CS4501 Cryptographic Protocols
Lecture 7: Interpolation, Linearity, Circuits

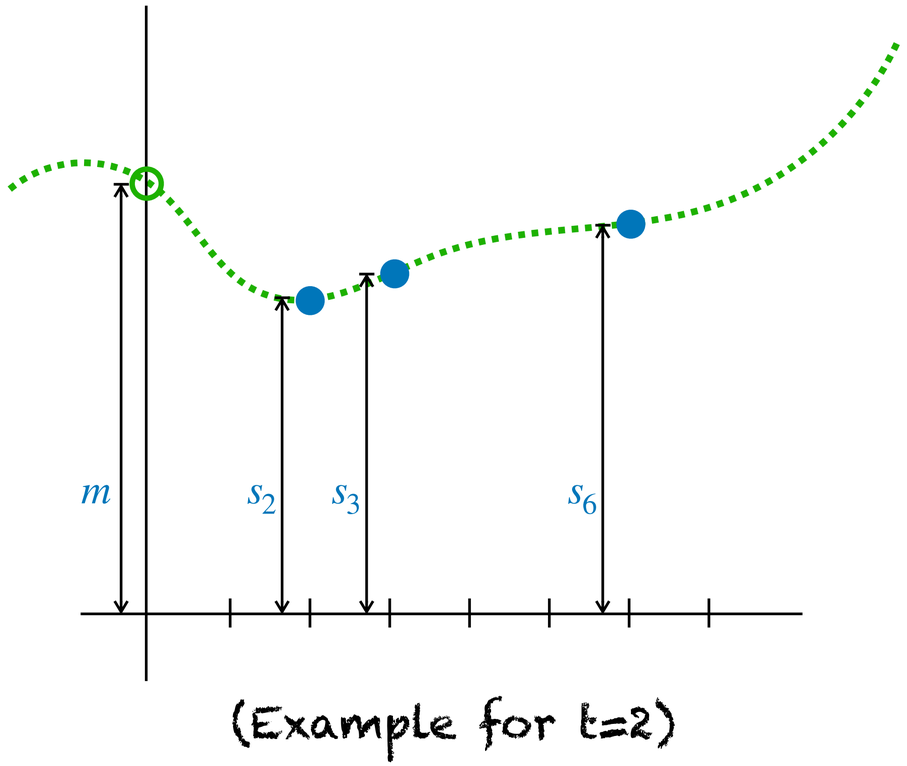
https://jackdoerner.net/teaching/#2026/Spring/CS4501

Interpolation: the Problem

- **Given:** pairwise distinct $(i_1, \ldots, i_{t+1}) \in \mathbb{F}_p^{t+1}$ and $\left(s_{i_1}, \ldots, s_{i_{t+1}}\right) \in \mathbb{F}_p^{t+1}$.
- **Find:** $f(x) = m + a_1 \cdot x + \ldots a_t \cdot x^t$ satisfying:

$$f(i_1) = m + a_1 \cdot i_1 + \ldots a_t \cdot i_1^t = s_{i_1}$$
$$f(i_2) = m + a_1 \cdot i_2 + \ldots a_t \cdot i_2^t = s_{i_2}$$
$$\ldots$$
$$f(i_{t+1}) = m + a_1 \cdot i_{t+1} + \ldots a_t \cdot i_{t+1}^t = s_{i_{t+1}}$$

This is a linear system with $t + 1$ equations and $t + 1$ variables, so...



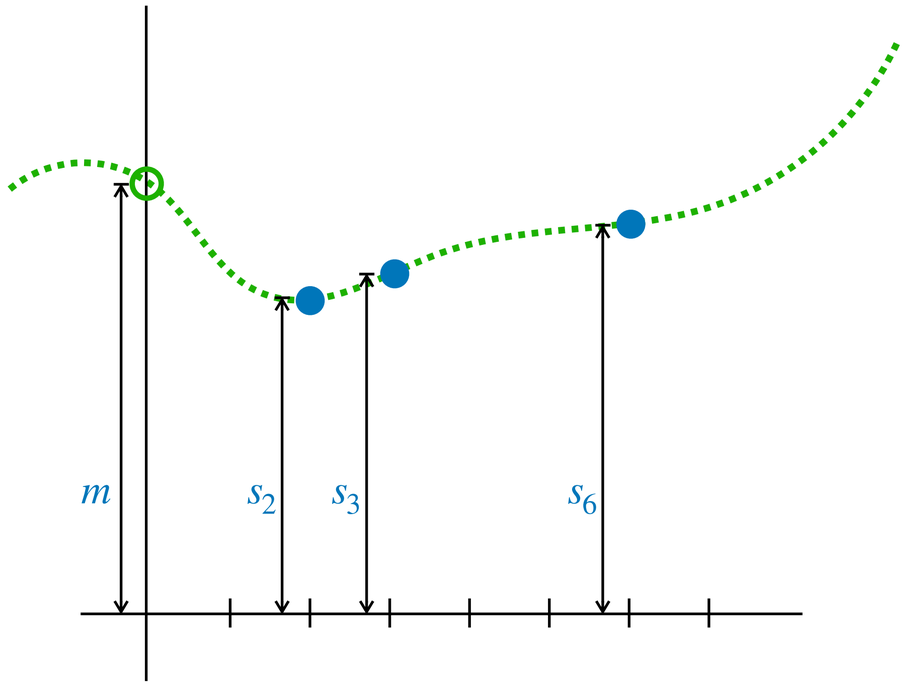
Interpolation: the Problem

- **Given:** pairwise distinct $(i_1, \ldots, i_{t+1}) \in \mathbb{F}_p^{t+1}$
  and $\left(s_{i_1}, \ldots, s_{i_{t+1}}\right) \in \mathbb{F}_p^{t+1}$.
- **Find:** $f(x) = m + a_1 \cdot x + \ldots a_t \cdot x^t$ satisfying:

$$\begin{bmatrix} 1 & i_1 & \cdots & i_1^t \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_{t+1} & \cdots & i_{t+1}^t \end{bmatrix} \begin{bmatrix} m \\ a_1 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{t+1} \end{bmatrix}$$

This is called a Vandermonde Matrix

It has determinant $\prod_{0 \le j < k \le t+1} (i_k - i_j) \neq 0$, therefore...



(Example for t=2)

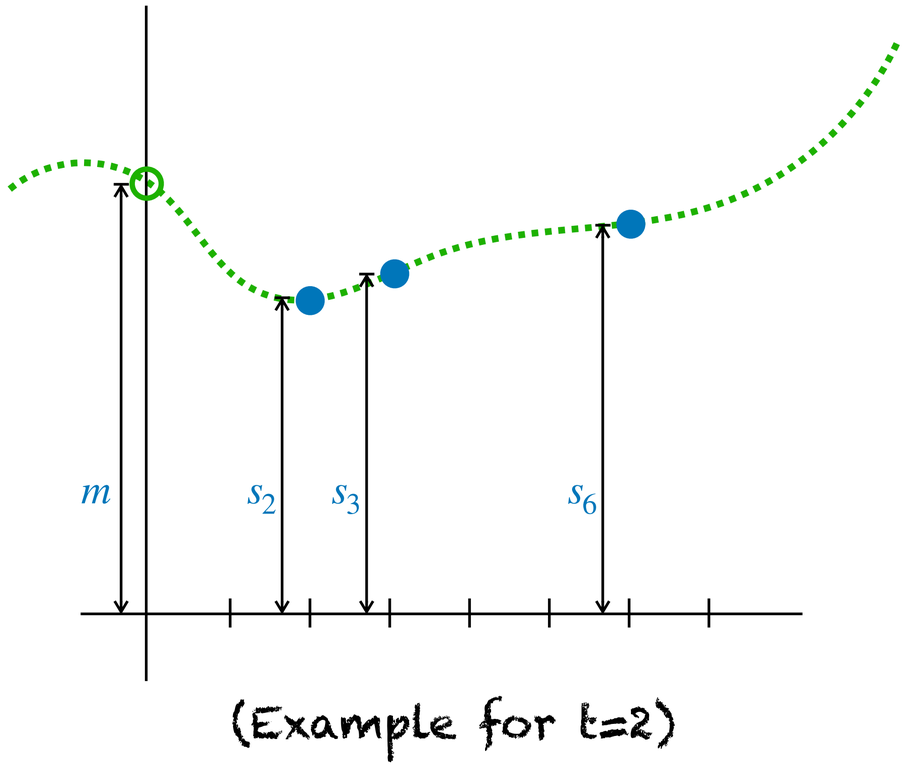
Interpolation: the Problem

- **Given:** pairwise distinct $(i_1, \ldots, i_{t+1}) \in \mathbb{F}_p^{t+1}$
  and $\left(s_{i_1}, \ldots, s_{i_{t+1}}\right) \in \mathbb{F}_p^{t+1}$.
- **Find:** $f(x) = m + a_1 \cdot x + \ldots a_t \cdot x^t$ satisfying:

$$\begin{bmatrix} m \\ a_1 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} 1 & i_1 & \cdots & i_1^t \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_{t+1} & \cdots & i_{t+1}^t \end{bmatrix}^{-1} \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{t+1} \end{bmatrix}$$

... it is always invertible.

Let's turn this into an *interpolation* algorithm for directly recovering any point on the polynomial!



(Chalkboard Proof)

Def 1 (Lagrange Basis): Given a set of $t+1$ distinct values $x_1, \ldots, x_{t+1} \in \mathbb{F}$, the $t+1$ Lagrange bases of degree $\leq t$ are defined $\forall i \in [t+1]$ as:

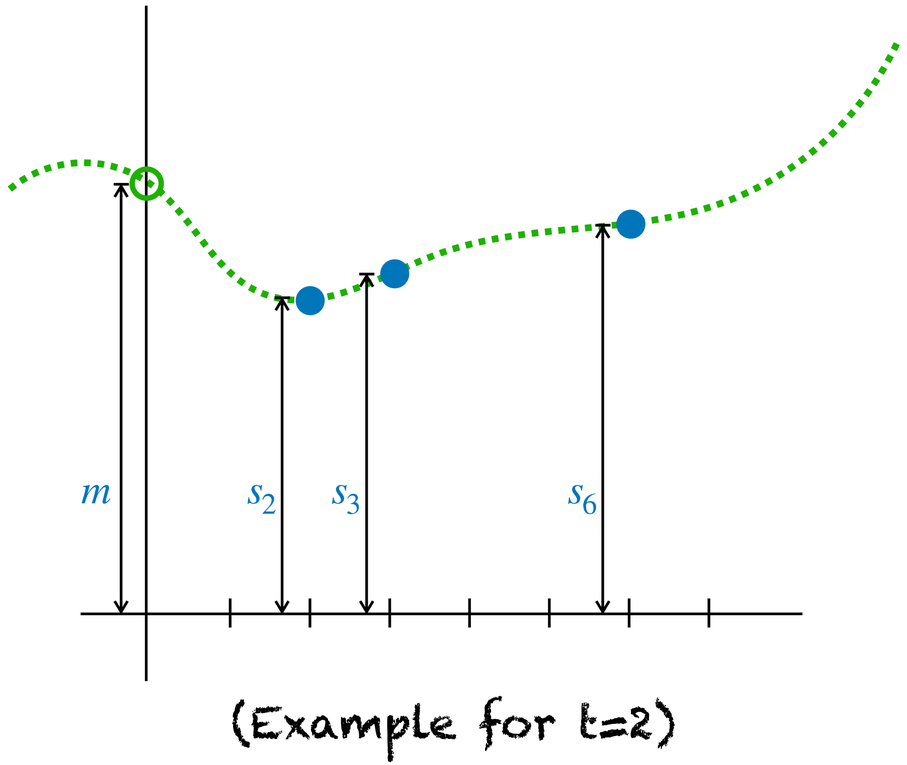
$$\ell_i(x) = \prod_{j \in [t+1] \setminus \{i\}} \frac{x - x_j}{x_i - x_j}$$

