MIT 6.875 & Berkeley CS276

GMW – Secure multi-party computation

Lecture 25

Credit: Some slides adapted from Yehuda Lindell

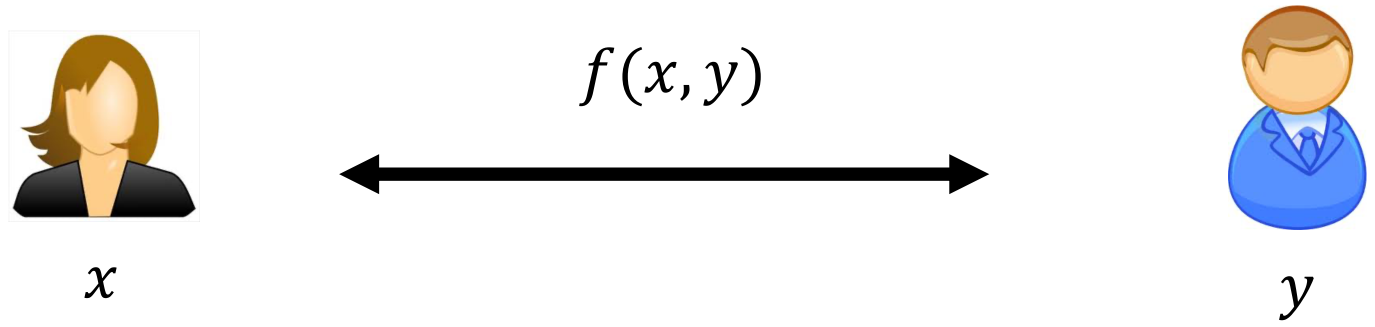
In this lecture

Recording…

Last time:
- Secure 2-party computation via Yao garbled circuits

This time the GMW protocol:
- Secure 2-party computation via secret sharing
- Extension to secure n-parth computation
- Extension to malicious security

Secure 2-party computation



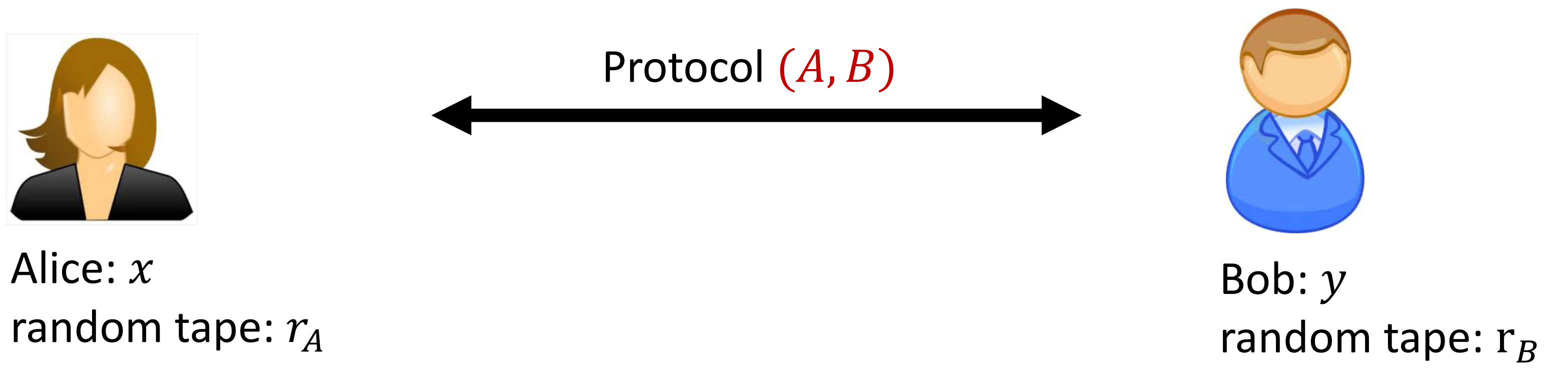
Alice and Bob compute $f(x, y)$ without sharing their private inputs with each other

Recall: adversarial models

- **Semi-honest/honest-but-curious:** Each party follows the protocol, but tries to learn additional information from the transcript

- **Malicious:** Parties can behave arbitrarily, even deviate from the protocol in order to learn additional information

Security in the semi-honest model



Definition: An efficient protocol $\langle A,B\rangle$ securely computes a deterministic function $f=(f_1,f_2)$ in the semi-honest model if there exist PPT simulators S_A and S_B such that for every $\{x,y\}\in\{0,1\}^*$, the following hold:

Correctness:

$$\Pr[out_A[\langle A(x),B(y)\rangle(1^n)],out_B[\langle A(x),B(y)\rangle(1^n)]=f(x,y)]=1$$

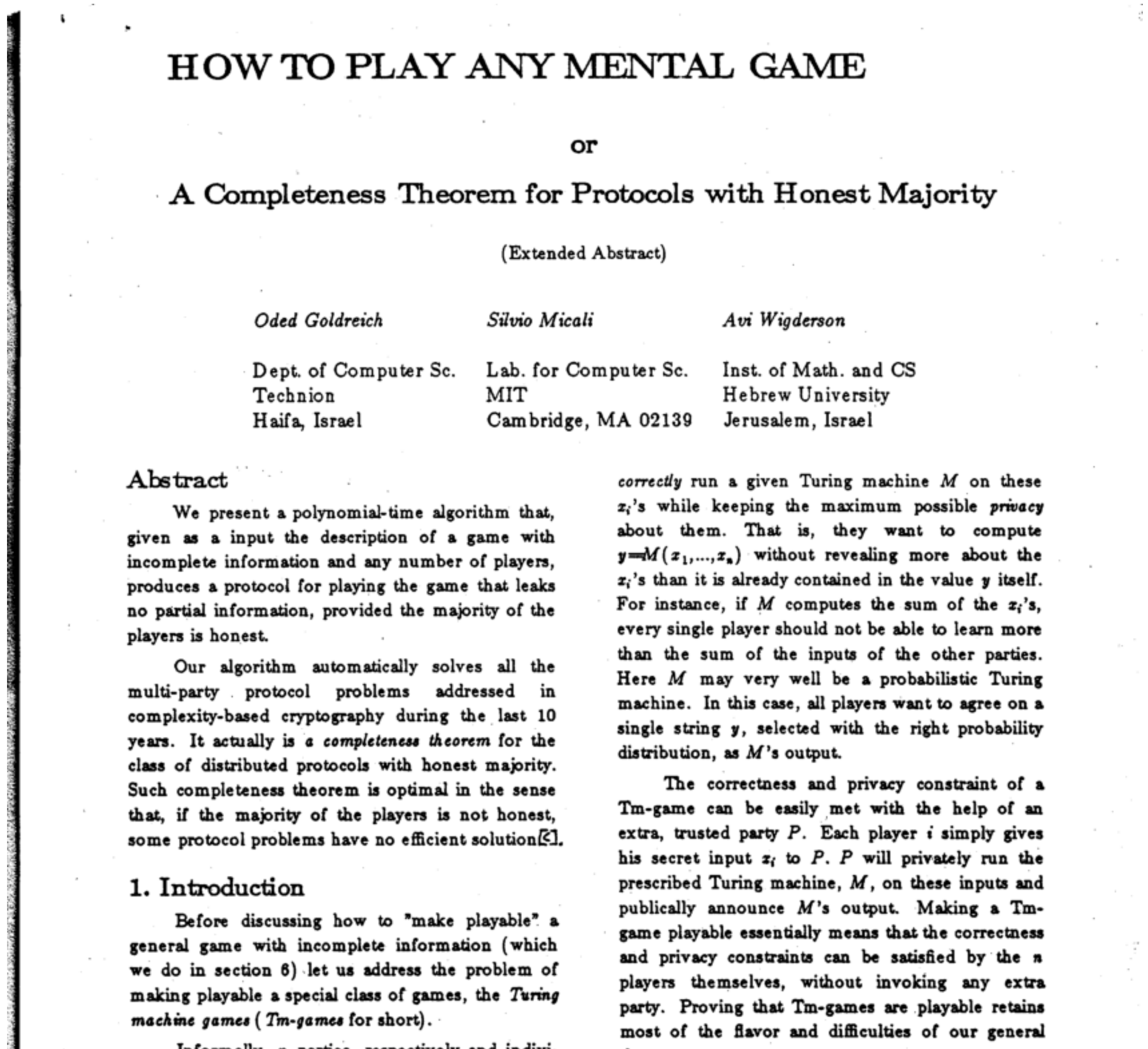
Security against semi-honest Alice:

$$S_A(x,f_1(x,y))\approx_c view_A(\langle A(x),B(y)\rangle)$$

Security against semi-honest Bob:

Symmetric

Goldreich-Micali-Wigderson (GMW) 1987



HOW TO PLAY ANY MENTAL GAME

or

A Completeness Theorem for Protocols with Honest Majority

(Extended Abstract)

Oded Goldreich — Dept. of Computer Sc., Technion, Haifa, Israel

Silvio Micali — Lab. for Computer Sc., MIT, Cambridge, MA 02139

Avi Wigderson — Inst. of Math. and CS, Hebrew University, Jerusalem, Israel

Abstract

We present a polynomial-time algorithm that, given as a input the description of a game with incomplete information and any number of players, produces a protocol for playing the game that leaks no partial information, provided the majority of the players is honest.

Our algorithm automatically solves all the multi-party protocol problems addressed in complexity-based cryptography during the last 10 years. It actually is *a completeness theorem* for the class of distributed protocols with honest majority. Such completeness theorem is optimal in the sense that, if the majority of the players is not honest, some protocol problems have no efficient solution[C].

1. Introduction

Before discussing how to "make playable" a general game with incomplete information (which we do in section 6) let us address the problem of making playable a special class of games, the *Turing machine games* (*Tm-games* for short).

Informally, n parties, respectively and indivi-

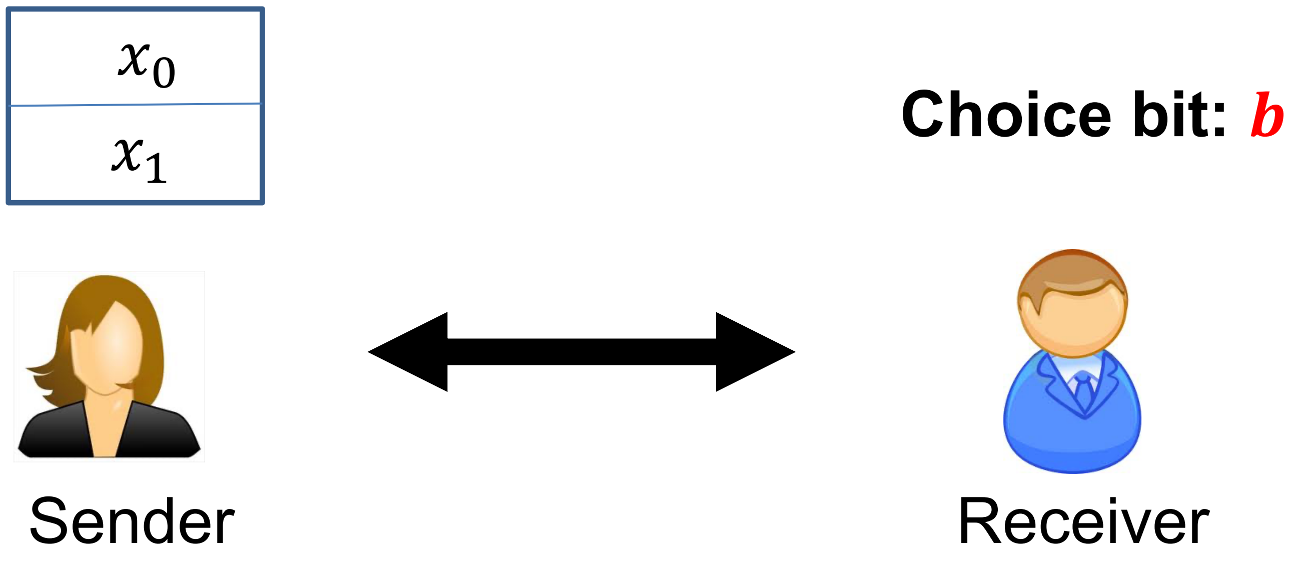
correctly run a given Turing machine M on these x_i's while keeping the maximum possible *privacy* about them. That is, they want to compute $y=M(x_1,...,x_n)$ without revealing more about the x_i's than it is already contained in the value y itself. For instance, if M computes the sum of the x_i's, every single player should not be able to learn more than the sum of the inputs of the other parties. Here M may very well be a probabilistic Turing machine. In this case, all players want to agree on a single string y, selected with the right probability distribution, as M's output.

