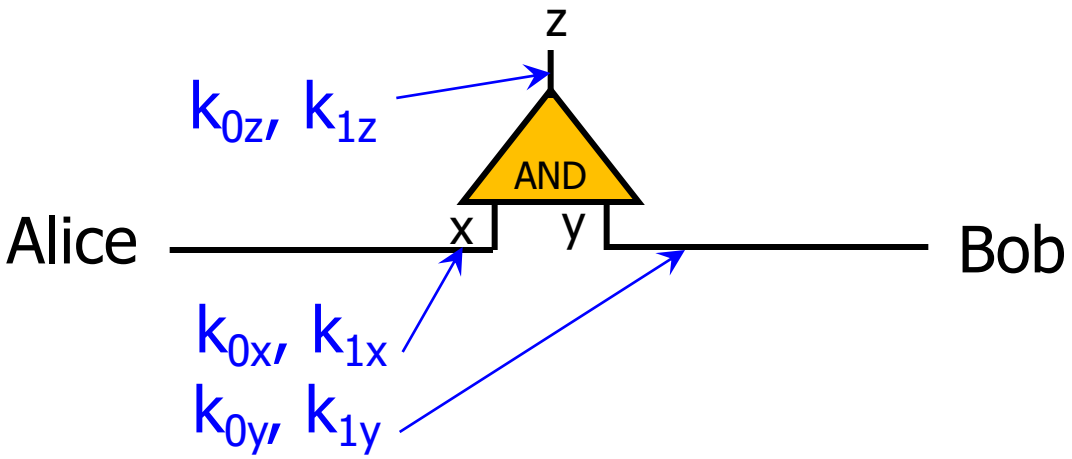
Alice picks two random keys for each wire

- 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 <u>each row of the truth table</u> 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

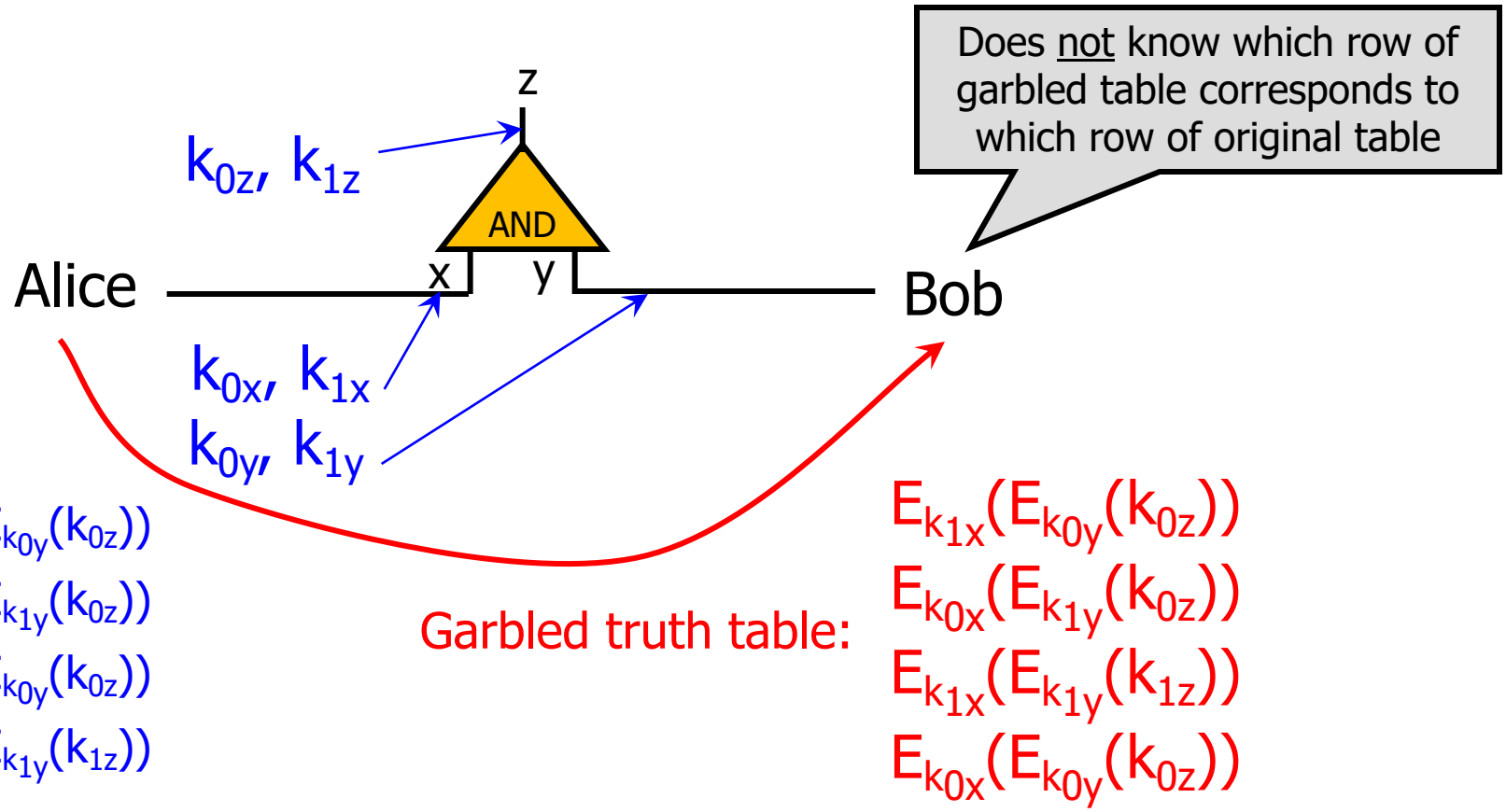
Encrypted truth table:

$E_{k0x}(E_{k0y}(k_{0z}))$
$E_{k0x}(E_{k1y}(k_{0z}))$
$E_{k1x}(E_{k0y}(k_{0z}))$
$E_{k1x}(E_{k1y}(k_{1z}))$

3. SEND GARBLED TRUTH TABLE

ILLINOIS INSTITUTE OF TECHNOLOGY

Alice randomly permutes ("garbles") encrypted truth table and sends it to Bob

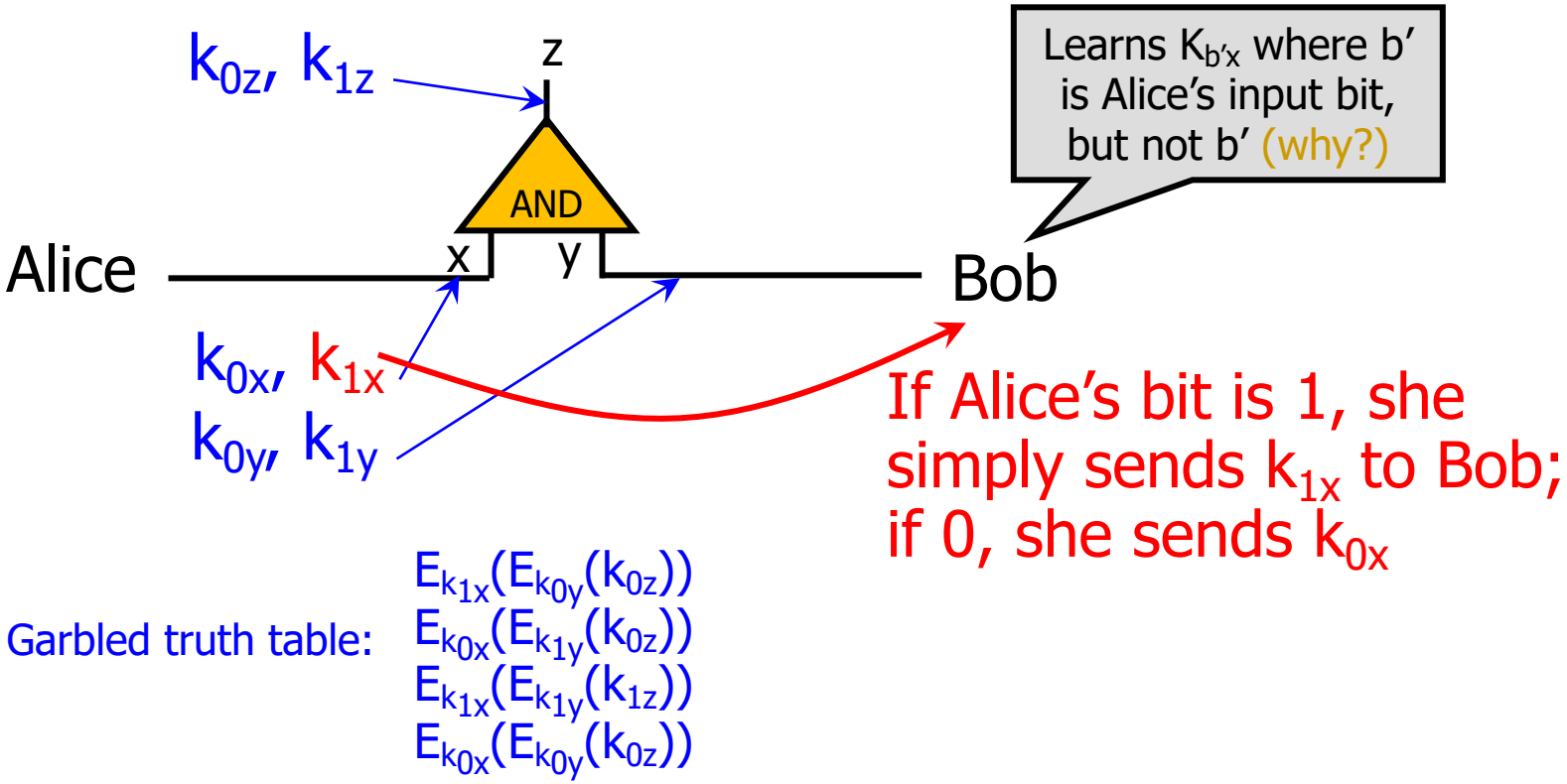


4. SEND KEYS FOR ALICE'S INPUTS

ILLINOIS INSTITUTE OF TECHNOLOGY