Def 2 (Lagrange Polynomial): Given a set of $t+1$ points $(x_1, y_1), \ldots, (x_{t+1}, y_{t+1}) \in \mathbb{F}^2$, with distinct x-coordinates, the degree $\leq t$ Lagrange polynomial is:

$$L(x) = \sum_{i \in [t+1]} y_i \cdot \ell_i(x)$$

Interpolating Shamir Shares Efficiently

- $\ell_1(x), \ldots, \ell_{t+1}(x)$ depend only upon $i_1, \ldots, i_{t+1}$, so any group of parties that wish to reconstruct can precompute the appropriate Lagrange bases. Then reconstruct is simply a linear combination.
- To recover m the parties only need to precompute $\ell_1(0), \ldots, \ell_{t+1}(0)$.
- If p is large, we can choose the indexes of the parties to make reconstruction more efficient! (e.g. so that we multiply by powers of 2)



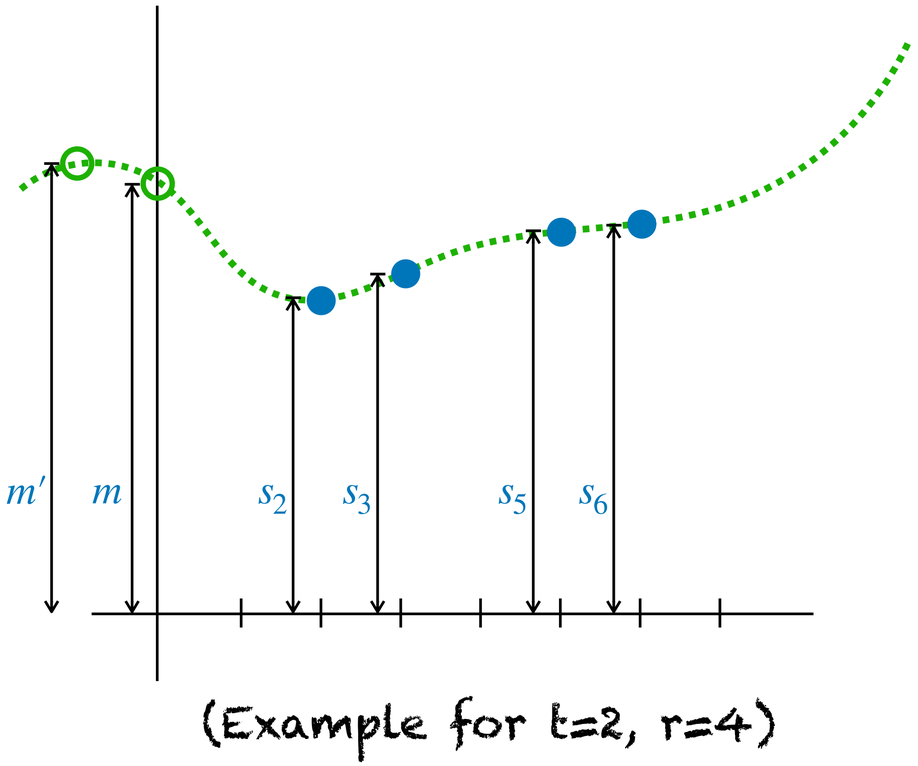
Memory-Efficient Secret Sharing

- **Question:** So far we encoded a single secret, achieved privacy against t corrupt parties, and required $t+1$ parties to reconstruct. What if we require $t+2$ parties to reconstruct (i.e. use a polynomial of degree $t+1$ instead of t)?

  Answer: We can encode two secrets!

  In general, for t corruptions and r reconstruction parties, you can encode $r-t$ secrets.
  This is called *packed* secret sharing.

  Checking that you still get correctness and privacy would be a good exercise for you!
  …but I'm not assigning it for homework.



Linear Secret Sharing Schemes

Definition 3: A secret sharing scheme is *linear* if and only if:

1. The message space is a group. That is, for some group $\mathbb{G}$ and every valid m, $m \in \mathbb{G}$.
2. The randomness consumed by the Share algorithm can be cast a vector $\vec{a}$ of elements of $\mathbb{G}$.
3. Each share s_i is a fixed, publicly known linear combination of m and $\vec{a}$.

Question: is Shamir Secret Sharing linear?

Answer: **Yes!** This means we can write the Share algorithm as a matrix multiplication.

$$\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} := \begin{bmatrix} 1 & i_1 & \cdots & i_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_n & \cdots & i_n^{n-1} \end{bmatrix} \begin{bmatrix} m \\ a_1 \\ \vdots \\ a_t \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

We don't strictly need a field here: for $i \in \mathbb{N}$ and $a \in \mathbb{G}$, $i \cdot a$ can be defined as a scalar product. That is, $i \cdot a = \underbrace{a + \ldots + a}_{i \text{ times}}$

However, if we aren't operating over a field, we might not be able to reconstruct!

Linear Secret Sharing Schemes

Now Suppose we add together sharings of two different values. What happens?

$$\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} := \begin{bmatrix} 1 & i_1 & \cdots & i_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_n & \cdots & i_n^{n-1} \end{bmatrix} \begin{bmatrix} m \\ a_1 \\ \vdots \\ a_t \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \begin{bmatrix} s'_1 \\ s'_2 \\ \vdots \\ s'_n \end{bmatrix} := \begin{bmatrix} 1 & i_1 & \cdots & i_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_n & \cdots & i_n^{n-1} \end{bmatrix} \begin{bmatrix} m' \\ a'_1 \\ \vdots \\ a'_t \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} + \begin{bmatrix} s'_1 \\ s'_2 \\ \vdots \\ s'_n \end{bmatrix} = \begin{bmatrix} 1 & i_1 & \cdots & i_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_n & \cdots & i_n^{n-1} \end{bmatrix} \begin{bmatrix} m \\ a_1 \\ \vdots \\ a_t \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & i_1 & \cdots & i_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_n & \cdots & i_n^{n-1} \end{bmatrix} \begin{bmatrix} m' \\ a'_1 \\ \vdots \\ a'_t \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & i_1 & \cdots & i_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_n & \cdots & i_n^{n-1} \end{bmatrix} \begin{bmatrix} m + m' \\ a_1 + a'_1 \\ \vdots \\ a_t + a'_t \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Linear Secret Sharing Schemes

Now Suppose we add together sharings of two different values. What happens?

$$\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} + \begin{bmatrix} s'_1 \\ s'_2 \\ \vdots \\ s'_n \end{bmatrix} = \begin{bmatrix} 1 & i_1 & \cdots & i_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_n & \cdots & i_n^{n-1} \end{bmatrix} \begin{bmatrix} m \\ a_1 \\ \vdots \\ a_t \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & i_1 & \cdots & i_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_n & \cdots & i_n^{n-1} \end{bmatrix} \begin{bmatrix} m' \\ a'_1 \\ \vdots \\ a'_t \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & i_1 & \cdots & i_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & i_n & \cdots & i_n^{n-1} \end{bmatrix} \begin{bmatrix} m+m' \\ a_1+a'_1 \\ \vdots \\ a_t+a'_t \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

We have a secret sharing of the sum of the values.

Observation: if a and a' are uniformly distributed in $\mathbb{F}$, then $a + a'$ is too.

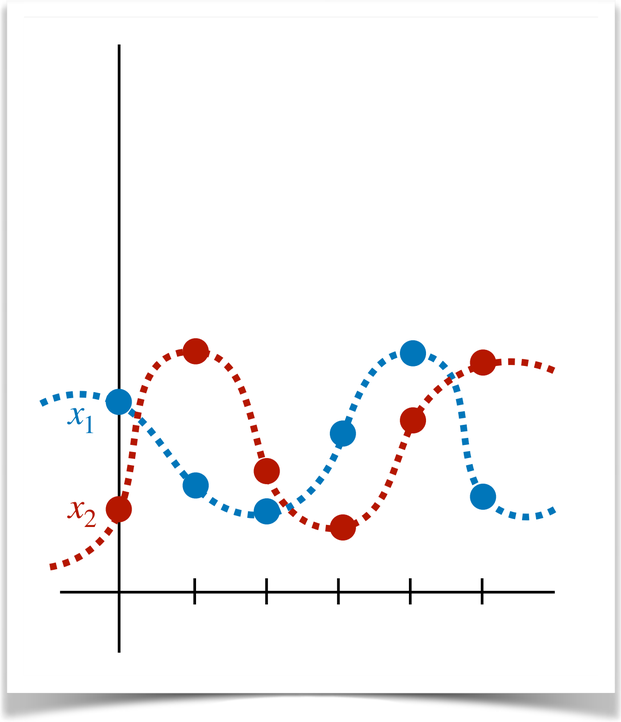
Thus if we have $f \leftarrow \mathscr{P}_{p,t,m}$ and $f' \leftarrow \mathscr{P}_{p,t,m'}$ then $g = f + f'$ is uniform in $\mathscr{P}_{p,t,(m+m')}$.