The correctness and privacy constraint of a Tm-game can be easily met with the help of an extra, trusted party P. Each player i simply gives his secret input x_i to P. P will privately run the prescribed Turing machine, M, on these inputs and publically announce M's output. Making a Tm-game playable essentially means that the correctness and privacy constraints can be satisfied by the n players themselves, without invoking any extra party. Proving that Tm-games are playable retains most of the flavor and difficulties of our general

GMW

- Construct a protocol for the semi-honest model
  - Based on OT
- “Compile it” to obtain a protocol that is secure for the malicious model
  - Compilation involves forcing the parties to follow the protocol

Recall: Oblivious Transfer (OT)



- Sender holds two bits x_0 and x_1.
- Receiver holds a choice bit b.
- Receiver should learn x_b, sender should learn nothing.

Generalization

- Can define 1-out-of-k oblivious transfer

- We will use 1-out-of-4 OT for 2-party GMW

1-out-of-2 OT

Input bits: (x_0, x_1)

Choice bit: b

Encrypt choice bit b

$c \longleftarrow \mathrm{Enc}(sk, b)$

$$\xleftarrow{\quad c \quad}$$

Homomorphically evaluate the selection function

$$\boldsymbol{SEL_{x_0,x_1}(b) = (x_1 \oplus x_0)b + x_0}$$

$$c' = \mathrm{Eval}(SEL_{x_0,x_1}(b), c)$$

$$\xrightarrow{\qquad\qquad}$$

Decrypt to get x_b

Bob's security: computational, from CPA-security of Enc.

Alice's security: statistical, from circuit-privacy of Eval.

1-out-of-4 OT

Input bits:
(x_0, x_1, x_2, x_3)

Choice bit: b_0, b_1

Encrypt choice bit b

$c \longleftarrow \mathrm{Enc}(sk, b)$

c

$\longleftarrow$

Homomorphically evaluate the selection function

$\boldsymbol{SEL_{x_0,x_1}(b)} =$
$\boldsymbol{(x_1 + x_0)b + x_0}$ mod 2

$c' = \mathrm{Eval}(SEL_{x_0,x_1}(b), c)$

$\longrightarrow$

Decrypt to get x_b

Ideas how to adapt this protocol to 1-out of-4?

1-out-of-4 OT

Input bits:
(x_0, x_1, x_2, x_3)

Choice bit: b_0, b_1

Encrypt choice bit b

$c \longleftarrow \mathrm{Enc}(sk, b_0, b_1)$

$$\xleftarrow{\quad c \quad}$$

Homomorphically evaluate the selection function

$SEL_{SEL_{x_0,x_1}(b_1), SEL_{x_2,x_3}(\mathrm{b}_1)}(b_0)$

$$\xrightarrow{\quad c' = \mathrm{Eval}(\ldots) \quad}$$

Decrypt to get $x_{b_0 b_1}$

2-party semi-honest GMW

- Let f be the function that the parties wish to compute
- Represent f as an arithmetic circuit with addition and multiplication gates (over GF[2]).
  - $x + y \, mod \, 2 = x \oplus y$
  - $x \ast y \, mod \, 2 = x$ AND y
  - Will only write $x + y$ and $x \ast y$ for simplicity

Random Shares Paradigm

- Compute gate-by-gate, revealing only random shares each time

- Let a be a private value:
  - Alice holds a random value a_1
  - Bob holds $a+a_1$
  - Note that without knowing a_1, $a+a_1$ is just a random value revealing nothing of a.
  - We say that the parties hold random shares of a.
- The computation will be such that all intermediate values are random shares (and so they reveal nothing).

Circuit Computation

- Stage 1: each party randomly shares its input with the other party
- Stage 2: compute gates of circuit as follows
  - Given random shares to the input wires, compute random shares of the output wires
- Stage 3: combine shares of the output wires in order to obtain actual output

Stage 1: randomly share input

input a | input b

choose random a_1 $\xrightarrow{a_2 = a + a_1}$ a_2

b_1 $\xleftarrow{b_1 = b + b_2}$ Choose random b_2

Stage 3: combine output from shares

At the end of Stage 2, they will each have shares of the output

input a

input b

$$\begin{tikzcd}
o_1 \arrow[r, "o_1"] & o_2 + = o \\
+ = o & \arrow[l, "o_2"']
\end{tikzcd}$$

Stage 2

Compute each gate at a time: given random shares of the input wires, compute random shares of the output wires

- Addition
- Multiplication

Addition

Parties should end up having shares of $a + b$

input a

a_1, b_1

input b

a_2, b_2

Ideas?