Alice sends the key corresponding to her input bit

- Keys are random, so Bob does not learn what this bit is

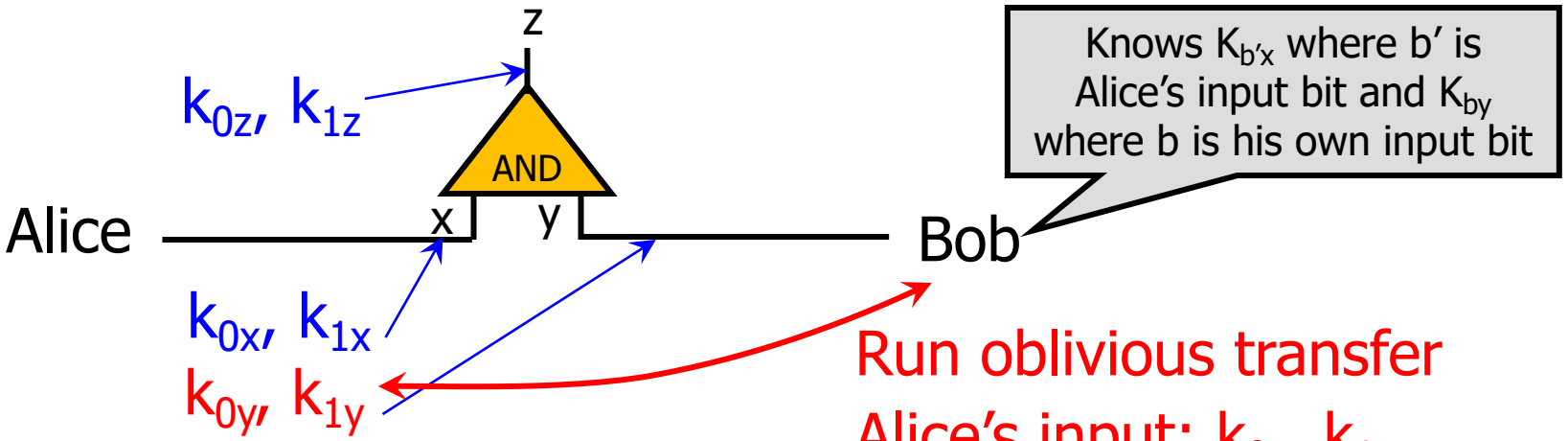


5. USE OT ON KEYS FOR BOB'S INPUT

ILLINOIS INSTITUTE OF TECHNOLOGY

Alice and Bob run oblivious transfer (OT) 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