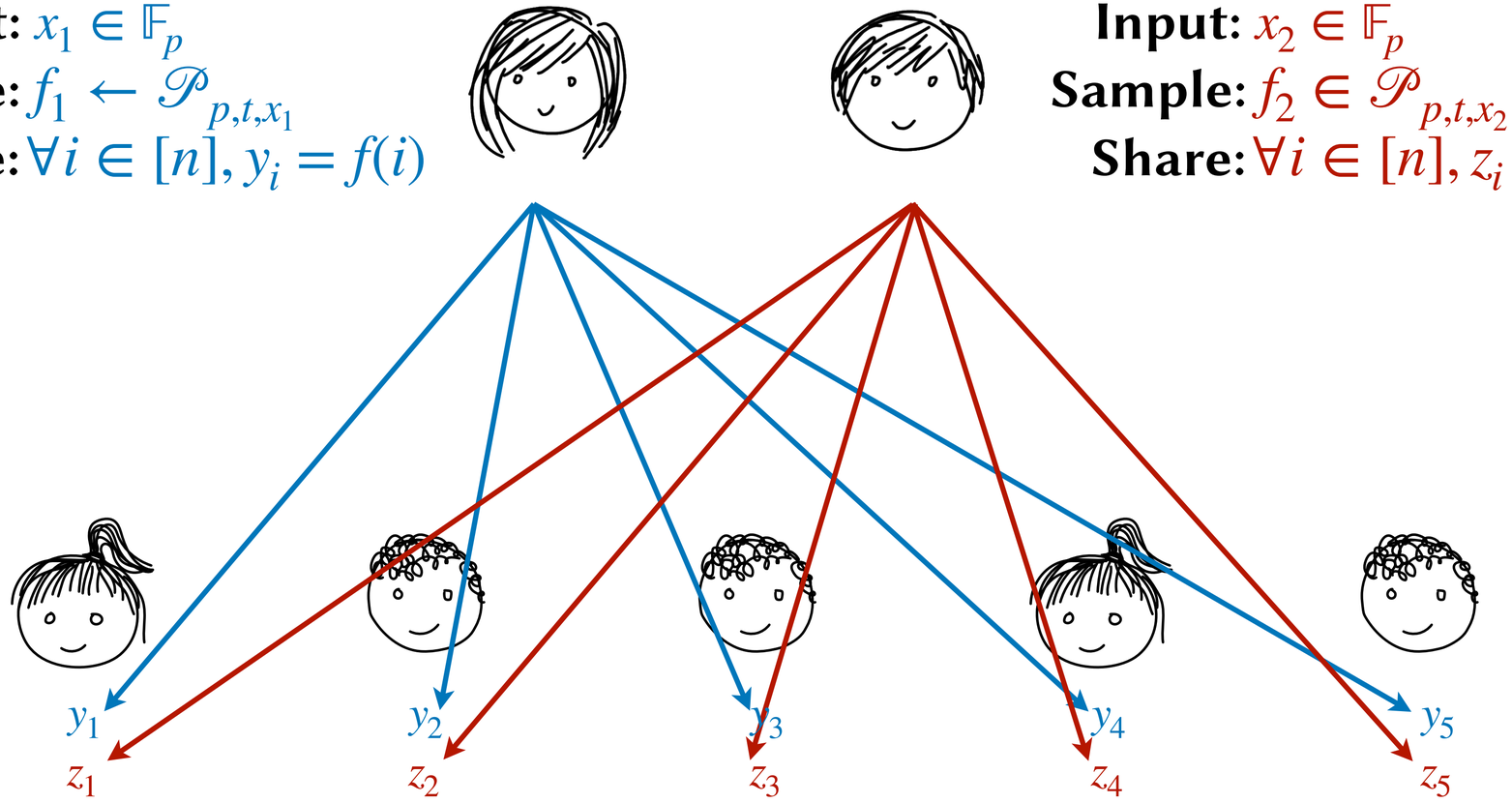
It fulfills our privacy definition! Notice that there is a correlation, between, g, f, f' though…

Two Dealers Distribute a Sum…

Input: $x_1 \in \mathbb{F}_p$
Sample: $f_1 \leftarrow \mathscr{P}_{p,t,x_1}$
Share: $\forall i \in [n], y_i = f(i)$

Input: $x_2 \in \mathbb{F}_p$
Sample: $f_2 \in \mathscr{P}_{p,t,x_2}$
Share: $\forall i \in [n], z_i = f(i)$

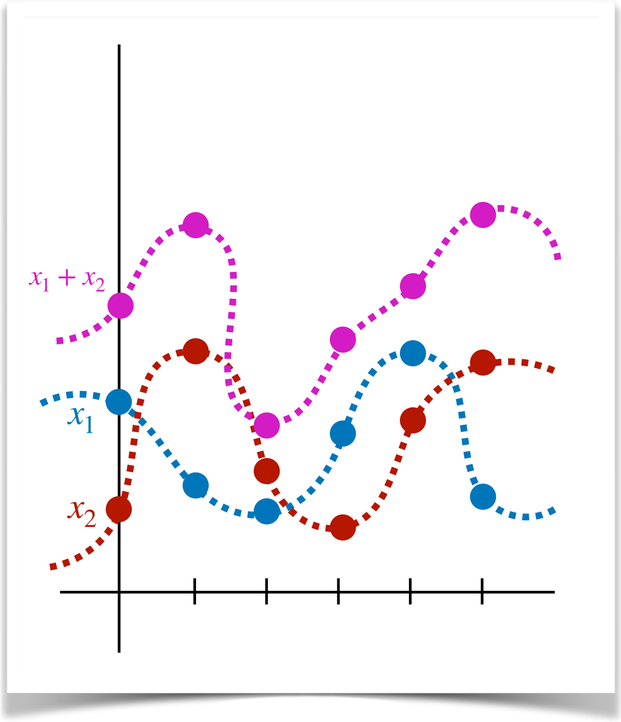




Two Dealers Distribute a Sum...

Input: $x_1 \in \mathbb{F}_p$
Sample: $f_1 \leftarrow \mathscr{P}_{p,t,x_1}$
Share: $\forall i \in [n], y_i = f(i)$

Input: $x_2 \in \mathbb{F}_p$
Sample: $f_2 \in \mathscr{P}_{p,t,x_2}$
Share: $\forall i \in [n], z_i = f(i)$



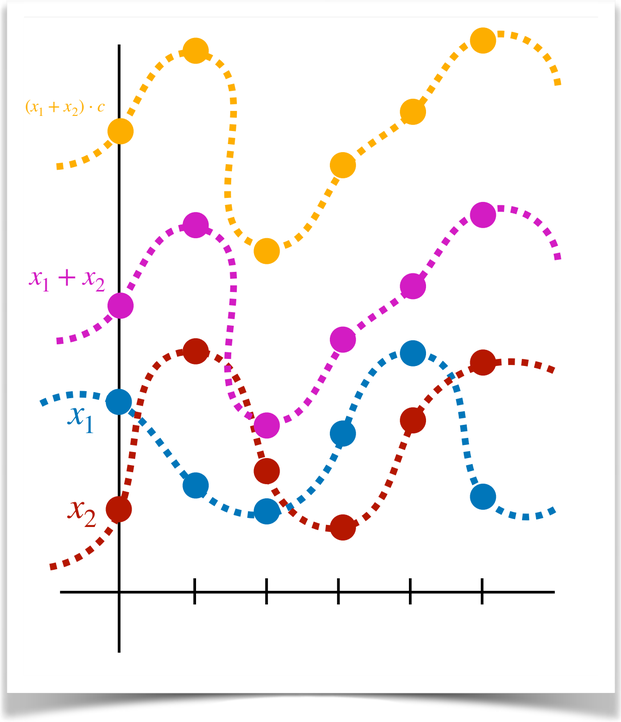
The parties can compute a sharing of the sum *without interacting*!
(no interaction $\Longrightarrow$ they cannot possibly learn anything new)

y_1	y_2	y_3	y_4	y_5
z_1	z_2	z_3	z_4	z_5
$w_1 = y_1 + z_1$	$w_2 = y_2 + z_2$	$w_3 = y_3 + z_3$	$w_4 = y_4 + z_4$	$w_5 = y_5 + z_5$

Two Dealers Distribute a Sum...

Input: $x_1 \in \mathbb{F}_p$
Sample: $f_1 \leftarrow \mathscr{P}_{p,t,x_1}$
Share: $\forall i \in [n], y_i = f(i)$

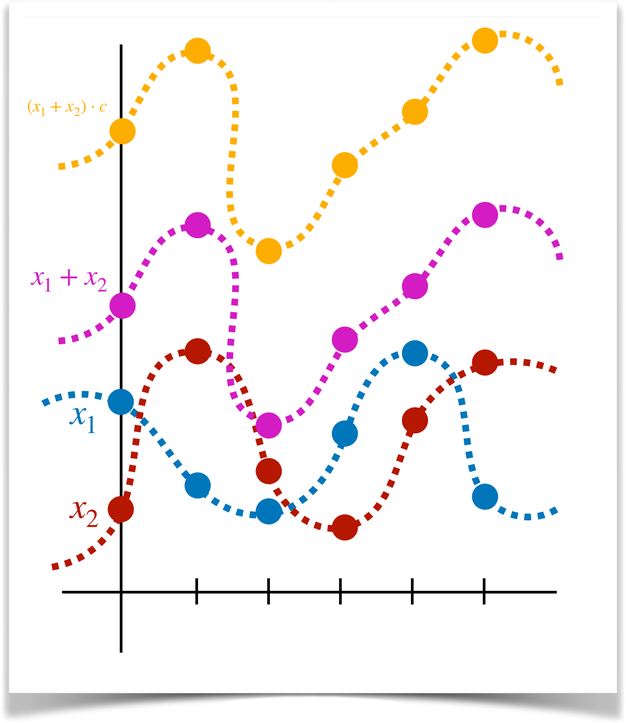
Input: $x_2 \in \mathbb{F}_p$
Sample: $f_2 \in \mathscr{P}_{p,t,x_2}$
Share: $\forall i \in [n], z_i = f(i)$



y_1	y_2	y_3	y_4	y_5
z_1	z_2	z_3	z_4	z_5
$w_1 = y_1 + z_1$	$w_2 = y_2 + z_2$	$w_3 = y_3 + z_3$	$w_4 = y_4 + z_4$	$w_5 = y_5 + z_5$
c	c	c	c	c
$v_1 = c \cdot w_1$	$v_2 = c \cdot w_2$	$v_3 = c \cdot w_3$	$v_4 = c \cdot w_4$	$v_5 = c \cdot w_5$

Because they have a degree-t Shamir sharing of $x_1 + x_2$, they can perform additional operations on it! e.g. scalar multiplication by c.

Two Dealers Distribute a Sum…



y_1	y_2	y_3	y_4	y_5
z_1	z_2	z_3	z_4	z_5
$w_1 = y_1 + z_1$	$w_2 = y_2 + z_2$	$w_3 = y_3 + z_3$	$w_4 = y_4 + z_4$	$w_5 = y_5 + z_5$
c	c	c	c	c
$v_1 = c \cdot w_1$	$v_2 = c \cdot w_2$	$v_3 = c \cdot w_3$	$v_4 = c \cdot w_4$	$v_5 = c \cdot w_5$

(Example for t=2)

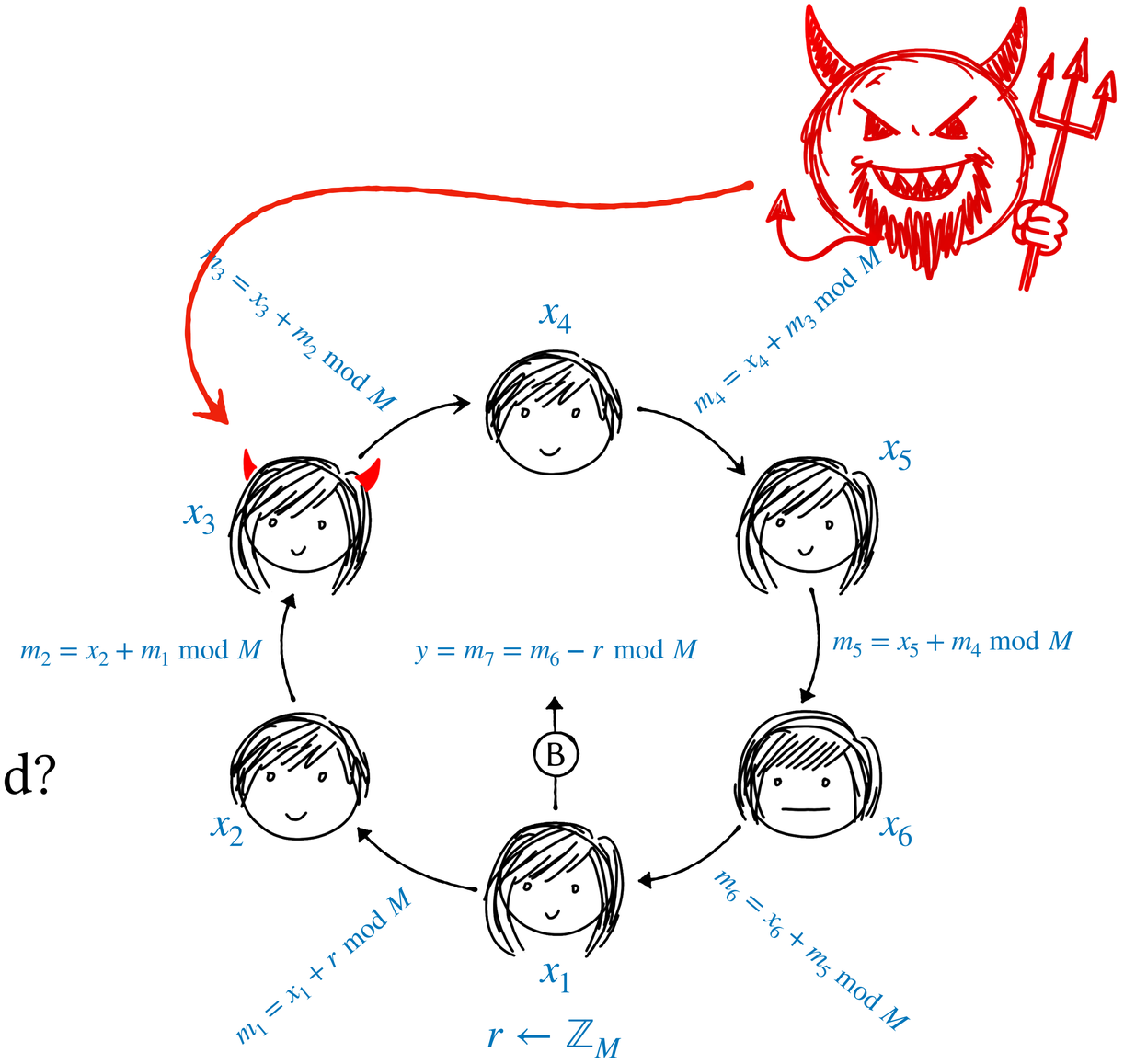
Notice that anyone who receives $t + 1$ shares shares can reconstruct $(x_1 + x_2) \cdot c$.