Addition

Parties should end up having shares of $a + b$

input a input b

a_1, b_1 a_2, b_2

$a_1 + b_1 = o_1$ simple local addition $a_2 + b_2 = o_2$

These are shares of $a + b$ because $o_1 + o_2 = a_1 + b_1 + a_2 + b_2 = a + b$

Invert a

Parties should end up having shares of $1 + a$

input a

a_1, b_1

input b

a_2, b_2

Ideas?

Invert a

Parties should end up having shares of $1 + a$

input a

a_1, b_1

$o_1 = 1 + a_1$

invert at one party

input b

a_2, b_2

$o_2 = a_2$

Multiplication

Parties should end up having shares of $a * b = (a_1 + a_2)(b_1 + b_2)$

input a

a_1, b_1

input b

a_2, b_2

Ideas?
Hint: OT

Multiplication

Parties should end up having shares of $a * b = (a_1 + a_2)(b_1 + b_2)$

input a

a_1, b_1

input b

a_2, b_2

Alice does not know Bob's shares, but there are only 4 possibilities: 00,01,10,11

Multiplication (cont’d)

- Alice prepares a table as follows:
  - Row 1 corresponds to Bob’s input 00
  - Row 2 corresponds to Bob’s input 01
  - Row 3 corresponds to Bob’s s input 10
  - Row 4 corresponds to Bob’s input 11

- Let r be a random bit chosen by Alice:
  - Row 1 contains the value $a \cdot b + r$ when $a_2=0, b_2=0$
  - Row 2 contains the value $a \cdot b + r$ when $a_2=0, b_2=1$
  - Row 3 contains the value $a \cdot b + r$ when $a_2=1, b_2=0$
  - Row 4 contains the value $a \cdot b + r$ when $a_2=1, b_2=1$

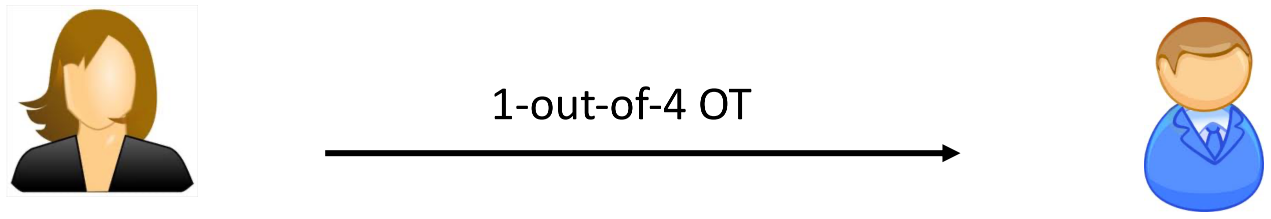
Concrete Example

- Assume: $a_1=0$, $b_1=1$
- Assume: $r=1$

Row	Bob's shares	Output value
1	$a_2=0, b_2=0$	$(0+0)\cdot(1+0)+1=1$
2	$a_2=0, b_2=1$	$(0+0)\cdot(1+1)+1=1$
3	$a_2=1, b_2=0$	$(0+1)\cdot(1+0)+1=0$
4	$a_2=1, b_2=1$	$(0+1)\cdot(1+1)+1=1$

Multiplication

Parties should end up having shares of $a * b = (a_1 + a_2)(b_1 + b_2)$



input a | input b

a_1, b_1 | a_2, b_2

- The parties run a 1-out-of-4 oblivious transfer protocol
- Alice plays the sender.
- Bob selects the row corresponding to a_2, b_2 so it obtains $ab\ + r$
- Alice has r, so they have a secret sharing of $a * b$ as desired

Computing a multi-gate circuit

- The shares outputting a gate are used to compute another gate
- Computation proceeds in this manner until the end
- Combine the shares of the parties only at the end

