Multiparty Computation

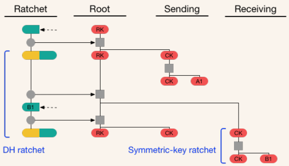
Hari

Cryptography

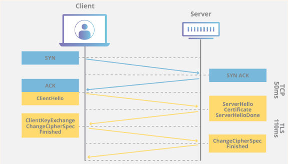
Cryptography

Secure Communication

- AES, RSA, etc.



Double Ratchet: Signal

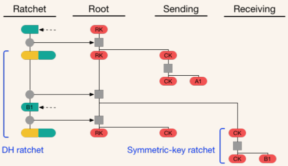


TLS: Cloudflare

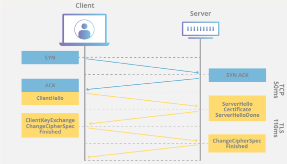
Cryptography

Secure Communication

- AES, RSA, etc.



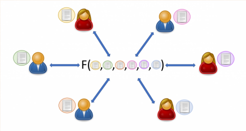
Double Ratchet: Signal

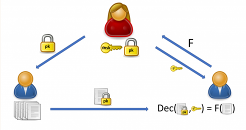


TLS: Cloudflare

Secure **Computation**

- Modern day constructions
- MPC, FE, FHE, and more
- Secure Voting and Auctions, Cryptocurrency, Secure ML





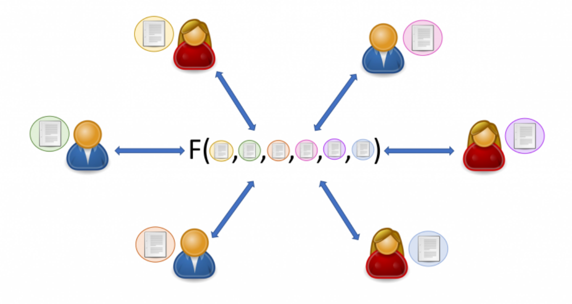
Secure Computation: COSIC

Secure Multiparty Computation

- Can we compute a function of data from multiple parties **securely**?

Secure Multiparty Computation

- Can we compute a function of data from multiple parties **securely**?



Secure MPC: Cosic

Secure Multiparty Computation

- We have n employees, who would like to compute the average salary without revealing individual salaries.

Secure Multiparty Computation

- We have n employees, who would like to compute the average salary without revealing individual salaries.
- Train a machine learning model without revealing training datasets

Secure Multiparty Computation

- We have n employees, who would like to compute the average salary without revealing individual salaries.
- Train a machine learning model without revealing training datasets
- Hiding auction bids on a smart contract

Secure Multiparty Computation

- First two party protocols introduced by Yao
- Generalized to multiple parties by Goldreich, Micali, and Widgerson

Secure Multiparty Computation

- First two party protocols introduced by Yao
- Generalized to multiple parties by Goldreich, Micali, and Widgerson

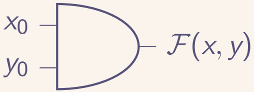
GMW Protocol

Introduction to GMW

Given a function $\mathcal{F}(x_1, x_2, \ldots x_n)$ representable as a Boolean circuit $\mathcal{C}$, how do we evaluate it?

Introduction to GMW

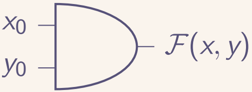
Given a function $\mathcal{F}(x_1, x_2, \ldots x_n)$ representable as a Boolean circuit $\mathcal{C}$, how do we evaluate it?



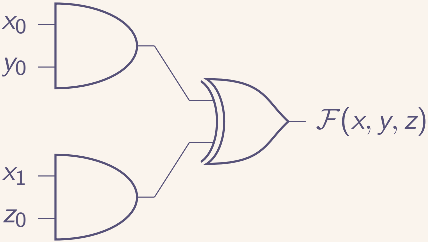
Example circuit

Introduction to GMW

Given a function $\mathcal{F}(x_1, x_2, \ldots x_n)$ representable as a Boolean circuit $\mathcal{C}$, how do we evaluate it?