Since $v_1, \ldots, v_5$ correspond to a uniform member of $\mathcal{P}_{p,t,(x_1+x_2)\cdot c}$,  learns nothing about x_1, x_2 beyond $(x_1 + x_2) \cdot c$.

…even if  also knows t shares of x_1, x_2. *Why?*

Example: n-Party Sum

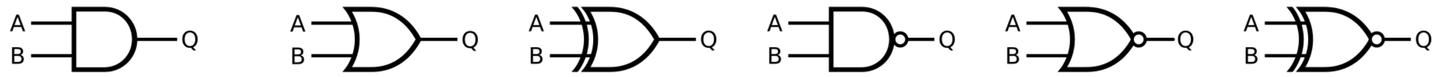
- In previous lectures we saw this protocol. We proved it was secure against one semi-honest corruption and demonstrated that it was insecure against two.
- Is there a way to achieve security against more corruptions? *How many?*
- How many *rounds of interaction* do we need?



Arithmetic Circuits:
How to Model Multi-Step Computations

Boolean Circuits

- In 1938 proved that boolean logic could be used to analyze digital computers.
- Since then the overwhelming majority of computer hardware has been digital, and the overwhelming majority of theory has concerned boolean circuits (or other models of equivalent expressive power, such as Turing machines).
- There are 2^4 truth tables for a 2-ary boolean logic gate. Of these, 2 have constant output (0,1) and 4 depend upon only one input wire (e.g. NOT). The other 10 are:
  AND, OR, XOR, NAND, NOR, XNOR, …



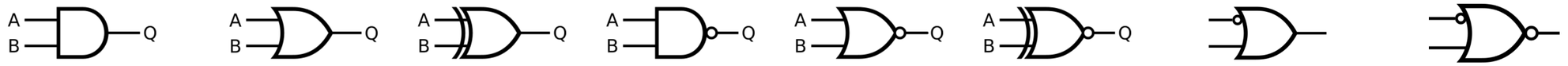
Quiz: what gates have these truth tables?

Input		Output
A	B	Q
0	0	1
0	1	1
1	0	0
1	1	1

Input		Output
A	B	Q
0	0	0
0	1	0
1	0	1
1	1	0

Boolean Circuits

- In 1938 proved that boolean logic could be used to analyze digital computers.
- Since then the overwhelming majority of computer hardware has been digital, and the overwhelming majority of theory has concerned boolean circuits (or other models of equivalent expressive power, such as Turing machines).
- There are 2^4 truth tables for a 2-ary boolean logic gate. Of these, 2 have constant output (0,1) and 4 depend upon only one input wire (e.g. NOT). The other 10 are:
  AND, OR, XOR, NAND, NOR, XNOR, IMPLY(×2), NIMPLY(×2)



- We use gates to build *circuits*. A circuit is a directed *acyclic* graph. The edges are *wires* and the nodes are *gates*, *inputs*, or *outputs*. Usually we allow *fan-out:* the output of a gate can connect to many inputs. Evaluation happens in topological order.

Note: Sorry hardware people - latches are forbidden here because they are cyclic.

Boolean Circuits

Every boolean function $f : \{0,1\}^n \to \{0,1\}^m$ can be represented as a boolean circuit. Let $x_1, \ldots, x_n$ be the n input wires and $z_1, \ldots, z_m$ be the m output wires.

1. For every $j \in [m]$ write down the truth table with respect to z_j. There are 2^n rows. Each row k maps some assignment of $x_1^k, \ldots, x_n^k$ to some assignment of z_j^k.

2. If row k has $z_j^k = 1$, then let $S_j^k = \{i \in [n] : x_i^k = 1\}$, $T_j^k = \{i \in [n] : x_i^k = 0\}$, and $C_j^k(x_1, \ldots, x_n) = \bigwedge_{i \in S_j^k} x_i \bigwedge_{i \in T_j^k} \neg x_i$. If row k has $z_j^k = 0$, then let $C_j^k(x_1, \ldots, x_n) = 0$.

3. Now you can compute $z_j = \bigvee_{k \in [2^n]} C_j^k(x_1, \ldots, x_n)$. This is *Disjunctive Normal Form.*

Note: The size of the circuit we define this way might be exponential!

Boolean Circuits

Every boolean function $f : \{0,1\}^n \to \{0,1\}^m$ can be represented as a boolean circuit. Let $x_1, \ldots, x_n$ be the n input wires and $z_1, \ldots, z_m$ be the m output wires.

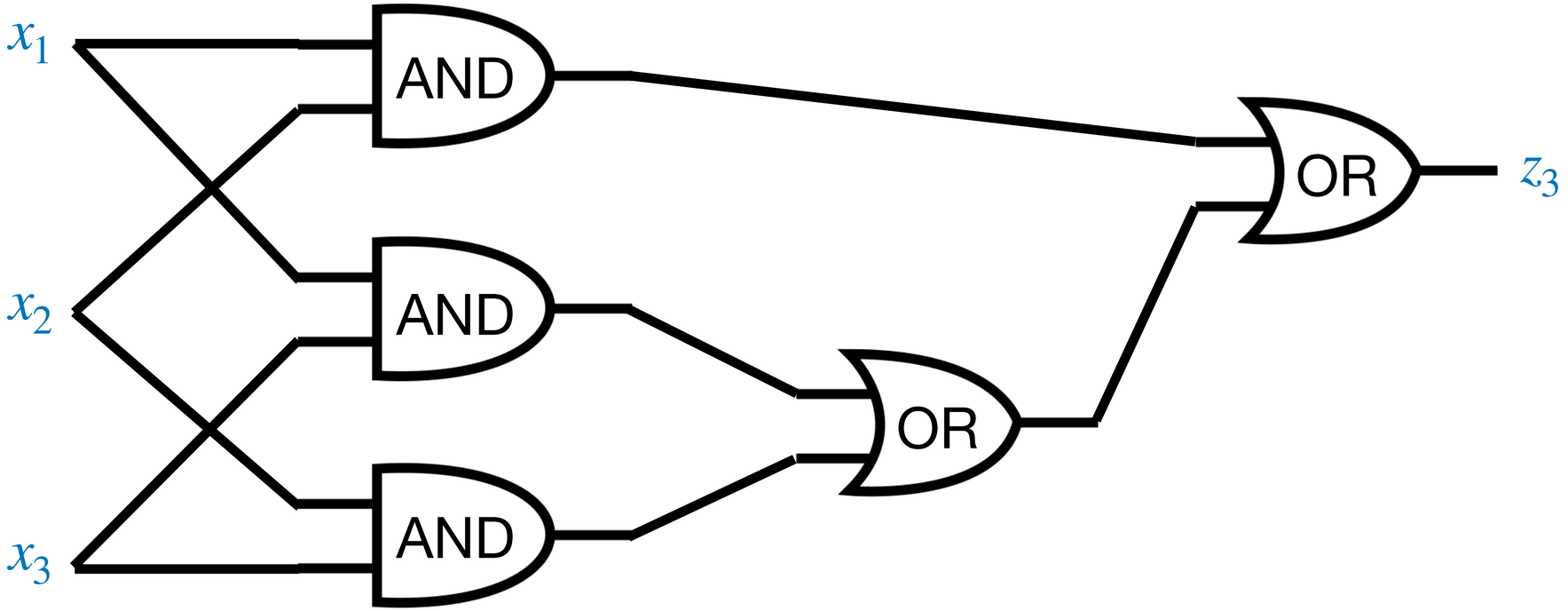
1. For every $j \in [m]$ write down the truth table with respect to z_j. There are 2^n rows. Each row k maps some assignment of $x_1^k, \ldots, x_n^k$ to some assignment of z_j^k.

2. If row k has $z_j^k = 1$, then let $S_j^k = \{i \in [n] : x_i^k = 1\}$, $T_j^k = \{i \in [n] : x_i^k = 0\}$, and $C_j^k(x_1, \ldots, x_n) = \bigwedge_{i \in S_j^k} x_i \bigwedge_{i \in T_j^k} \neg x_i$. If row k has $z_j^k = 0$, then let $C_j^k(x_1, \ldots, x_n) = 0$.

3. Now you can compute $z_j = \bigvee_{k \in [2^n]} C_j^k(x_1, \ldots, x_n)$. This is *Disjunctive Normal Form.*

Any set of boolean gates that can be used to express all functions is called *complete.* Above we used ($\wedge$, $\vee$, $\neg$) but other combinations work, such as ($\wedge$, $\oplus$, 1).

(**Small Quiz**: What is this Circuit?)



Answer: *2-of-3 threshold.*

Arithmetic Circuits

- In this class, we will think about *arithmetic circuits* over $\mathbb{F}_p$. Each wire contains a value from $\mathbb{F}_p$. Each gate is a 2-ary function $\mathbb{F}_p^2 \to \mathbb{F}_p$. There are p^{p^2} possible gates!
- In order represent any n-ary function $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$ as an *arithmetic circuit*, we need a *complete* set of gates for $\mathbb{F}_p$. Any idea which ones?
- Since $(\cdot, +)$ are the fundamental operations on $\mathbb{F}_p$, it had better be those! We also need a constant (for algebraists: not every possible input includes a generator).

Claim: any function $f : \mathbb{F}_p^n \to \mathbb{F}_p^m$ can be expressed using $(\cdot, +, 1)$. Can you see how?

Pf Sketch: To compute the j^{th} output $z_j \in \mathbb{F}_p$, write the truth table and find the *multivariate* Lagrange polynomial that passes through the points defined by the rows of the truth table. This Lagrange polynomial computes f using $(\cdot, +)$ and constants.