Semi-honest security

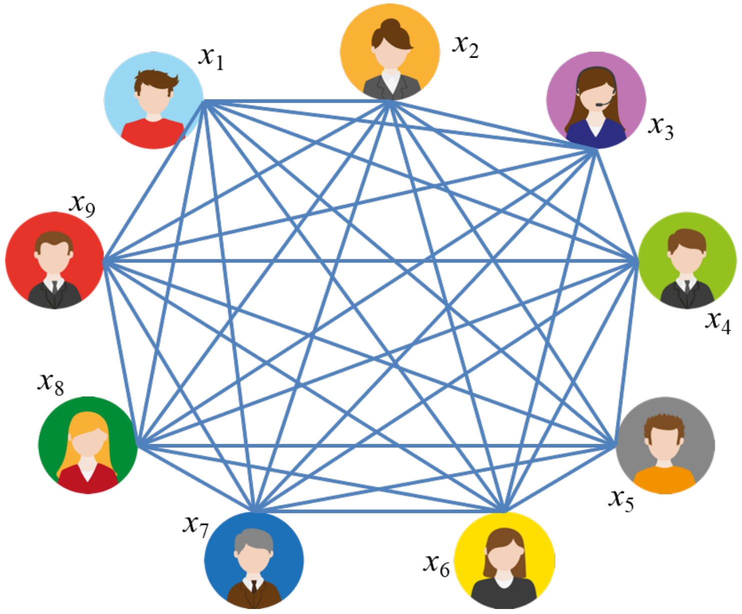
Theorem: assuming OT, any efficient functionality f can be securely computed in the semi-honest model

- XOR (+ mod 2) and AND (* mod 2) are universal
- Reduction to the oblivious transfer protocol
- Assuming security of the OT protocol, parties only see random values until the end. Therefore, simulation is straightforward.

Properties of GMW

- Interactive protocol
- How many rounds of communication?
  - O(depth) for depth of circuit because parallel gates can be computed in parallel

Can be generalized to multi-party computation

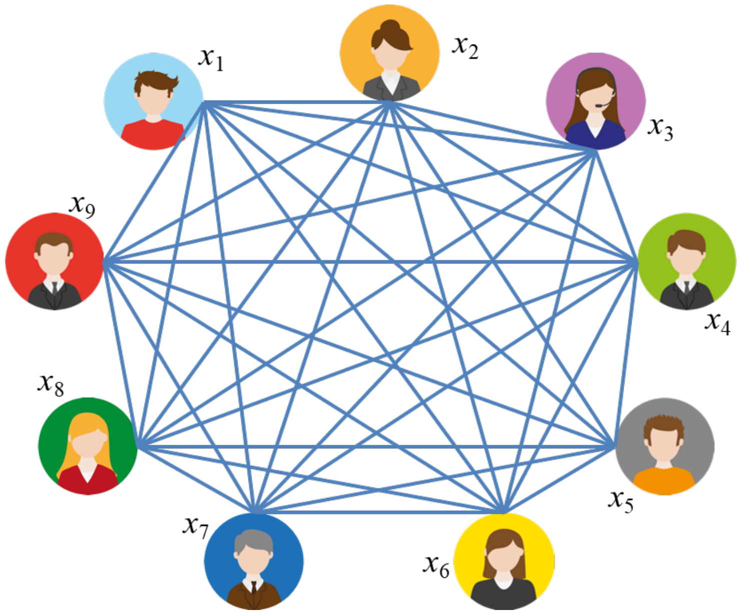


n parties

- Each user's private input is secret shared in n shares
- Any gate operates on two wires, so let us consider private values a and b of which everyone has a share $(a_1, \dots, a_n)$ and $(b_1, \dots, b_n)$
- Generalizes to n-party easier than garbled circuits

How would you compute $a + b$?

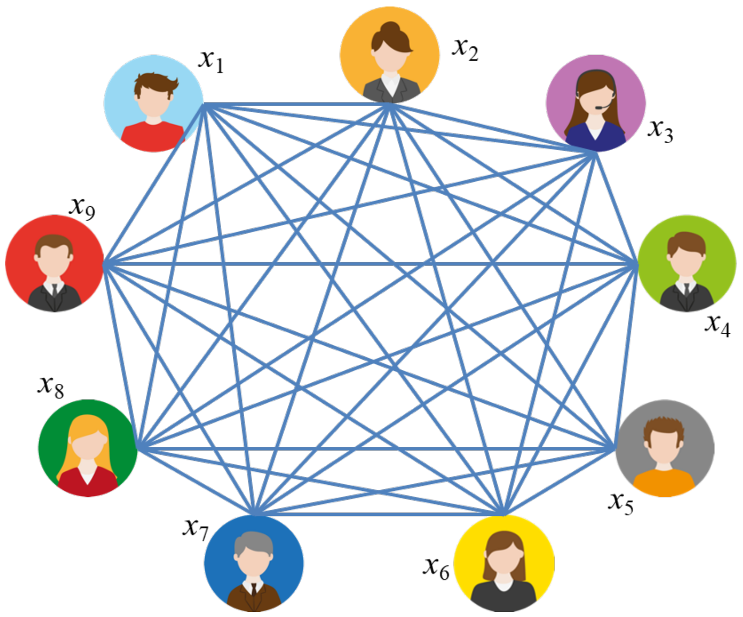
Can be generalized to multi-party computation