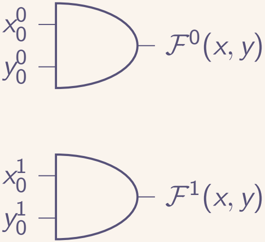
Example circuit



Slightly more complicated circuit

Introduction to GMW

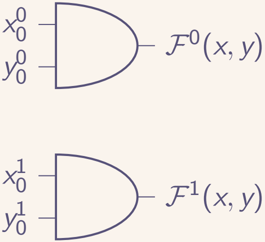
- Idea: Each party gives a "piece" of their data to the other



Example split circuit (Two parties)

Introduction to GMW

- Idea: Each party gives a "piece" of their data to the other



Example split circuit (Two parties)

- With these input pieces, each party can evaluate the circuit
- Known as **secret sharing**

Secret Sharing

- For each bit a, we can split it into n shares as following:

Secret Sharing

- For each bit a, we can split it into n shares as following:
- Choose random bits $r_0, r_1, r_2, \ldots, r_{n-1}$

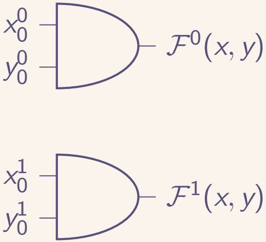
Secret Sharing

- For each bit a, we can split it into n shares as following:
- Choose random bits $r_0, r_1, r_2, \ldots, r_{n-1}$
- Set $r_n = a \oplus r_0 \oplus r_1 \oplus \ldots \oplus r_{n-1}$

Secret Sharing

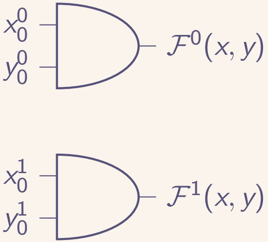
- For each bit a, we can split it into n shares as following:
- Choose random bits $r_0, r_1, r_2, \ldots, r_{n-1}$
- Set $r_n = a \oplus r_0 \oplus r_1 \oplus \ldots \oplus r_{n-1}$
- Provide each party with one of the r_i
- Notice that all parties must get together to find a
- The r_i are known as **secret shares** or **shares**

GMW



Example split circuit (Two parties)

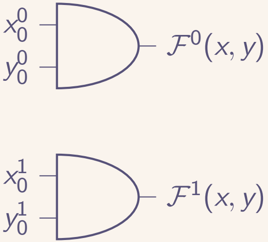
GMW



Example split circuit (Two parties)

- Party x creates the shares x_0^0, x_0^1 of bit x_0
- Party y creates the shares y_0^0, y_0^1 of bit y_0

GMW



Example split circuit (Two parties)

- Party x creates the shares x_0^0, x_0^1 of bit x_0
- Party y creates the shares y_0^0, y_0^1 of bit y_0
- Party x gets x_0^0, y_0^0, Party y gets x_0^1, x_0^1

How do we evaluate this "shared" circuit?

Evaluation

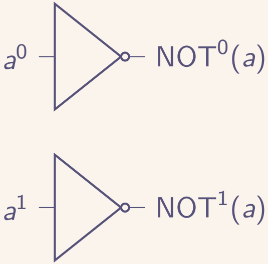
- Firstly, we assume our circuit only has AND, NOT, and XOR gates (which is universal)

Evaluation

- Firstly, we assume our circuit only has AND, NOT, and XOR gates (which is universal)
- When we evaluate a gate, **we want to get a secret share of the gate output**
- This allows parties to continue evaluating the next gate