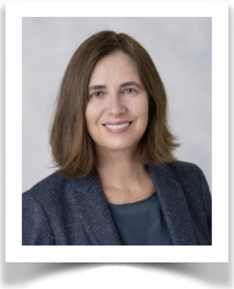
Note: The degree of the polynomial we define this way might be exponential!

Generalization Note

Note that boolean circuits correspond to arithmetic circuits over $\mathbb{F}_p$ with $p = 2$.
$\wedge$ is equivalent to $\cdot$ in $\mathbb{F}_2$, and $\oplus$ is equivalent to $+$.

Putting the Pieces Together

A First Look at the BGW Protocol


- First described by Ben-Or, Goldwasser, Widgerson, 1988
- Securely computes arithmetic circuits over a finite field
- Achieves perfect security…
  - …against an unbounded semi-honest $\mathscr{A}$ statically corrupting up to $t < n/2$ parties.
  - …against an unbounded malicious $\mathscr{A}$ statically corrupting up to $t < n/3$ parties.
  - We will see later that this is the best you can do if you want perfect security!
- Serves as the basis for more advanced protocols that handle adaptive corruption.
- Number of rounds grows with multiplicative depth of the function computed.

A First Look at the BGW Protocol

Let's Consider a Simplified Setting:

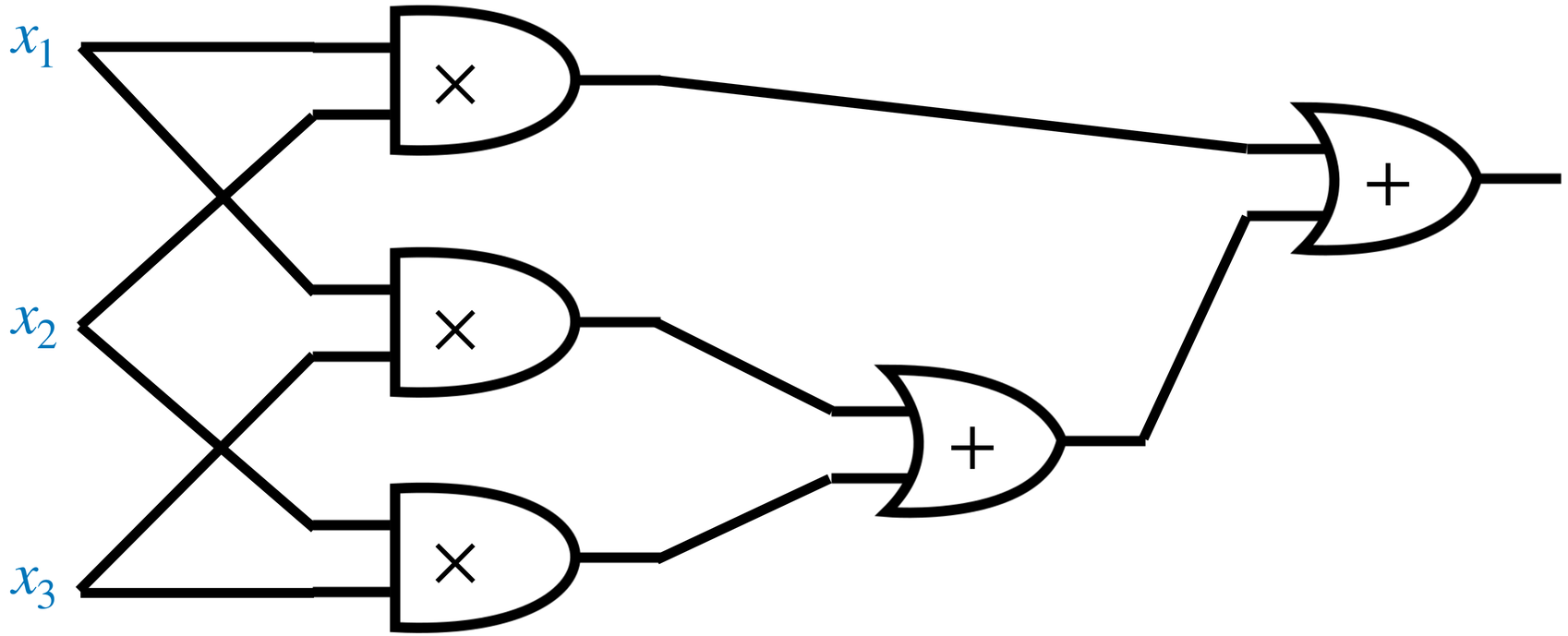
- Let $p > n > t$ be integers. We wish to compute a public circuit C representing a deterministic n-ary function $f : \mathbb{F}_p^n \to \mathbb{F}_p$, and give all parties the same output z.

- We will assume every pair of parties can communicate over a secure (private and authenticated) channel, and that their communication is *synchronous* (i.e. it proceeds in rounds and everyone knows when a round starts and ends).

Notation: for any $a \in \mathbb{F}_p$, let $\langle a \rangle$ denote an entire Shamir sharing of a, and let $\langle a \rangle_i$ denote the i^{th} share. That is, let $\langle a \rangle = (\langle a \rangle_1, \ldots, \langle a \rangle_n) \leftarrow \mathsf{Share}_{p,n,t}(a)$. Notice that these are random variables even if a is fixed!

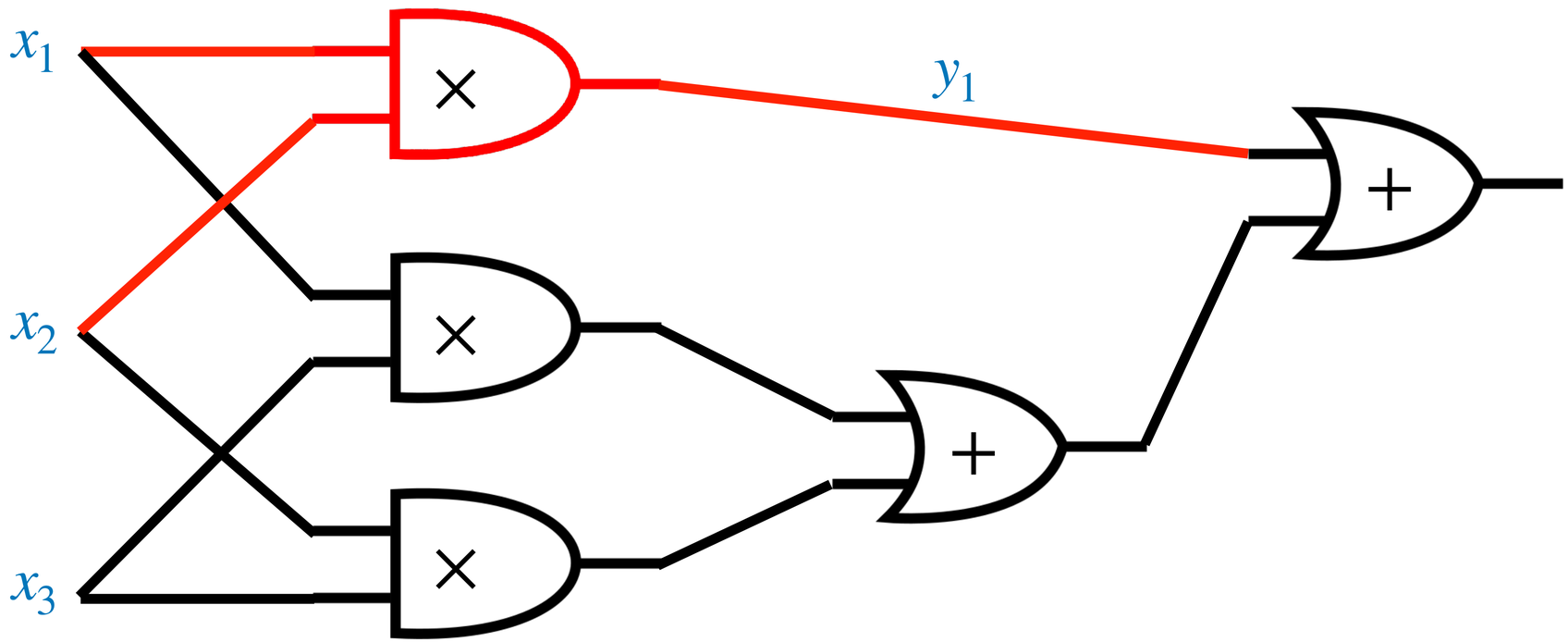
A First Look at the BGW Protocol

A Trusted Third Party who knew all the inputs would evaluate the circuit one gate at a time in topological order starting from the inputs.



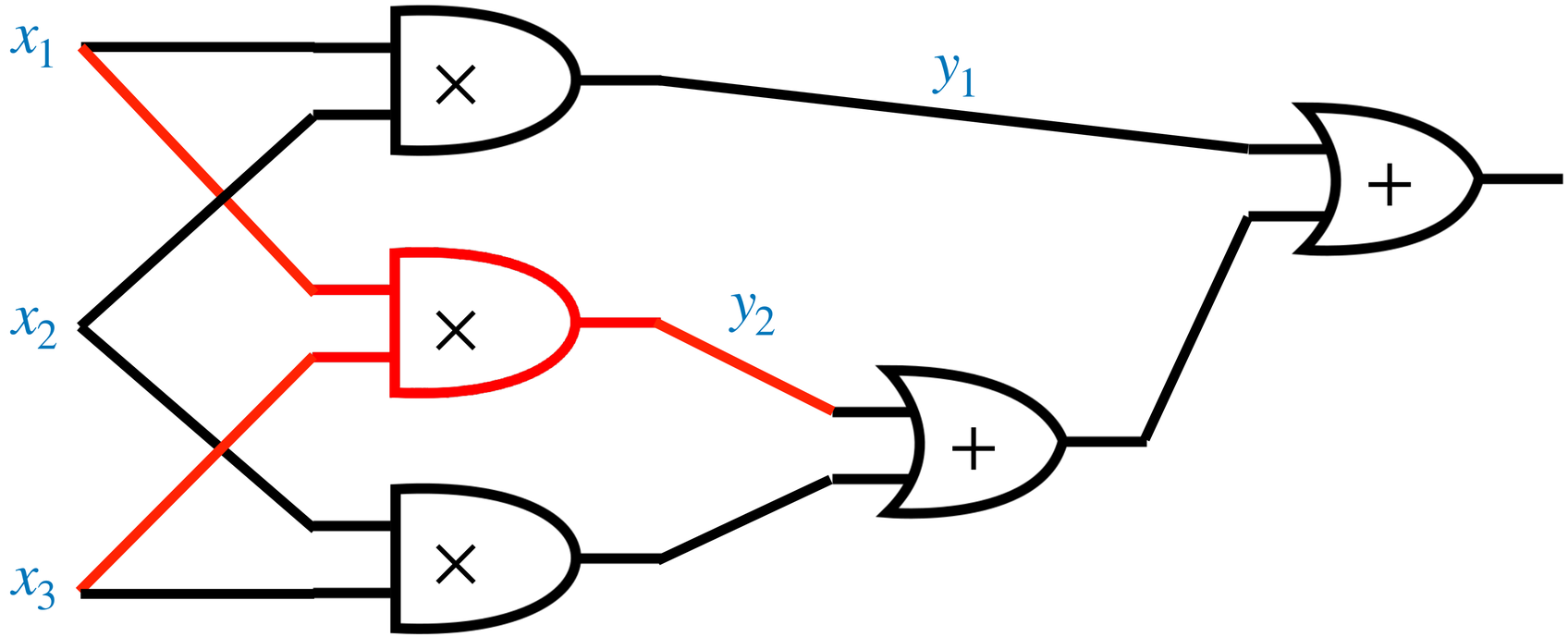
A First Look at the BGW Protocol

A Trusted Third Party who knew all the inputs would evaluate the circuit one gate at a time in topological order starting from the inputs.



