数据隐私方法伦理和实践

Methodology, Ethics and Practice of Data Privacy

秘密共享, 安全多方计算

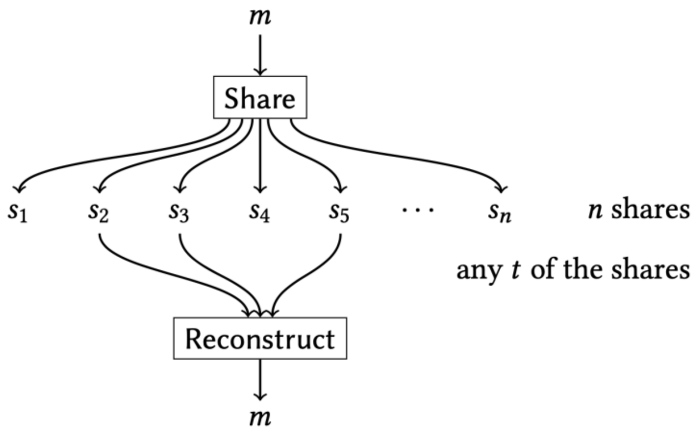
Secret Sharing, MPC

张兰
中国科学技术大学 计算机学院
2020春季

1. Secret Sharing

Threshold Scheme

- A (t,w)-threshold scheme
  - Sharing key K among a set of *w* users
  - Any *t* users can recover the key
  - Any *t-1* users can not do so



“

The security definition of secret sharing is that the shares must leak absolutely no information about the secret, until the number of shares passes the threshold value.

A Simple 2-out-of-2 Scheme

Construction 3.5(2-out-of-2 TSSS)

$\mathcal{M} = \{0,1\}^\ell$ $t = 2$ $n = 2$	$\underline{\text{Share}(m):}$ $s_1 \leftarrow \{0,1\}^\ell$ $s_2 := s_1 \oplus m$ return (s_1, s_2)	$\underline{\text{Reconstruct}(s_1, s_2):}$ return $s_1 \oplus s_2$

Example If we want to share the string $m = 1101010001$ then the Share algorithm might choose

$$s_1 := 0110000011$$

$$s_2 := s_1 \oplus m$$

$$= 0110000011 \oplus 1101010001 = 1011010010.$$

Then the secret can be reconstructed by xoring the two shares together, via:

$$s_1 \oplus s_2 = 0110000011 \oplus 1011010010 = 1101010001 = m.$$

Threshold Scheme

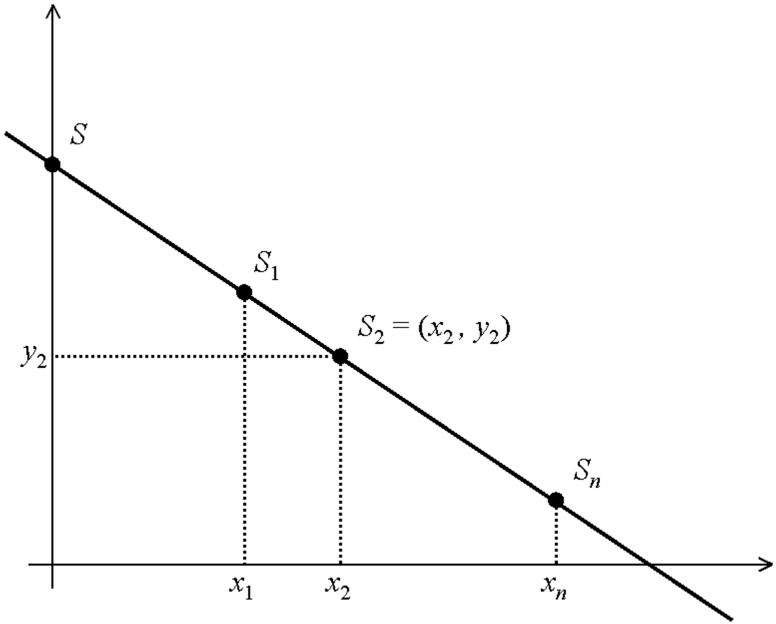
- A (t,w)-threshold scheme
  - Sharing key K among a set of *w* users
  - Any *t* users can recover the key
  - Any *t-1* users can not do so
- Schemes
  - A Simple 2-out-of-2 Scheme
  - Polynomial Interpolation
  - Shamir Secret Sharing
  - Visual Secret Sharing

Polynomial Interpolation

- **Principle:** d + 1 points determine a unique degree-d polynomial.

- If f is a polynomial that can be written as $f(x) = \sum_{i=0}^{d} f_i x^i$, then we say that f is a **degree-d** polynomial.

Polynomial Interpolation



Polynomials Over the Reals

- **Theorem 3.8 (Poly Interpolation)** Let $\{(x_1, y_1), \ldots, (x_{d+1}, y_{d+1})\} \subseteq \mathbb{R}^2$ be a set of points whose x_i values are all distinct. Then there is a unique degree-d polynomial f with real coefficients that satisfies $y_i = f(x_i)$ for all i .

Shamir Secret Sharing

- **Idea.** We have just seen that any d + 1 points on a degree-d polynomial are enough to uniquely reconstruct the polynomial. So a natural approach for secret sharing is to let each user's share be a point on a polynomial.

- **Shamir Secret Sharing** Share a secret $m \in Z_p$ with threshold t, first choose a degree-$(t - 1)$ polynomial f that satisfies $f(0) \equiv m \bmod p$, with all other coefficients chosen uniformly in Z_p. The i-th user receives the point $(i, f(i) \bmod p)$ on the polynomial.