Evaluation: NOT

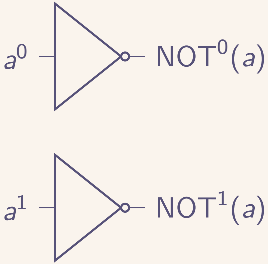
$$a^0 \oplus a^1 = a$$



Shared NOT Circuit

Evaluation: NOT

$$a^0 \oplus a^1 = a$$



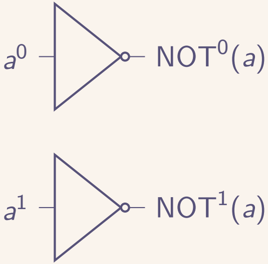
Shared NOT Circuit

- Notice that if we set a^0 to $\text{NOT}(a^0)$, we get

$$\text{NOT}(a^0) \oplus a^1 = \text{NOT}(a)$$

Evaluation: NOT

$$a^0 \oplus a^1 = a$$



Shared NOT Circuit

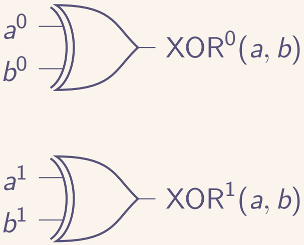
- Notice that if we set a^0 to NOT(a^0), we get

$$\text{NOT}(a^0) \oplus a^1 = \text{NOT}(a)$$

- One party just has to flip their share

Evaluation: XOR

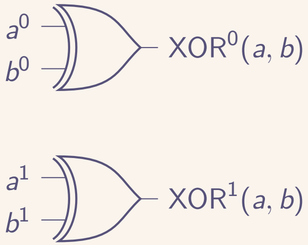
$$a^0 \oplus a^1 = a,\ b^0 \oplus b^1 = b$$



Shared XOR Circuit

Evaluation: XOR

$$a^0 \oplus a^1 = a,\ b^0 \oplus b^1 = b$$

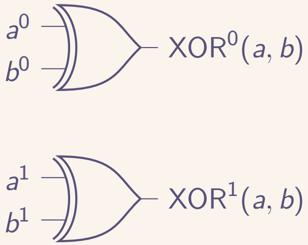


Shared XOR Circuit

- Notice that $(a^0 \oplus b^0) \oplus (a^1 \oplus b^1) = a \oplus b$

Evaluation: XOR

$$a^0 \oplus a^1 = a,\ b^0 \oplus b^1 = b$$

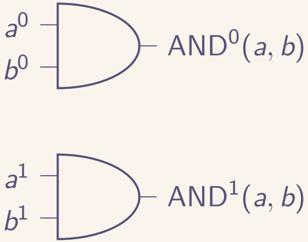


Shared XOR Circuit

- Notice that $(a^0 \oplus b^0) \oplus (a^1 \oplus b^1) = a \oplus b$
- Then we can have each party evaluate $\mathrm{XOR}^i(a, b) = a^i \oplus b^i$

Evaluation: AND

$$a^0 \oplus a^1 = a,\ b^0 \oplus b^1 = b$$



Shared AND Circuit

How do we do this?

Detour: Oblivious Transfer

Oblivious Transfer

Definition (One out of n Oblivious Transfer)

We have two parties, the sender $\mathcal{S}$ and receiver $\mathcal{R}$.
$\mathcal{S}$ has n secrets $x_0, x_1, \ldots x_n$.
$\mathcal{R}$ has a selection value v from 1 to n.

Oblivious Transfer

Definition (One out of n Oblivious Transfer)

We have two parties, the sender $\mathcal{S}$ and receiver $\mathcal{R}$.
$\mathcal{S}$ has n secrets $x_0, x_1, \ldots x_n$.
$\mathcal{R}$ has a selection value v from 1 to n.
An **Oblivious Transfer** protocol is a protocol where

- $\mathcal{R}$ receives x_v without learning any of the other secrets
- $\mathcal{S}$ does not learn v

Oblivious Transfer

- Seems like magic: we can achieve it with public key cryptography (ex. RSA or Discrete Log)
- (See this paper to see some examples, there are many)

Oblivious Transfer

- Seems like magic: we can achieve it with public key cryptography (ex. RSA or Discrete Log)
- (See this paper to see some examples, there are many)
- Here we treat it like a gnome inside a magic box, purpose is not to explain OT

