Secure Multi-Party Computation with Honest Majority

Vassilis Zikas

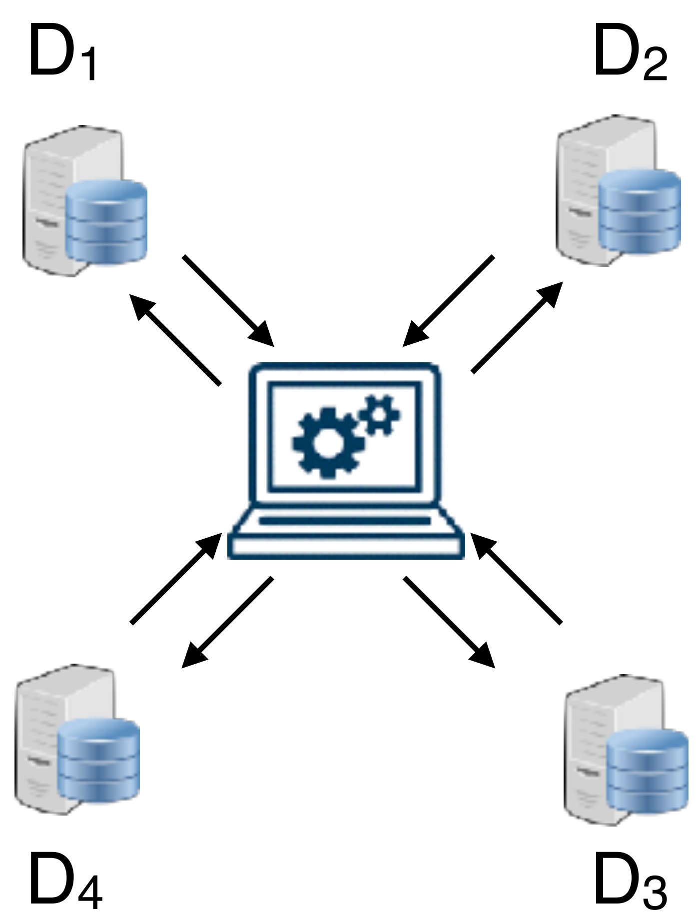
RPI

MPC School

IIT Mumbai

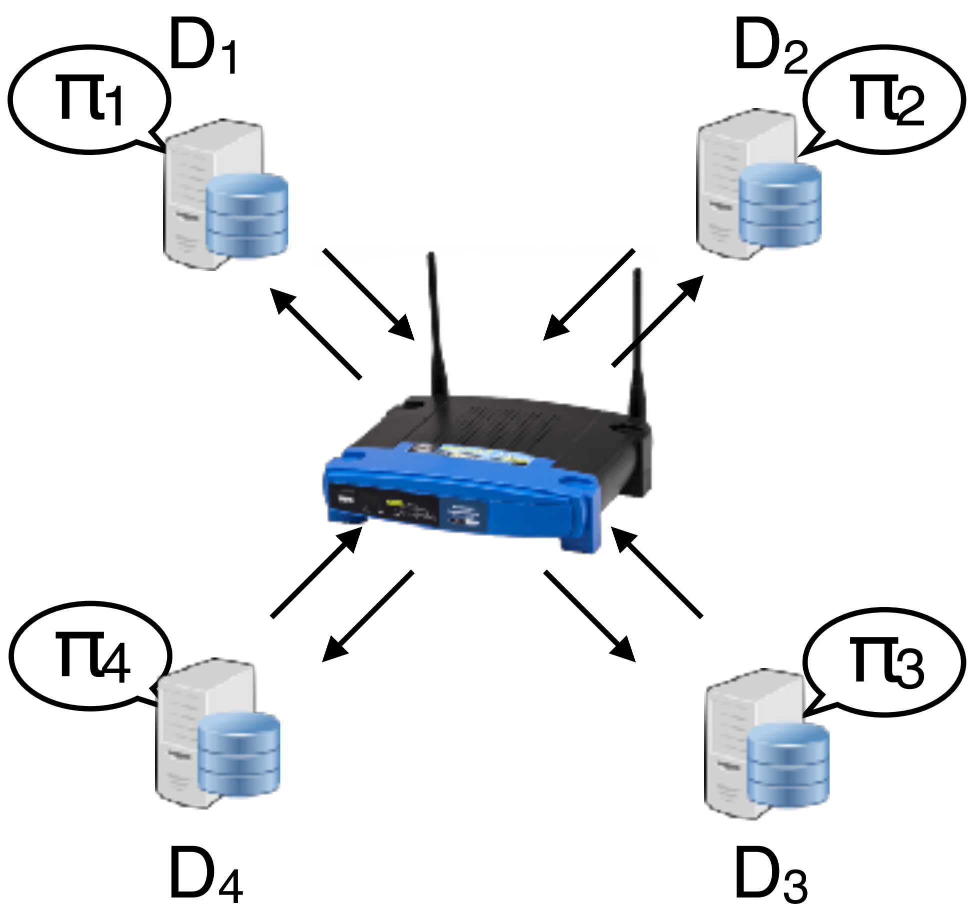
Secure Multi-Party Computation (MPC)

MPC: The general task



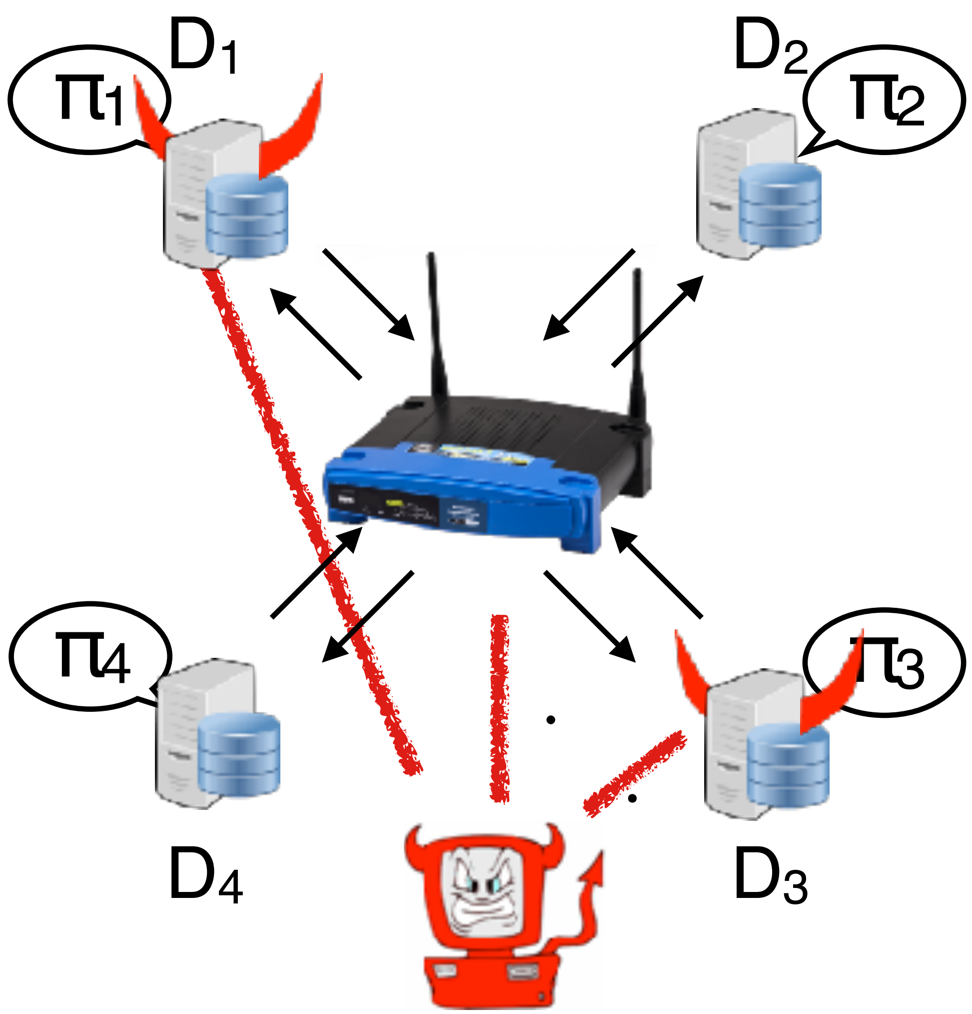
Secure Multi-Party Computation (MPC)

MPC: The general task



Secure Multi-Party Computation (MPC)

MPC: The general task



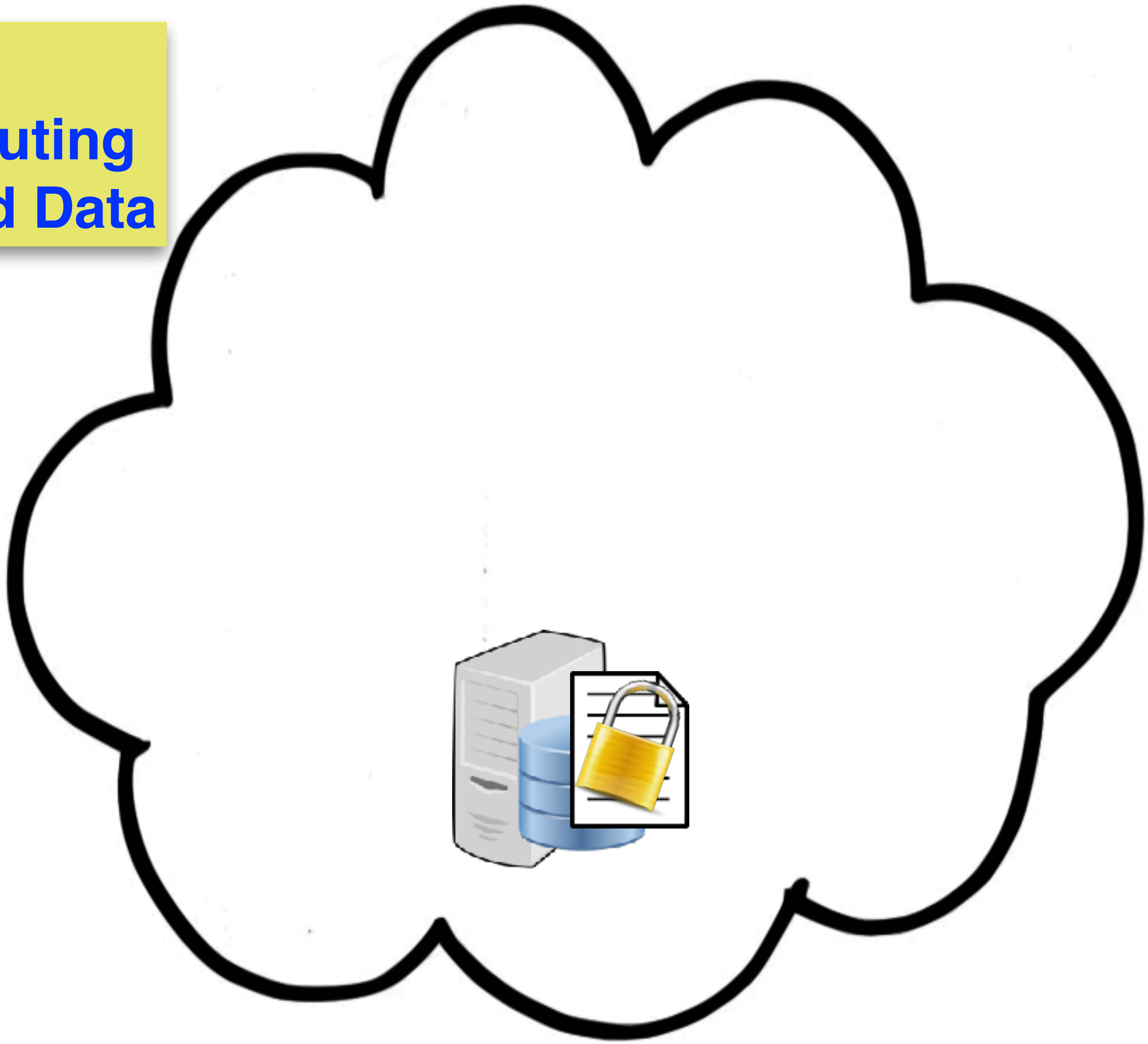
Protocol π is secure if for any such cheaters:

- *(privacy)* Whatever the adversary learns he could compute by himself
- *(correctness)* Honest (uncorrupted) parties learn their correct outputs

MPC in Action: A Toy Example

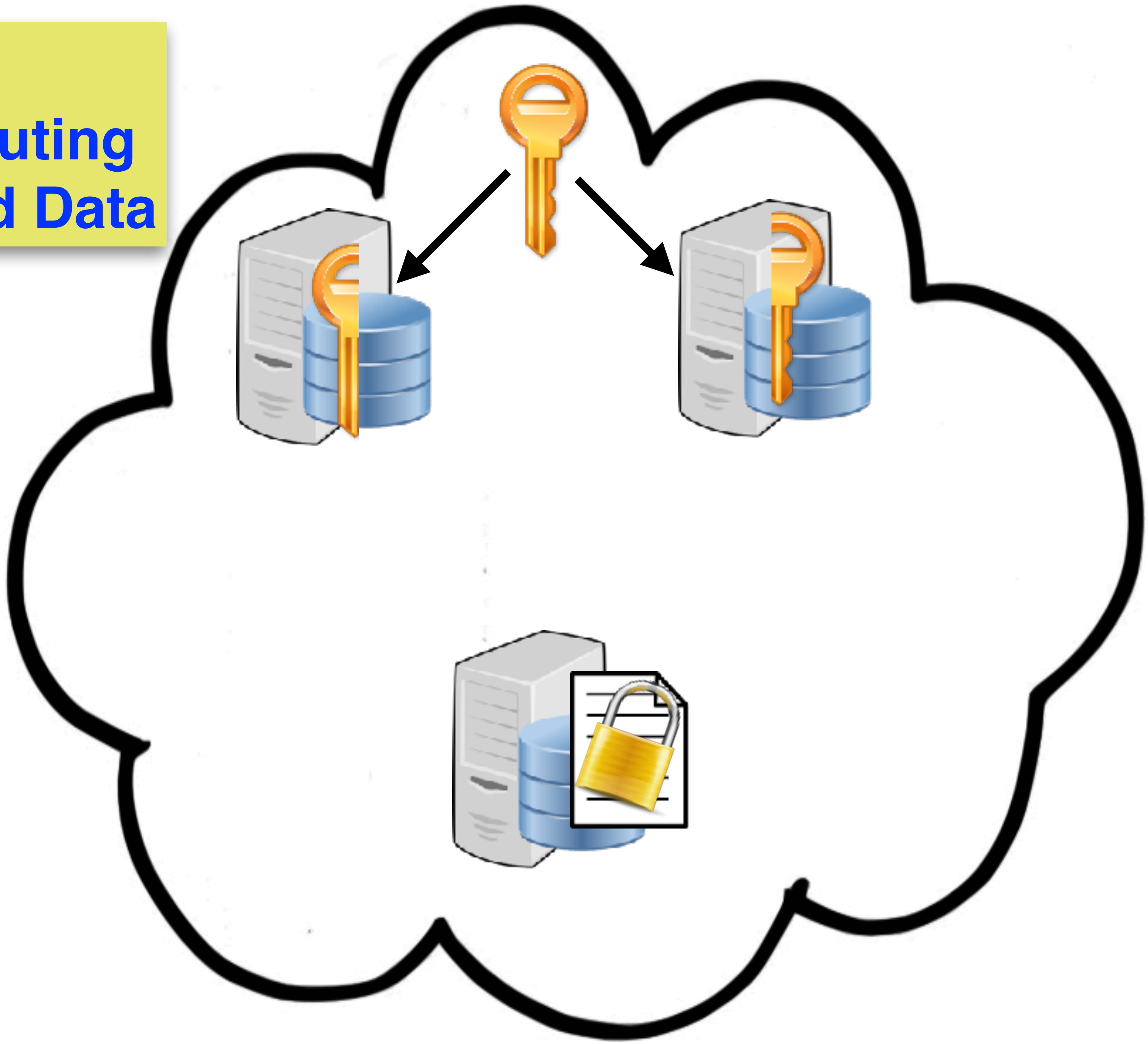
Example:

Cloud Computing on Encrypted Data



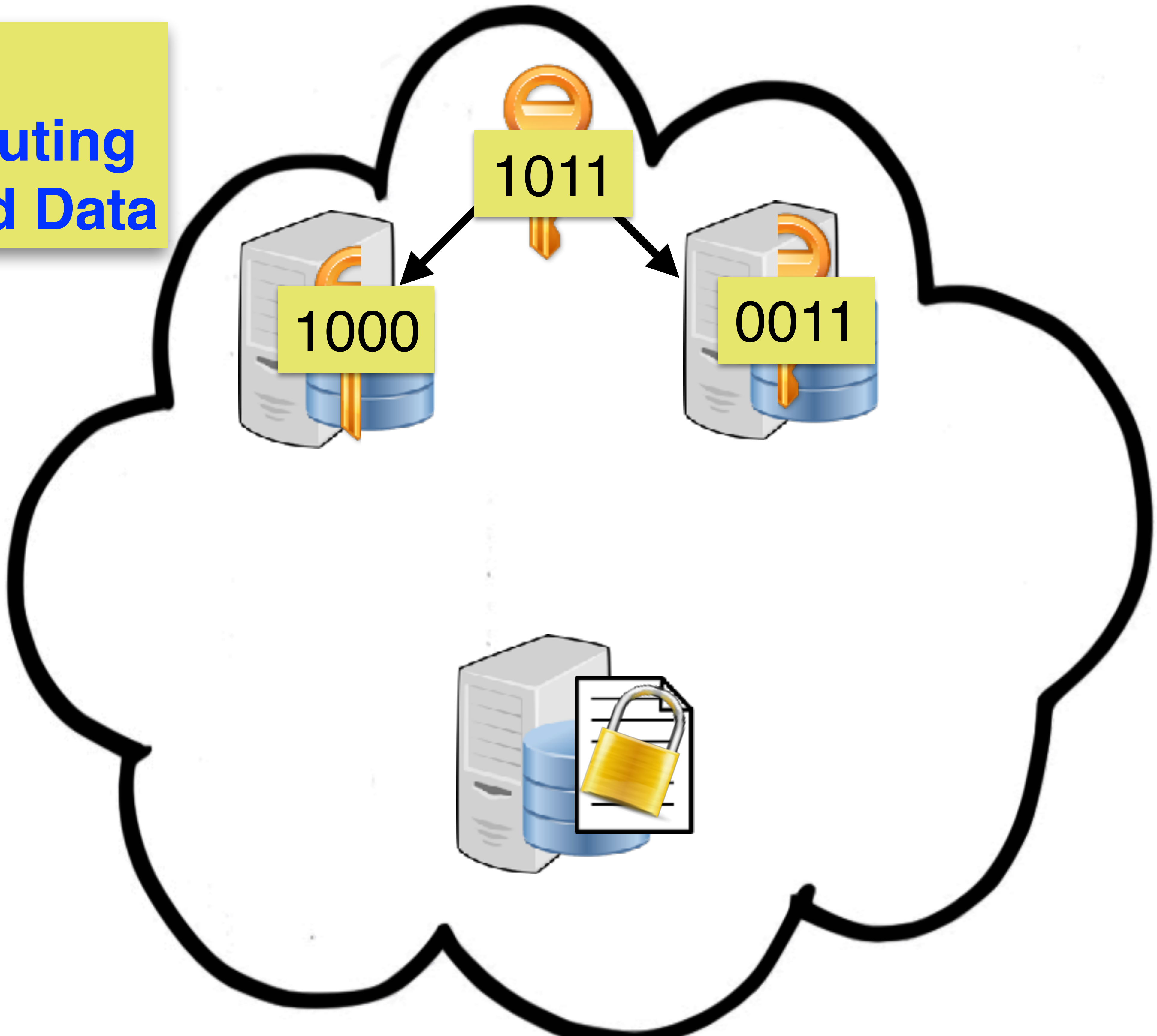
MPC in Action: A Toy Example

Example:
Cloud Computing on Encrypted Data



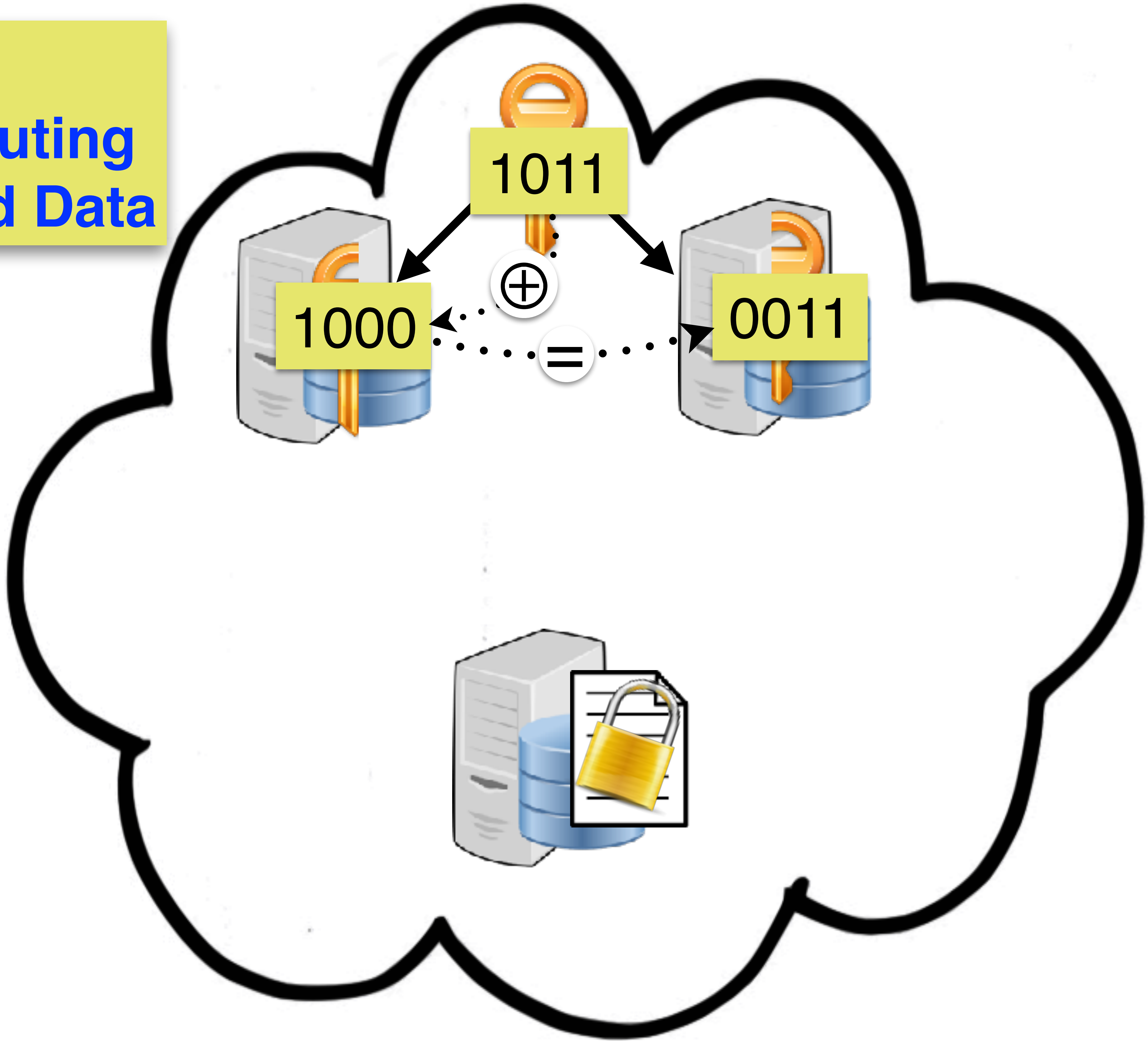
MPC in Action: A Toy Example

Example:
Cloud Computing
on Encrypted Data



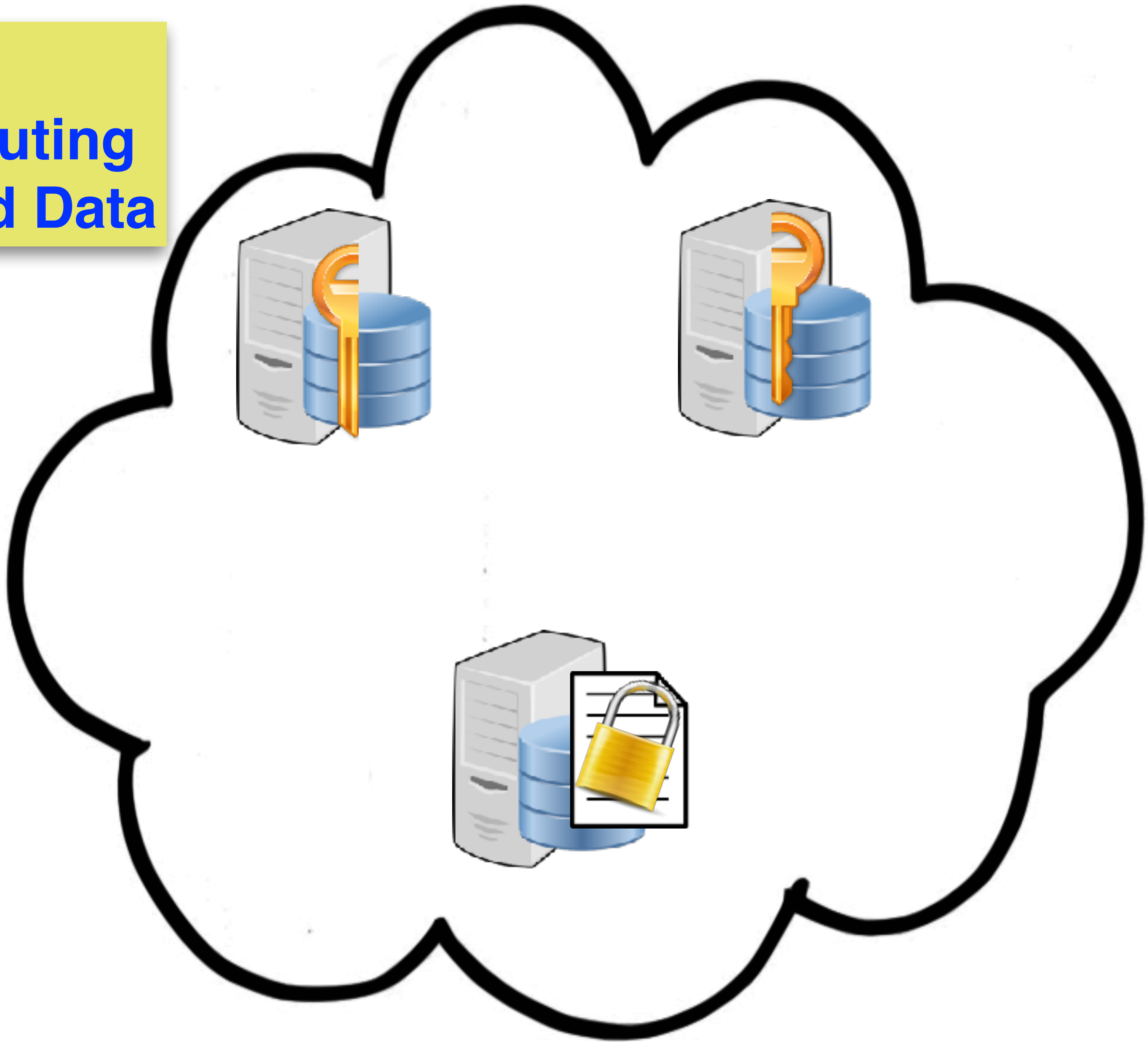
MPC in Action: A Toy Example

Example:
Cloud Computing on Encrypted Data



MPC in Action: A Toy Example

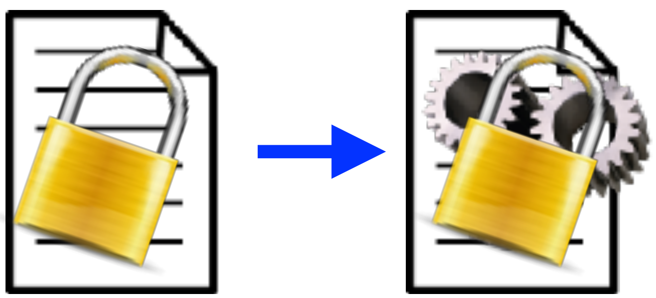
Example:
Cloud Computing on Encrypted Data

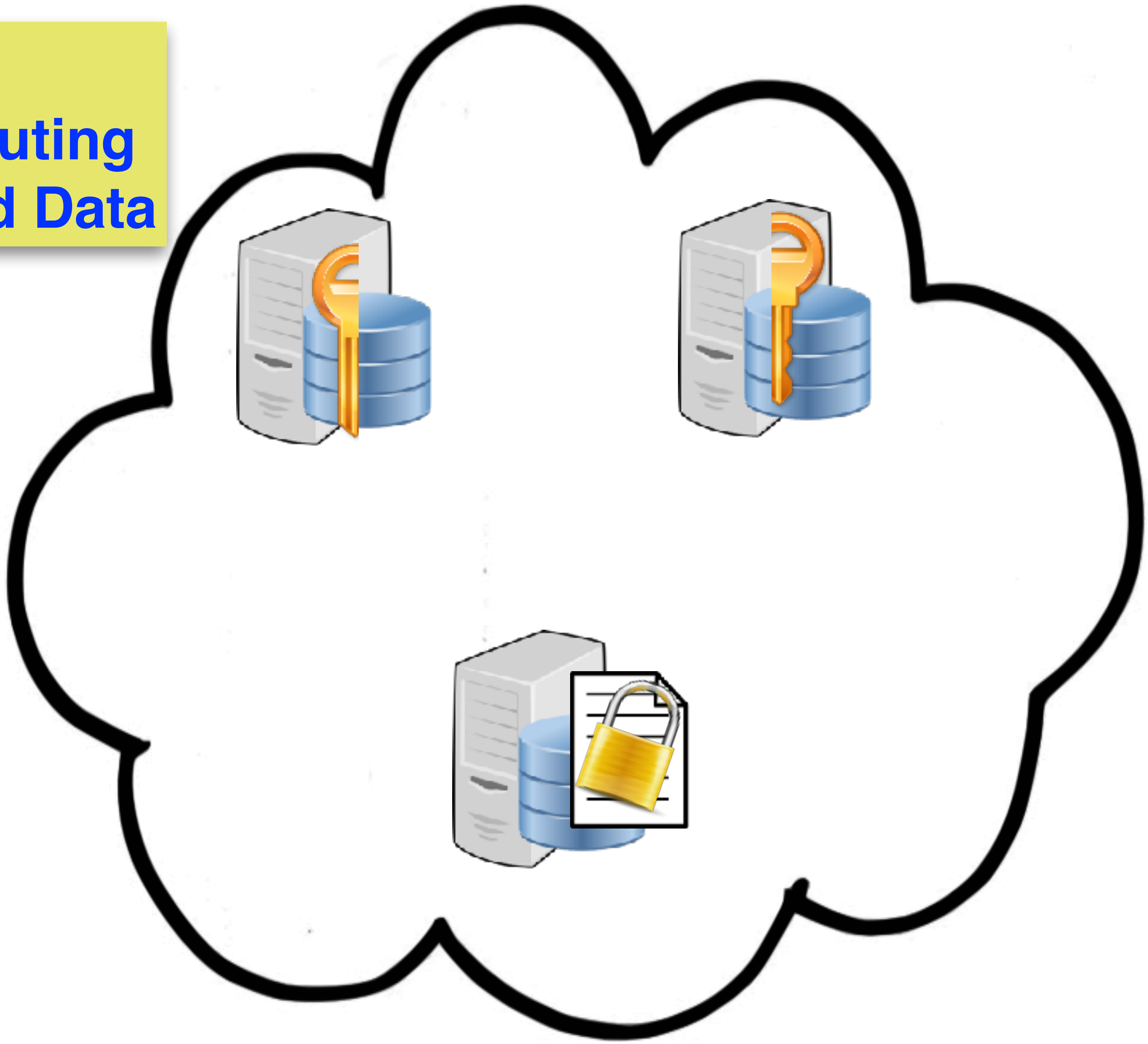


MPC in Action: A Toy Example

Example:
Cloud Computing on Encrypted Data

Goal

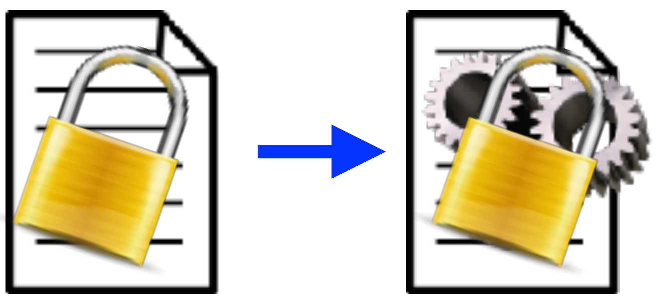


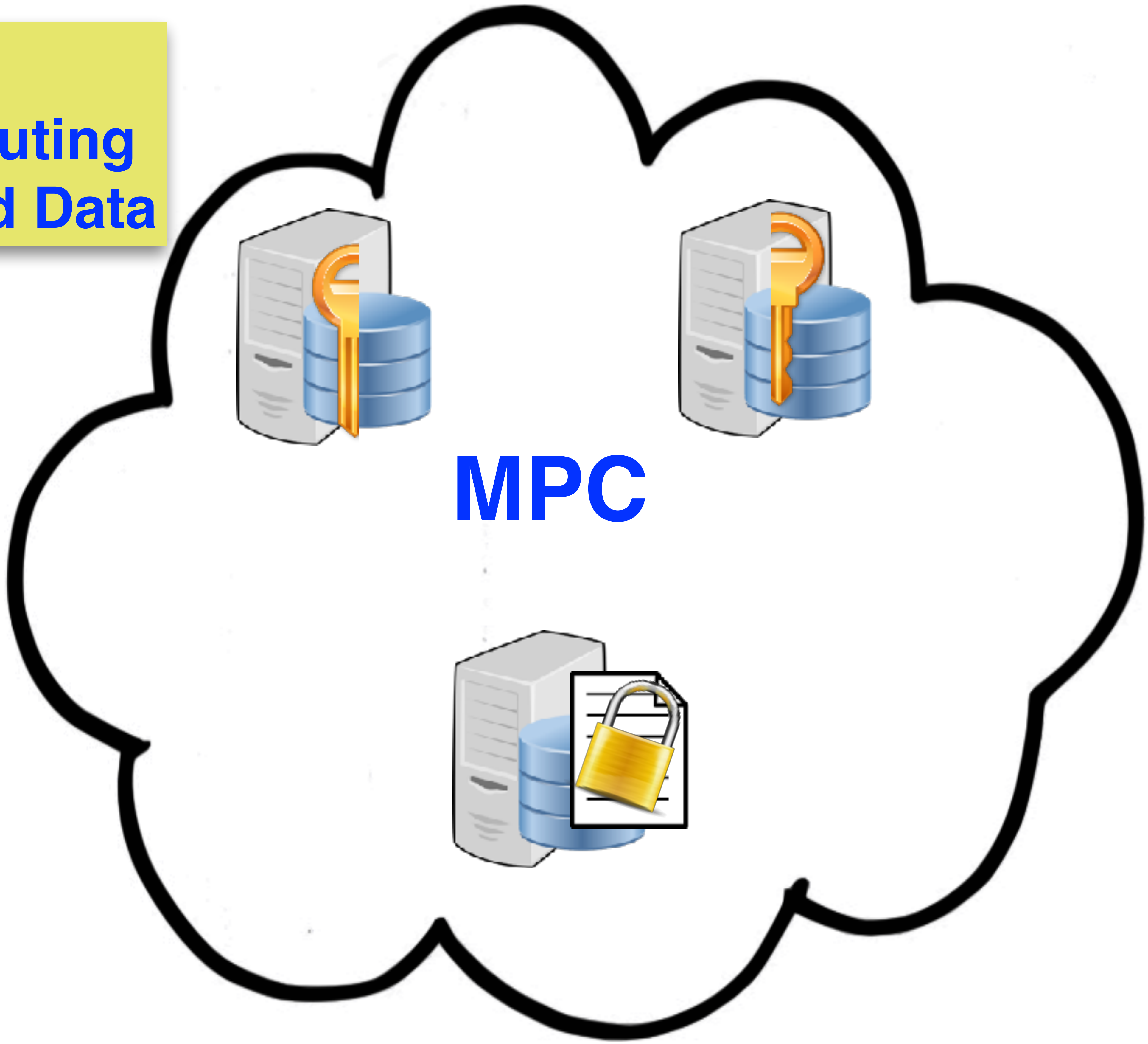


MPC in Action: A Toy Example

Example:
Cloud Computing on Encrypted Data

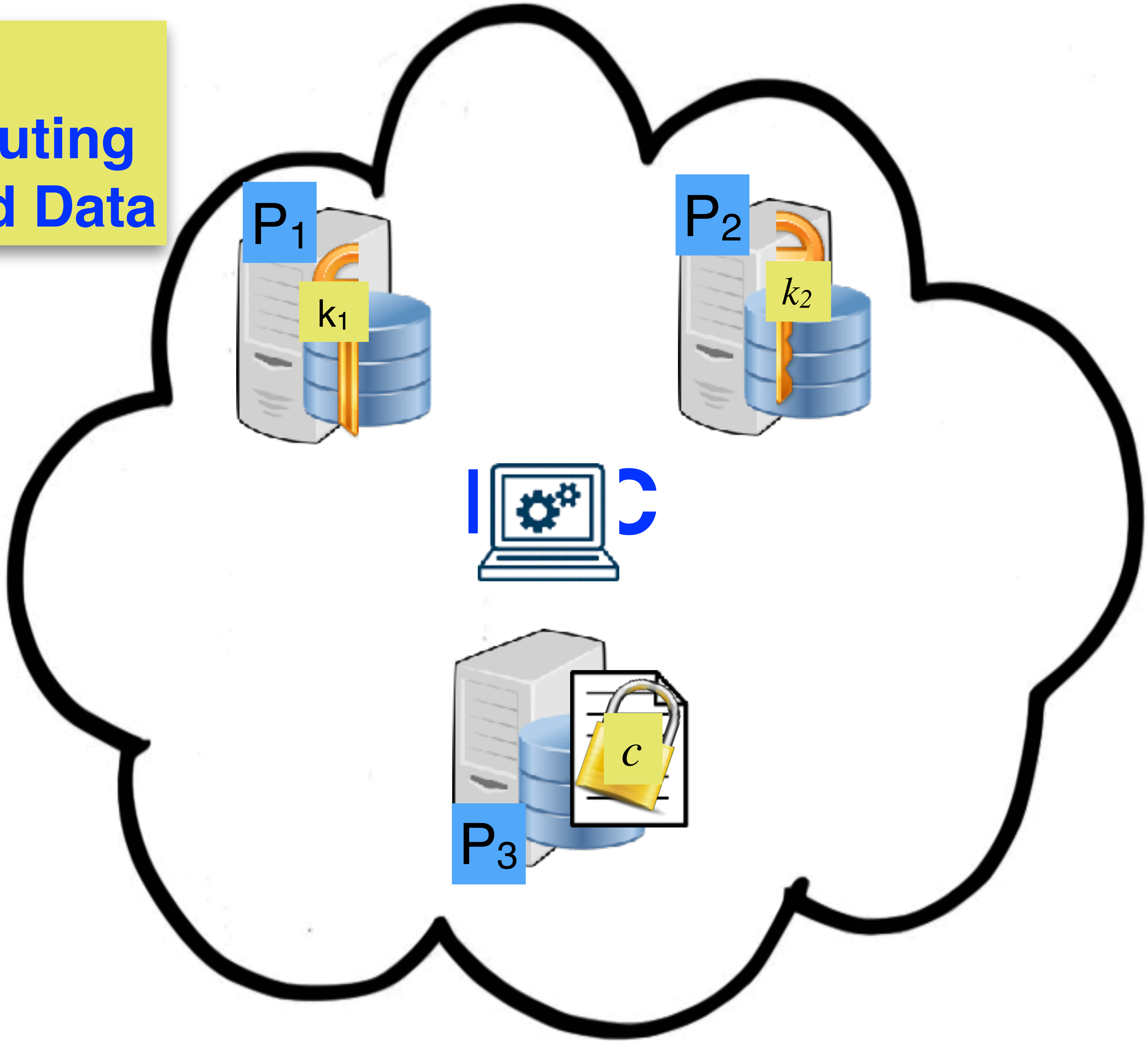
Goal





MPC in Action: A Toy Example

Example:
Cloud Computing on Encrypted Data



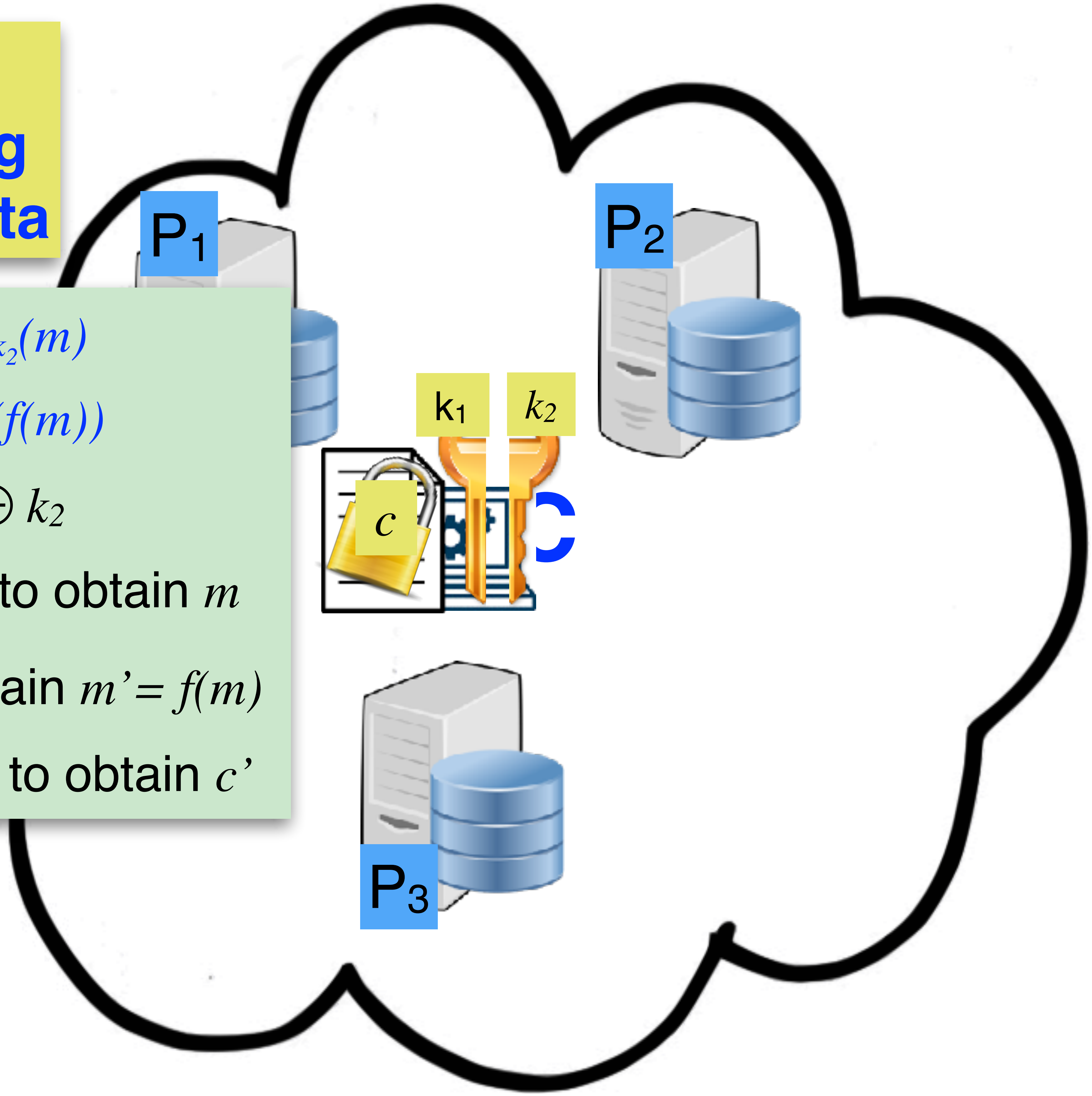
MPC in Action: A Toy Example

Example:
Cloud Computing on Encrypted Data

Inputs: $k_1, k_2, c=Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. Apply $f(\cdot)$ to m to obtain $m' = f(m)$
4. Re-encrypt m' with k to obtain c'



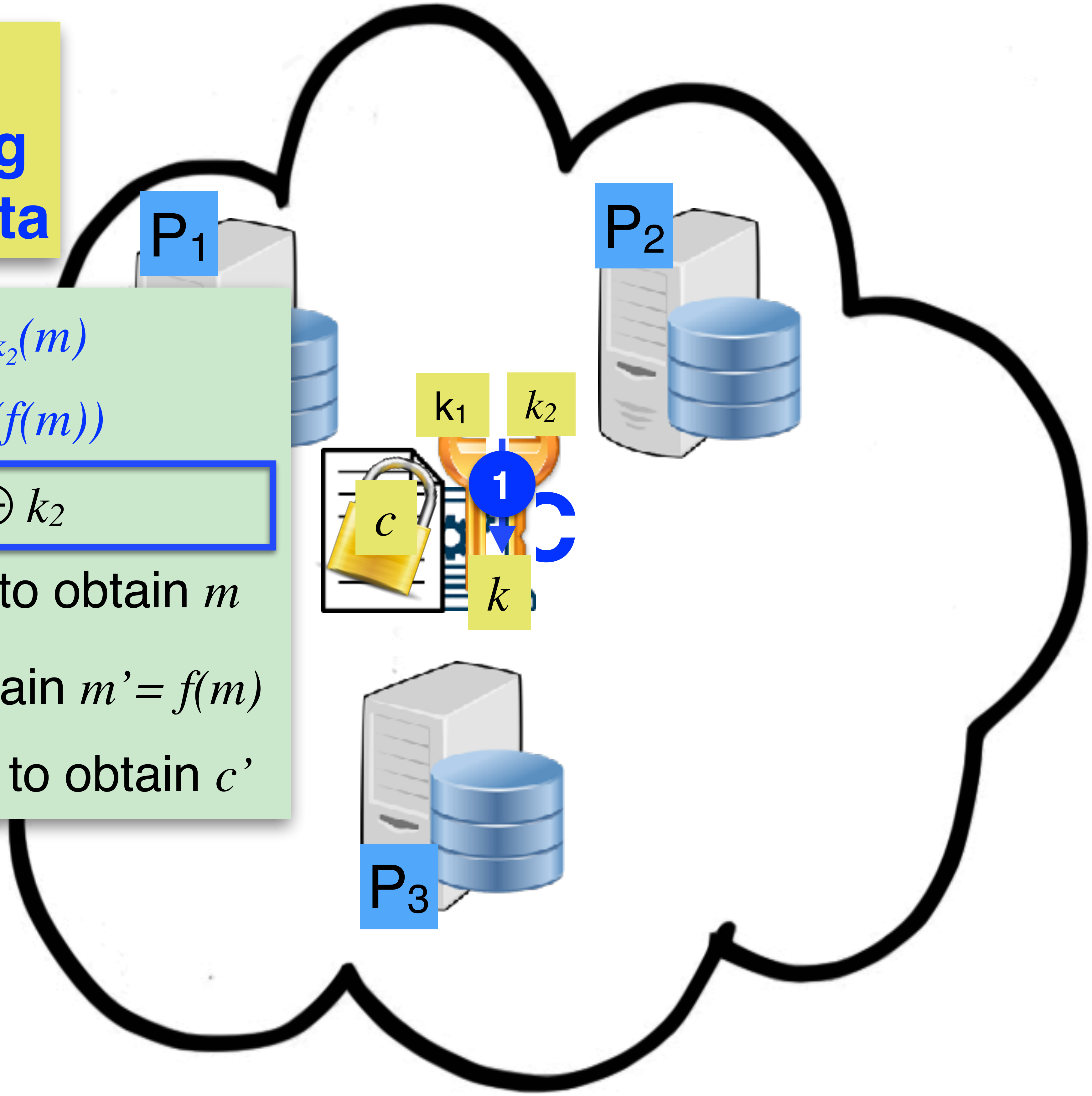
MPC in Action: A Toy Example

Example:
Cloud Computing on Encrypted Data

Inputs: $k_1, k_2, c = Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. Apply $f(\cdot)$ to m to obtain $m' = f(m)$
4. Re-encrypt m' with k to obtain c'