n parties

Addition is the same: add local shares $a_i \oplus b_i$

Can be generalized to multi-party computation



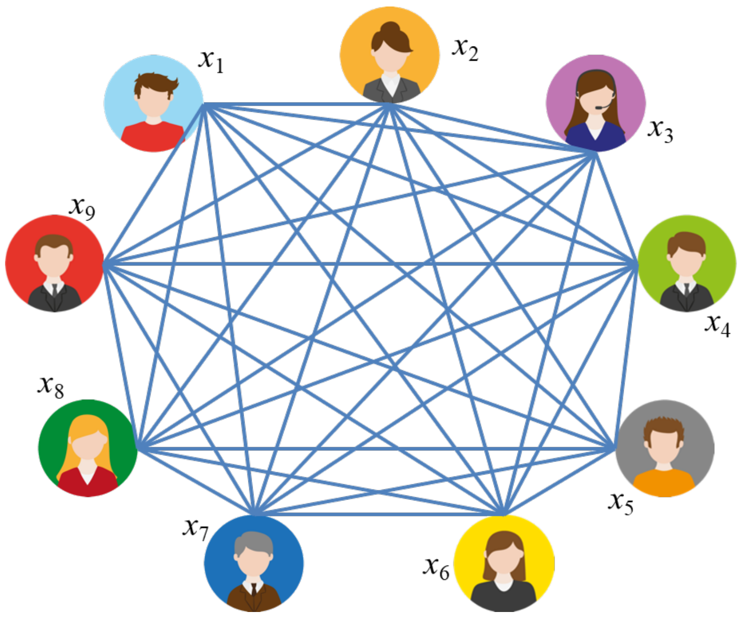
n parties

Multiplication: want $a * b = (a_1 + \cdots + a_n) * (b_1 + \cdots + b_n)$

(all bits and mod 2)

Ideas?

Can be generalized to multi-party computation



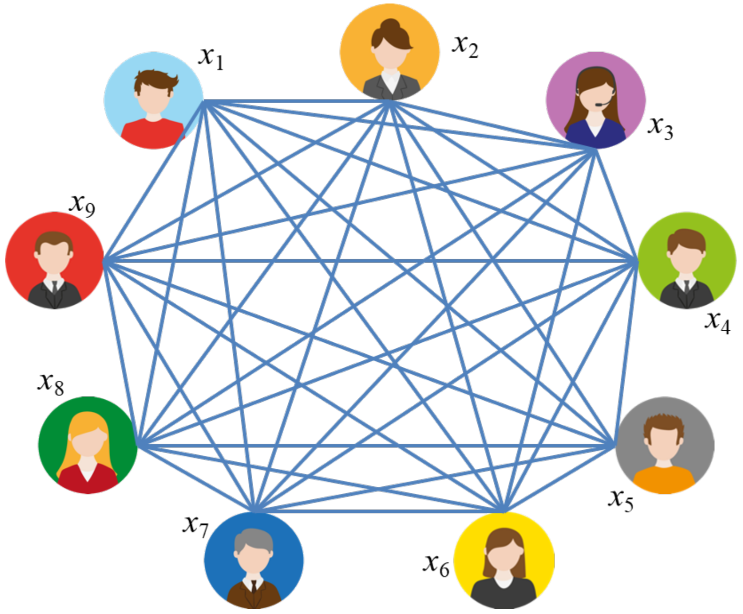
n parties

Multiplication: want $a * b = (a_1 + \cdots + a_n) * (b_1 + \cdots + b_n)$

$$= \sum_i a_i b_i + \sum_{i \neq j} a_i b_j$$

How to compute each term?