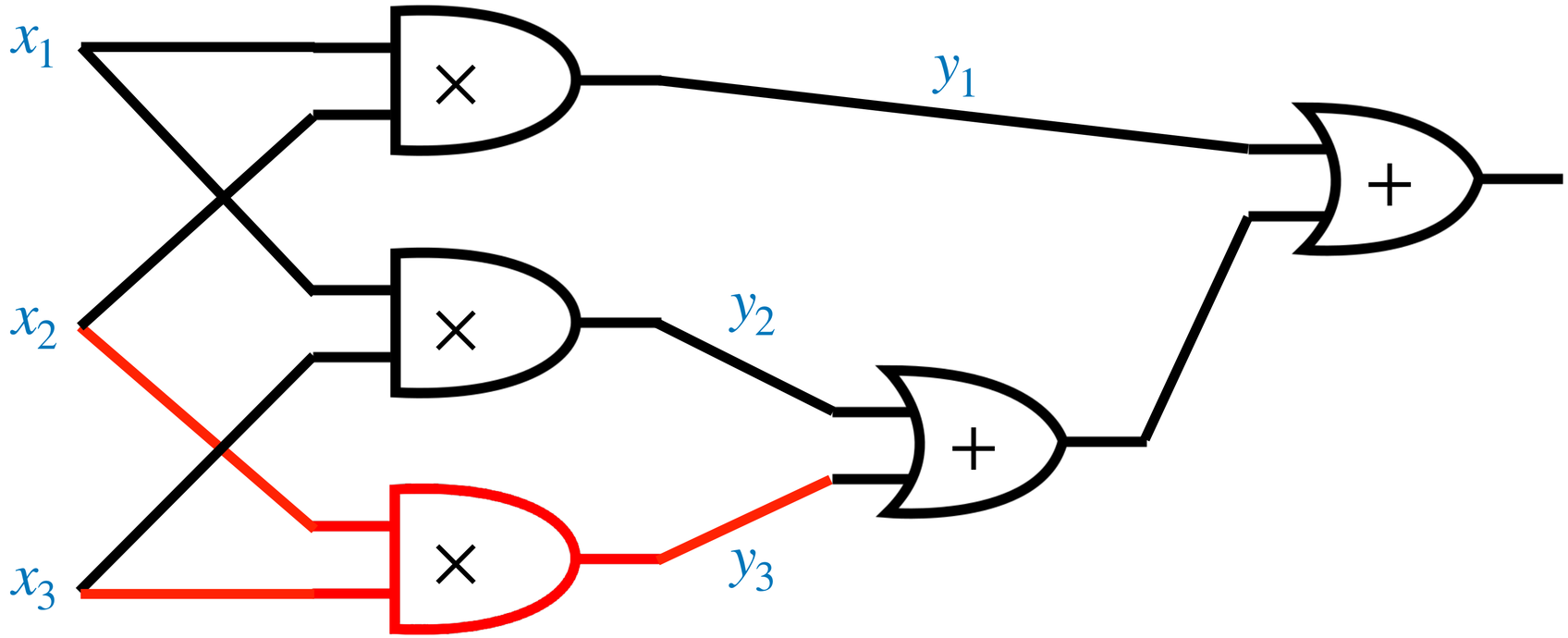
A First Look at the BGW Protocol

A Trusted Third Party who knew all the inputs would evaluate the circuit one gate at a time in topological order starting from the inputs.



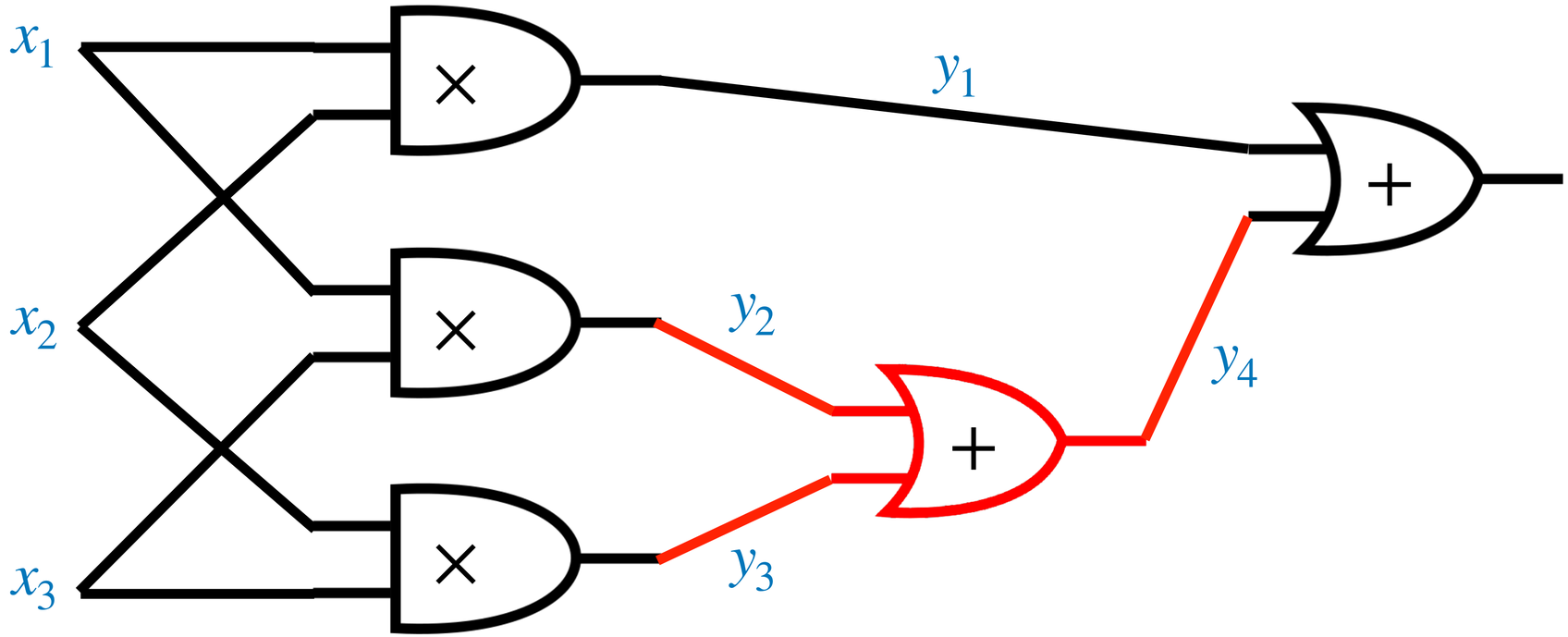
A First Look at the BGW Protocol

A Trusted Third Party who knew all the inputs would evaluate the circuit one gate at a time in topological order starting from the inputs.



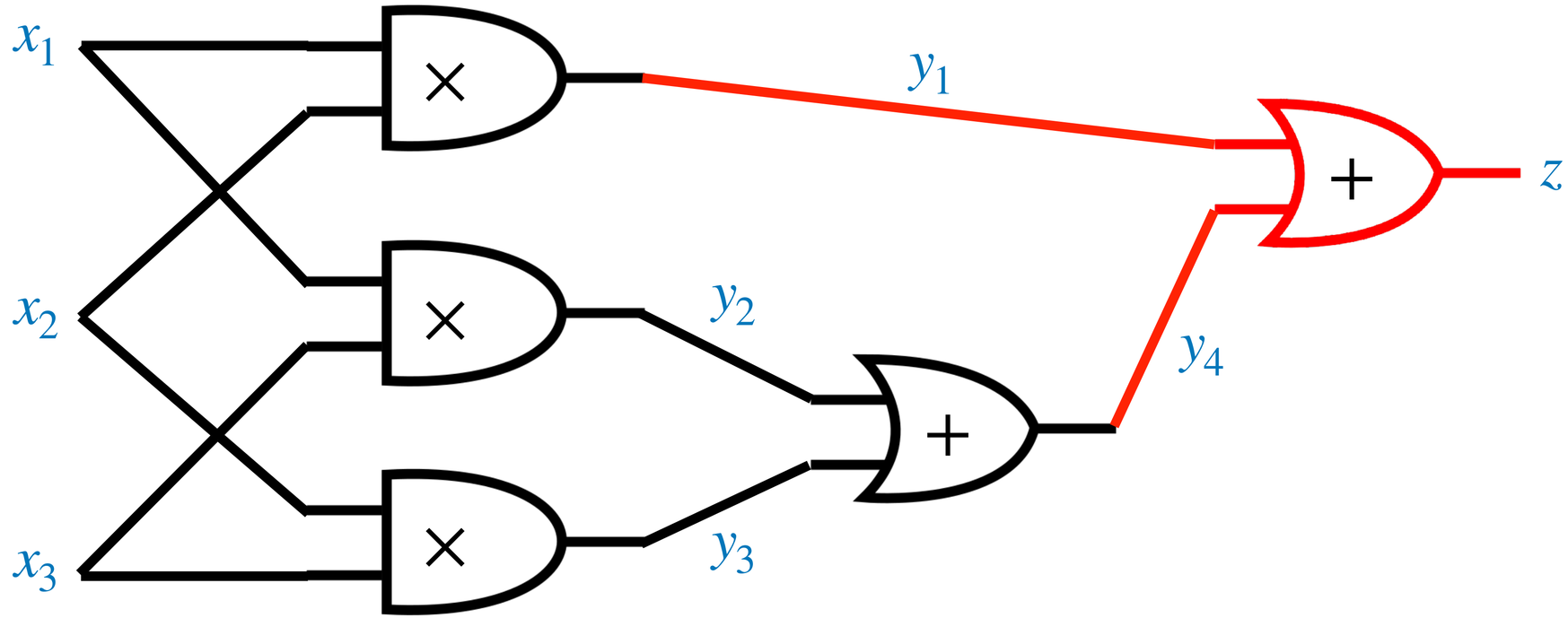
A First Look at the BGW Protocol

A Trusted Third Party who knew all the inputs would evaluate the circuit one gate at a time in topological order starting from the inputs.