MPC in Action: A Toy Example

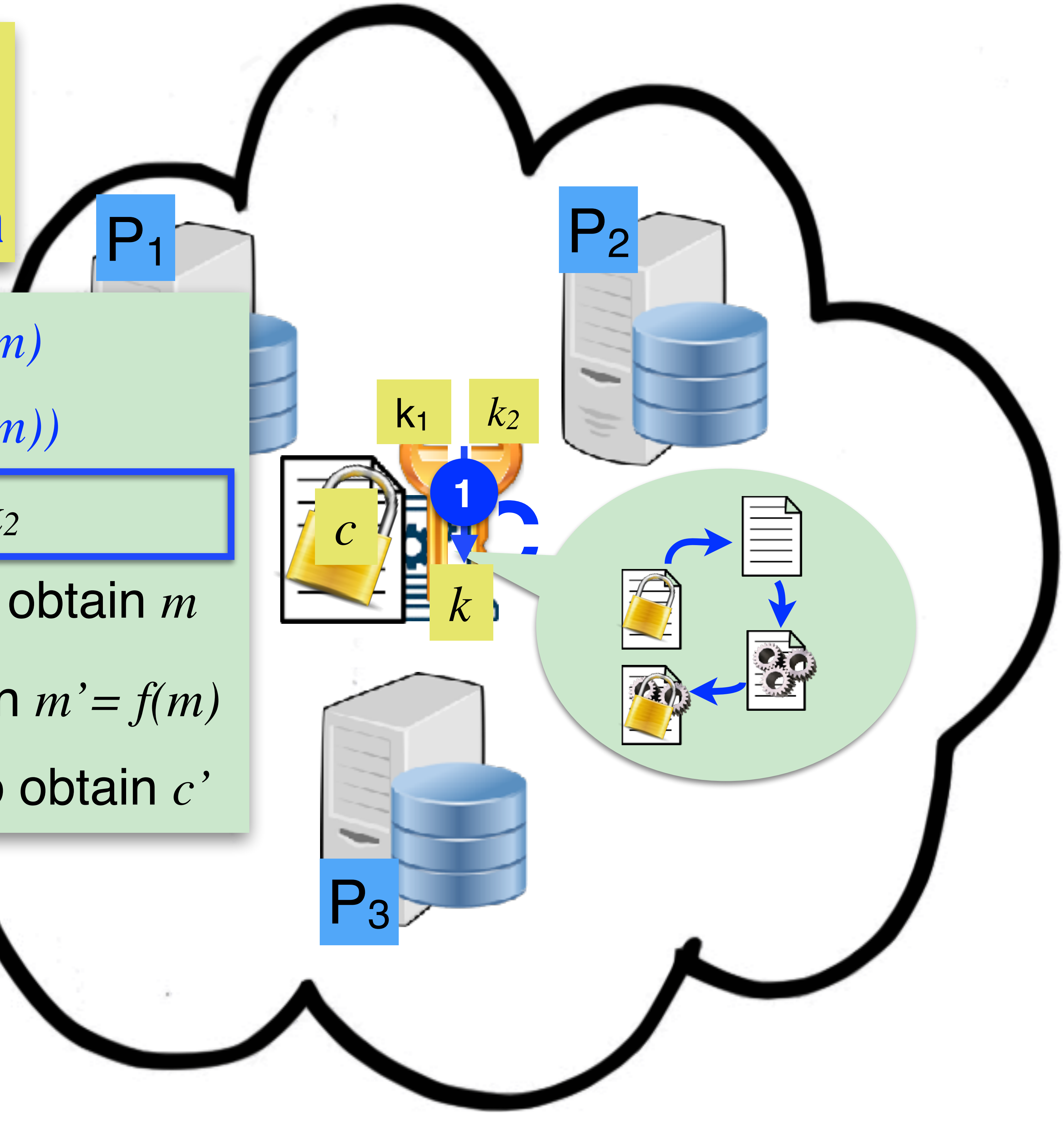
Example:

Cloud Computing on Encrypted Data

Inputs: $k_1, k_2, c = Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. Apply $f(\cdot)$ to m to obtain $m' = f(m)$
4. Re-encrypt m' with k to obtain c'



MPC in Action: A Toy Example

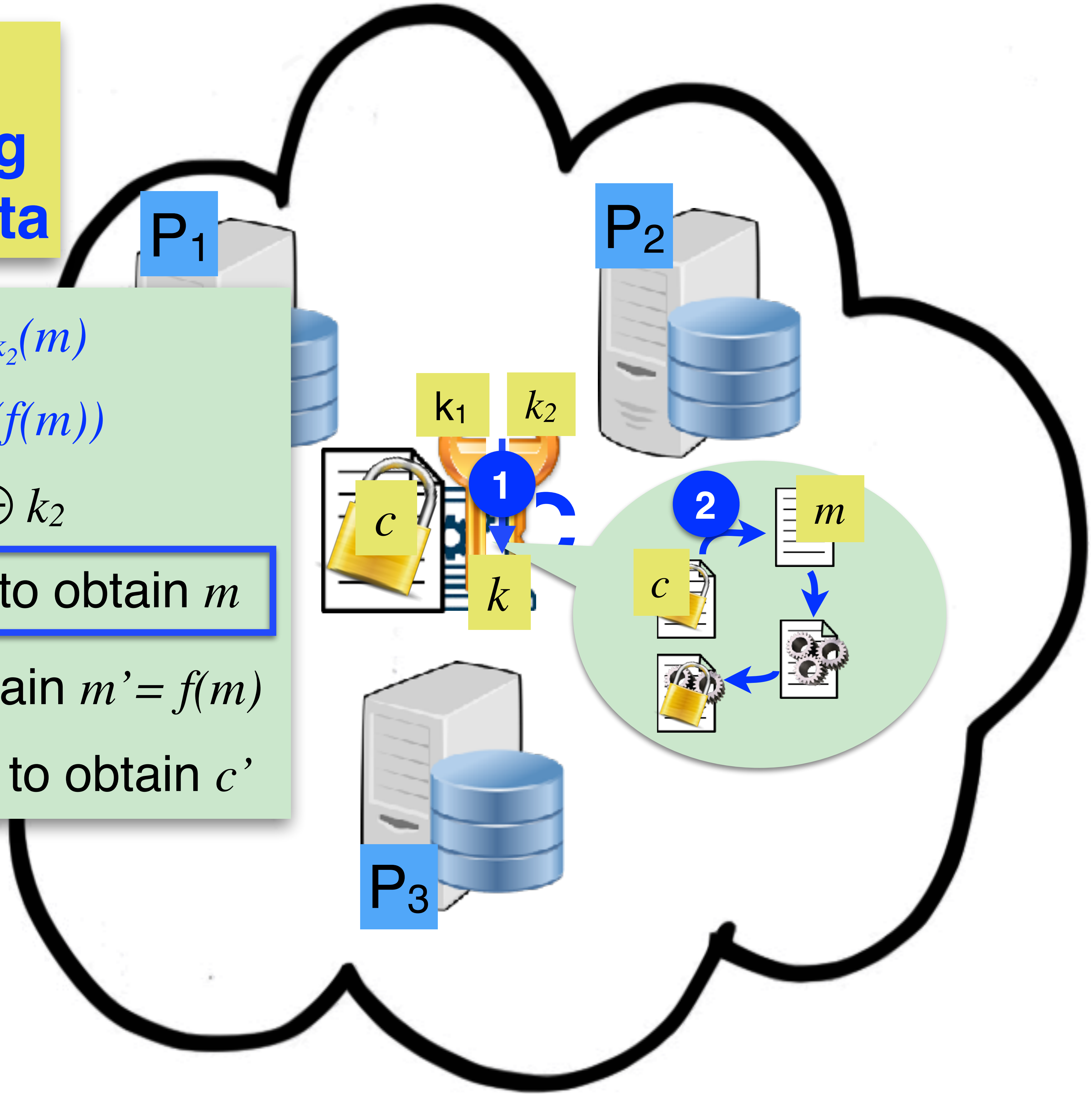
Example:

Cloud Computing on Encrypted Data

Inputs: $k_1, k_2, c=Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. Apply $f(\cdot)$ to m to obtain $m' = f(m)$
4. Re-encrypt m' with k to obtain c'



MPC in Action: A Toy Example

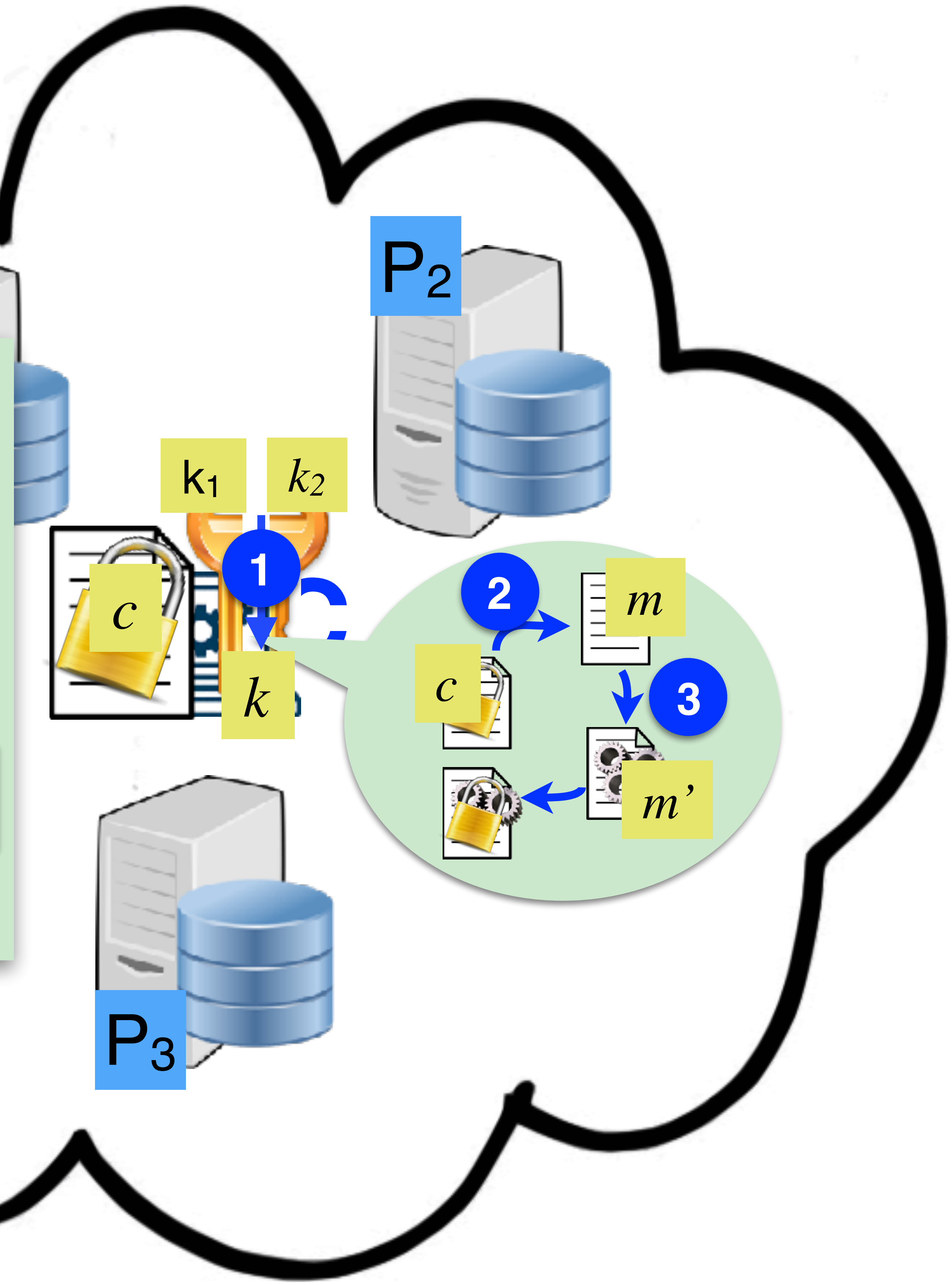
Example:

Cloud Computing on Encrypted Data

Inputs: $k_1, k_2, c = Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. $\boxed{\text{Apply } f(\cdot) \text{ to } m \text{ to obtain } m' = f(m)}$
4. Re-encrypt m' with k to obtain c'



MPC in Action: A Toy Example

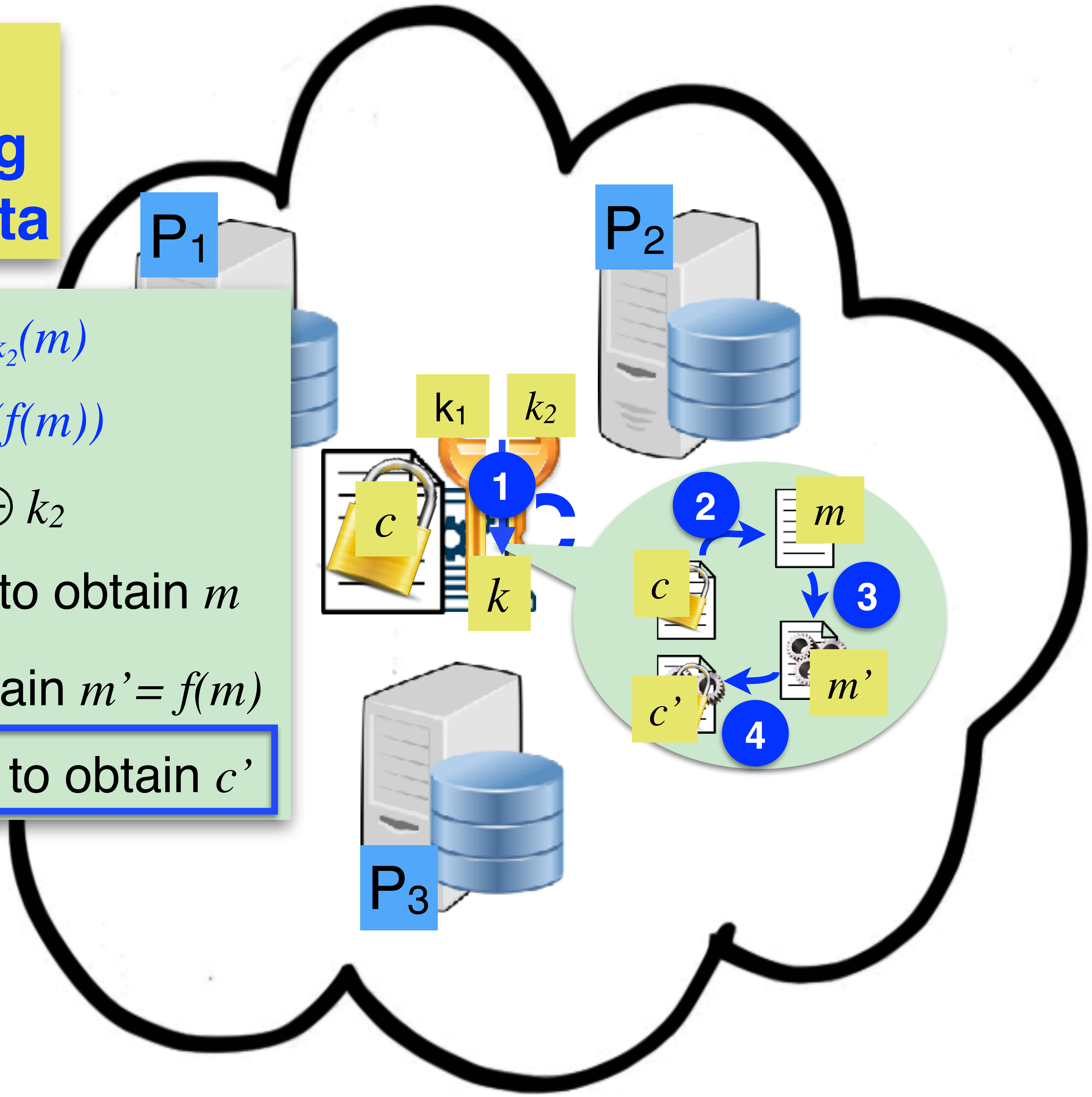
Example:

Cloud Computing on Encrypted Data

Inputs: $k_1, k_2, c = Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. Apply $f(\cdot)$ to m to obtain $m' = f(m)$
4. Re-encrypt m' with k to obtain c'



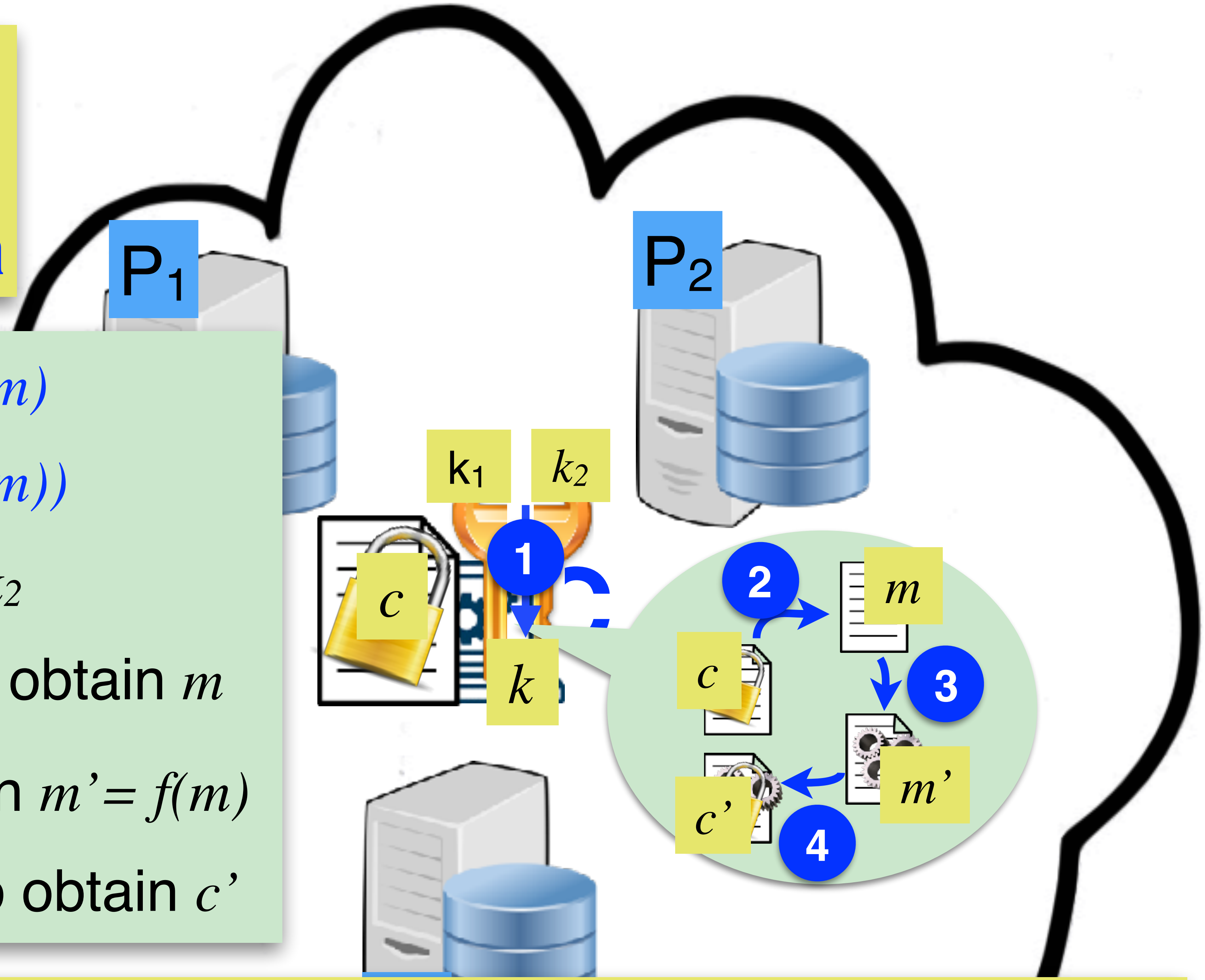
MPC in Action: A Toy Example

Example:
Cloud Computing on Encrypted Data

Inputs: $k_1, k_2, c = Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. Apply $f(\cdot)$ to m to obtain $m' = f(m)$
4. Re-encrypt m' with k to obtain c'



Goal: Perform this computation securely

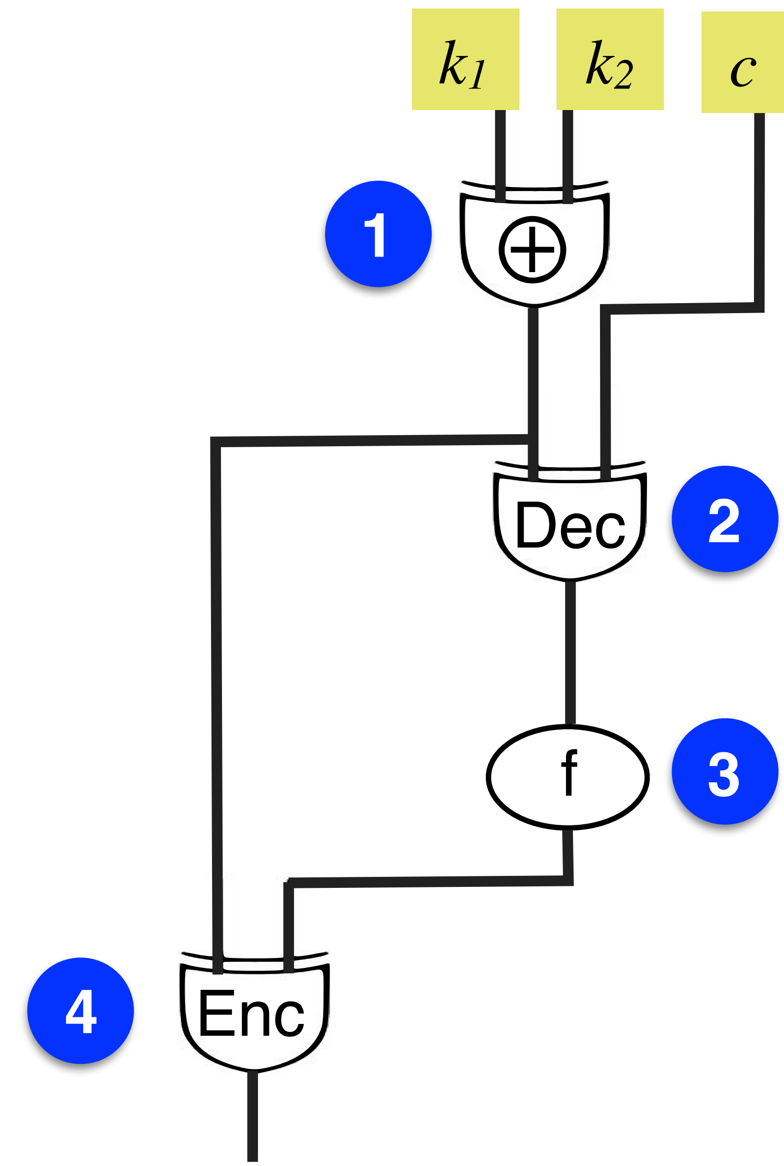
- *(privacy)* No (corrupted) server learns the key or the plaintext
- *(correctness)* The result is the encrypted data after the computation

MPC in Action: A Toy Example

Inputs: $k_1, k_2, c = Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. Apply $f(\cdot)$ to m to obtain $m' = f(m)$
4. Re-encrypt m' with k to obtain c'