Shamir Secret Sharing

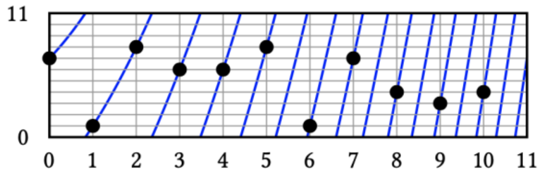
Example

Here is an example of 3-out-of-5 secret sharing over $\mathbb{Z}_{11}$ (so $p = 11$). Suppose the secret being shared is $m = 7 \in \mathbb{Z}_{11}$. The Share *algorithm chooses a random degree-2 polynomial with constant coefficient 7.*

Let's say that the remaining two coefficients are chosen as $f_2 = 1$ and $f_1 = 4$, resulting in the following polynomial:

$$f(\boldsymbol{x}) = 1\,\boldsymbol{x}^2 + 4\,\boldsymbol{x} + 7$$

This is the same polynomial illustrated in the previous example:



Shamir Secret Sharing

Example(Continued)

For each user $i \in \{1, \ldots, 5\}$, *we distribute the share* $(i, f(i) \,\%\, 11)$. *These shares correspond to the highlighted points in the mod-11 picture above.*

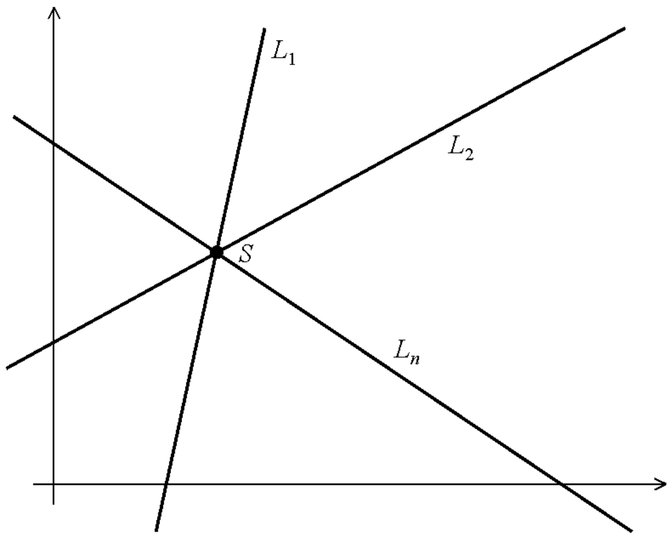
user (i)	$f(i)$	share $(i, f(i) \,\%\, 11)$
1	$f(1) = 12$	$(1, 1)$
2	$f(2) = 19$	$(2, 8)$
3	$f(3) = 28$	$(3, 6)$
4	$f(4) = 39$	$(4, 6)$
5	$f(5) = 52$	$(5, 8)$

- Because Share is a randomized algorithm, there are **many valid** sharings of the same secret.

Blakley's Scheme

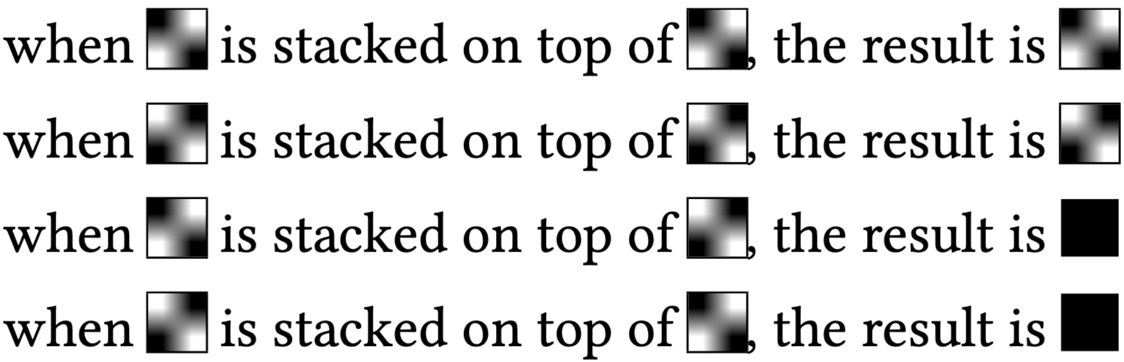
- Secret is a point in an t-dimensional space
- Dealer gives each user a hyper-plane passing the secret point
- Any t users can recover the common point

Geometry View



Visual Secret Sharing

- a simple visual secret sharing scheme that is inspired by the following observations:



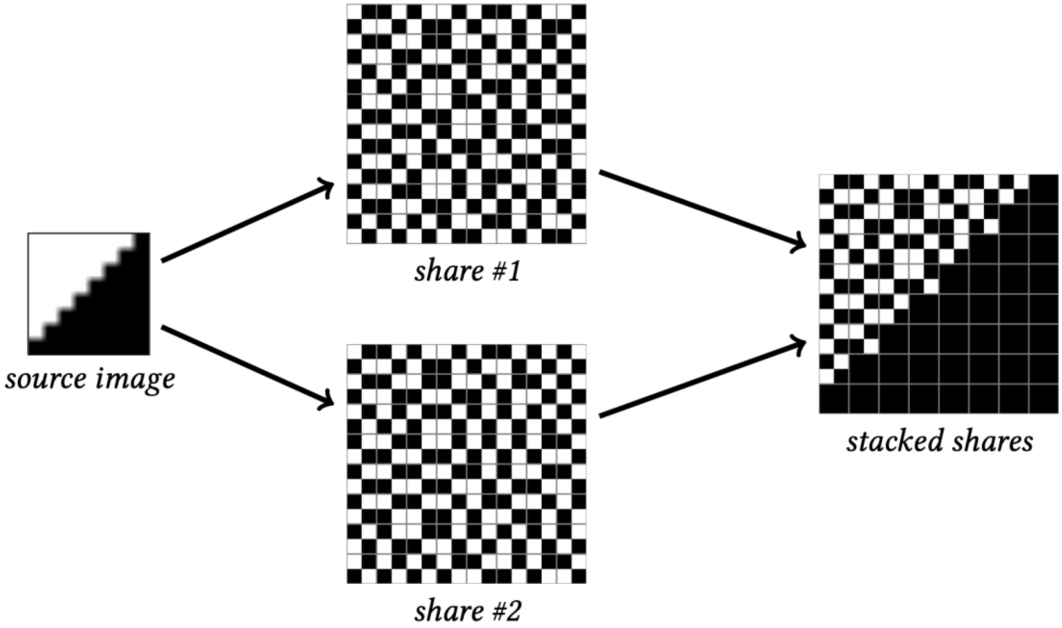
Visual Secret Sharing

» Construction 3.14

<u>Share(m):</u>
initialize empty images s_1, s_2, with dimensions twice that of m
for each position (i, j) in m:
randomly choose $b_1 \leftarrow \{$, $\}$
if $m[i, j]$ is a white pixel: set $b_2 := b_1$
if $m[i, j]$ is a black pixel: set b_2 to the "opposite" of b_1 (*i.e.*, $\{$, $\} \setminus \{b_1\}$)
add 2×2 block b_1 to image s_1 at position $(2i, 2j)$
add 2×2 block b_2 to image s_2 at position $(2i, 2j)$
return (s_1, s_2)

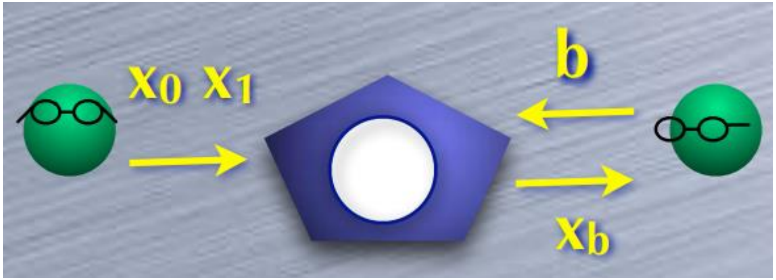
Visual Secret Sharing

» **Example**



Oblivious Transfer

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



2. MPC

Scenario 1: Private Dating

- Alice and Bob meet at a pub
  - If both of them want to date together – they will find out
  - If Alice doesn't want to date – she won't learn his intentions
  - If Bob doesn't want to date – he won't learn her intentions

Scenario 1: Private Dating

- Alice and Bob meet at a pub
  - If both of them want to date together – they will find out
  - If Alice doesn't want to date – she won't learn his intentions
  - If Bob doesn't want to date – he won't learn her intentions

Solution: use a trusted bartender

Scenario 2: Private Auction

» Many parties wish to execute a private auction
  - The highest bid wins
  - Only the highest bid (and bidder) is revealed

Scenario 2: Private Auction

» Many parties wish to execute a private auction
  - The highest bid wins
  - Only the highest bid (and bidder) is revealed