Can be generalized to multi-party computation



n parties

Multiplication: want $a * b = (a_1 + \cdots + a_n) * (b_1 + \cdots + b_n)$

$$= \sum_i a_i b_i + \sum_{i \neq j} a_i b_j$$

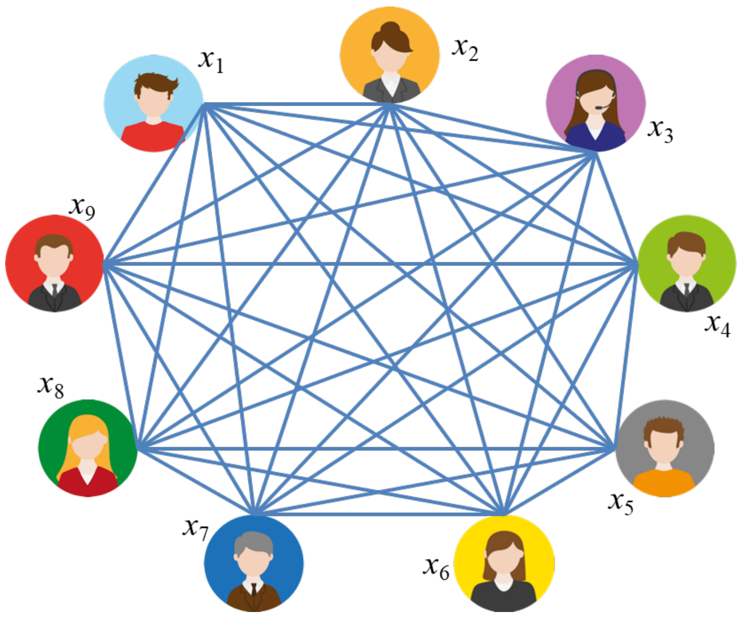
To compute shares of $a_i b_i$: party i multiplies its shares

To compute shares of $a_i b_j$: parties j and i use 2-party GMW with secrets a_i and b_j

Each party i adds up all these shares to obtain $(a * b)_i$

Intuition for malicious security

What can a malicious party do?



Corrupt the shares (which corrupts the final result, e.g. easy to flip the result)
Corrupt multiple parties hoping to reconstruct private inputs
Choose bad randomness
Cheat during OT

Maliciously-secure MPC

- Malicious security is significantly more complex both definitionally and construction-wise than semi-honest security
- We will now study malicious security only informally
- Let’s start with a security definition…

The real/ideal model paradigm

- An attacker corrupted some parties
- Ideal model: parties send inputs to a trusted party, who computes the function and sends the outputs
- Real model: parties run a real protocol with no trusted help

Informally: a protocol is secure if an attacker receives no more information about the private inputs in the real protocol than in the ideal model

- Since essentially **no** attacks can be carried out in the ideal model, security is implied

The Security Definition

- A protocol Π securely computes a function **f** if:
  - For every real-model adversary **A**, there exists an ideal-model simulator **S**, such that for every set of inputs
  - the result of a real execution of Π with **A** is computationally indistinguishable from the result of an ideal execution with **S** (where the trusted party computes **f**).
- The result of an execution is defined by the output vector of the honest parties and adversary

Meaning of the Definition

- Interpretation 1:
  - Security in the ideal model is absolute. Since no attacks are possible in the ideal model, we obtain that the same is also true of the real model.
- Interpretation 2:
  - Anything that an adversary could have learned/done in the real model, it could have also learned/done in the ideal model.

Properties of the Definition

Privacy:

- The ideal-model adversary cannot learn more about the honest party's input than what is revealed by the function output.
- Thus, the same is true of the real-model adversary.
- Otherwise, the REAL and IDEAL could be easily distinguished.

Correctness:

- In the ideal model, the function is always computed correctly.
- Thus, the same is true in the real-model.
- Otherwise, the REAL and IDEAL could be easily distinguished

The GMW Paradigm

- Construct a protocol for the semi-honest model
- "Compile it" to obtain a protocol that is secure for the malicious model
  - Compilation involves forcing the parties to follow the protocol
- It may be more efficient to work differently

Three steps for malicious security

1. Force the adversary to use a fixed input
2. Force the adversary to use a uniform random tape

- Each party commits to its input and provides commitments to all parties
- Parties run a (maliciously-secure) coin tossing protocol at the end of which each party has a random tape and the other parties have commitments to this party's tape

3. Force the adversary to follow the protocol exactly (consistently with their fixed input and random tape)

- Run the protocol as with semi-honest GMW but each party adds a ZK proof that it performed the right computation against the secret inputs and randomness in the commitments

Summary

- GMW provides a secure multi-party protocol via OT and secret sharing
- Malicious security is achieved by compiling a semi-honest protocol using commitments, coin tossing and ZK proofs
- This was last lecture for Berkeley, next is optional for Berkeley