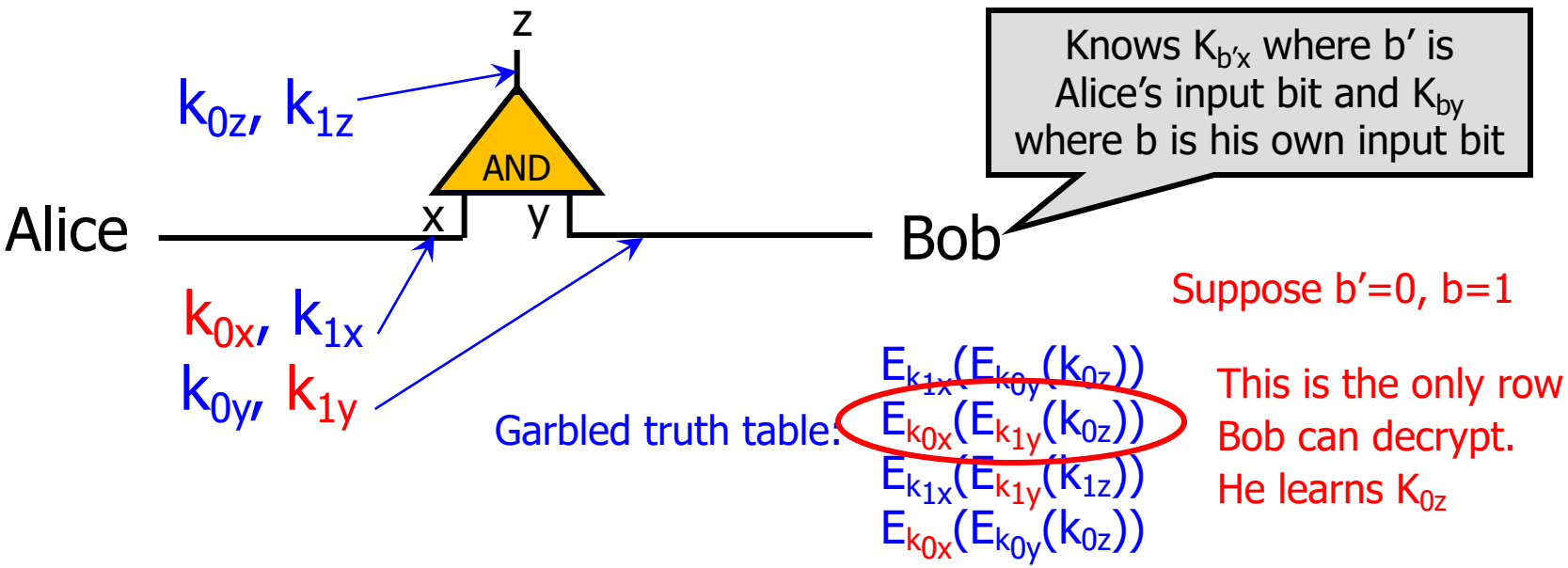
Garbled truth table:

$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$
$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$
$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$

6. EVALUATE GARBLED GATE

Using the two keys that he learned, Bob decrypts <u>exactly one of the output-wire keys</u>