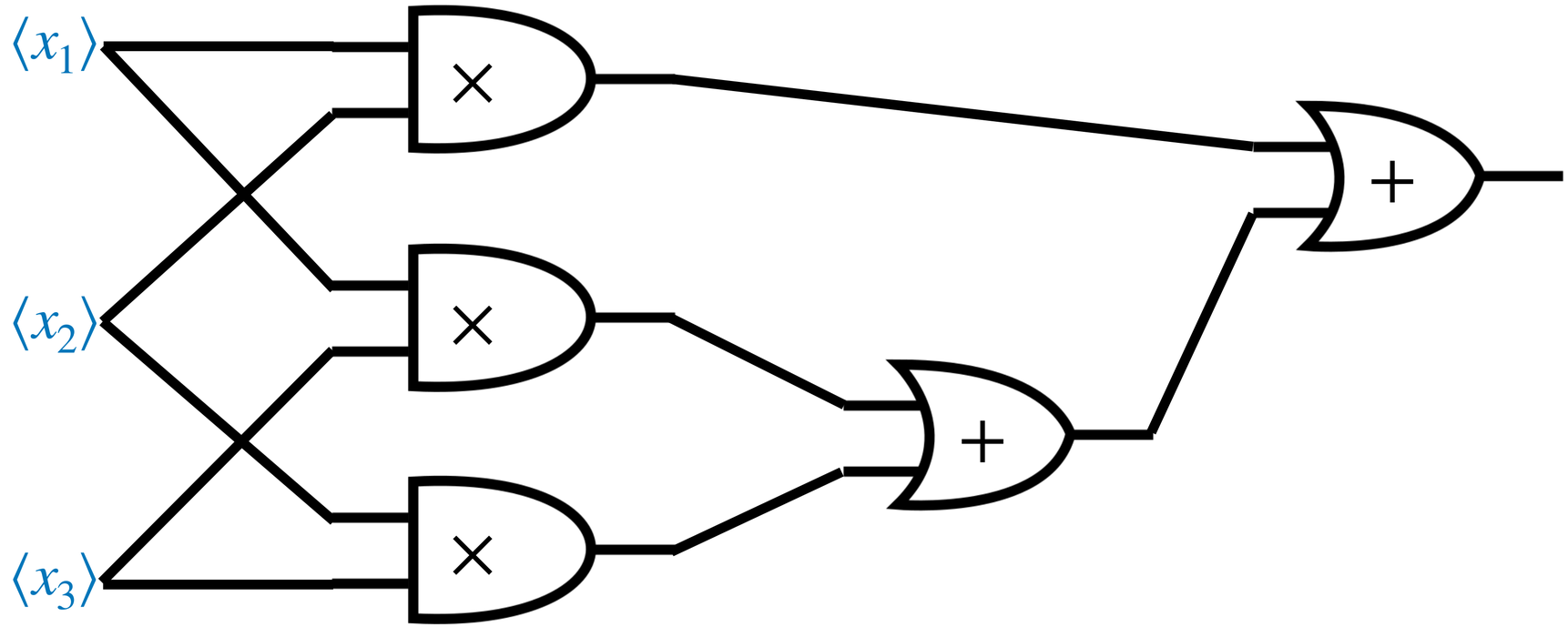
A First Look at the BGW Protocol

A Trusted Third Party who knew all the inputs would evaluate the circuit one gate at a time in topological order starting from the inputs.



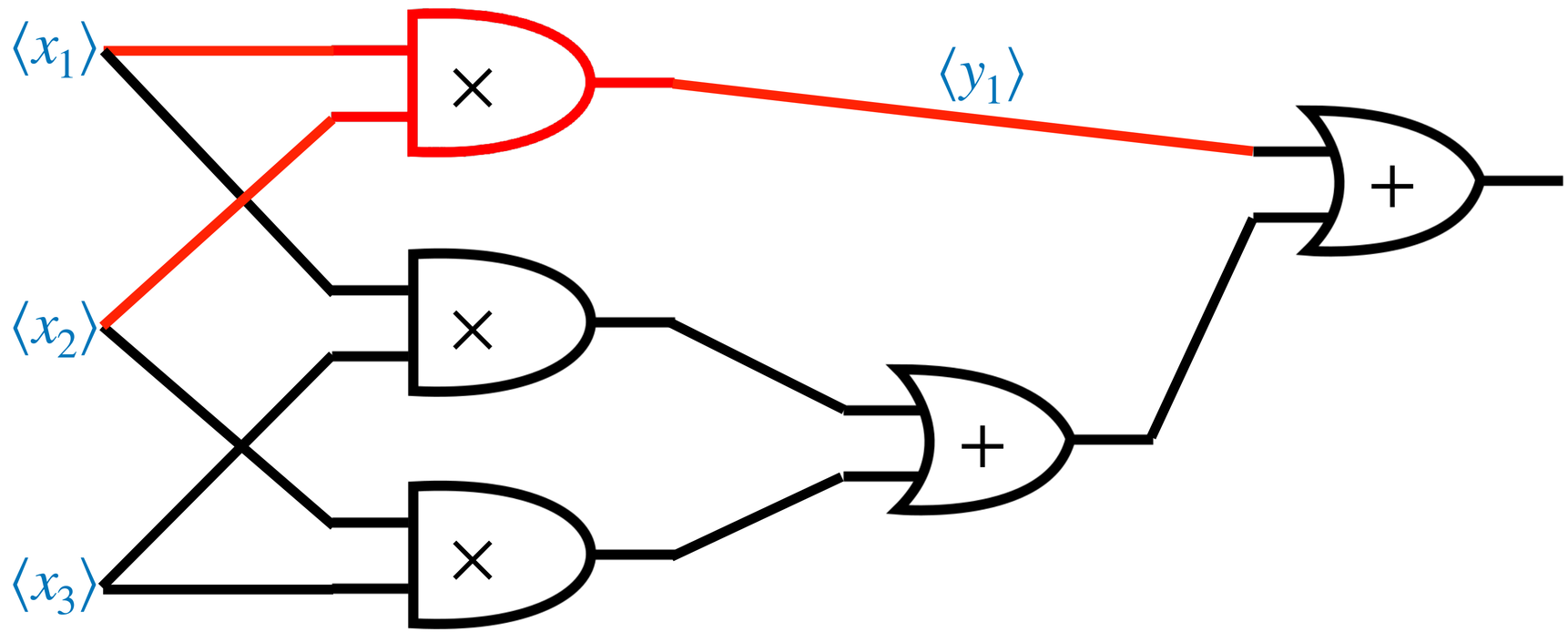
A First Look at the BGW Protocol

The BGW Protocol proceeds exactly like this, except that all wire values are *Shamir-shared*. Each party knows one *share* of the value on each wire.



A First Look at the BGW Protocol

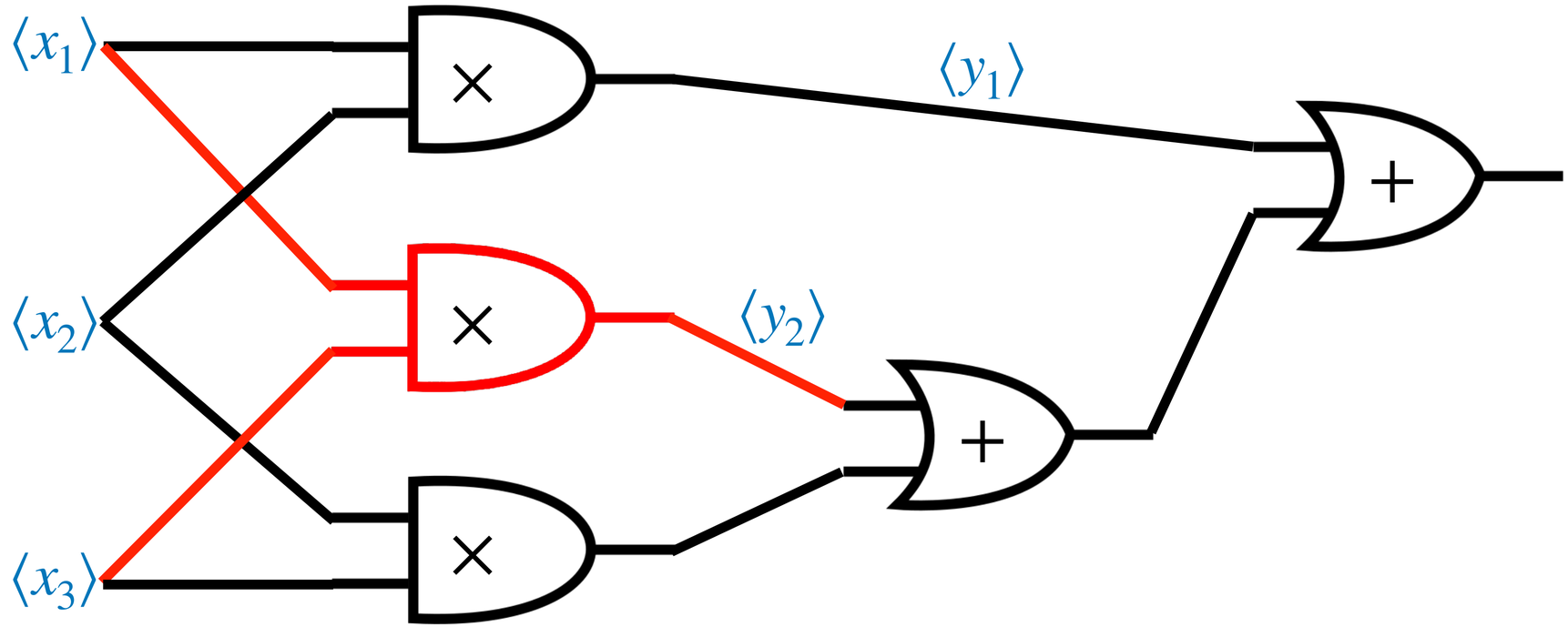
The BGW Protocol proceeds exactly like this, except that all wire values are *Shamir-shared*. Each party knows one *share* of the value on each wire.



Each Gate operates on *shares* instead of *logical values*. Some gates involve interaction.

A First Look at the BGW Protocol

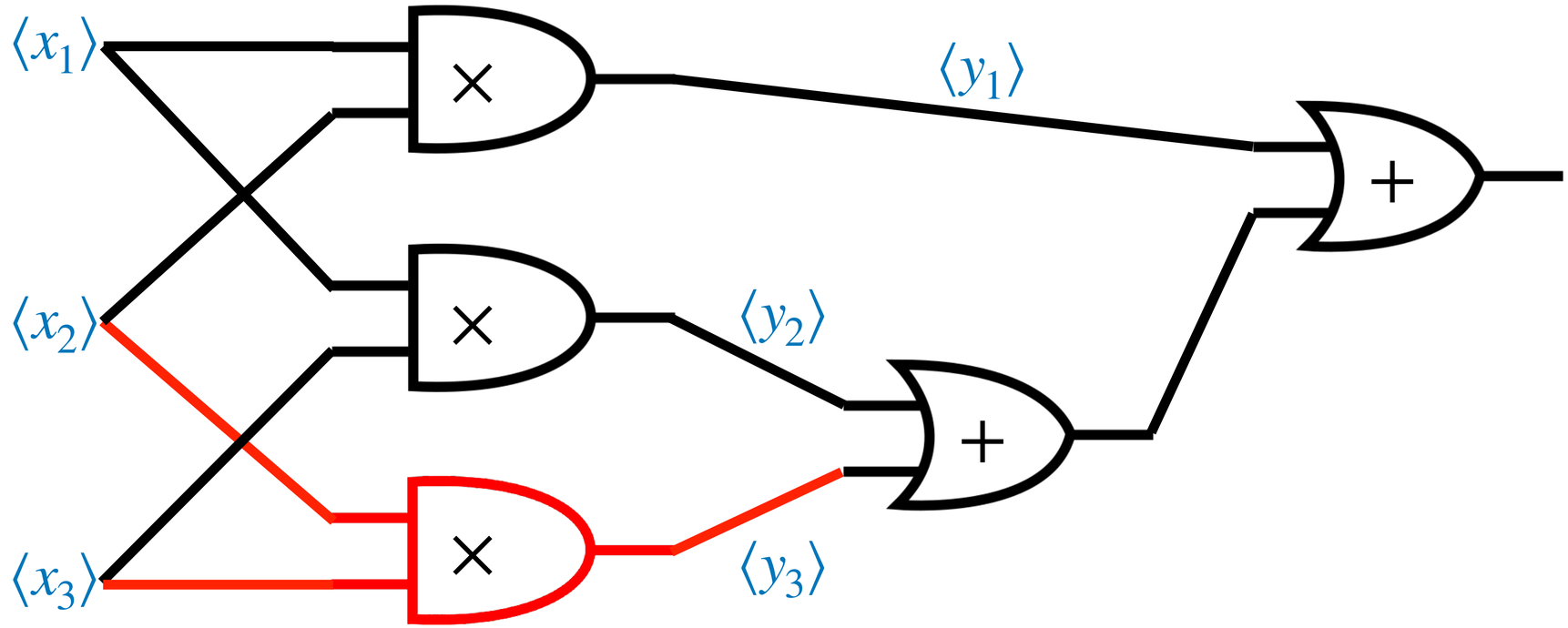
The BGW Protocol proceeds exactly like this, except that all wire values are *Shamir-shared*. Each party knows one *share* of the value on each wire.