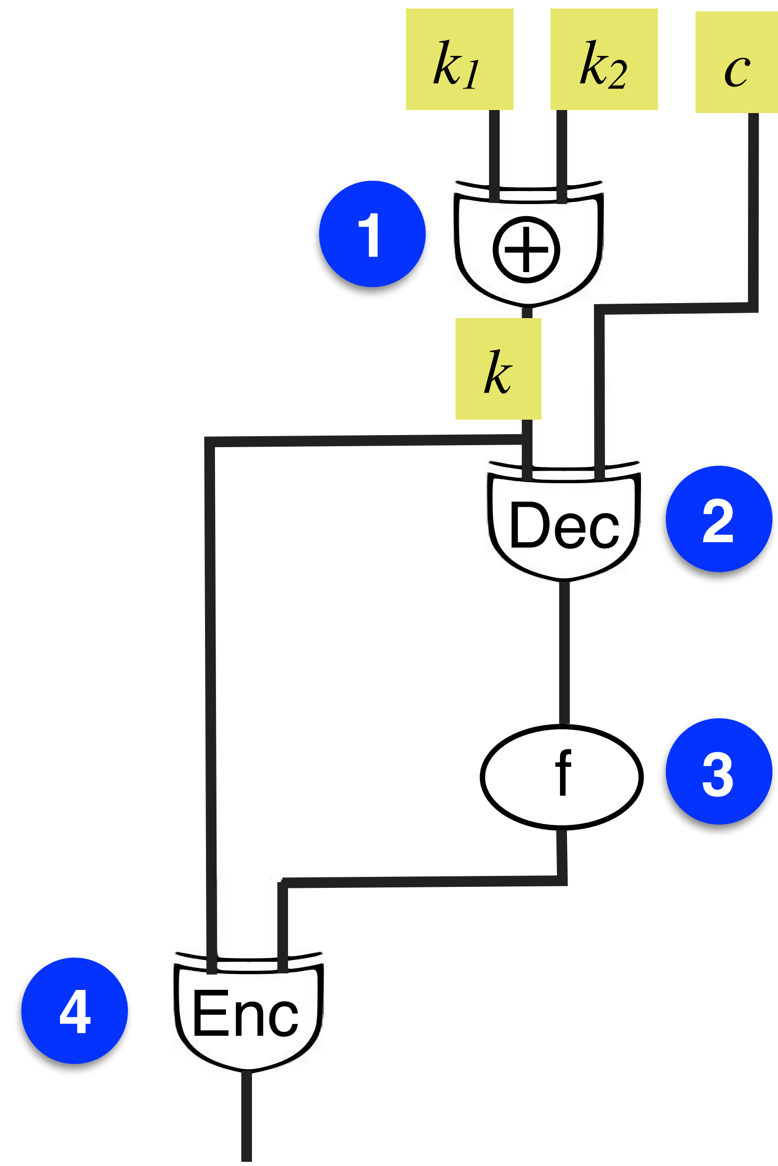


MPC in Action: A Toy Example

Inputs: $k_1, k_2, c = Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. Apply $f(\cdot)$ to m to obtain $m' = f(m)$
4. Re-encrypt m' with k to obtain c'

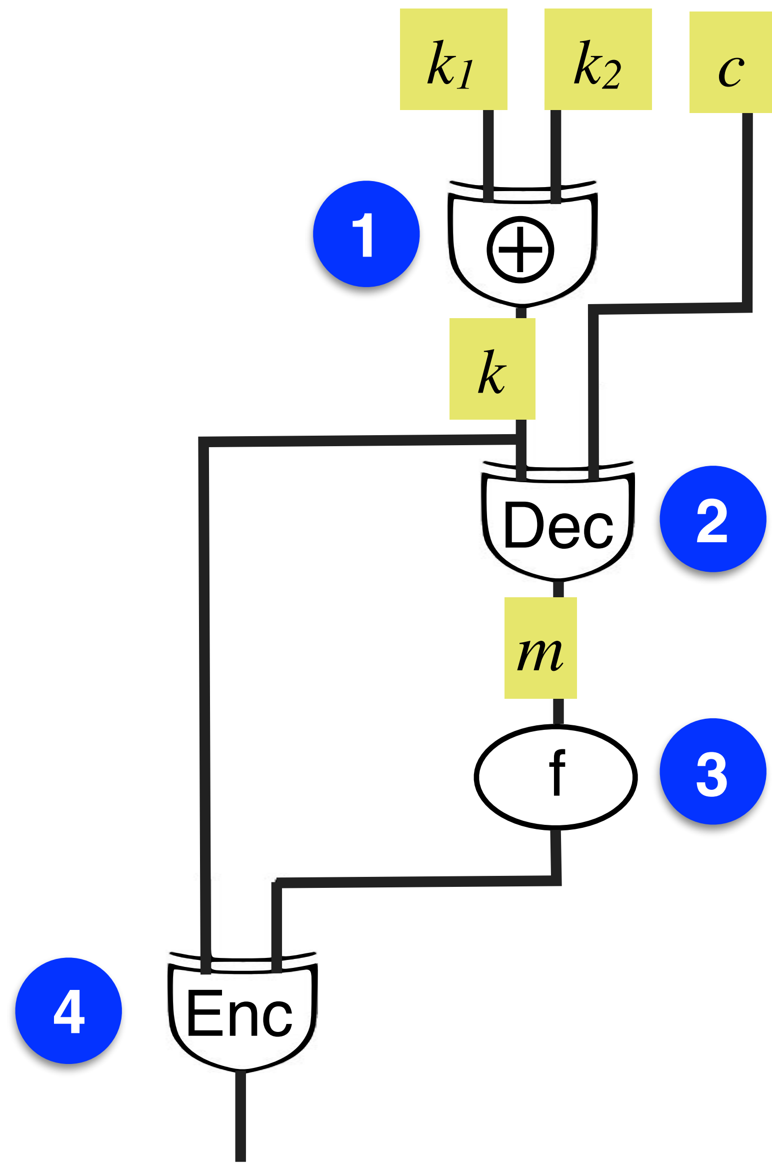


MPC in Action: A Toy Example

Inputs: $k_1, k_2, c = Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. Apply $f(\cdot)$ to m to obtain $m' = f(m)$
4. Re-encrypt m' with k to obtain c'

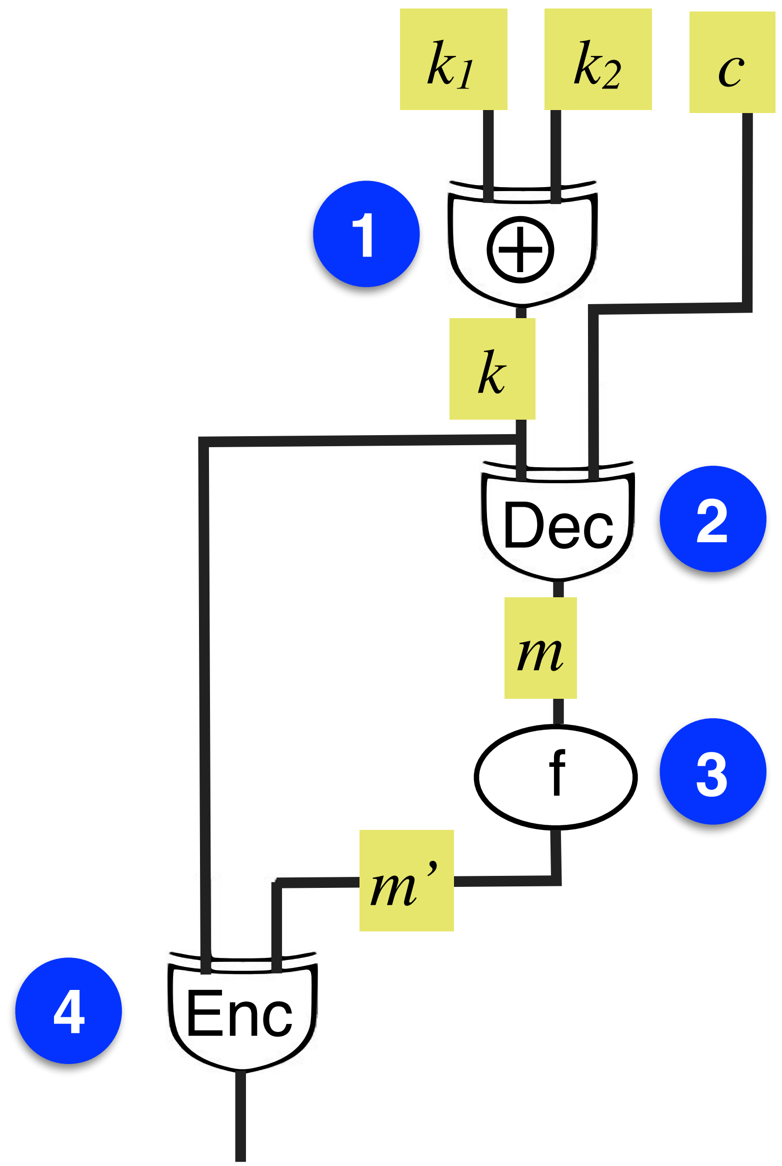


MPC in Action: A Toy Example

Inputs: $k_1, k_2, c = Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. Apply $f(\cdot)$ to m to obtain $m' = f(m)$
4. Re-encrypt m' with k to obtain c'

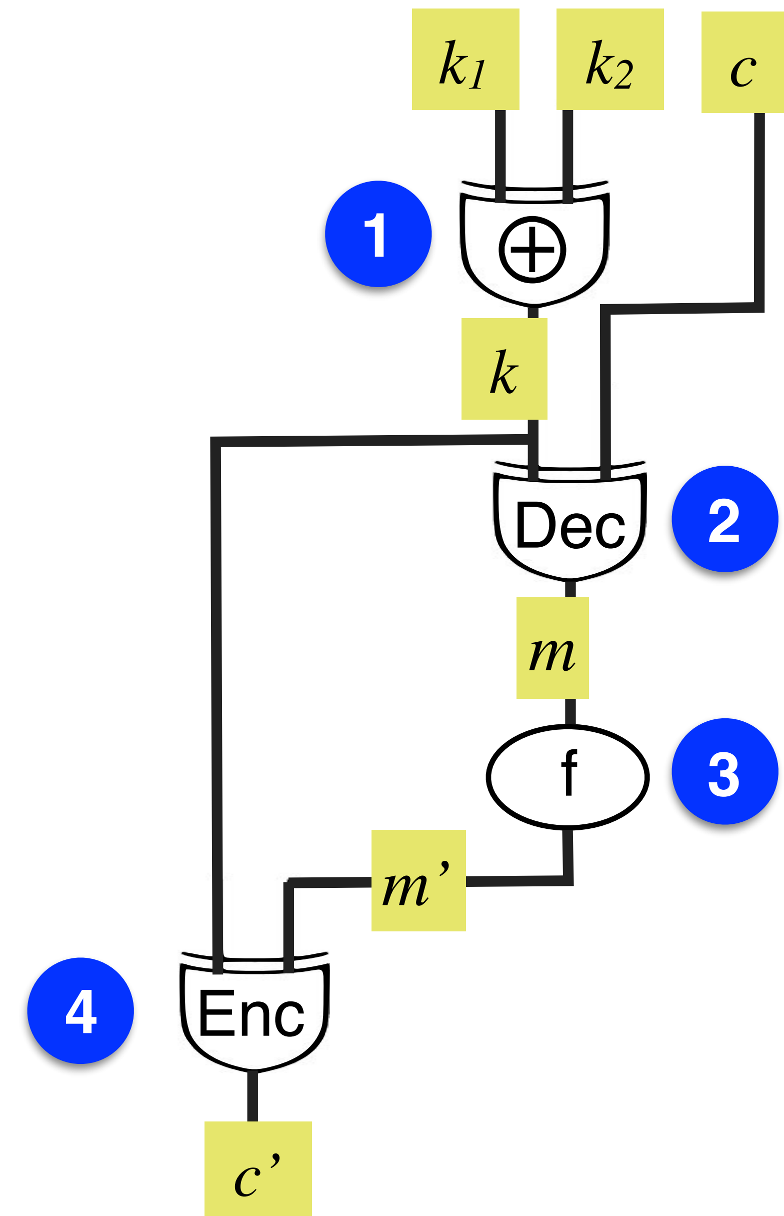


MPC in Action: A Toy Example

Inputs: $k_1, k_2, c = Enc_{k=k_1 \oplus k_2}(m)$

Task: Compute $c' = Enc_k(f(m))$

1. Reconstruct $k := k_1 \oplus k_2$
2. Decrypt c with key k to obtain m
3. Apply $f(\cdot)$ to m to obtain $m' = f(m)$
4. Re-encrypt m' with k to obtain c'



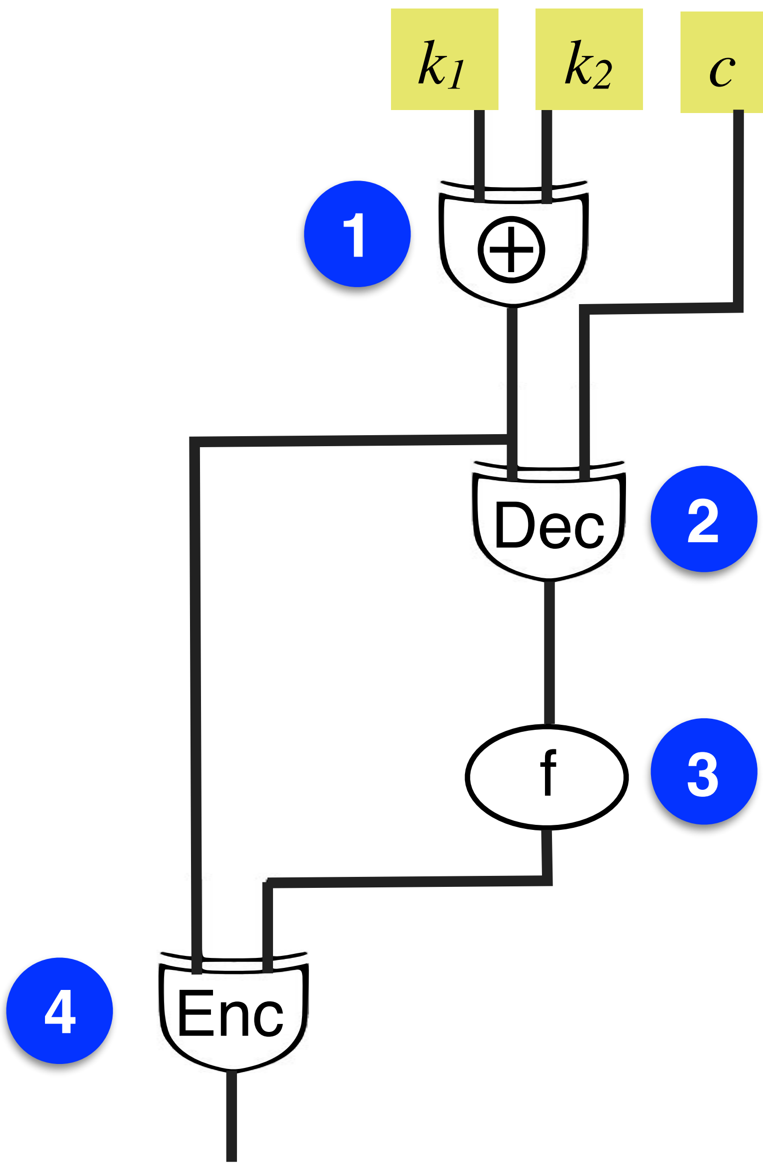
MPC in Action: A Toy Example

$m = m_L || m_R$

Example:

$Enc_k(m) := m \oplus k$, $f(m) = m_L \oplus m_R || m_R$

$Dec_k(c) := c \oplus k$



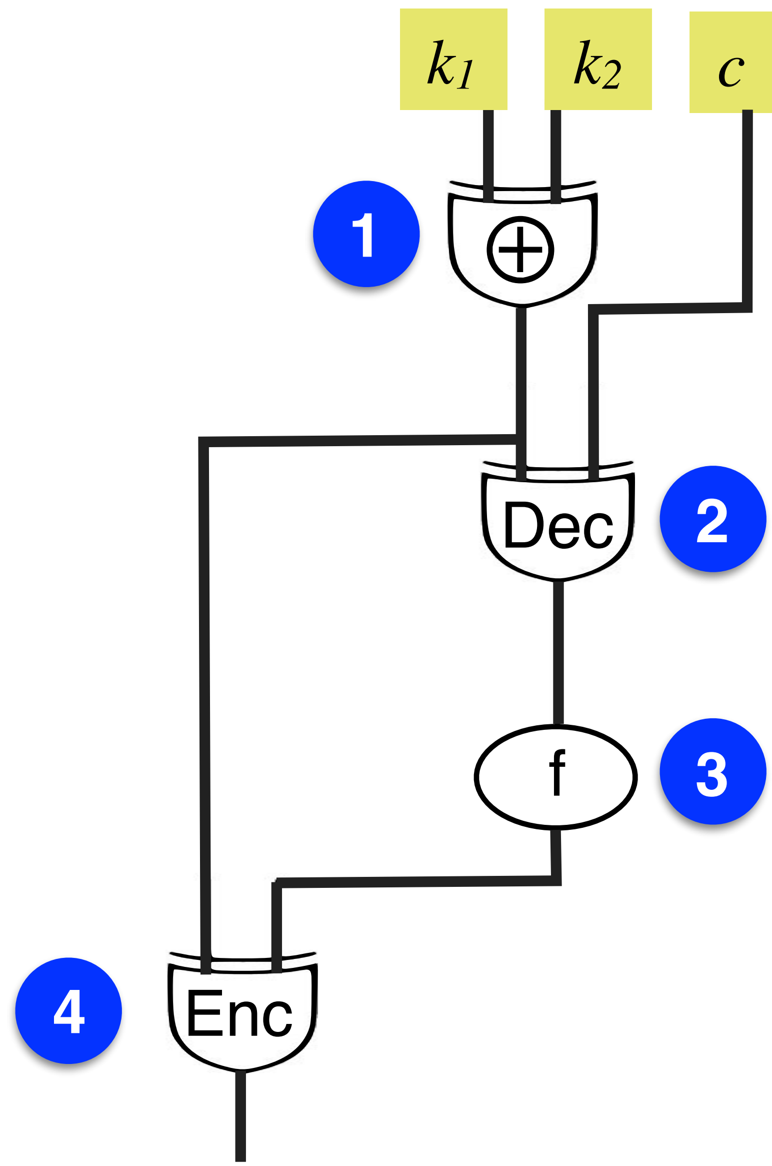
MPC in Action: A Toy Example

$m=m_L||m_R$

Example:

$Enc_k(m):= m \oplus k$, $f(m) = m_L \oplus m_R || m_R$

$Dec_k(c):= c \oplus k$



Tool: (Additive) Secret Sharing *[s]* of secret *s*

- Choose random s_1, s_2, s_3 s.t. $s_1 \oplus s_2 \oplus s_3 = s$
- Hand s_i to P_i

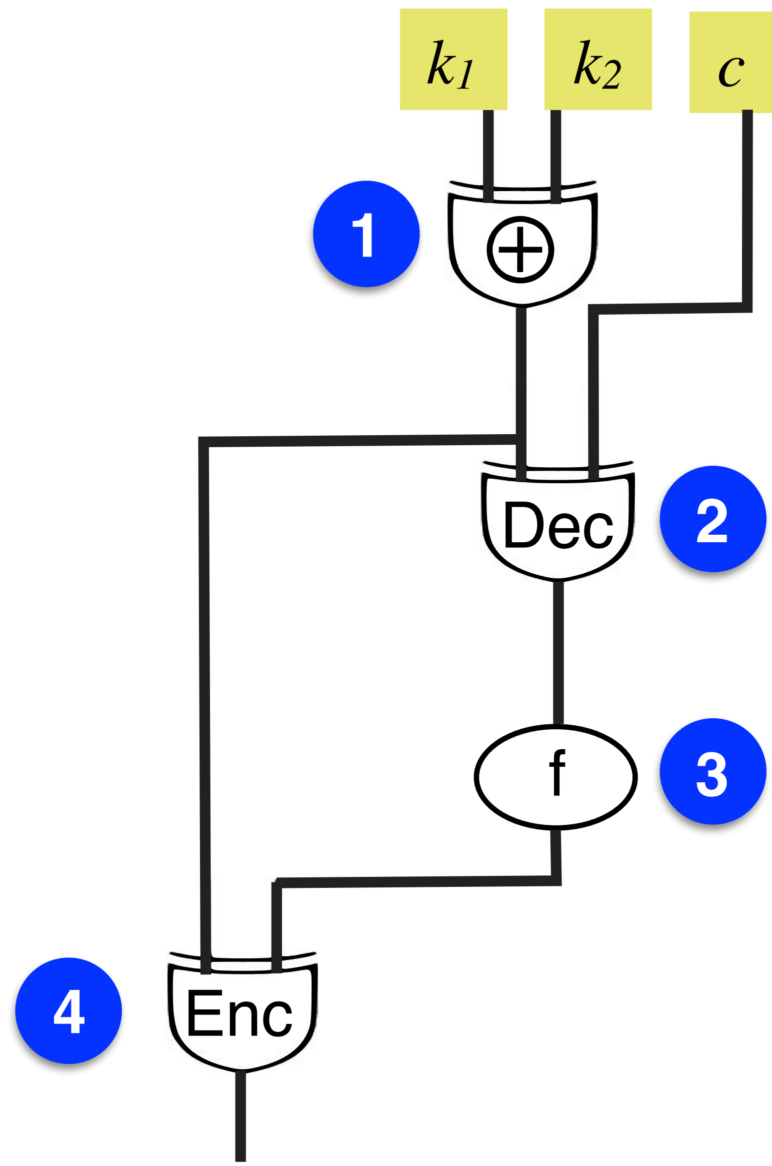
MPC in Action: A Toy Example

Example:

$m = m_L || m_R$

$Enc_k(m) := m \oplus k \quad , f(m) = m_L \oplus m_R || m_R$

$Dec_k(c) := c \oplus k$



Tool: (Additive) Secret Sharing *[s]* of secret *s*

- Choose random s_1, s_2, s_3 s.t. $s_1 \oplus s_2 \oplus s_3 = s$
- Hand s_i to P_i

Any subset gets no info on s

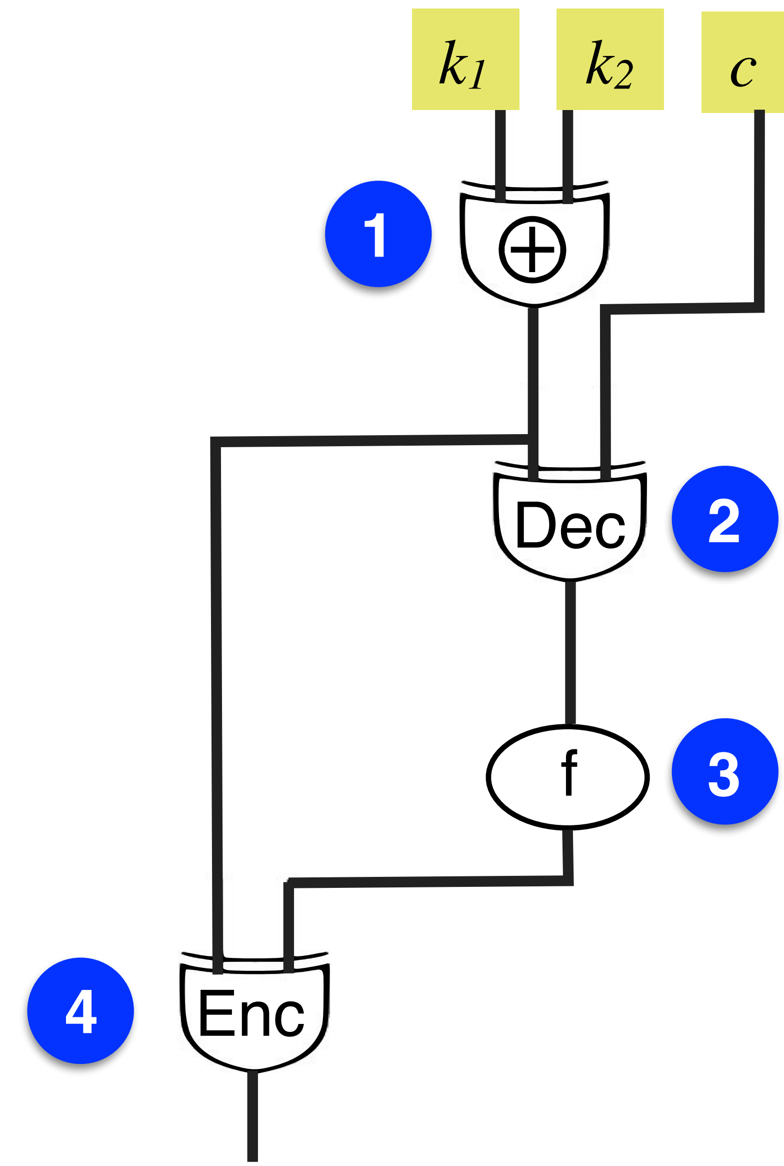
MPC in Action: A Toy Example

$m = m_L || m_R$

Example:

$Enc_k(m) := m \oplus k$, $f(m) = m_L \oplus m_R || m_R$

$Dec_k(c) := c \oplus k$



Tool: (Additive) Secret Sharing *[s]* of secret *s*

- Choose random s_1, s_2, s_3 s.t. $s_1 \oplus s_2 \oplus s_3 = s$
- Hand s_i to P_i

Any subset gets no info on s

Protocol: Traverse the circuit gate by gate where instead of the wires' values compute sharing of these values

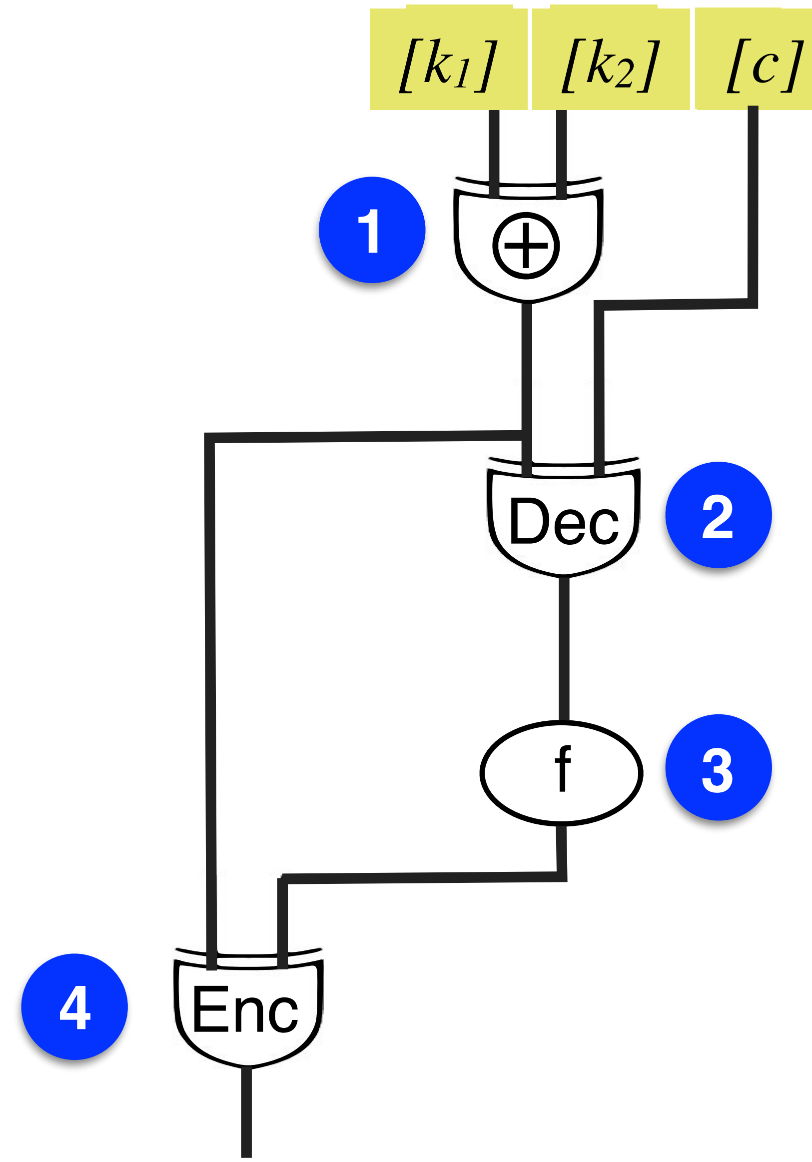
MPC in Action: A Toy Example

$m = m_L || m_R$

Example:

$Enc_k(m) := m \oplus k \quad , f(m) = m_L \oplus m_R || m_R$

$Dec_k(c) := c \oplus k$



Tool: (Additive) Secret Sharing *[s]* of secret *s*

- Choose random s_1, s_2, s_3 s.t. $s_1 \oplus s_2 \oplus s_3 = s$
- Hand s_i to P_i

Any subset gets no info on s

Protocol: Traverse the circuit gate by gate where instead of the wires' values compute sharing of these values

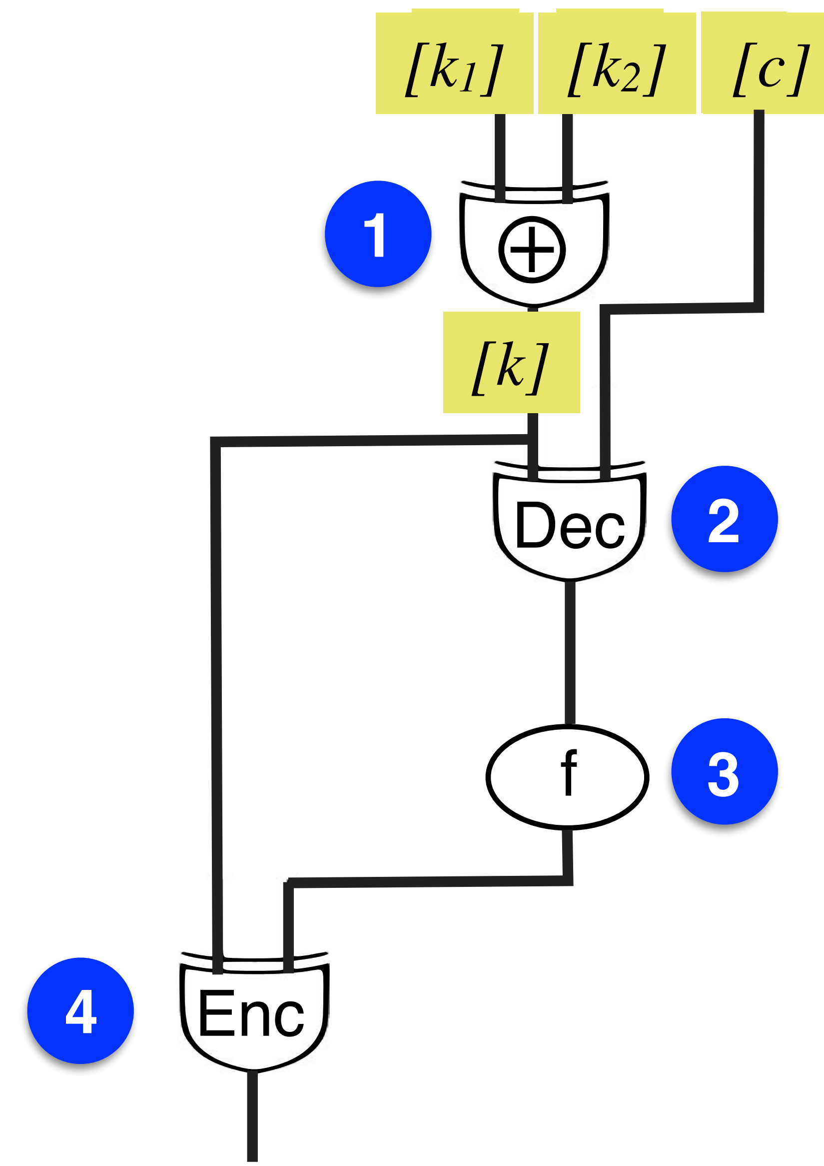
MPC in Action: A Toy Example

Example:

$m = m_L || m_R$

$Enc_k(m) := m \oplus k$, $f(m) = m_L \oplus m_R || m_R$

$Dec_k(c) := c \oplus k$



Tool: (Additive) Secret Sharing *[s]* of secret *s*

- Choose random s_1, s_2, s_3 s.t. $s_1 \oplus s_2 \oplus s_3 = s$
- Hand s_i to P_i

Any subset gets no info on s

Protocol: Traverse the circuit gate by gate where instead of the wires' values compute sharing of these values

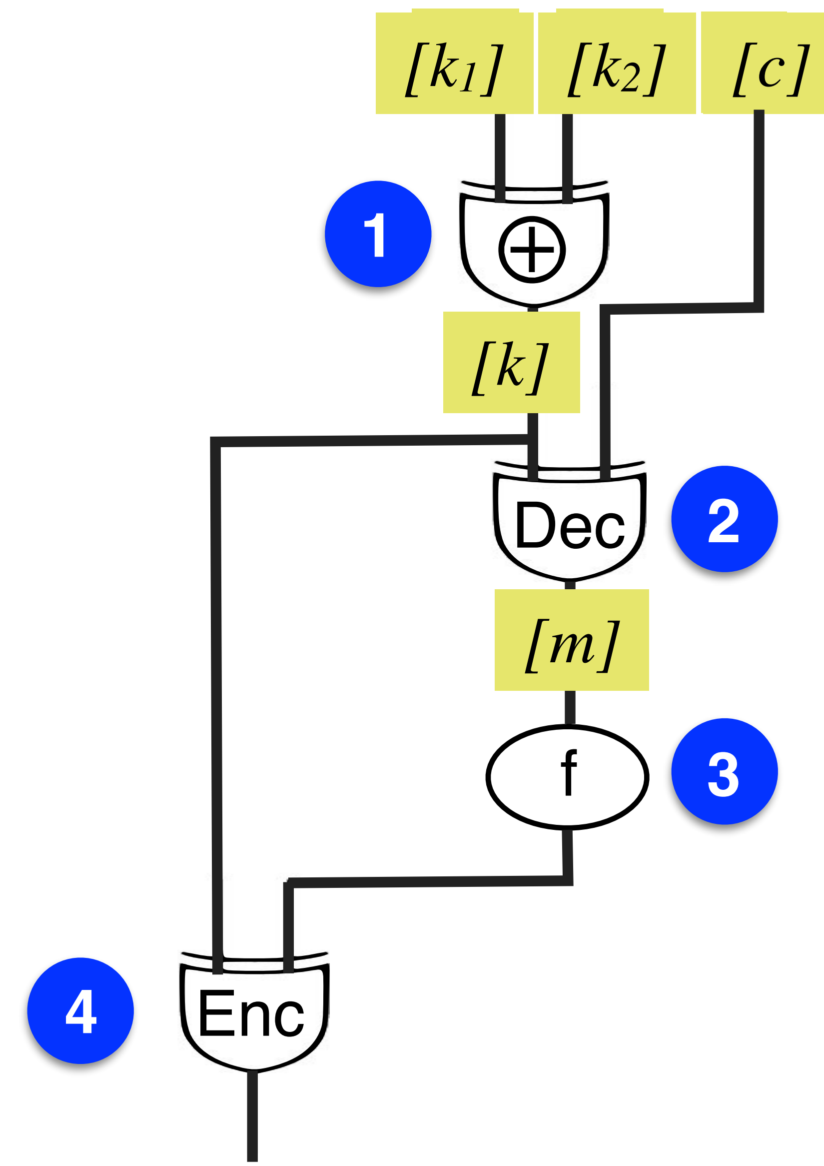
MPC in Action: A Toy Example

Example:

$m = m_L || m_R$

$Enc_k(m) := m \oplus k \quad , f(m) = m_L \oplus m_R || m_R$

$Dec_k(c) := c \oplus k$



Tool: (Additive) Secret Sharing *[s]* of secret *s*

- Choose random s_1, s_2, s_3 s.t. $s_1 \oplus s_2 \oplus s_3 = s$
- Hand s_i to P_i

Any subset gets no info on s

Protocol: Traverse the circuit gate by gate where instead of the wires' values compute sharing of these values

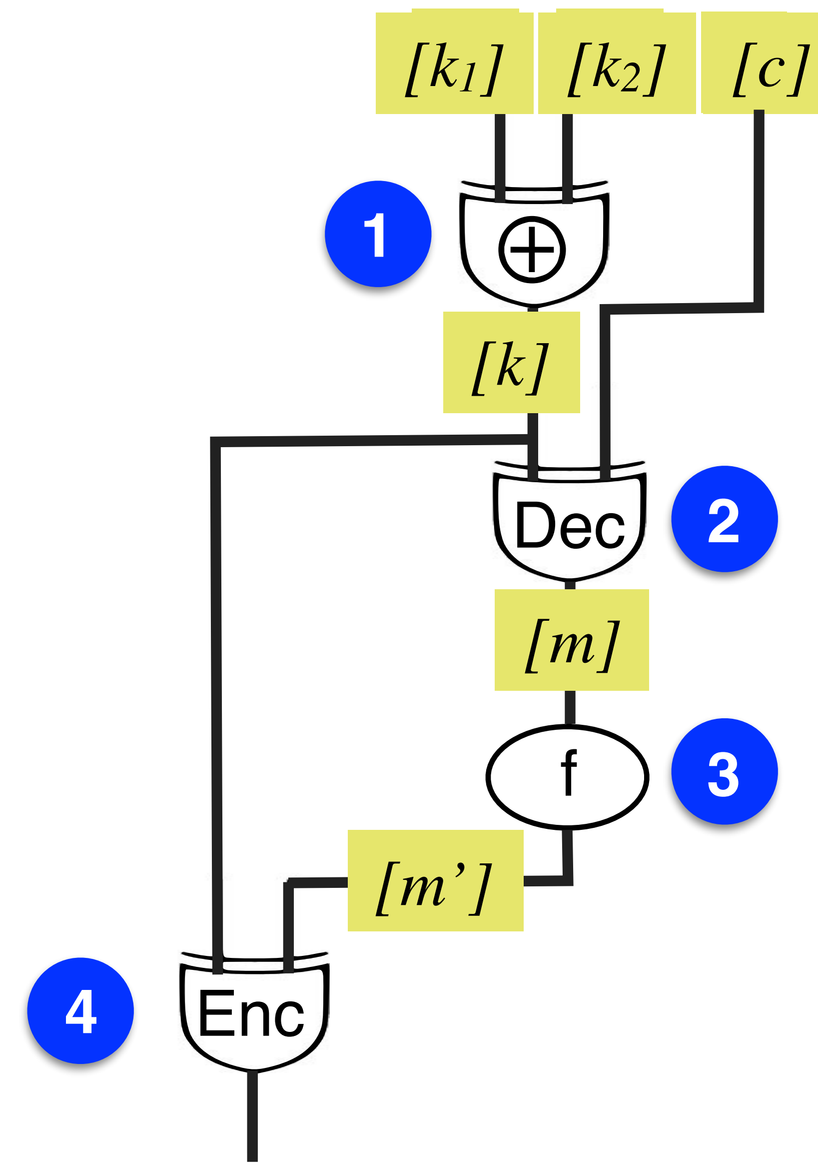
MPC in Action: A Toy Example

$m = m_L || m_R$

Example:

$Enc_k(m) := m \oplus k \quad , f(m) = m_L \oplus m_R || m_R$

$Dec_k(c) := c \oplus k$



Tool: (Additive) Secret Sharing *[s]* of secret *s*

- Choose random s_1, s_2, s_3 s.t. $s_1 \oplus s_2 \oplus s_3 = s$
- Hand s_i to P_i

Any subset gets no info on s

Protocol: Traverse the circuit gate by gate where instead of the wires' values compute sharing of these values

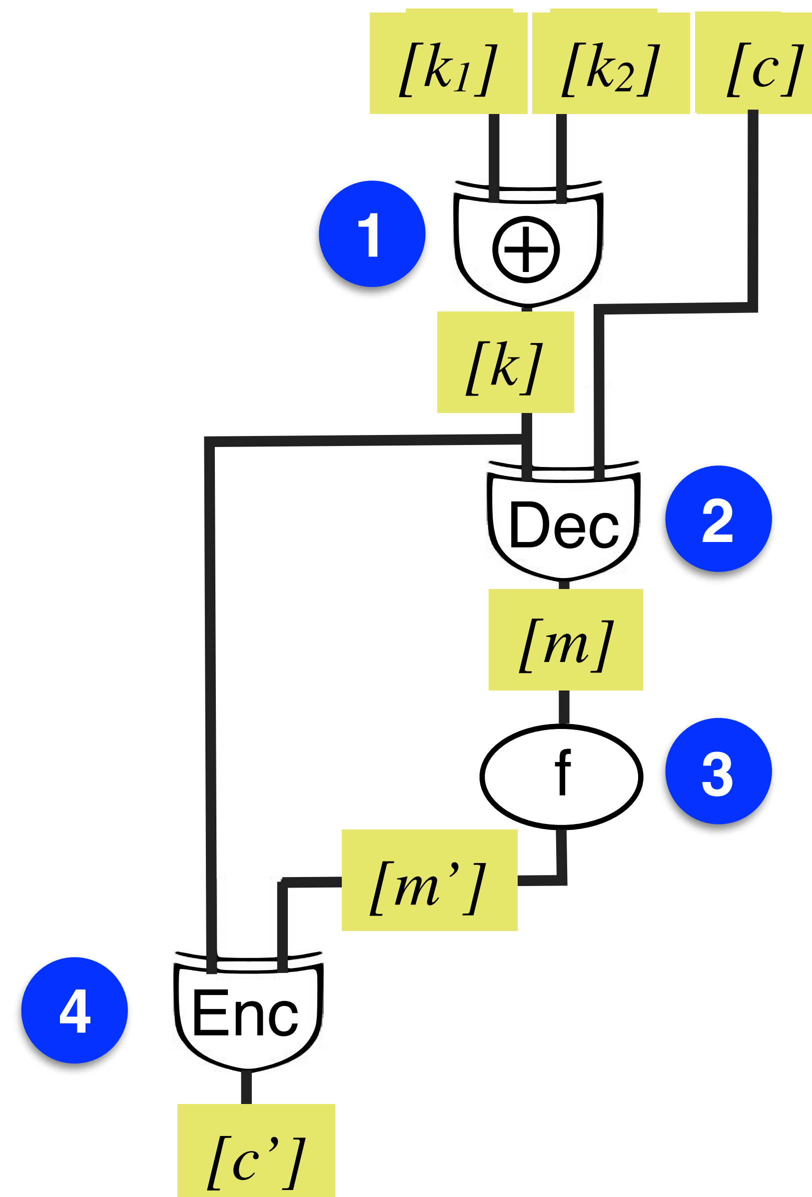
MPC in Action: A Toy Example

$m = m_L || m_R$

Example:

$Enc_k(m) := m \oplus k \quad , f(m) = m_L \oplus m_R || m_R$

$Dec_k(c) := c \oplus k$



Tool: (Additive) Secret Sharing *[s]* of secret *s*

- Choose random s_1, s_2, s_3 s.t. $s_1 \oplus s_2 \oplus s_3 = s$
- Hand s_i to P_i

Any subset gets no info on s

Protocol: Traverse the circuit gate by gate where instead of the wires' values compute sharing of these values

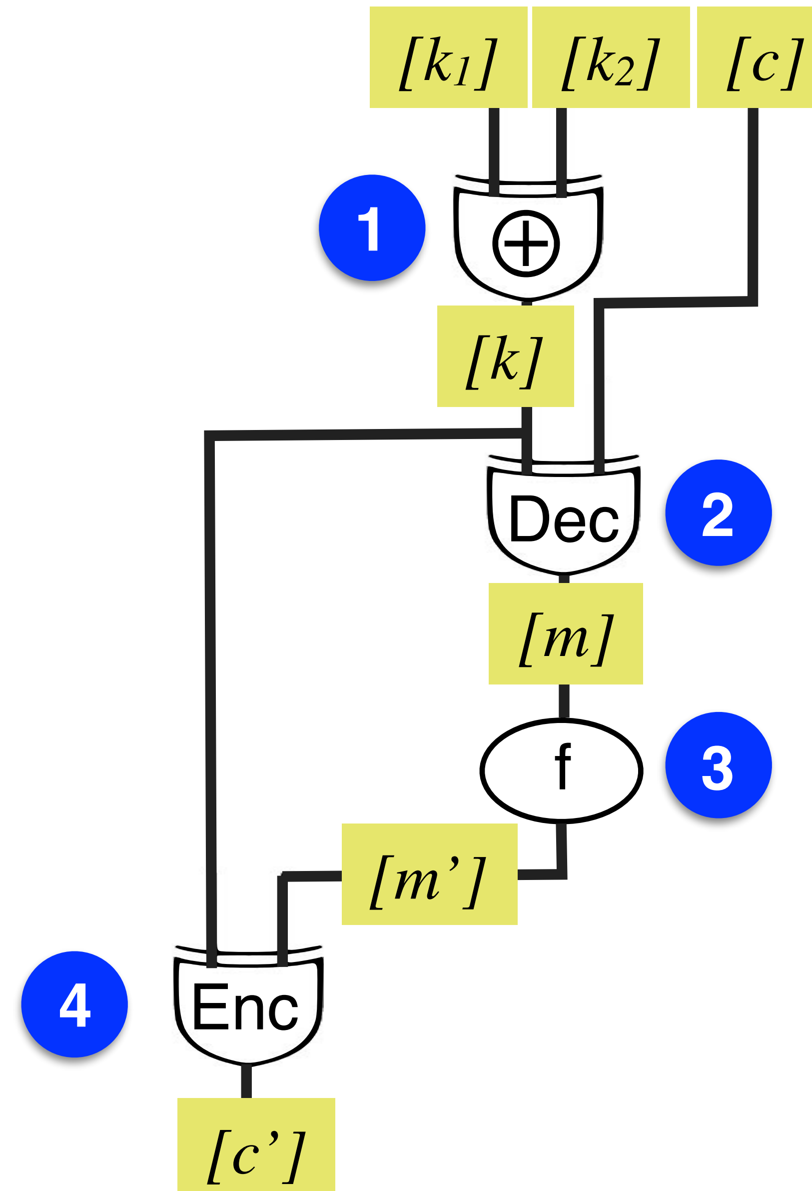
MPC in Action: A Toy Example

$m = m_L || m_R$

Example:

$Enc_k(m) := m \oplus k \quad , f(m) = m_L \oplus m_R || m_R$

$Dec_k(c) := c \oplus k$



MPC in Action: A Toy Example

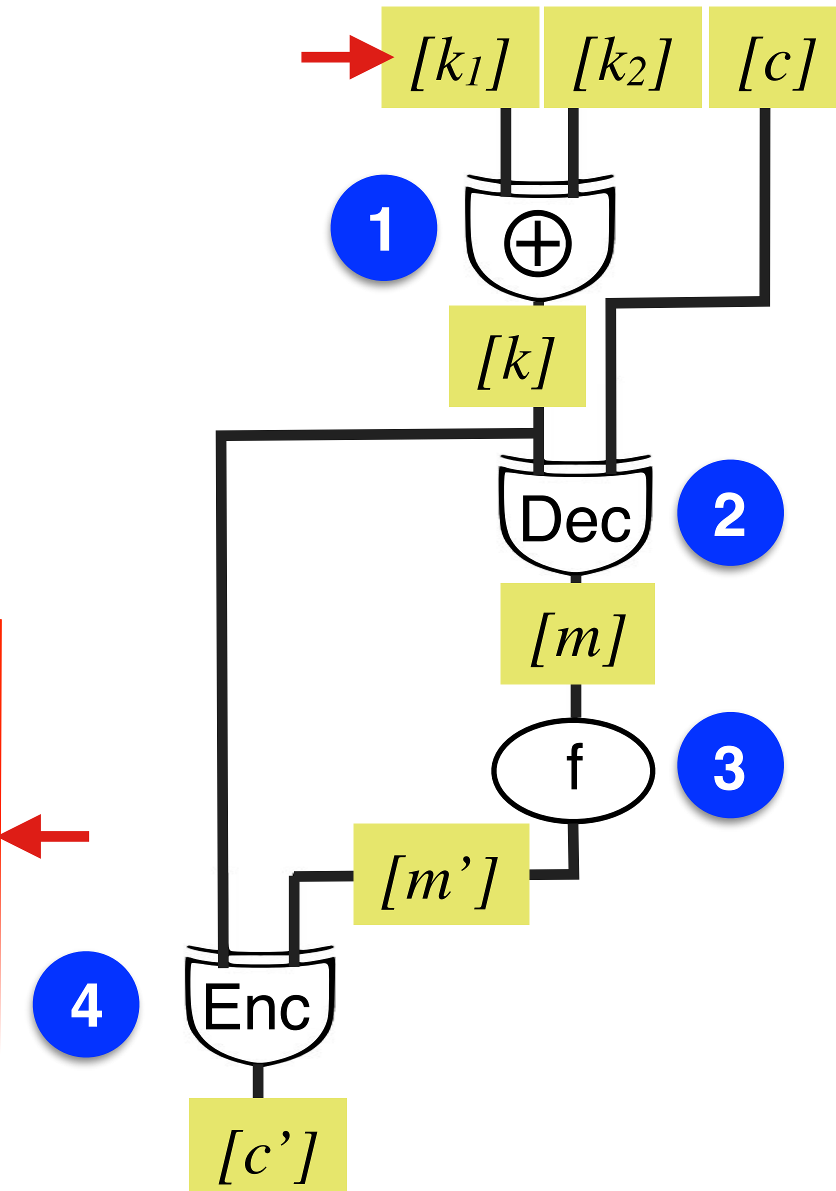
$m = m_L || m_R$

Example:

$Enc_k(m) := m \oplus k$, $f(m) = m_L \oplus m_R || m_R$

$Dec_k(c) := c \oplus k$

	P_1	P_2	P_3	
P_1 (k_1)	k_{11}	k_{12}	k_{13}	$[k_1]$
P_2 (k_2)	k_{21}	k_{22}	k_{23}	$[k_2]$
P_3 (c)	c_1	c_2	c_{13}	$[c]$



MPC in Action: A Toy Example

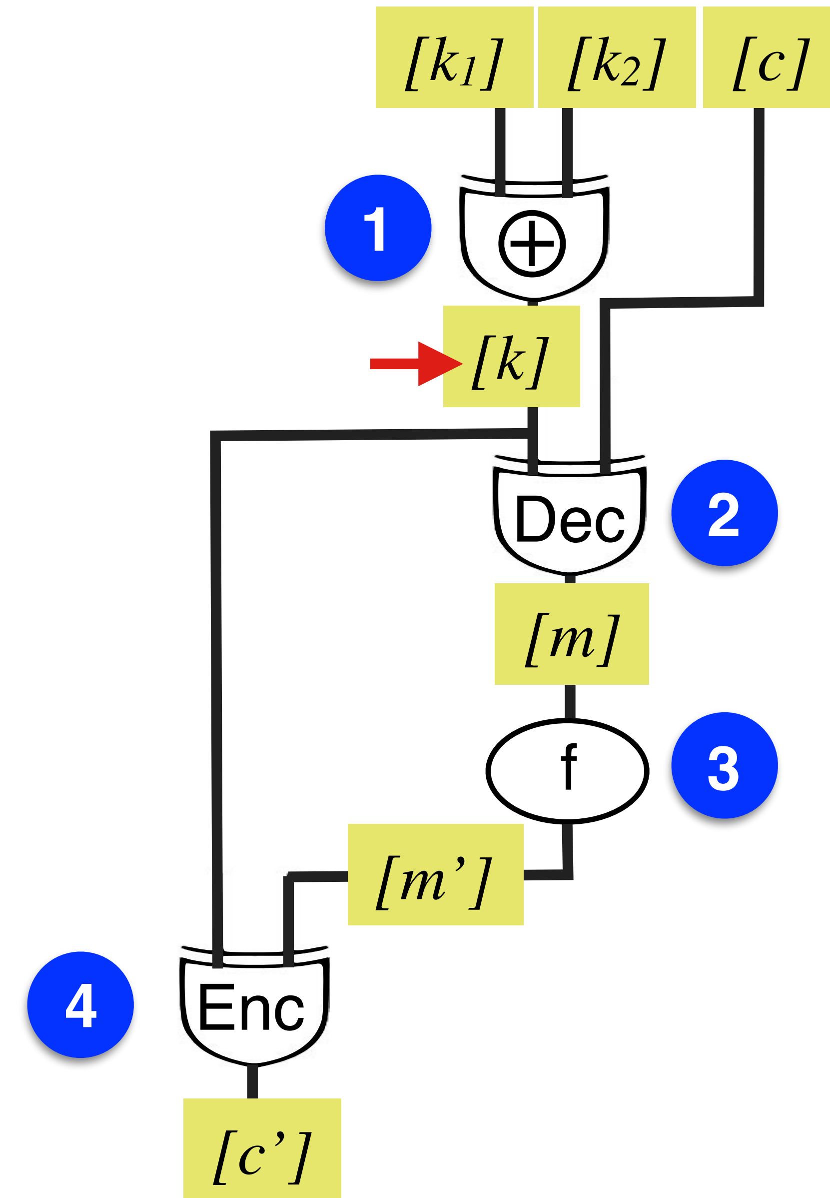
$m = m_L || m_R$

Example:

$$Enc_k(m) := m \oplus k \quad , f(m) = m_L \oplus m_R || m_R$$

$$Dec_k(c) := c \oplus k$$

	P_1	P_2	P_3	
P_1 (k_1)	k_{11}	k_{12}	k_{13}	$[k_1]$
P_2 (k_2)	k_{21}	k_{22}	k_{23}	$[k_2]$
P_3 (c)	c_1	c_2	c_{13}	$[c]$
1	$k_{11} \oplus k_{21}$	$k_{12} \oplus k_{22}$	$k_{13} \oplus k_{23}$	$[k_1 \oplus k_2] = [k]$



MPC in Action: A Toy Example

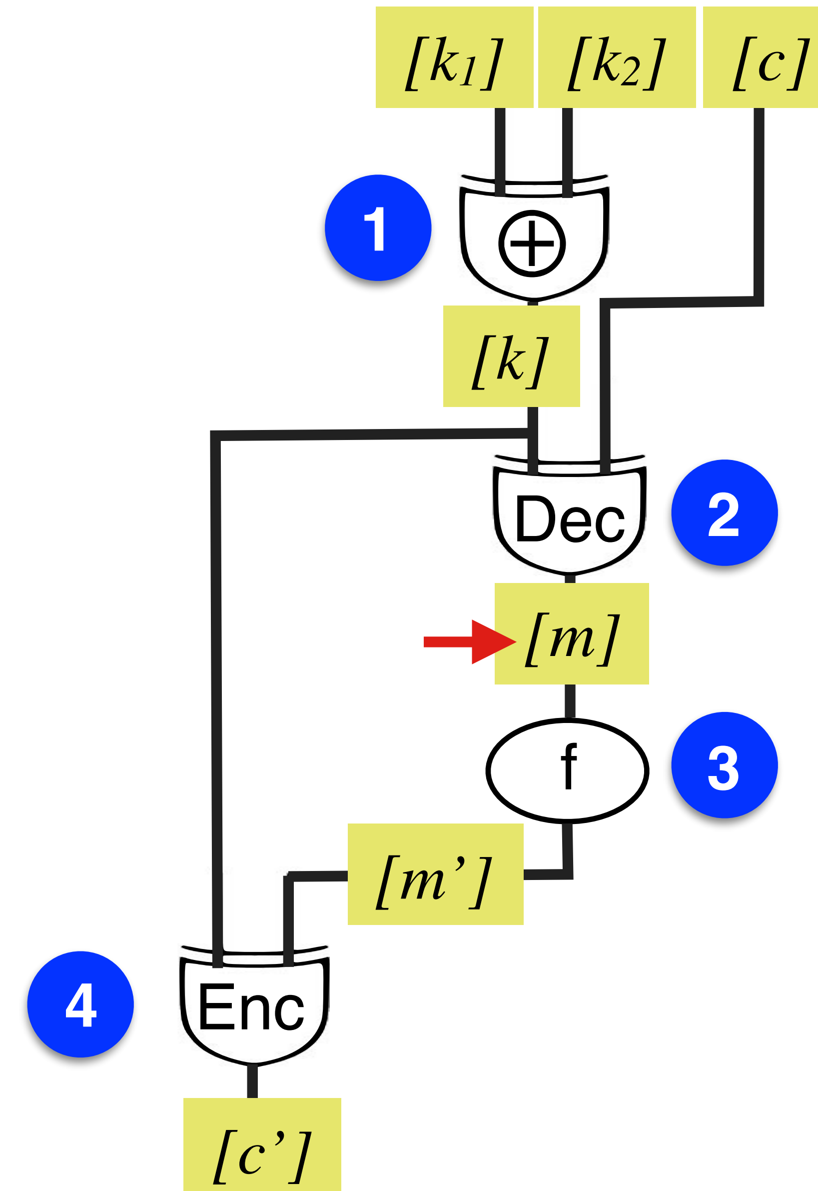
$m = m_L || m_R$

Example:

$$Enc_k(m) := m \oplus k \quad , f(m) = m_L \oplus m_R || m_R$$

$$Dec_k(c) := c \oplus k$$

	P_1	P_2	P_3	
P_1 (k_1)	k_{11}	k_{12}	k_{13}	$[k_1]$
P_2 (k_2)	k_{21}	k_{22}	k_{23}	$[k_2]$
P_3 (c)	c_1	c_2	c_{13}	$[c]$
1	$k_{11} \oplus k_{21}$	$k_{12} \oplus k_{22}$	$k_{13} \oplus k_{23}$	$[k_1 \oplus k_2] = [k]$
2	$c_1 \oplus k_{11} \oplus k_{21}$	$c_2 \oplus k_{12} \oplus k_{22}$	$c_3 \oplus k_{13} \oplus k_{23}$	$[c + k_1 \oplus k_2] = [m]$



MPC in Action: A Toy Example

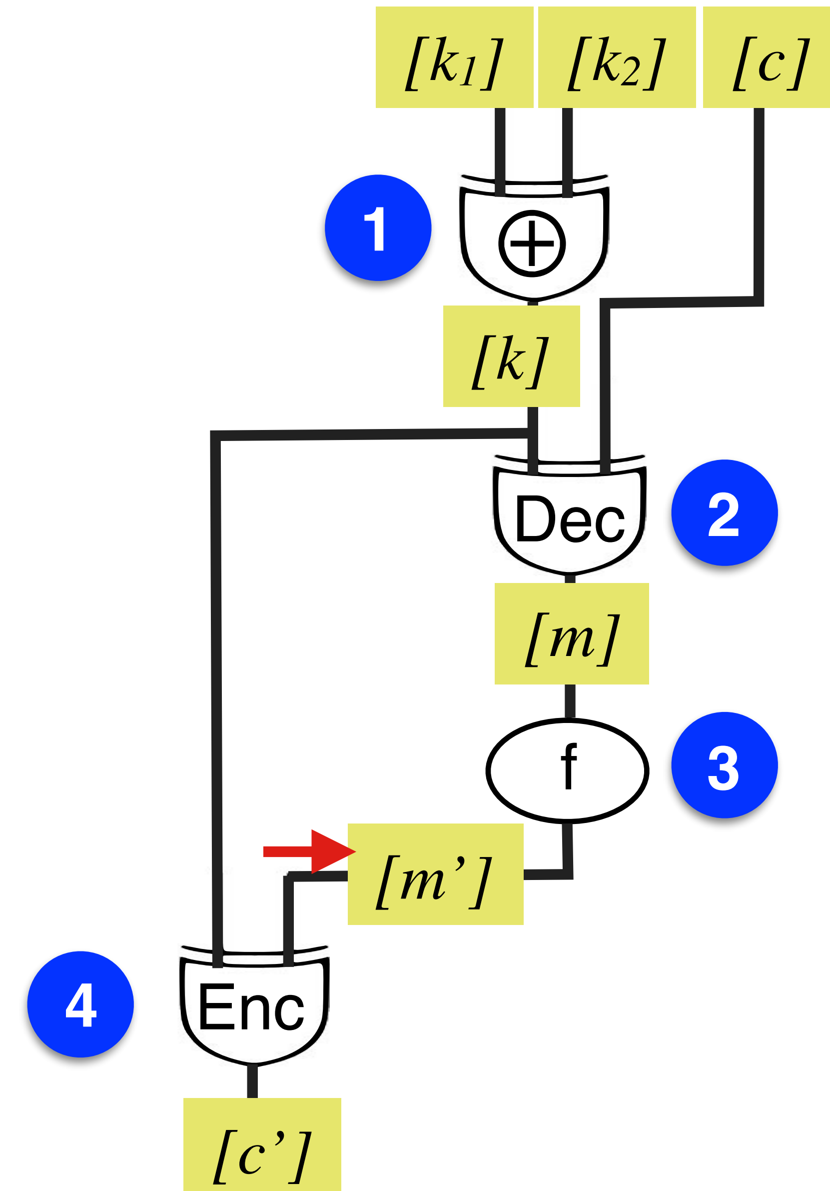
Example:

$m = m_L || m_R$

$$Enc_k(m) := m \oplus k \quad , f(m) = m_L \oplus m_R || m_R$$

$$Dec_k(c) := c \oplus k$$

	P_1	P_2	P_3	
P_1 (k_1)	k_{11}	k_{12}	k_{13}	$[k_1]$
P_2 (k_2)	k_{21}	k_{22}	k_{23}	$[k_2]$
P_3 (c)	c_1	c_2	c_{13}	$[c]$
1	$k_{11} \oplus k_{21}$	$k_{12} \oplus k_{22}$	$k_{13} \oplus k_{23}$	$[k_1 \oplus k_2] = [k]$
2	$c_1 \oplus k_{11} \oplus k_{21}$	$c_2 \oplus k_{12} \oplus k_{22}$	$c_3 \oplus k_{13} \oplus k_{23}$	$[c + k_1 \oplus k_2] = [m]$
3	$m'_1 = f(\ )$	$m'_2 = f(\ )$	$m'_3 = f(\ )$	$[f(m)] = [m']$



MPC in Action: A Toy Example

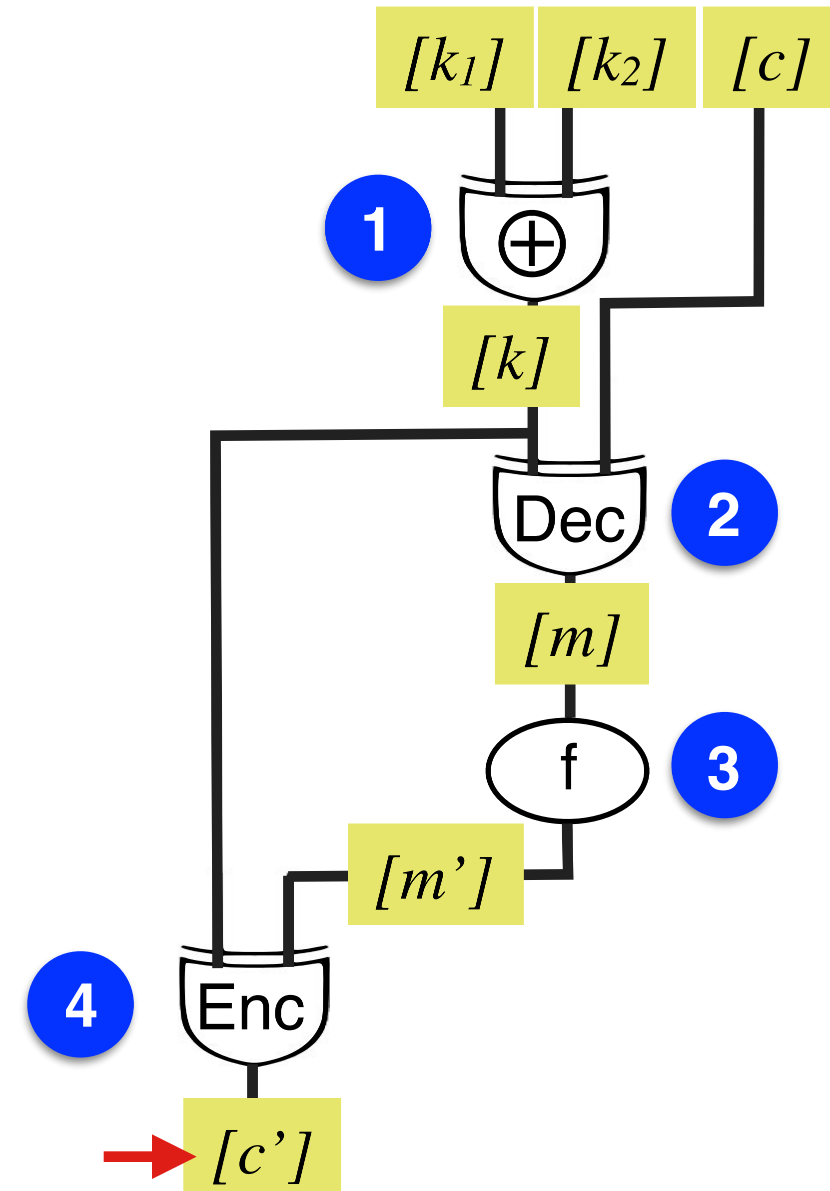
$m = m_L || m_R$

Example:

$Enc_k(m) := m \oplus k$, $f(m) = m_L \oplus m_R || m_R$

$Dec_k(c) := c \oplus k$

	P_1	P_2	P_3	
P_1 (k_1)	k_{11}	k_{12}	k_{13}	$[k_1]$
P_2 (k_2)	k_{21}	k_{22}	k_{23}	$[k_2]$
P_3 (c)	c_1	c_2	c_{13}	$[c]$
1	$k_{11} \oplus k_{21}$	$k_{12} \oplus k_{22}$	$k_{13} \oplus k_{23}$	$[k_1 \oplus k_2] = [k]$
2	$c_1 \oplus k_{11} \oplus k_{21}$	$c_2 \oplus k_{12} \oplus k_{22}$	$c_3 \oplus k_{13} \oplus k_{23}$	$[c + k_1 \oplus k_2] = [m]$
3	$m'_1 = f(\ )$	$m'_2 = f(\ )$	$m'_3 = f(\ )$	$[f(m)] = [m']$
4	$m'_1 \oplus k_{11} \oplus k_{21}$	$m'_2 \oplus k_{12} \oplus k_{22}$	$m'_3 \oplus k_{13} \oplus k_{23}$	$[m' + k] = [c']$



MPC in Action: A Toy Example

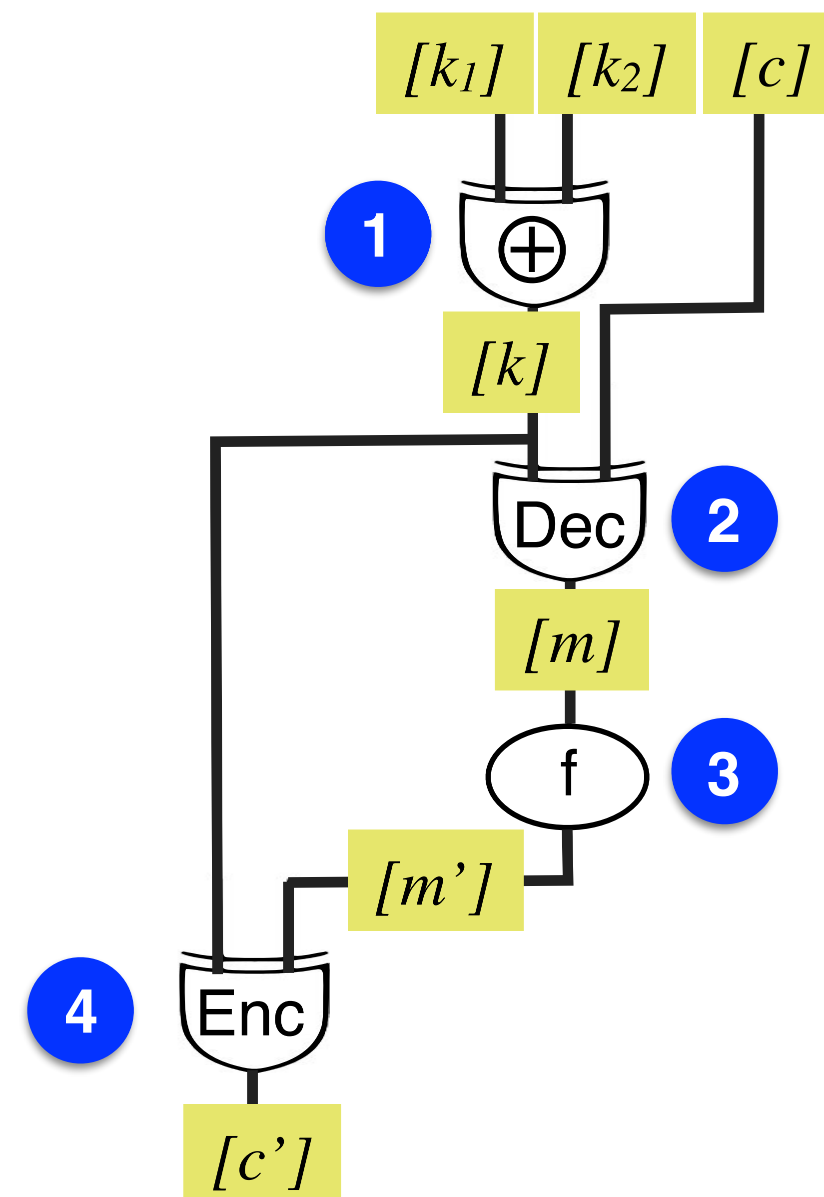
Example:

$m = m_L || m_R$

$Enc_k(m) := m \oplus k \quad , f(m) = m_L \oplus m_R || m_R$

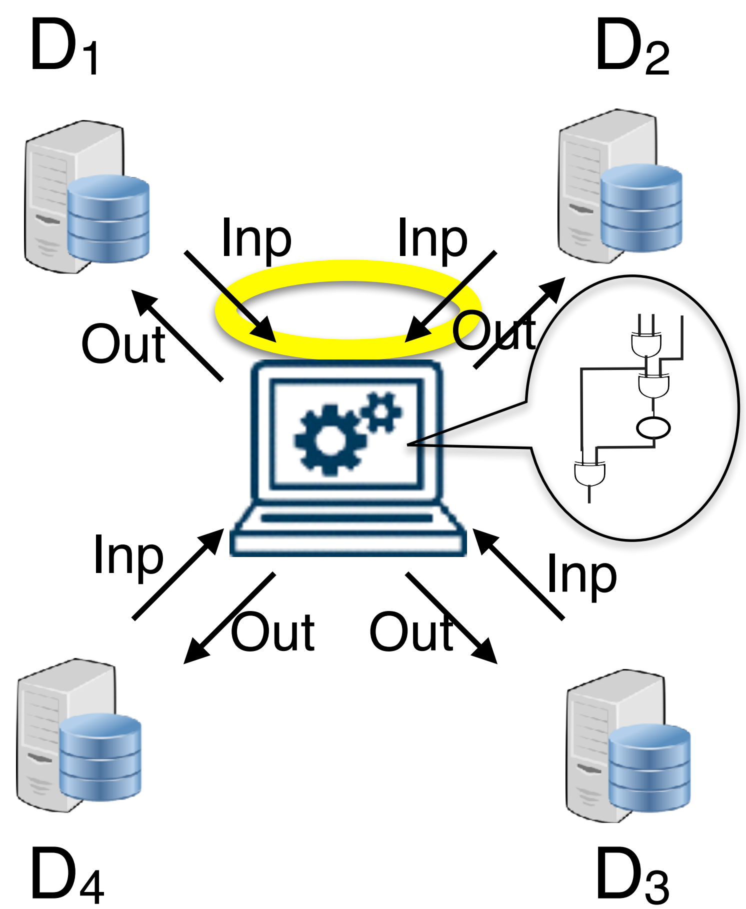
$Dec_k(c) := c \oplus k$

	P_1	P_2	P_3	
P_1 (k_1)	k_{11}	k_{12}	k_{13}	$[k_1]$
P_2 (k_2)	k_{21}	k_{22}	k_{23}	$[k_2]$
P_3 (c)	c_1	c_2	c_{13}	$[c]$
1	$k_{11} \oplus k_{21}$	$k_{12} \oplus k_{22}$	$k_{13} \oplus k_{23}$	$[k_1 \oplus k_2] = [k]$
2	$c_1 \oplus k_{11} \oplus k_{21}$	$c_2 \oplus k_{12} \oplus k_{22}$	$c_3 \oplus k_{13} \oplus k_{23}$	$[c + k_1 \oplus k_2] = [m]$
3	$m'_1 = f(\ )$	$m'_2 = f(\ )$	$m'_3 = f(\ )$	$[f(m)] = [m']$
4	$m'_1 \oplus k_{11} \oplus k_{21}$	$m'_2 \oplus k_{12} \oplus k_{22}$	$m'_3 \oplus k_{13} \oplus k_{23}$	$[m' + k] = [c']$