Solution: use a trusted auctioneer

Scenario 3: Private Set Intersection

» Intelligence agencies holds lists of potential terrorists

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

Mossad

MI5

FBI

Scenario 3: Private Set Intersection

» Intelligence agencies holds lists of potential terrorists

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

Solution: use a trusted party

Mossad

MI5

FBI

Trust me

Scenario 4: Online Poker

- Play online poker reliably

Scenario 4: Online Poker

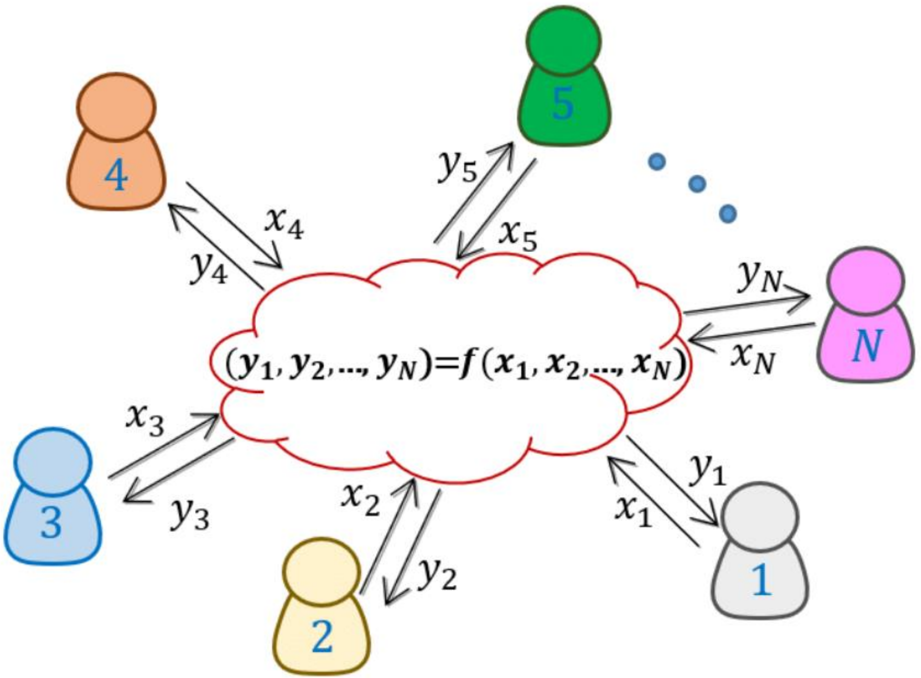
- Play online poker reliably

Solution: use a trusted party

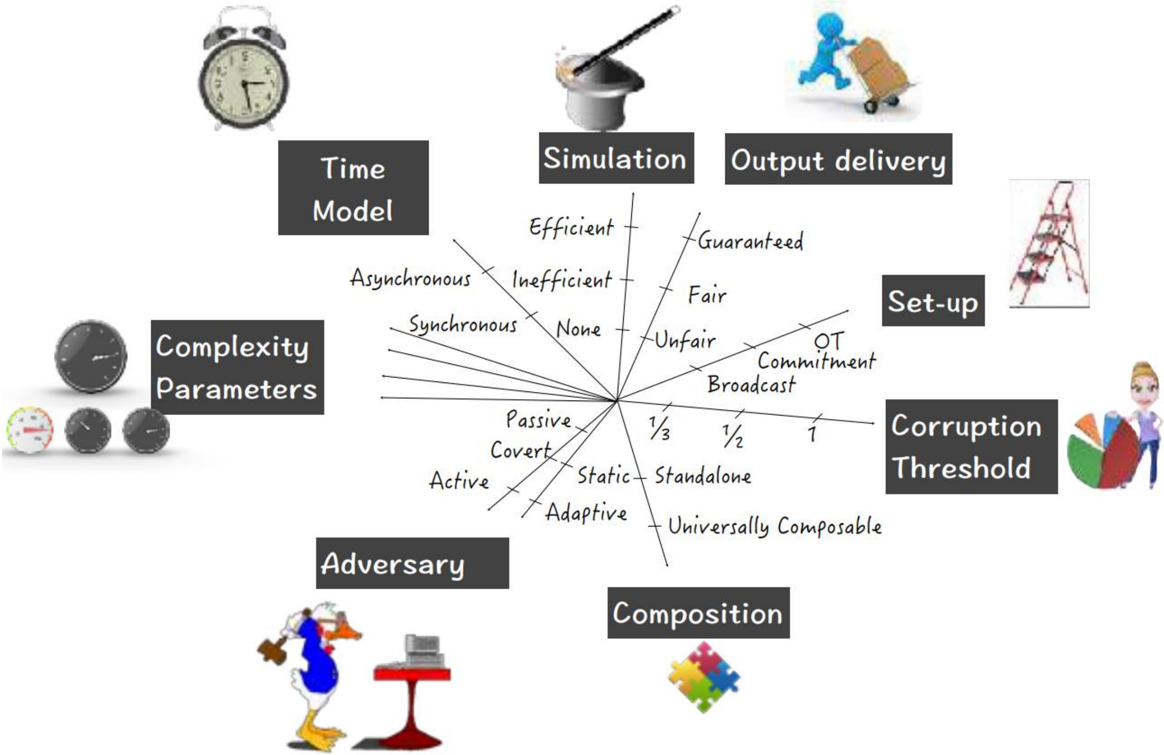
Secure Multiparty Computation

- **Goal:** use a protocol to emulate the trusted party

Secure Multiparty Computation

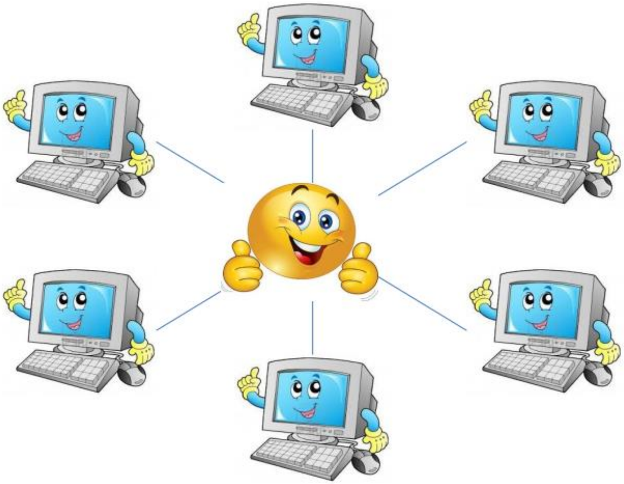


MPC dimensions



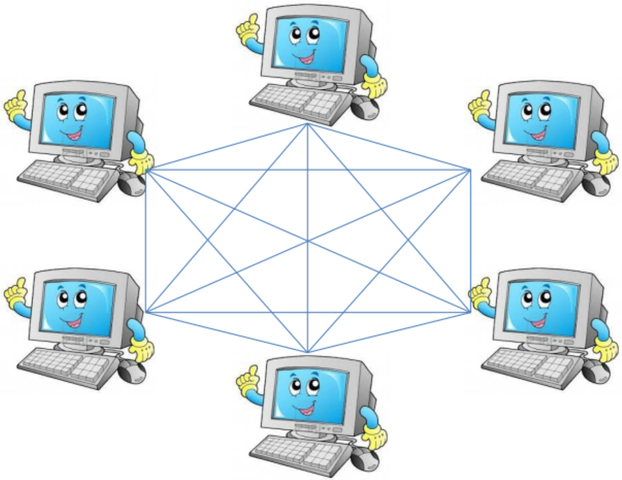
Ideal World

- 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

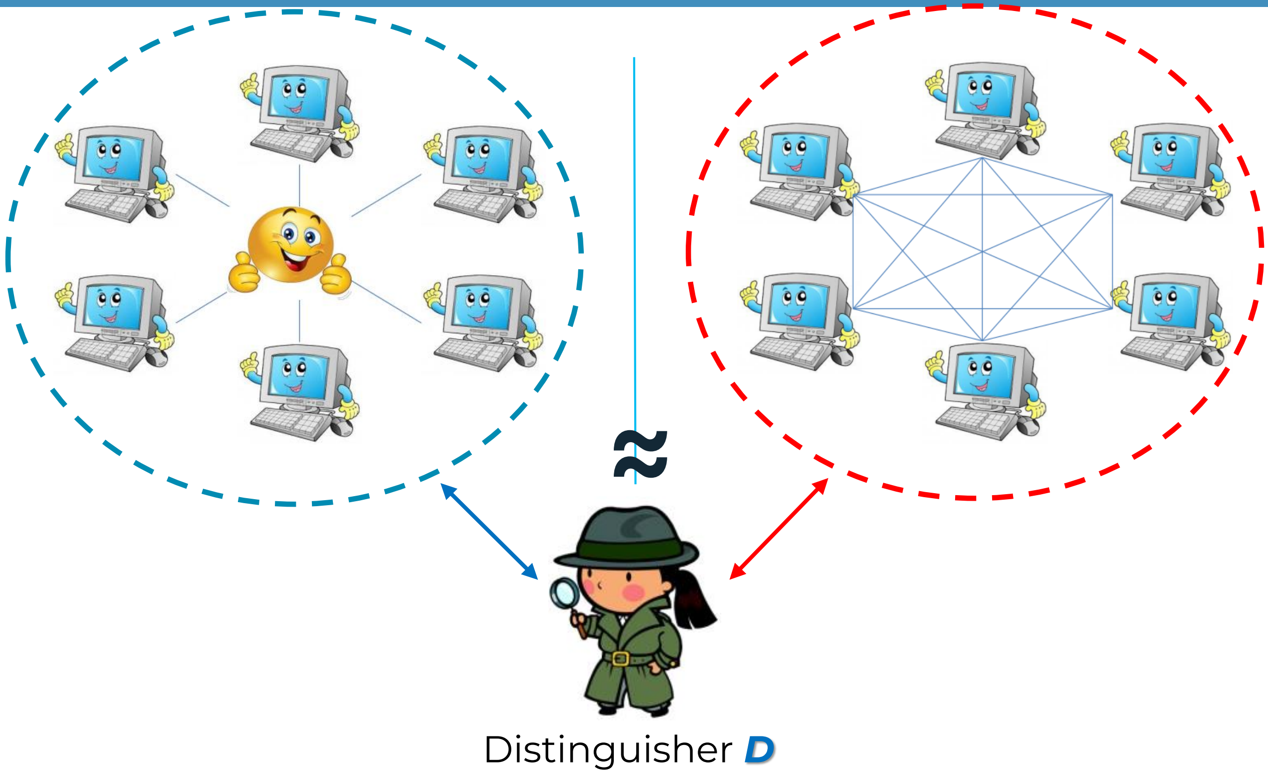


Real World