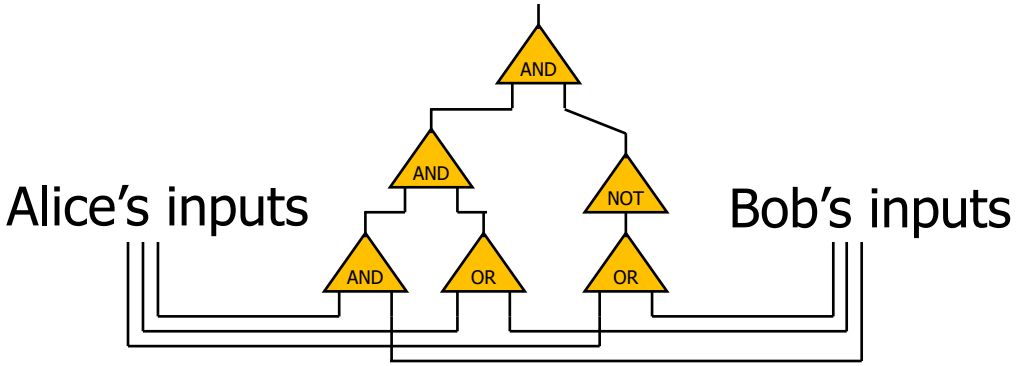
- Bob does not learn if this key corresponds to 0 or 1
  - Why is this important?



7. EVALUATE ENTIRE CIRCUIT

In this way, Bob evaluates entire garbled circuit (SFE- Secure Function Evaluation)

- 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

ILLINOIS INSTITUTE OF TECHNOLOGY

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!

EXAMPLE CIRCUIT FOR SECURE COMPARISON

ILLINOIS INSTITUTE OF TECHNOLOGY

- **X > Y (?) for <u>two-bit inputs</u> X=x1x0, Y= y1y0**

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

(x1 AND (NOT y1)) OR (NOT (x1 XOR y1)) AND (x0 AND (NOT y0))

- **Can convert any "program" into a (reasonably "small") circuit**
  - Size of circuit: number of wires (as a function of number of input wires)
- **Can convert a truth-table into a circuit**
  - Directly, with size of circuit exponentially large
  - In general, finding a small/smallest circuit from truth-table is notoriously hard

AGAINST ACTIVE ADVERSARIES

ILLINOIS INSTITUTE OF TECHNOLOGY

- **In Secure Function Evaluation (SFE), Alice builds and encrypts the circuit, and Bob evaluates the encrypted circuit**

- **What can an active adversary accomplish?**
  - Alice: encrypt a different circuit
  - Bob: report a different output

- **Cut-and-choose**
  - Alice prepares m circuits
  - Bob picks one to execute
  - Alice reveals secrets for all others

- **Multiple circuits**
  - Bob evaluates k out of m circuits, verifies the others
  - Note: must ensure Bob's inputs for all circuits are the same

OUTLINE

- Intro to Secure Multiparty Computation
- Security Definition
- Oblivious Transfer
- Exploring Yao's Protocol with Garbled Circuits
- An Open Source Tool

SMC/MPC HISTORY

- 1982 Yao's Garbled Circuits
- 2004 Fairplay
- 2016 FairplayMP, Obliv-C, ObliVM, FastGC, TASTY, SPDZ, EMP, TinyOT, ShareMind, PCF, Sharemonad, Fresco, Wysteria, …

  Plus, many schemes that have never been implemented!

FAIRPLAY

https://www.cs.huji.ac.il/project/Fairplay/home.html

- **Implementation of SFE**
- **Function specified as programs**
- **Compiler converts it to circuits**

```
program Millionaires {
   type int = Int<4>; // 4-bit
   integer
   type AliceInput = int;
   type BobInput = int;
   type AliceOutput = Boolean; type
   BobOutput = Boolean;
   type Output = struct { AliceOutput
   alice, BobOutput bob};
   type Input = struct { AliceInput
   alice, BobInput bob};

   function Output out(Input inp)
   { out.alice = inp.alice > inp.bob;
   out.bob = inp.bob > inp.alice; }

}
```