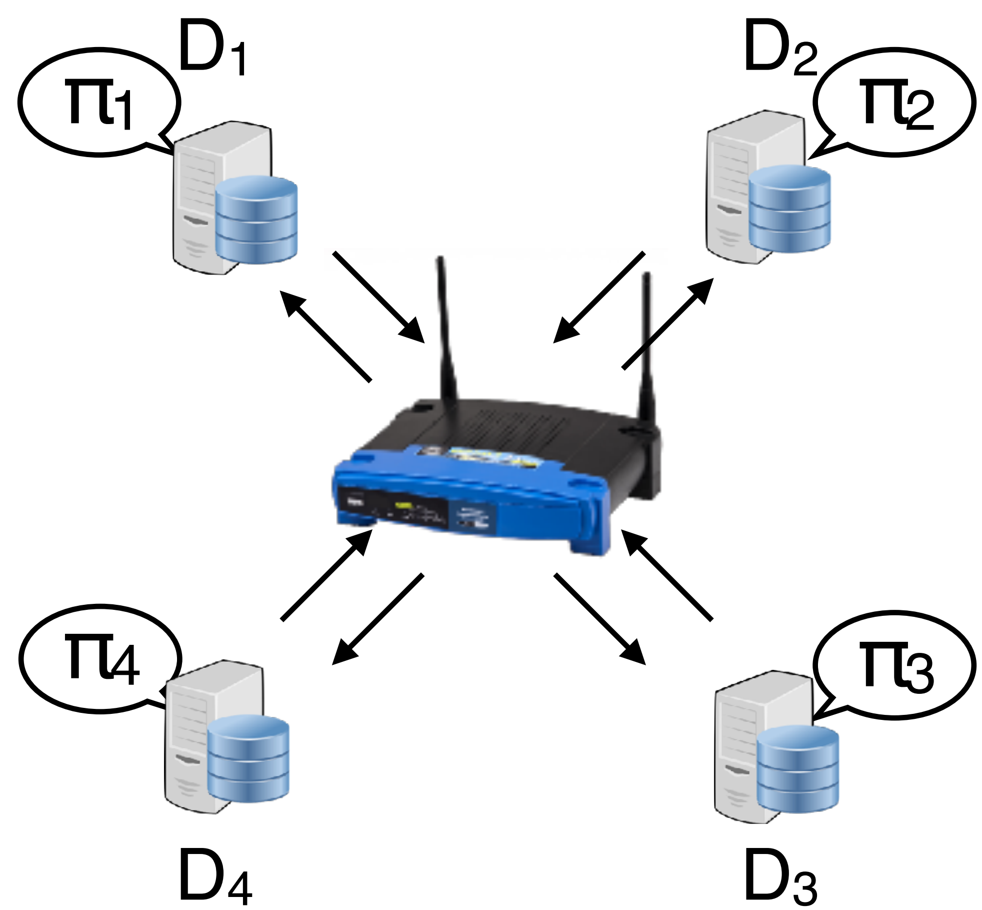
Back to MPC Security

Ideal World: Specification



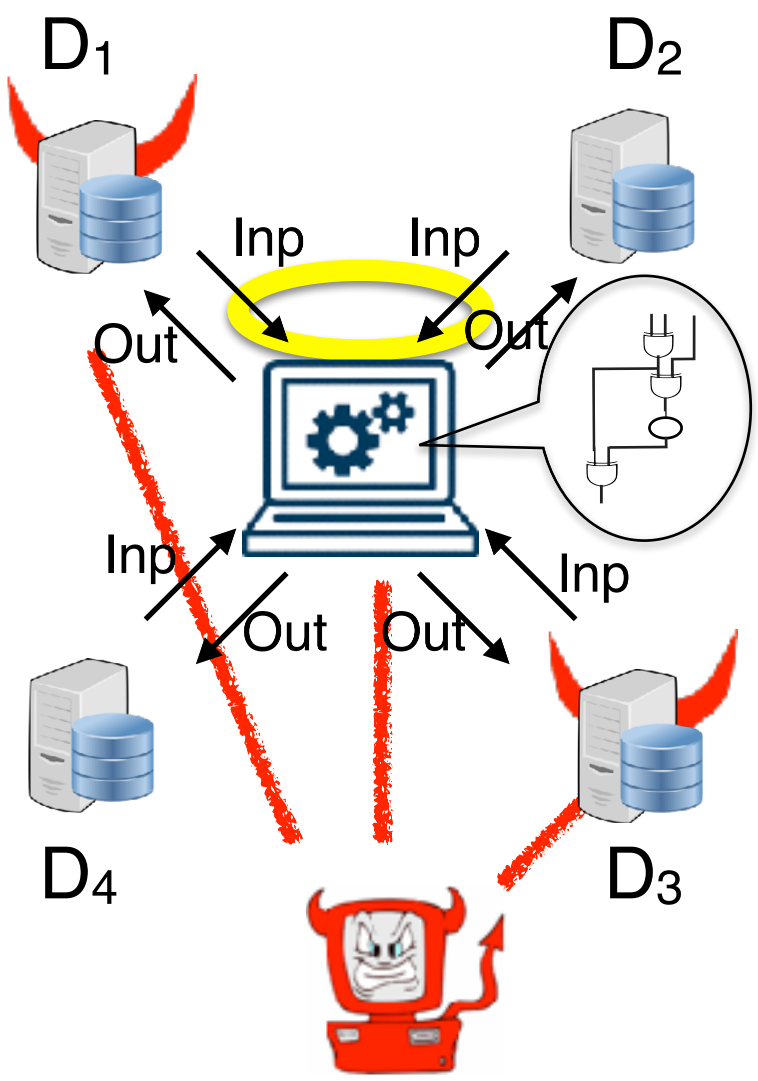
≈

Real World: Protocol



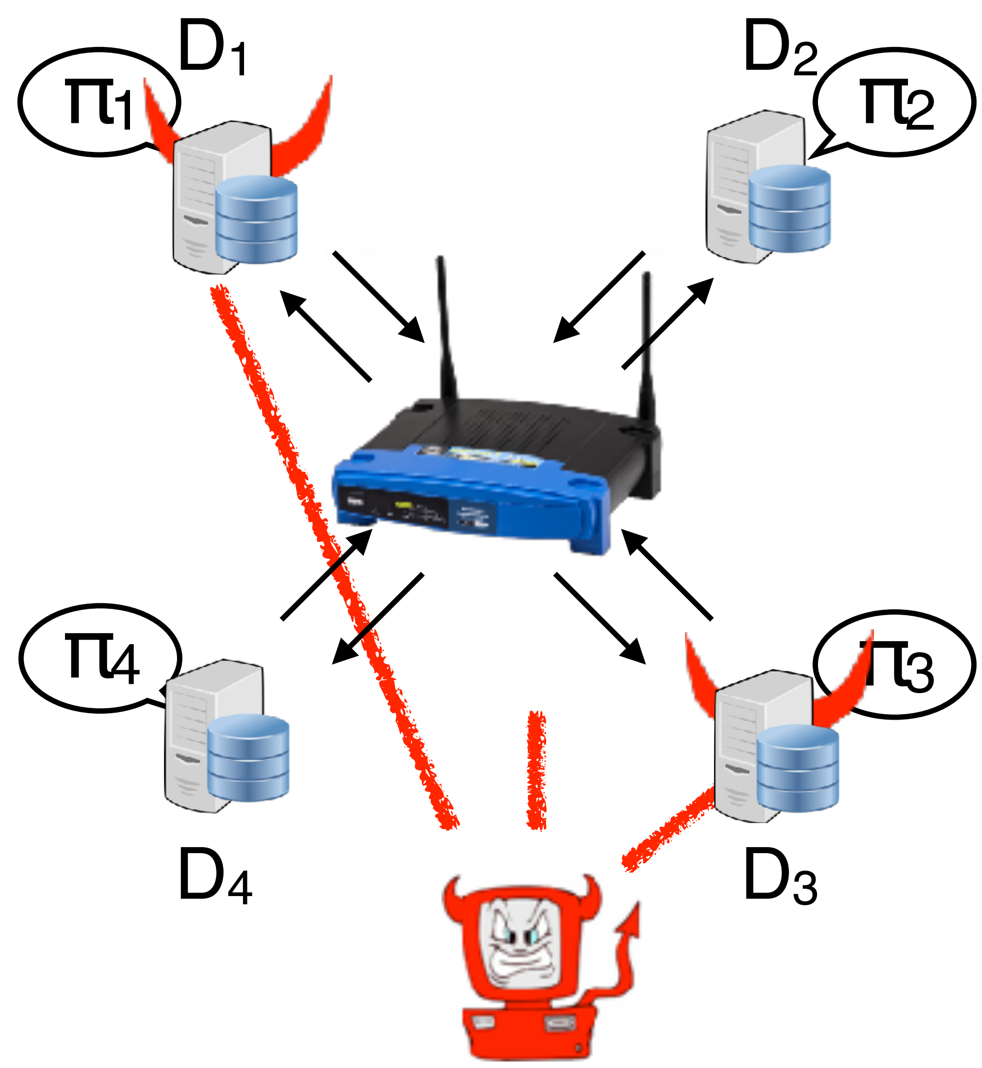
Back to MPC Security

Ideal World: Specification



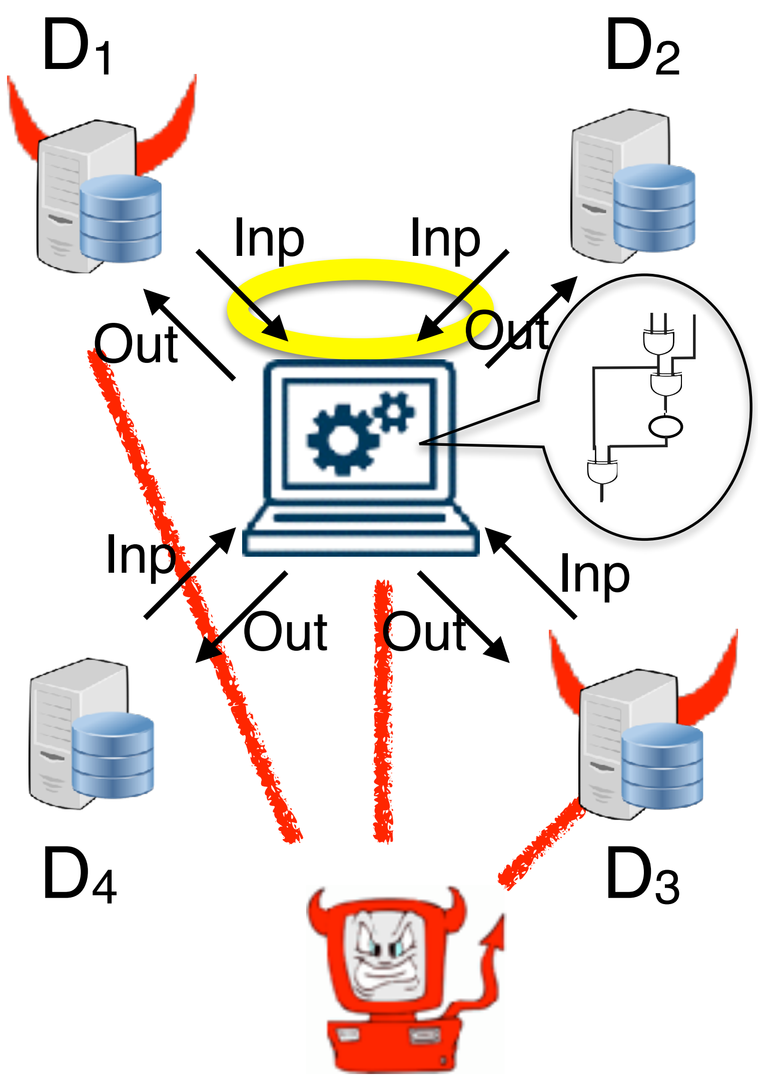
≈

Real World: Protocol

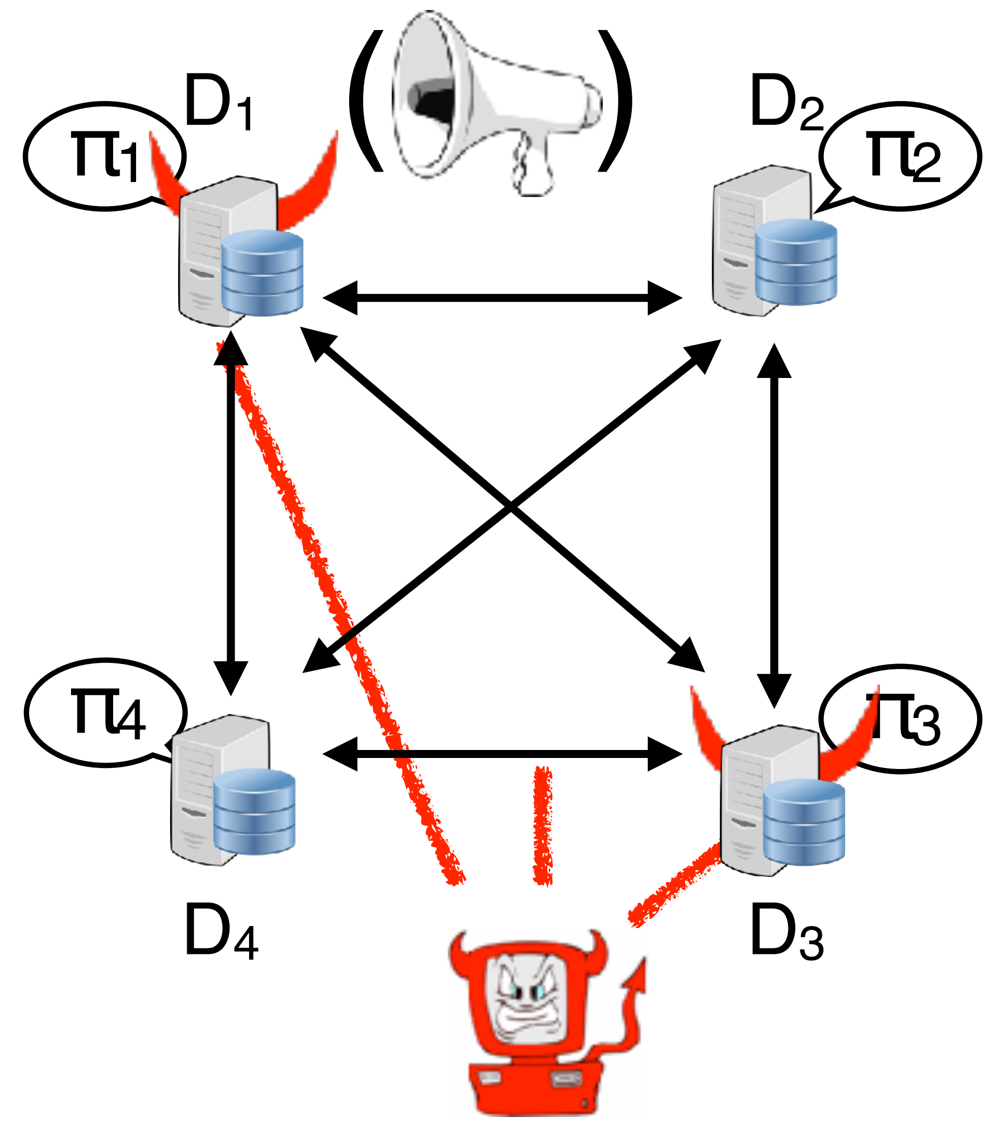


Back to MPC Security

Ideal World: Specification



Real World: Protocol



$\approx$

Model

- n players
- Computation over $(\mathbb{F}, \oplus, \otimes)$ — E.g. $(\mathbb{Z}_p, +, \cdot)$
- Communication: Point-to-point secure channels (and Broadcast)
- Synchrony: Messages sent in round i are delivered by round i+1

The adversary

Corruption Types

- Passive (semi-honest): Corrupted parties follow their protocol but try to learn more information than allowed from their joint view
- Active (malicious): Corrupted parties misbehave arbitrarily

Computing Power

- Unbounded (information theoretic security): The adversary can perform arbitrary (even exponential) computation
  - Security is unconditional
- Bounded (Computational or cryptographic security): The adversary can perform polynomial-time computation
  - Security is guaranteed under hardness assumptions, e.g., DDH, RSA, Factoring, …

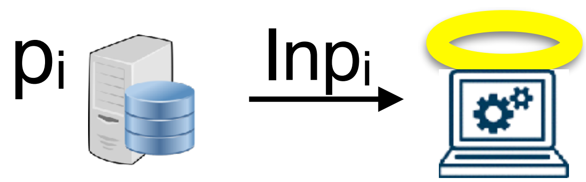
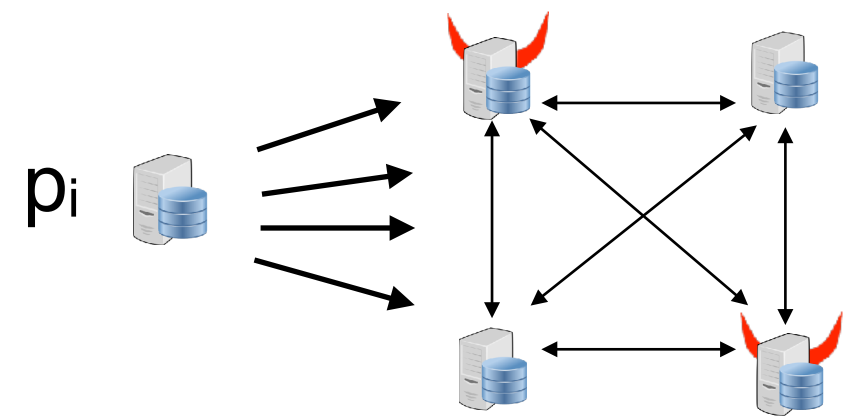
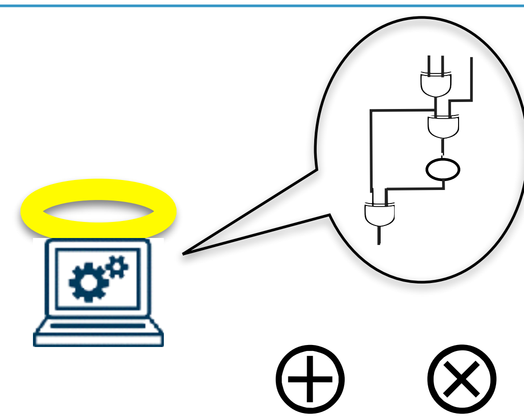
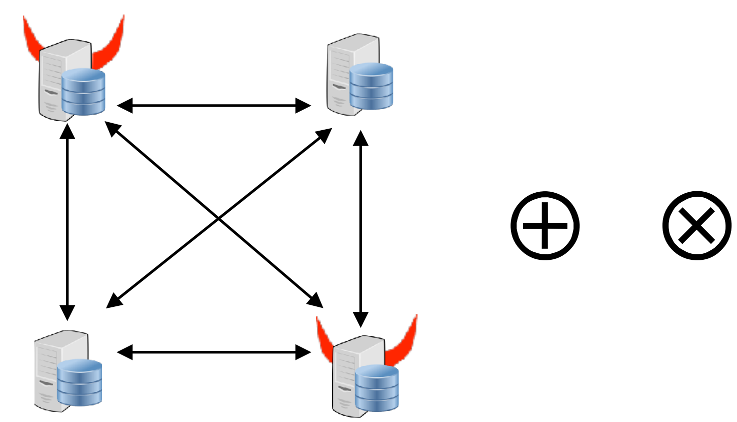
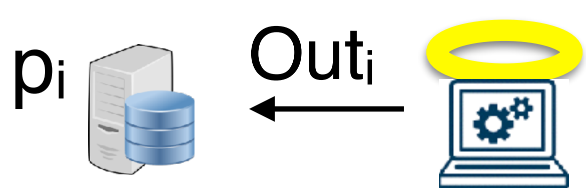
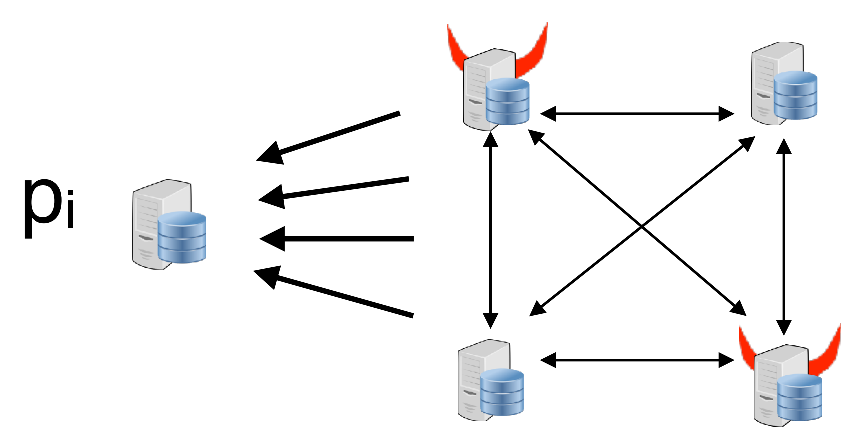
Known Feasibility Results

Adv. Type	Security	Corruption Bound	Requires
semi-honest (passive)	Information theoretic (IT)	t<n/2 [BGW88,CCD88]	Sec. channels
	Computational	t<n [GMW87]	Sec. channels + OT
malicious (active)	information theoretic	t<n/3 [BGW88,CCD88]	Sec. channels
	computational (or IT w. negligible error)	t<n/2 [GMW87,RB89]	Broadcast
	computational without fairness	t<n [GMW87]	Broadcast + OT

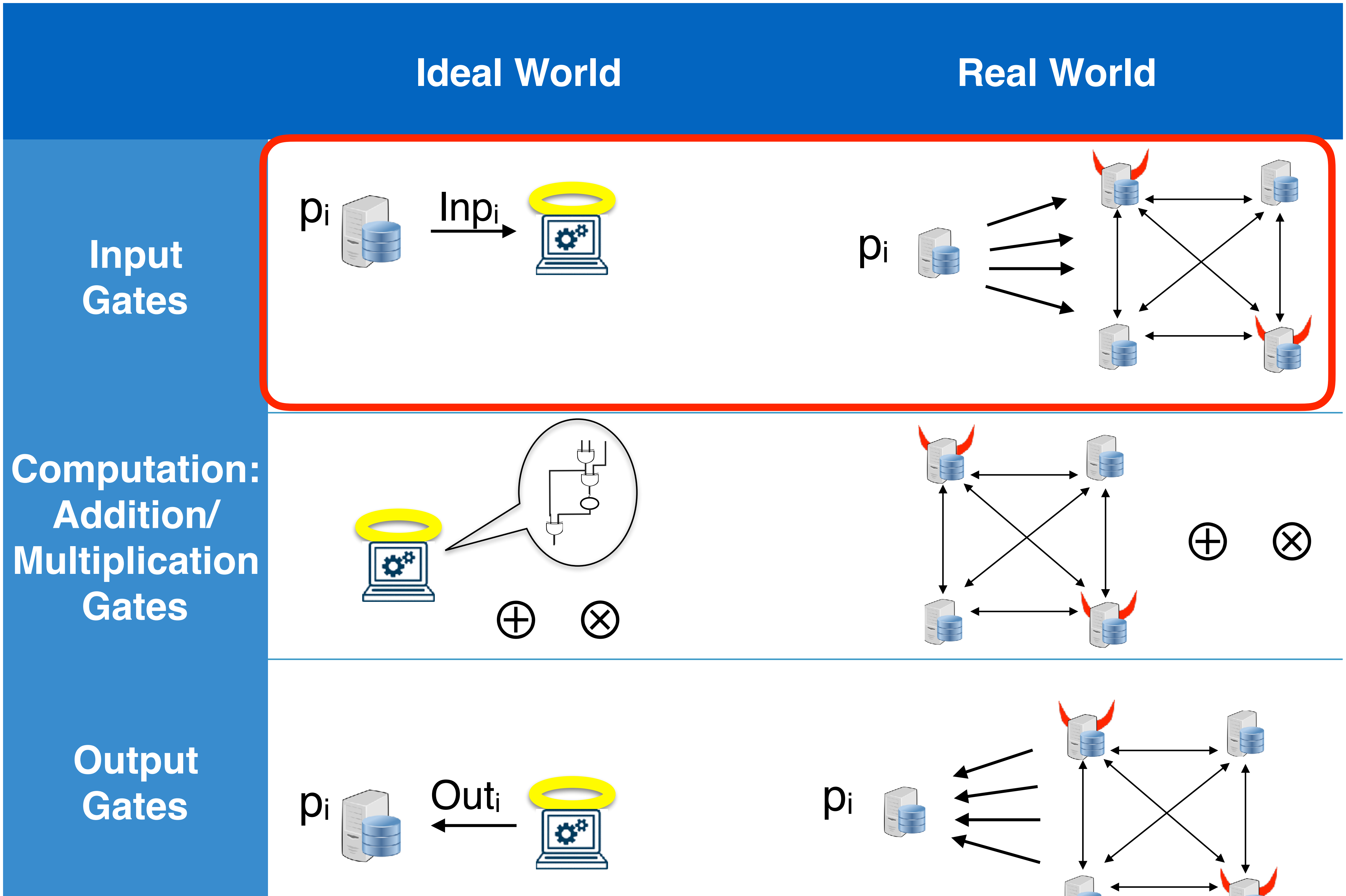
Known Feasibility Results

Adv. Type	Security	Corruption Bound	Requires
semi-honest (passive)	Information theoretic (IT)	t<n/2 [BGW88,CCD88]	Sec. channels
	Computational	t<n [GMW87]	Sec. channels + OT
malicious (active)	information theoretic	t<n/3 [BGW88,CCD88]	Sec. channels
	computational (or IT w. negligible error)	t<n/2 [GMW87,RB89]	Broadcast
	computational without fairness	t<n [GMW87]	Broadcast + OT

MPC Goal

	Ideal World	Real World
Input Gates		
Computation: Addition/ Multiplication Gates		
Output Gates		

MPC Goal



Ideal World
Real World
Input Gates
pi
Inpi
pi
Computation: Addition/ Multiplication Gates
⊕ ⊗
⊕ ⊗
Output Gates
pi
Outi
pi

Secret Sharing (Informal)

A secret-sharing scheme allows an honest dealer D to distribute a secret s among players in a set P, such that

- any *non-qualified* subset of players has no information about s,
- every *qualified* subset of players can collaboratively reconstruct the secret.