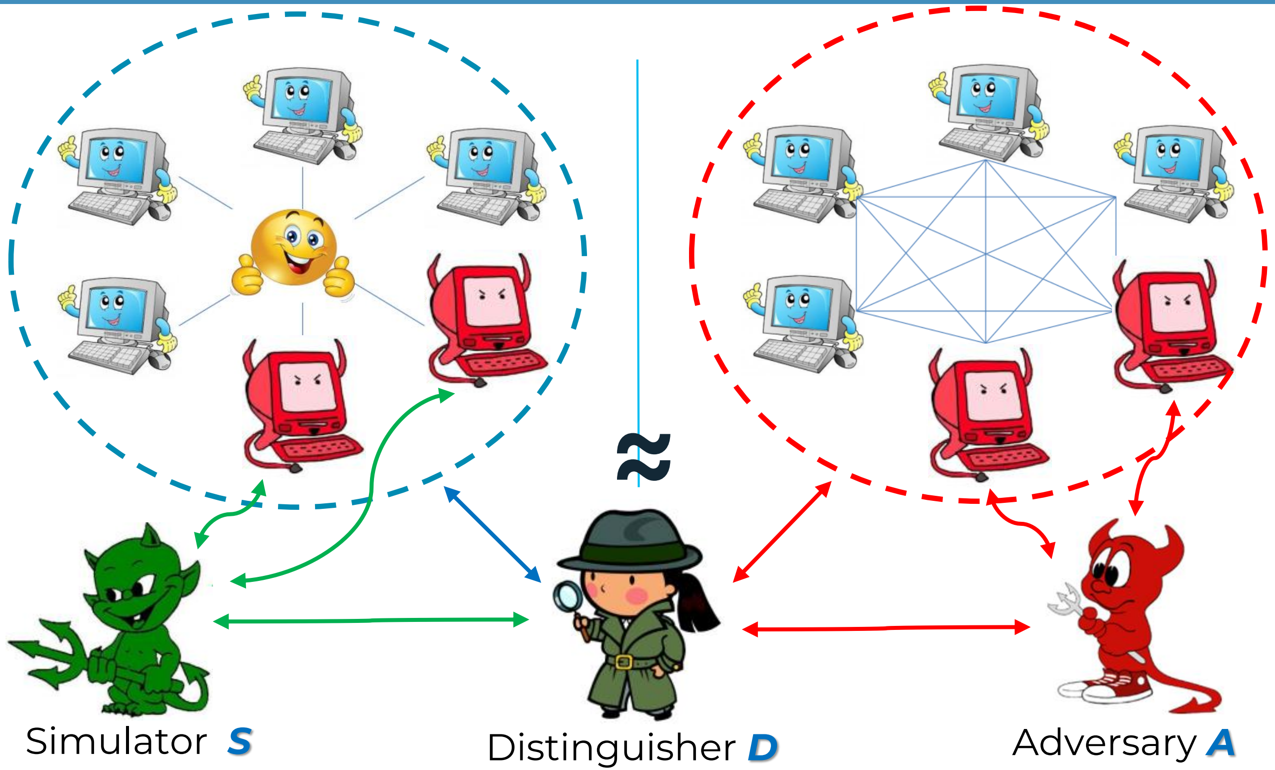
- Parties run a protocol π on inputs $(x_1,\ldots x_n)$



Simulation-Based Security



Simulation-Based Security



Simulation-Based Security

- The distinguisher ***D***:
  - Gives inputs to parties
  - Gets back output from parties and from adversary/simulator
  - Guesses which world it is real/ideal
- Protocol π securely computes y if $\forall \mathbf{A} \exists \mathbf{S} \forall \mathbf{D}$ distinguishing success is "small"

For every real-model adversary ***A***, there exists an ideal-model simulator ***S*** , such that the result of a real execution of π with ***A*** is indistinguishable from the result of an ideal execution with ***S***

The Definition

- A definition of an MPC task involves defining:
  - **Functionality**: what do we want to compute?
  - **Security type**: how strong protection do we want?
  - **Adversarial model**: what do we want to protect against?
  - **Network model**: in what setting are we going to do it?