Each Gate operates on *shares* instead of *logical values*. Some gates involve interaction.

A First Look at the BGW Protocol

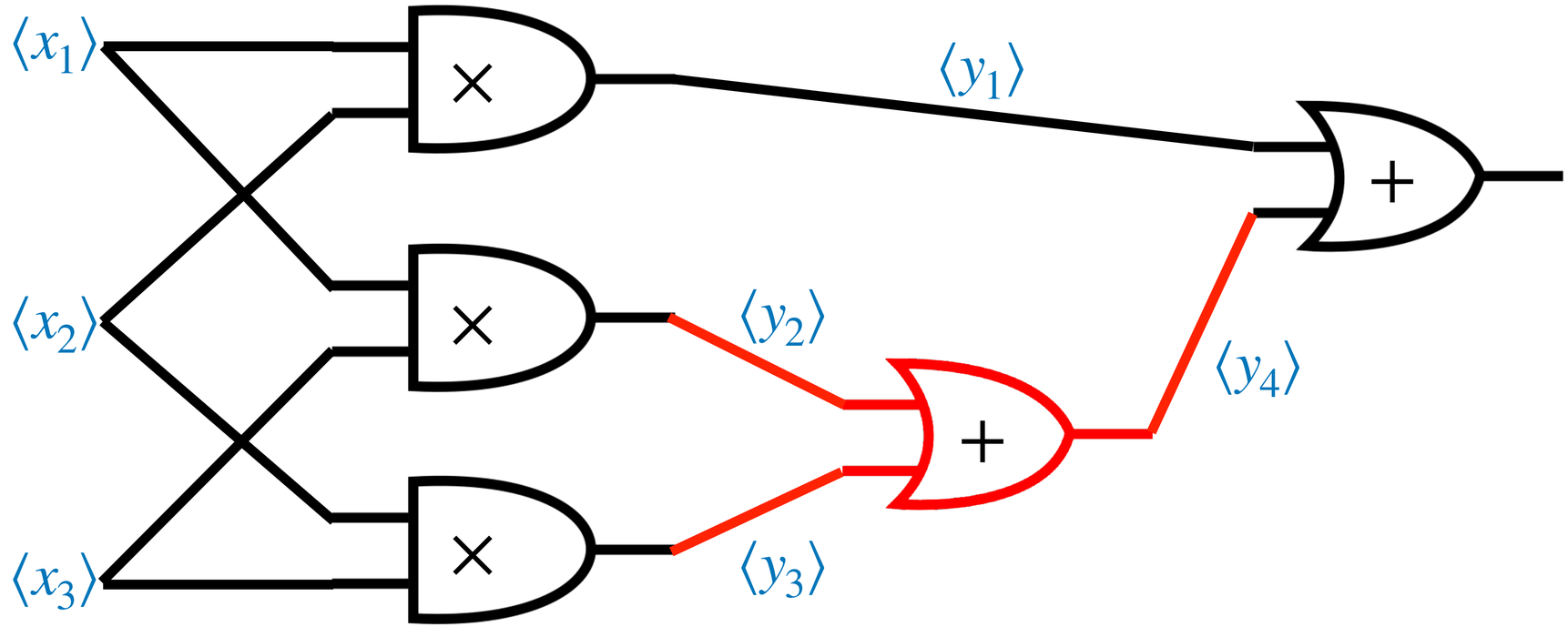
The BGW Protocol proceeds exactly like this, except that all wire values are *Shamir-shared*. Each party knows one *share* of the value on each wire.



Each Gate operates on *shares* instead of *logical values*. Some gates involve interaction.

A First Look at the BGW Protocol

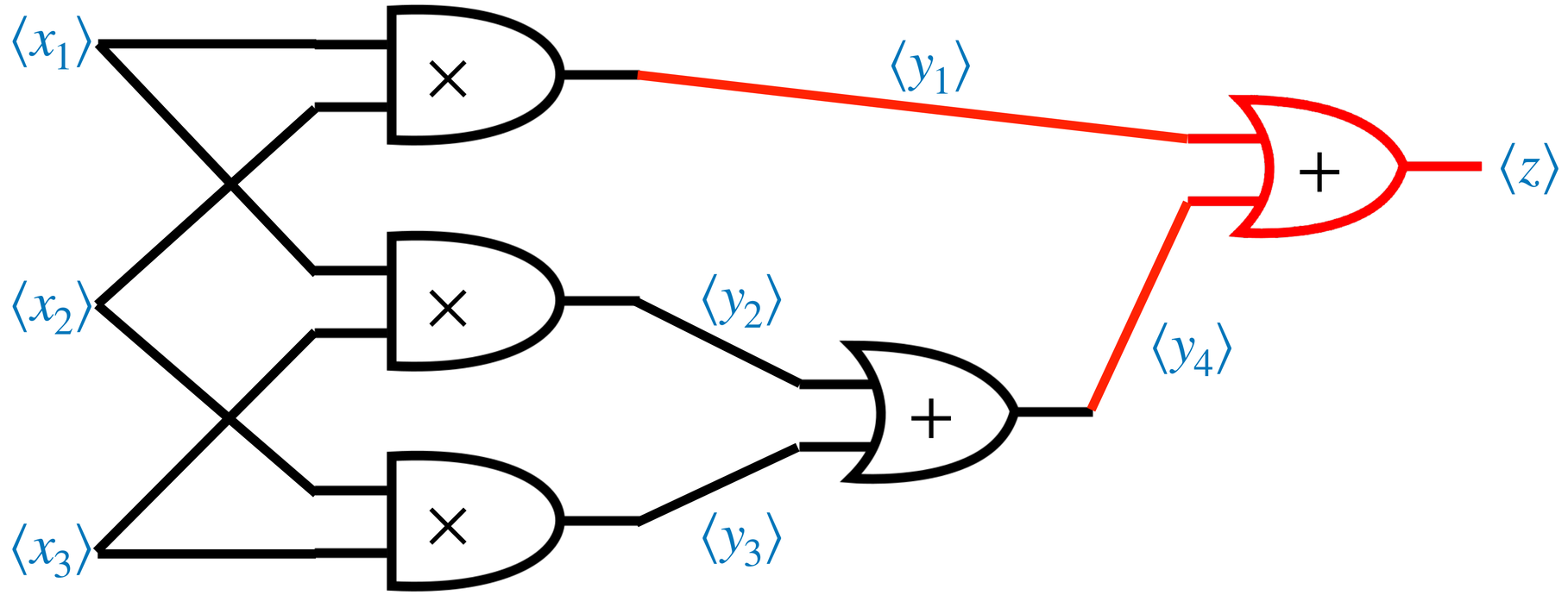
The BGW Protocol proceeds exactly like this, except that all wire values are *Shamir-shared*. Each party knows one *share* of the value on each wire.



Addition Gates can be evaluated non-interactively as we have just seen!

A First Look at the BGW Protocol

The BGW Protocol proceeds exactly like this, except that all wire values are *Shamir-shared*. Each party knows one *share* of the value on each wire.



Addition Gates can be evaluated non-interactively as we have just seen!

A First Look at the BGW Protocol

In order to complete the picture, we need to supply inputs and reveal outputs. Adding these actions to the circuit evaluation you've just seen gives us three phases:

1. Input Sharing: every P_i with input x_i computes $\langle x_i \rangle \leftarrow \mathsf{Share}_{p,n,t}(x_i)$ and sends $\langle x_i \rangle_j$ to every P_j for $j \in [n] \backslash \{i\}$.

2. Circuit Evaluation: the parties traverse the circuit C in topological order, evaluating each gate to produce shares of its output wire, until they collectively obtain $\langle z \rangle$ (i.e. each P_i learns *only* $\langle z \rangle_i$).

   If f is linear, then C contains only addition and scalar multiplication gates $\implies$ this phase is non-interactive.

3. Output Reconstruction: Each P_i sends $\langle z \rangle_i$ to all other parties. Since all parties now have *complete* knowledge of $\langle z \rangle$, they can each compute $z := \mathsf{Recon}_{p,n,t}([n], \langle z \rangle)$ and output z.

A Proof Sketch of Security for *Linear* Functions

Theorem 1: Let $p > n > t$. Assuming synchronicity and secure channels, every *linear deterministic* n-ary function $f : \mathbb{F}_p^n \to \mathbb{F}_p$ with a single output can be securely computed in the presence of a semi-honest $\mathscr{A}$ that statically corrupts up to $n - 1$ parties.

Pf Sketch: because f is deterministic and $\mathscr{A}$ is semi-honest it suffices to show *correctness* and *simulatability* individually.

Correctness follows directly from the correctness and linearity of Shamir sharing.

Simulatability comes from the following claim.

Claim 1: Let $I = \{i_1, \ldots, i_t\} \subset [n]$. Then there exists an algorithm Sim such that for every $\vec{x} \in \mathbb{F}_p^n$ we have $\mathsf{Sim}(I, x_I, f(\vec{x})) \equiv \mathsf{VIEW}_I$.

A Proof Sketch of Security for *Linear* Functions

Claim 1: Let $I = \{i_1, \ldots, i_t\} \subset [n]$. Then there exists an algorithm Sim such that for every $\vec{x} \in \mathbb{F}_p^n$ we have $\mathsf{Sim}(I, x_I, f(\vec{x})) \equiv \mathsf{VIEW}_I$.

In Phase 1: the adversary $\mathscr{A}$ receives t shares of x_i from each honest party (one share for each party it has corrupted). By the privacy property of Shamir-sharing Sim can sample identically-distributed shares without knowing the honest parties' inputs by running their code with input 0. Since $\mathscr{A}$ is semi-honest, Sim can run the corrupt parties' code normally on their inputs (which Sim knows) in order to simulate their messages.

In Phase 2: there is no interaction. Sim can predict the shares of z that the corrupt parties should obtain by running their code on the data in their views.

In Phase 3: $\mathscr{A}$ receives 1 share of z from each honest party. Sim knows z already, and it knows the shares that the corrupt parties possess from phase 2. Since these *completely fix* the honest parties shares, it can perfectly compute those shares by interpolating! ∎

Next Time: How do We Multiply?

CS4501 Cryptographic Protocols
Lecture 7: Interpolation, Linearity, Circuits

https://jackdoerner.net/teaching/#2026/Spring/CS4501