RUN FAIRPLAY

ILLINOIS INSTITUTE OF TECHNOLOGY

A. Examples for compiling an SFDL program:

---

1. run_bob -c progs/Billionaires.txt
2. run_alice -c progs/Billionaires.txt

(-c for compile, progs/Billionaires.txt - SFDL program to compile)

Both commands produce the same two output files:

1. progs/Billionaires.txt.Opt.circuit (an SHDL circuit)
2. progs/Billionaires.txt.Opt.fmt

B. Example for running Bob (should be first):

---

run_bob -r progs/Billionaires.txt S&b~n2#m8_Q 4

(-r for run,
progs/Billionaires.txt - program to run,
3rd parameter - crazy string for random seed,
4th parameter - OT type [1-4], 4 is the best one)

C. Example for running Alice (should be second):

---

run_alice -r progs/Billionaires.txt 5miQ^0s1 humus.cs.huji.ac.il

(-r for run,
progs/Billionaires.txt - program to run,
3rd parameter - crazy string for random seed,
4th parameter - hostname where Bob is)

D. Some general comments:

---

1. As with regular programs - first compile, then run.
2. Bob & Alice use fixed port (no. 3496) for TCP/IP communication.

SAMPLE SFDL PROGRAMS

Billionaires

```
/* Check which of two Billionaires is richer */

program Billionaires {
        type int = Int<32>; // 32-bit integer
        type AliceInput = int;
        type BobInput = int;
        type AliceOutput = Boolean;
        type BobOutput = Boolean;
        type Output = struct {AliceOutput alice, BobOutput bob};
        type Input = struct {AliceInput alice, BobInput bob};

        function Output output(Input input) {
                output.alice = (input.alice > input.bob);
                output.bob = (input.bob > input.alice);
}

}
```

SAMPLE SFDL PROGRAMS

AND

```
/* Compute AND of two bytes */

program And {
        const N=8;
        type Byte = Int<N>;
        type AliceInput = Byte;
        type BobInput = Byte;
        type AliceOutput = Byte;
        type BobOutput = Byte;
        type Input = struct {AliceInput alice, BobInput bob};
        type Output = struct {AliceOutput alice, BobOutput bob};

        function Output output(Input input) {
        output.alice = (input.bob & input.alice);
        output.bob = (input.bob & input.alice);
}

}
```

SAMPLE SFDL PROGRAMS

ILLINOIS INSTITUTE OF TECHNOLOGY

Median

```
/* Compute the Median of two sorted arrays */

program Median {
// Constants
const inp_size = 10;

// Type Definitions
type Elem = Int<16>;
type AliceInput = Elem[inp_size];
type AliceOutput = Int<16>;
type BobInput = Elem[inp_size];
type BobOutput = Int<16>;
type Input = struct {AliceInput alice, BobInput bob};
type Output = struct {AliceOutput alice, BobOutput bob};

// Function Definitions
// This is the main function
function Output output(Input input) {
var Int<8> i;
var Int<8> ai;
var Int<8> bi;
ai=0; bi=0;

for (i = 1 to inp_size-1) {
                    if (input.alice[ai] >= input.bob[bi]) bi = bi + 1;
                    else ai = ai + 1;
}
if (input.alice[ai] < input.bob[bi])
                    { output.alice = input.alice[ai];
                      output.bob = input.alice[ai]; }
else { output.alice = input.bob[bi];
          output.bob = input.bob[bi]; }
}
}
```

FAIRPLAY PERFORMANCE

https://www.cs.huji.ac.il/project/Fairplay/home.html

Function	Gates	OTs
AND	32	8
Billionaires	254	32
KDS	1229	6
Median	4383	160

Function	LAN	WAN
AND	0.41	2.57
Billionaires	1.25	4.01
KDS	0.49	3.38
Median	7.09	16.63

ACKNOWLEDGMENTS

Note: Some of the slides in this lecture are based on material created by

- Dr. Nikita Borisov at UIUC
- Dr. Vitaly Shmatikov at Cornell University
- Dr. Jaideep Vaidya at Rutgers University
- Dr. Li Xiong at Emory University
- Fairplay SFE Team