The Functionality

- The code of the trusted party

- We will focus on secure function evaluation (SFE), the trusted party computes $y=f(x_1,\ldots x_n)$
  - Deterministic vs. randomized
  - Single public output vs. private outputs
  - Reactive vs. non-reactive

Security Type

- **Computational:**
  - The real & ideal worlds are computationally indistinguishable
- **Statistical (information-theoretic):** all-powerful distinguisher, negligible error probability
  - The real & ideal worlds are statistically close
- **Perfect:** all-powerful distinguisher, zero error probability
  - The real & ideal worlds are identically distributed

Adversarial Model (1)

- Adversarial behavior
  - **Semi honest**: honest-but-curious. corrupted parties follow the protocol honestly, adversary tries to learn more information.
  - **Fail stop**: same as semi honest, but corrupted parties can prematurely halt.
  - **Malicious**: corrupted parties can deviate from the protocol in an arbitrary way

Adversarial Model (2)

- Adversarial power
  - **Polynomial time**: computational security, normally requires cryptographic assumptions, e.g., encryption, signatures, oblivious transfer
  - **Computationally unbounded**: an all-powerful adversary, information-theoretic security

Adversarial Model (3)

- Adversarial corruption
  - **Static**: the set of corrupted parties is defined before the execution of the protocol begins. Honest parties are always honest, corrupted parties are always corrupted
  - **Adaptive**: adversary can decide which parties to corrupt during the course of the protocol, based on information it dynamically learns
  - **Mobile**: adversary can "jump" between parties. Honest parties can become corrupted, corrupted parties can become honest again

Adversarial Model (4)

- Number of corrupted parties
  - **Threshold adversary**: Denote by $t \leq n$, an upper bound on # corruptions
    - No honest majority, e.g., two-party computation
    - Honest majority, i.e., $t \leq n/2$
    - Two-thirds majority, i.e., $t \leq n/3$
  - **General adversary structure:**
    - Protection against specific subsets of parties

Communication Model (1)

- **Point-to-point**: fully connected network of pairwise channels. (Partial networks: star, chain)
  - Unauthenticated channels
  - Authenticated channels: in the computational setting
  - Private channels: in the IT setting
- **Broadcast**: additional broadcast channel

Communication Model (2)

- Message delivery:
  - **Synchronous**: the protocol proceeds in rounds. Every message that is sent arrives within an known time frame
  - **Asynchronous (eventual delivery)**: the adversary can impose arbitrary (finite) delay on any message
  - **Fully Asynchronous**: the adversary has full control over the network, can even drop messages

Execution Environment

- **Stand alone**:
  - A single protocol execution at any given time (isolated from the rest of the world)
- **Concurrent general composition**:
  - Arbitrary protocols are executed concurrently
  - An Internet-like setting
  - Requires a strictly stronger definition. Captured by the universal composability (UC) framework
  - Impossible in general without a trusted setup assumption (e.g., common reference string)

Feasibility Results

- Malicious setting
  - For $t \le n/3$, every f can be securely computed with perfect security [BGW'88,CCD'88]
  - For $t \le n/2$, every f can be securely computed with statistical security [RB'89]
  - For $t \le n$, assuming OT, every f can be securely computed with abort and computational [GMW'87]
- Semi-honest setting
  - For $t \le n/2$, every f can be securely computed with perfect security [BGW'88,CCD'88]
  - For $t \le n$, assuming OT, every f can be securely computed with computational security[GMW'87]

Defining Privacy

- First pick an input $(x_1, \ldots, x_n)$ for the protocol and make a run of the protocol on this input.
- Then pick some $C \subset \{P_1, \ldots, P_n\}$ with $|C| \leq t$ and consider the values $\{view_j\}_{P_j \in C}$, where $view_j$ is the view of party P_j in the execution.

- **leaked** values : the values $\{view_j\}_{Pj \in C}$.
- **allowed** values: the values $\{x_j, y_j\}_{Pj \in C}$, (the input and output of each party)

Defining Privacy

- A protocol is **private** if it always holds that the leaked values contain no more information than the allowed values.

- A protocol is private if it always holds that the leaked values can be computed efficiently from the allowed values.

MPC Techniques

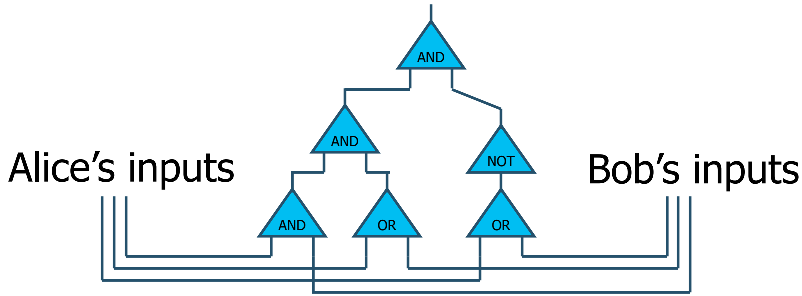
» Secure Multiparty Computation
  - Yao‘s Garbled Circuit
  - Secret sharing schemes
  - Homomorphic Encryption

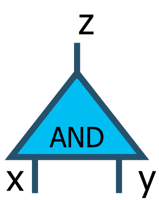
Yao's Theorem

- The first completeness theorem for secure computation.
- It states that for ANY efficiently computable function, there is a secure two-party protocol in the semi-honest model.
  - Therefore, in theory there is no need to design protocols for specific functions.
  - Surprising!