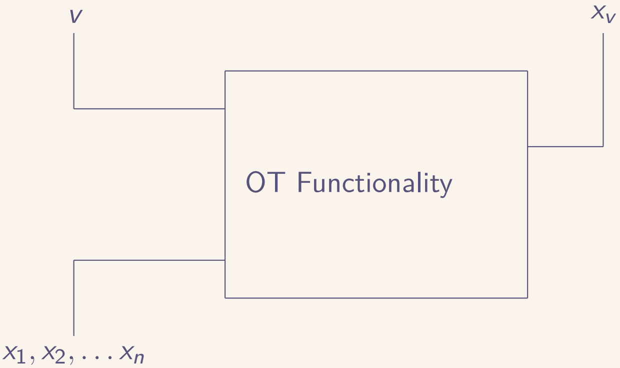
Oblivious Transfer

- Seems like magic: we can achieve it with public key cryptography (ex. RSA or Discrete Log)
- (See this paper to see some examples, there are many)
- Here we treat it like a gnome inside a magic box, purpose is not to explain OT
- (small fun fact: existence of OT is equivalent to existence of MPC, see this)

Oblivious Transfer

- Seems like magic: we can achieve it with public key cryptography (ex. RSA or Discrete Log)
- (See this paper to see some examples, there are many)
- Here we treat it like a gnome inside a magic box, purpose is not to explain OT
- (small fun fact: existence of OT is equivalent to existence of MPC, see this)
- (ask me after for an OT example)

Oblivious Transfer

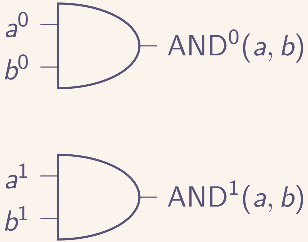


Oblivious Transfer

Back to GMW!

Evaluation: AND

$$a^0 \oplus a^1 = a,\ b^0 \oplus b^1 = b$$



Shared AND Circuit

How do we do this?

Evaluation: AND

- The second party has 4 possible values for its share values:

$$a^1 \in \{0, 1\}$$

$$b^1 \in \{0, 1\}$$

Evaluation: AND

- The second party has 4 possible values for its share values:

$$a^1 \in \{0, 1\}$$

$$b^1 \in \{0, 1\}$$

- From the first party's perspective: 4 possible values for AND(a, b):

$$(a^0 \oplus a^1) \wedge (b^0 \oplus b^1)$$

Evaluation: AND

- The second party has 4 possible values for its share values:

$$a^1 \in \{0, 1\}$$

$$b^1 \in \{0, 1\}$$

- From the first party's perspective: 4 possible values for AND(a, b):

$$(a^0 \oplus a^1) \wedge (b^0 \oplus b^1)$$

Evaluation: AND

- The first party can select a random bit $r \in \{0, 1\}$, and create 4 "possible" secret shares

$$\begin{pmatrix} r \oplus \big((a^0 \oplus 0) \wedge (b^0 \oplus 0)\big) \\ r \oplus \big((a^0 \oplus 0) \wedge (b^0 \oplus 1)\big) \\ r \oplus \big((a^0 \oplus 1) \wedge (b^0 \oplus 0)\big) \\ r \oplus \big((a^0 \oplus 1) \wedge (b^0 \oplus 1)\big) \end{pmatrix}$$

Evaluation: AND

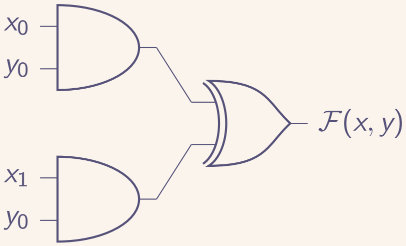
- The first party can select a random bit $r \in \{0, 1\}$, and create 4 "possible" secret shares

$$\begin{pmatrix} r \oplus \left((a^0 \oplus 0) \wedge (b^0 \oplus 0)\right) \\ r \oplus \left((a^0 \oplus 0) \wedge (b^0 \oplus 1)\right) \\ r \oplus \left((a^0 \oplus 1) \wedge (b^0 \oplus 0)\right) \\ r \oplus \left((a^0 \oplus 1) \wedge (b^0 \oplus 1)\right) \end{pmatrix}$$

- Then the first party uses 1 out of 4 Oblivious Transfer to send the share
- Notice that we get a secret share of the AND value!