Yao's Protocol

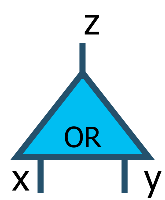
- Compute any function securely
  - … in the semi-honest model
- First, convert the function into a boolean circuit





Truth table:

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

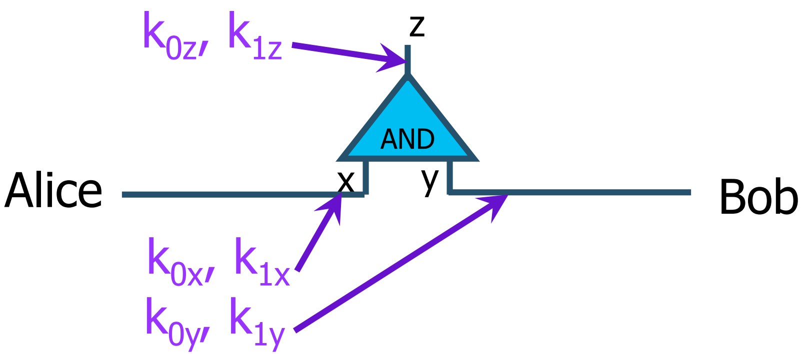


Truth table:

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

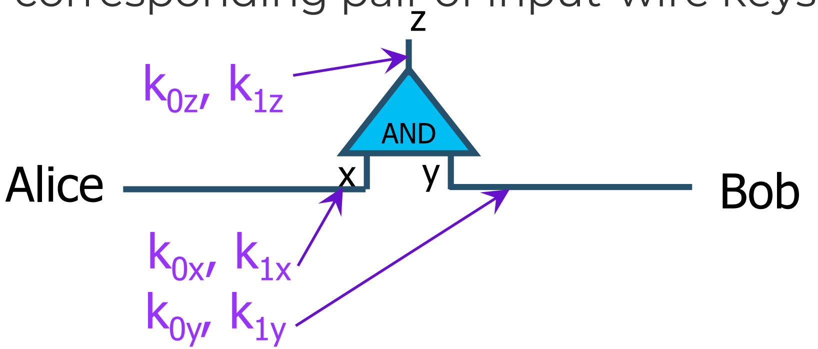
1: Pick Random Keys For Each Wire

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



Original truth table:

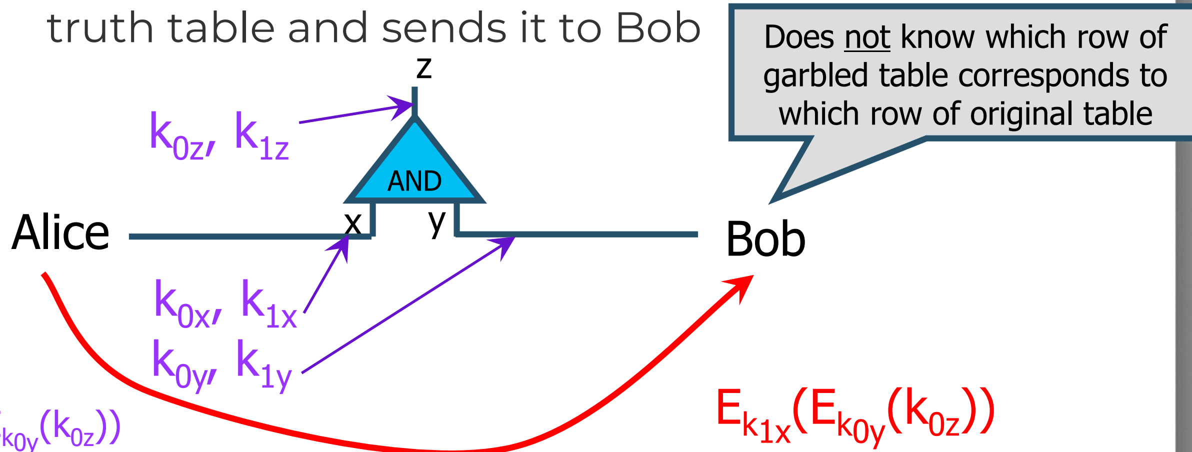
x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

Encrypted truth table:

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

3: Send Garbled Truth Table

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



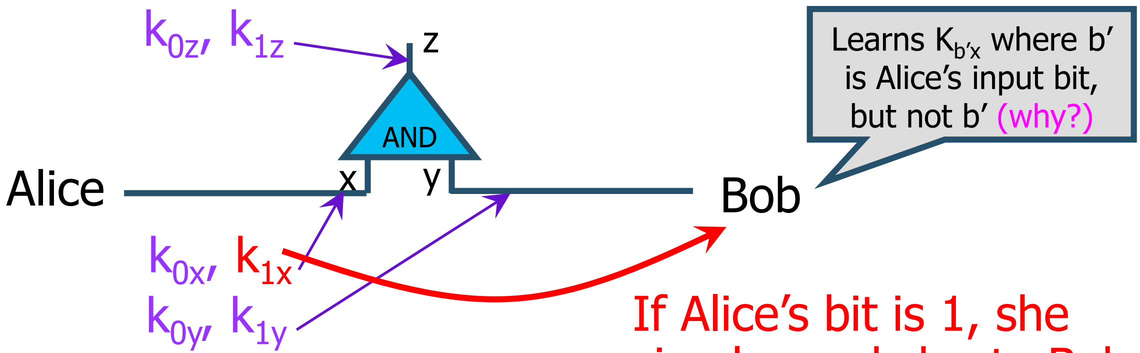
$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$
$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$
$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$

Garbled truth table:

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

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



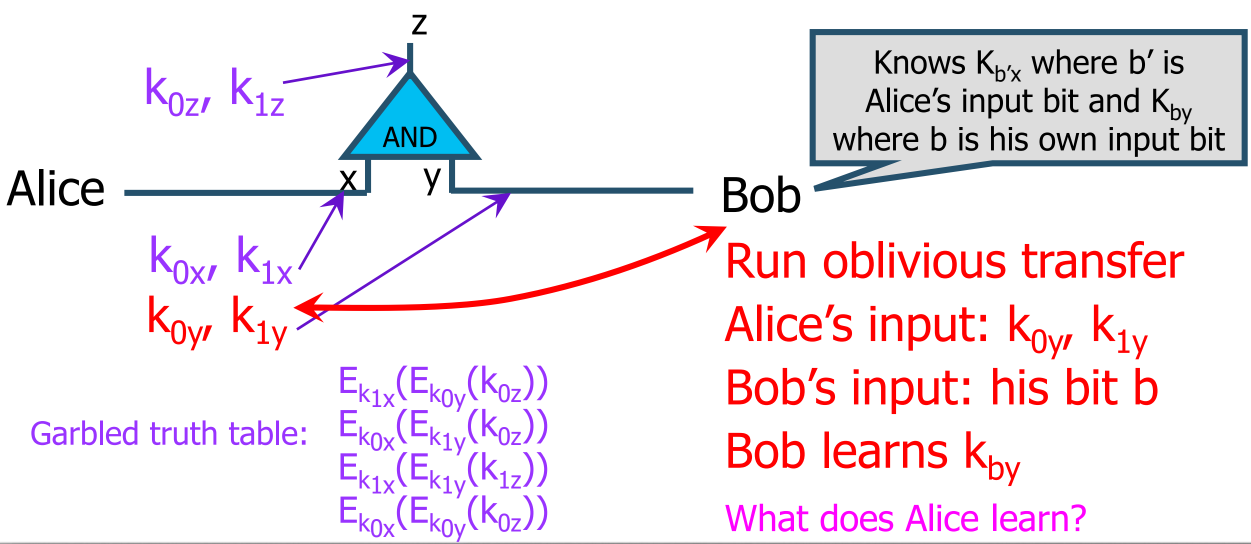
If Alice's bit is 1, she simply sends k_{1x} to Bob; if 0, she sends k_{0x}

Garbled truth table:

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

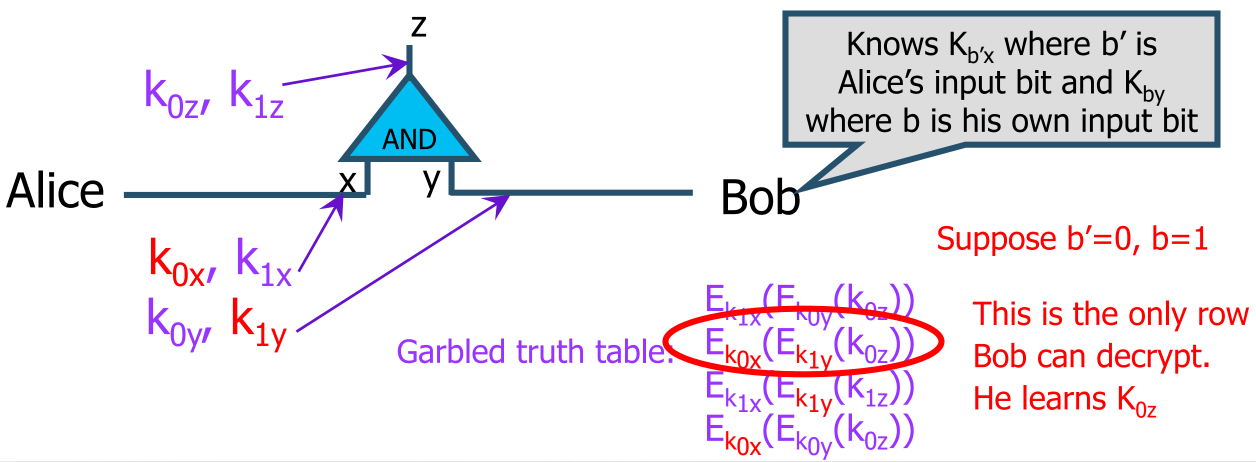
5: Use OT on Keys for Bob's Input

- 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?



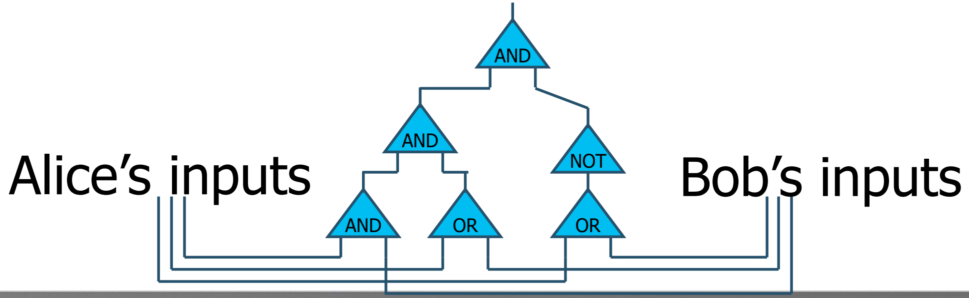
Evaluation of Garbed Circuit

» Given the two keys (one for Alice and one for Bob) representing the inputs of a gate, we can easily obtain the key representing the output of the gate.

- Only need to decrypt the corresponding entry using both keys from Alice and Bob
- But we do not know which entry it is? We can decrypt all entries. Suppose each cleartext contains some redundancy (like a hash value). Then only decryption of the right entry can yield such redundancy.

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?)



Finishing the Evaluation

- At the end of evaluation, Bob gets the keys representing the output bits of circuit.
  - Alice sends Bob a table of the keys for each output bit.
  - Bob translates the keys back to the output bits.
- For privacy, we need to be careful:
  - The topology of the circuit should be the same for all circuits of a particular input size.
  - Then privacy is guaranteed.

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!

How to ensure Efficiency?

» Computation and Communication?
  - Communication optimization: reducing information that must be shared between the two parties;
  - Execution optimization: allowing for the same number of gates to be executed in a shorter amount of time.
  - Circuit optimization: reducing the number of gates needed to compute a function.

Privacy ≠ Security

- As we have mentioned, secure computation only deals with the process of computing the function
  - It does not ask whether or not the function should be computed

Privacy and Secure Computation

- Secure computation can be used to solve any distributed data-mining problem
- A two-stage process:
  - Decide that the function/algorithm should be computed – an issue of **privacy**
  - Apply secure computation techniques to compute it securely – **security**
- But, not every privacy problem can be cast as a distributed computation

THANKS!

Any questions?

You can find me at:

» zhanglan@ustc.edu.cn