GMW

- And we're done! We can evaluate any circuit, and then at the end we reconstruct the secret with the shares.



Example Circuit

GMW: Multiple Parties

- How do we do multiple parties?

GMW: Multiple Parties

- How do we do multiple parties?
- NOT gates: Only one person still needs to flip their share

$$\neg(a^0 \oplus a^1 \oplus a^2 \dots) = (\neg a^0) \oplus a^1 \oplus a^2 \dots$$

- XOR gates: All parties still xor their shares together

$$(a^0 \oplus a^1 \oplus a^2 \dots) \oplus (b^0 \oplus b^1 \oplus b^2 \dots)$$
$$= (a^0 \oplus b^0) \oplus (a^1 \oplus b^1) \oplus \dots$$

GMW: Multiple Parties

- How do we do multiple parties?
- NOT gates: Only one person still needs to flip their share

$$\neg(a^0 \oplus a^1 \oplus a^2 \dots) = (\neg a^0) \oplus a^1 \oplus a^2 \dots$$

- XOR gates: All parties still xor their shares together

$$\begin{aligned}&(a^0 \oplus a^1 \oplus a^2 \dots) \oplus (b^0 \oplus b^1 \oplus b^2 \dots) \\ &= (a^0 \oplus b^0) \oplus (a^1 \oplus b^1) \oplus \dots\end{aligned}$$

- AND gates: ???

GMW: Multiple Parties

- AND gates: We can see

$$
\begin{aligned}
&(a^0 \oplus a^1 \oplus a^2 \dots) \wedge (b^0 \oplus b^1 \oplus b^2 \dots) \\
&= \left(\bigoplus_{i \in [n]} a^i \wedge b^i \right) \oplus \left(\bigoplus_{i \neq j} a^i \wedge b^j \right)
\end{aligned}
$$

- The left side can be computed by each party locally (AND each share)

GMW: Multiple Parties

- AND gates: We can see

$$(a^0 \oplus a^1 \oplus a^2 \ldots) \wedge (b^0 \oplus b^1 \oplus b^2 \ldots)$$
$$= \left(\bigoplus_{i \in [n]} a^i \wedge b^i \right) \oplus \left(\bigoplus_{i \neq j} a^i \wedge b^j \right)$$

- The left side can be computed by each party locally (AND each share)
- Shares of every pair of parties on the right can be obtained through the OT protocol

GMW: Multiple Parties

- AND gates: We can see

$$(a^0 \oplus a^1 \oplus a^2 \ldots) \wedge (b^0 \oplus b^1 \oplus b^2 \ldots)$$
$$= \left(\bigoplus_{i \in [n]} a^i \wedge b^i \right) \oplus \left(\bigoplus_{i \neq j} a^i \wedge b^j \right)$$

- The left side can be computed by each party locally (AND each share)
- Shares of every pair of parties on the right can be obtained through the OT protocol
- **Reduce the multiparty problem into a set of two party problems**

GMW Summary

- We can evaluate circuits now!

GMW Summary

- We can evaluate circuits now!
- We first split up each input bit into shares for each party

GMW Summary

- We can evaluate circuits now!
- We first split up each input bit into shares for each party
- Each party can then evaluate the circuit gate by gate using their shares

GMW Summary

- We can evaluate circuits now!
- We first split up each input bit into shares for each party
- Each party can then evaluate the circuit gate by gate using their shares
- In the end they can reconstruct the output by using output shares

More Fun Things

- This is just the surface: many more questions
- Can you evaluate functions securely if there are malicious parties?
- Can we improve round complexity and communication efficiency?

More Fun Things

- Very useful in the world of secure computation
- For example, say the circuit is a circuit which takes in an ML classifier and an image, and outputs the classification
- Allows doctors to provide medical images / data to ML models without violating HIPAA

More Fun Things

Bonus: Funny card trick (related to MPC)!!