Threshold Secret Sharing

Secret Sharing: A *t-out-of-n* secret sharing scheme for $P=\{p_1, \ldots, p_n\}$ consists of a pair of protocols: $(\mathrm{Share}, \mathrm{Reconstruct})$ with the following properties

- Share allows a Dealer D to distribute a given value s among the parties in P. It is probabilistic and uses secure channels to distribute the shares.
- $\mathrm{Reconstruct}$ allows to later on reconstruct the shared value.

Threshold Secret Sharing

Secret Sharing: A *t-out-of-n* secret sharing scheme for $P=\{p_1, \ldots, p_n\}$ consists of a pair of protocols: (Share, Reconstruct) with the following properties

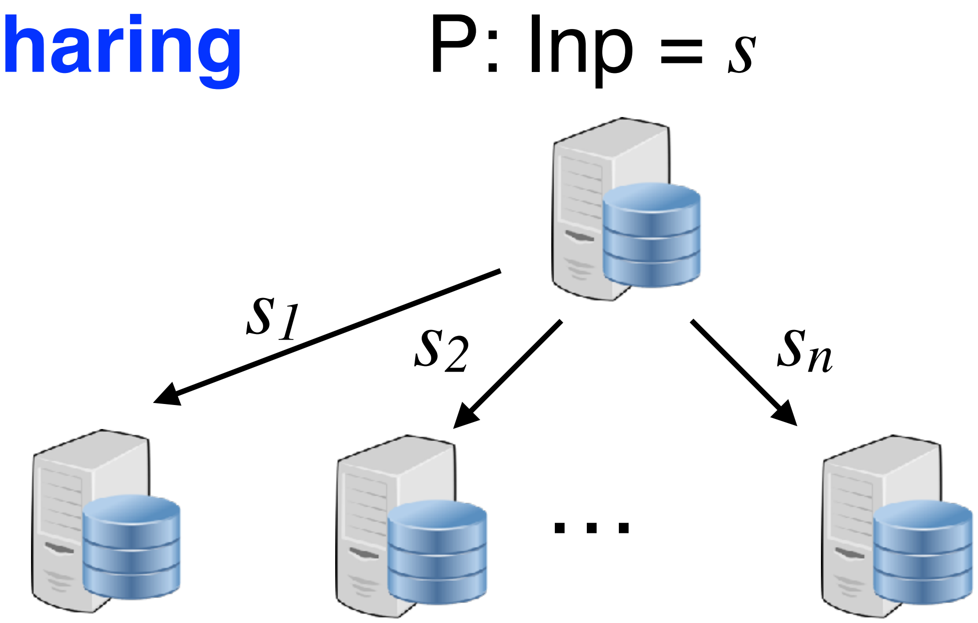
- Share allows a Dealer D to distribute a given value s among the parties in P. It is probabilistic and uses secure channels to distribute the shares.
- Reconstruct allows to later on reconstruct the shared value.

Security properties:

- **(correctness)** Given the shares of any t parties, *Reconstruct* should output the secret s.
- **(t-privacy)** The shares of any t-1 parties include not information about s.

Threshold Secret Sharing

Example: (n-out-of-n) Additive Secret Sharing

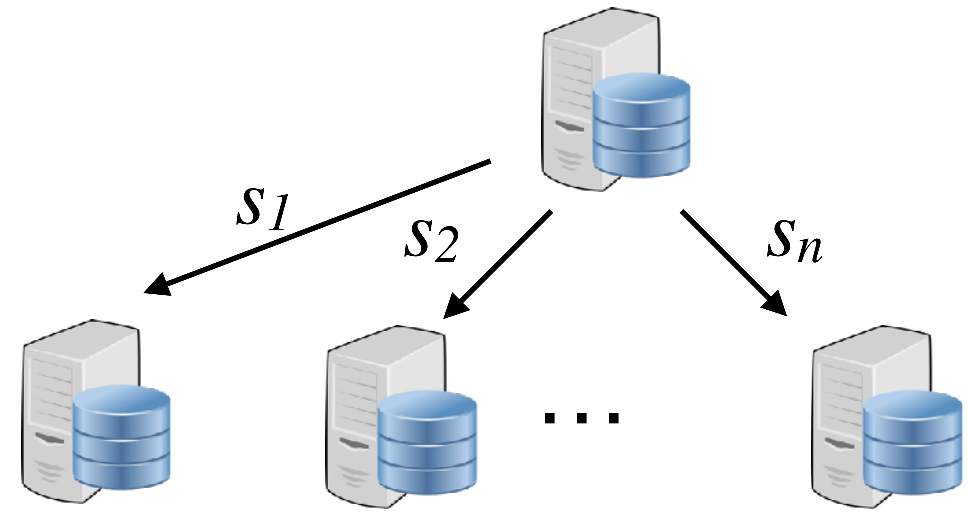


- Share: Dealer p sharing s:
  - Choose n values $s_1, \ldots, s_n \in \mathbb{Z}_p$ uniformly at random s.t. $\sum_{i=1}^{n} s_i = s \pmod{p}$
  - Send s_i to player p_i
- Reconstruct:
  - The parties add their shares to recover s

Threshold Secret Sharing

Example: (n-out-of-n) Additive Secret Sharing

P: Inp = s

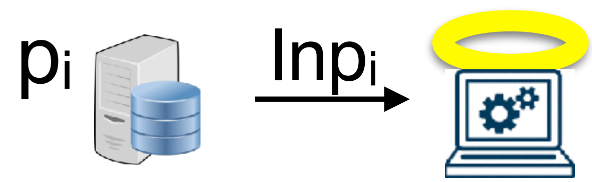
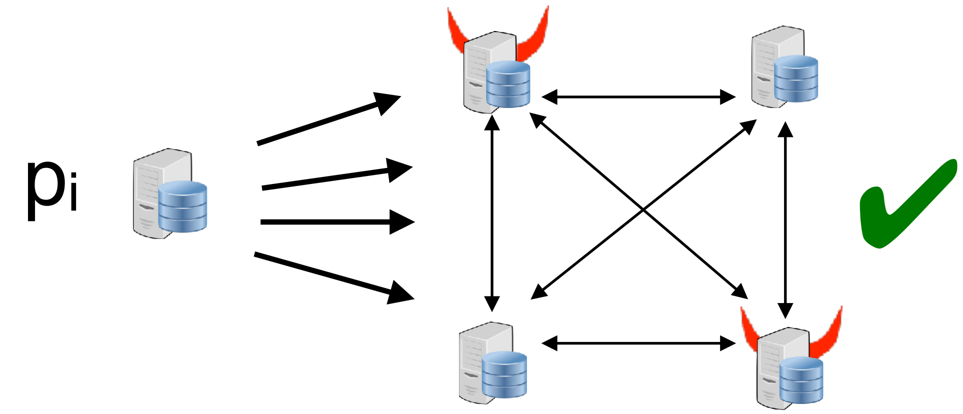
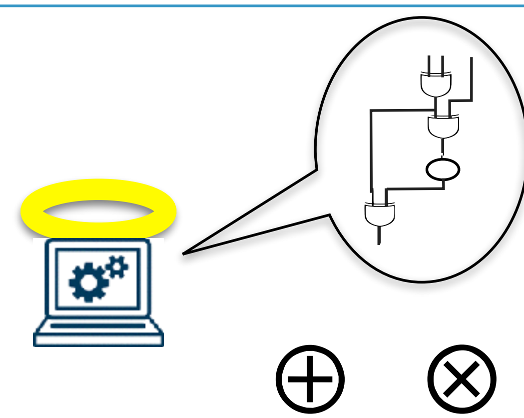
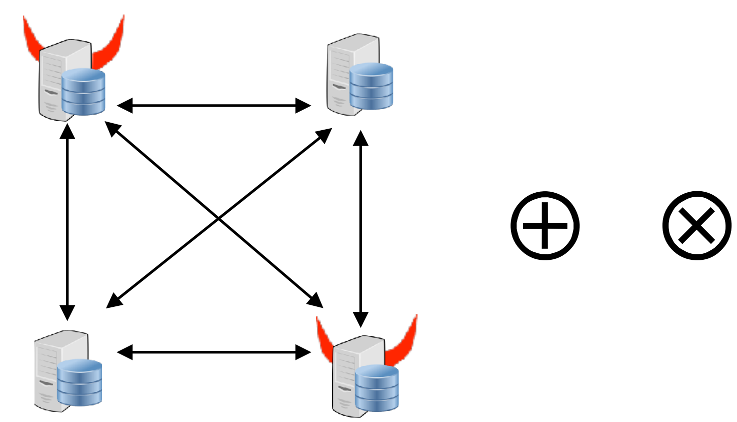
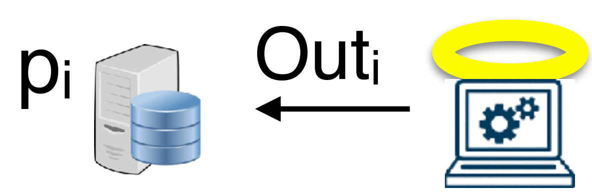
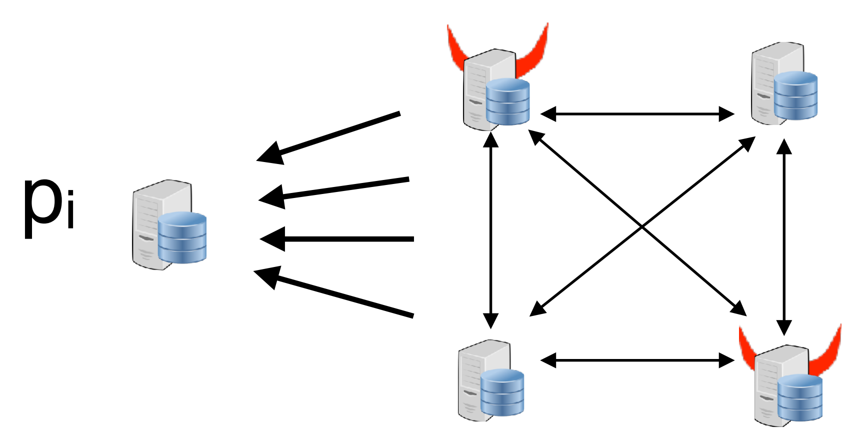


- Share: Dealer p sharing s:
  - Choose n values $s_1, \ldots, s_n \in \mathbb{Z}_p$ uniformly at random s.t. $\sum_{i=1}^{n} s_i = s \pmod{p}$
  - Send s_i to player p_i
- Reconstruct:
  - The parties add their shares to recover s

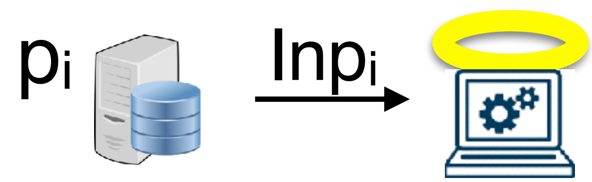
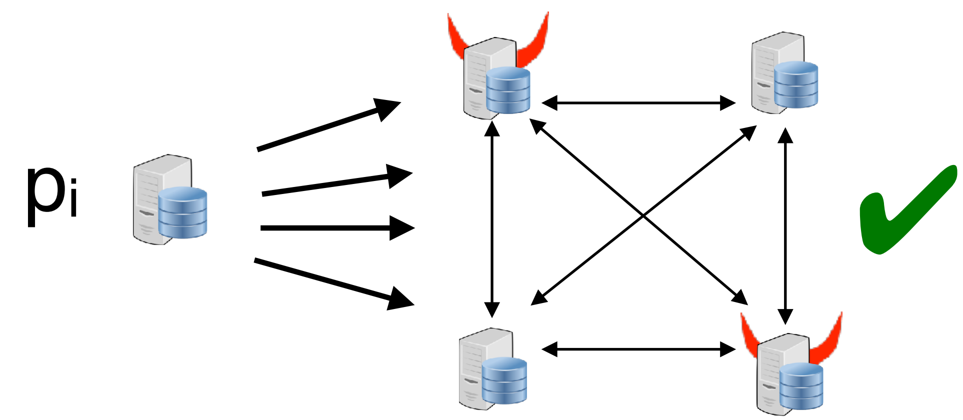
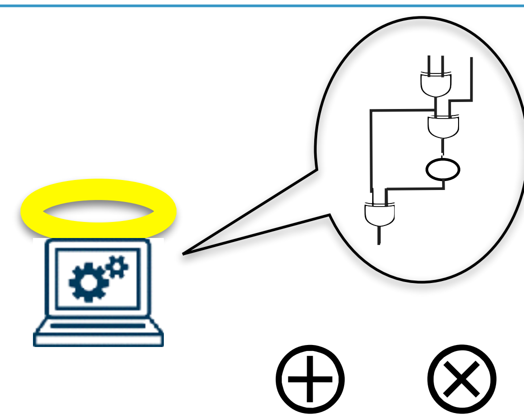
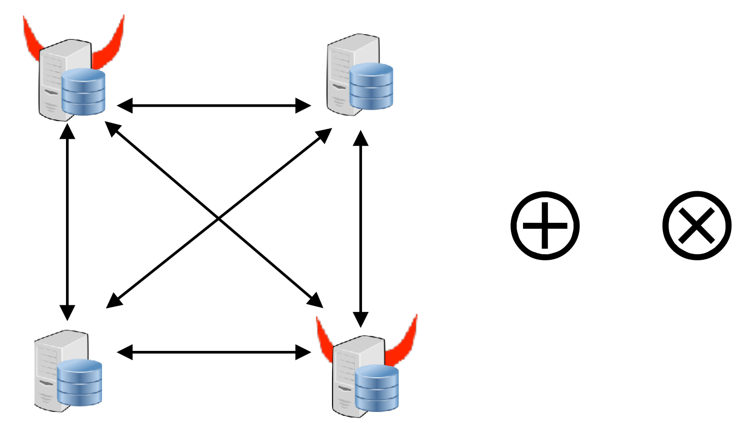
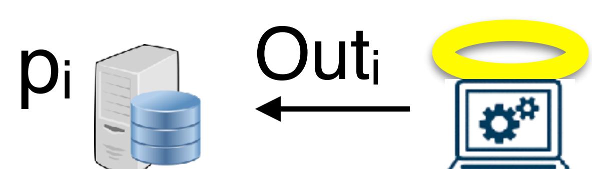
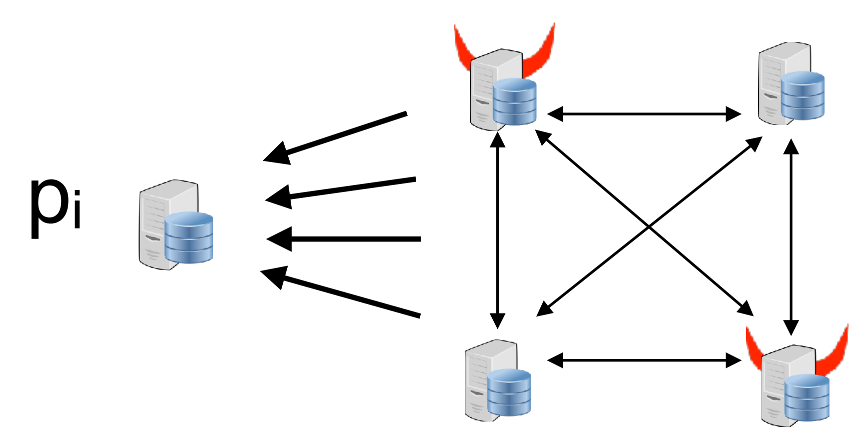
Security:

- **(correctness)** Given the shares of any n parties, *Reconstruct* outputs the secret s by summing them.
- **(n-privacy)** The shares of any n-1 parties include not information about s since the missing share perfectly blinds the secret.

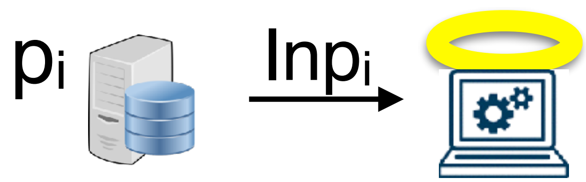
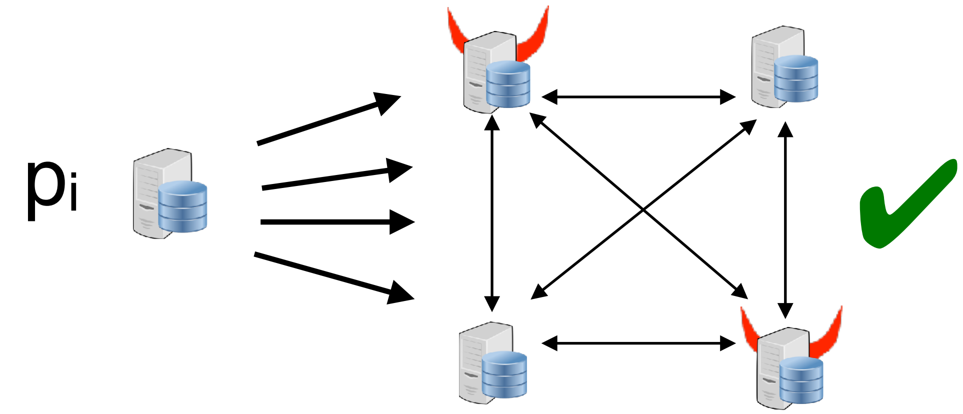
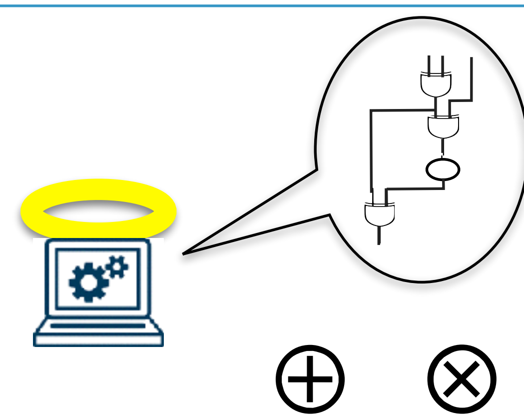
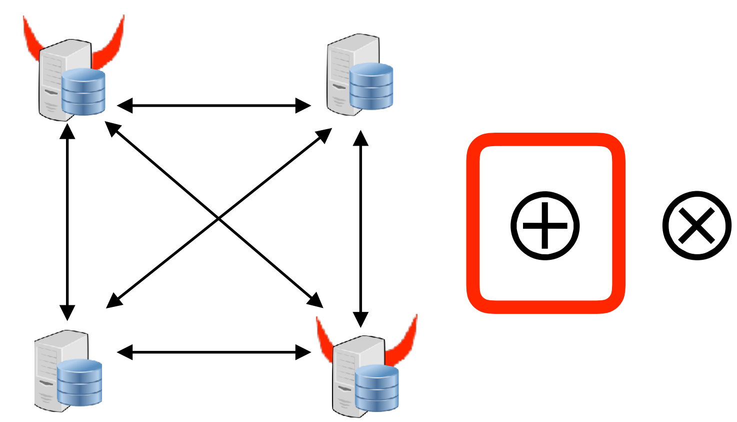
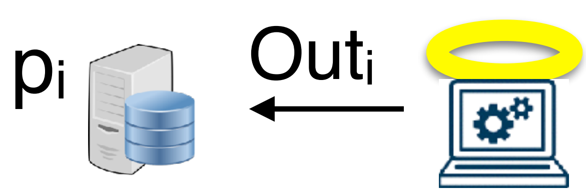
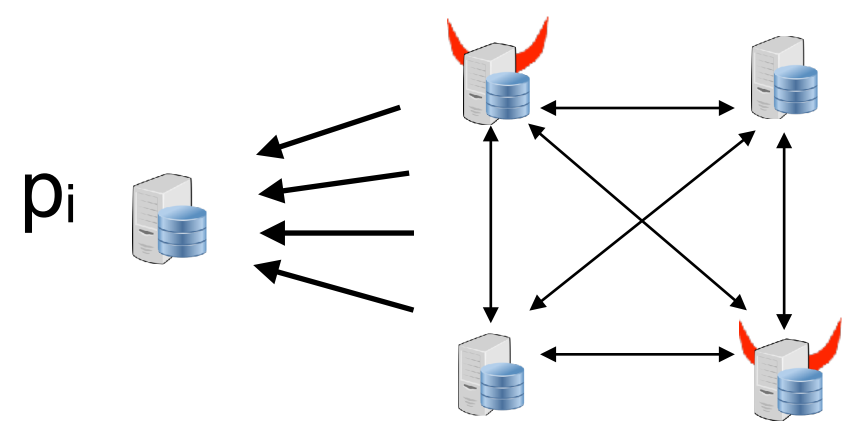
MPC Goal

	Ideal World	Real World
Input Gates		
Computation: Addition/ Multiplication Gates		
Output Gates		

MPC Goal

	Ideal World	Real World
Input Gates		
Computation: Addition/ Multiplication Gates		
Output Gates		

MPC Goal

	Ideal World	Real World
Input Gates		
Computation: Addition/ Multiplication Gates		
Output Gates		

Linear Secret Sharing

We say that a sharing $(s_1, \ldots, s_n)$ is **linear** if the shares are computed as a linear function of s and random values. That is if there exists a **constant** $n \times (m+1)$ matrix A such that for random values $r_1, \ldots, r_m$:

$$\begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} A_{10} & A_{11} & \cdots & A_{1m} \\ \vdots & & & \vdots \\ A_{n0} & A_{n1} & \cdots & A_{nm} \end{bmatrix} \begin{bmatrix} s \\ r_1 \\ \vdots \\ r_m \end{bmatrix}$$

Linear Secret Sharing

We say that a sharing $(s_1, \ldots, s_n)$ is **linear** if the shares are computed as a linear function of s and random values. That is if there exists a **constant** $n \times (m+1)$ matrix A such that for random values $r_1, \ldots, r_m$:

$$\begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} A_{10} & A_{11} & \cdots & A_{1m} \\ \vdots & & & \vdots \\ A_{n0} & A_{n1} & \cdots & A_{nm} \end{bmatrix} \begin{bmatrix} s \\ r_1 \\ \vdots \\ r_m \end{bmatrix}$$

Example: n-out-of-n (additive) sharing

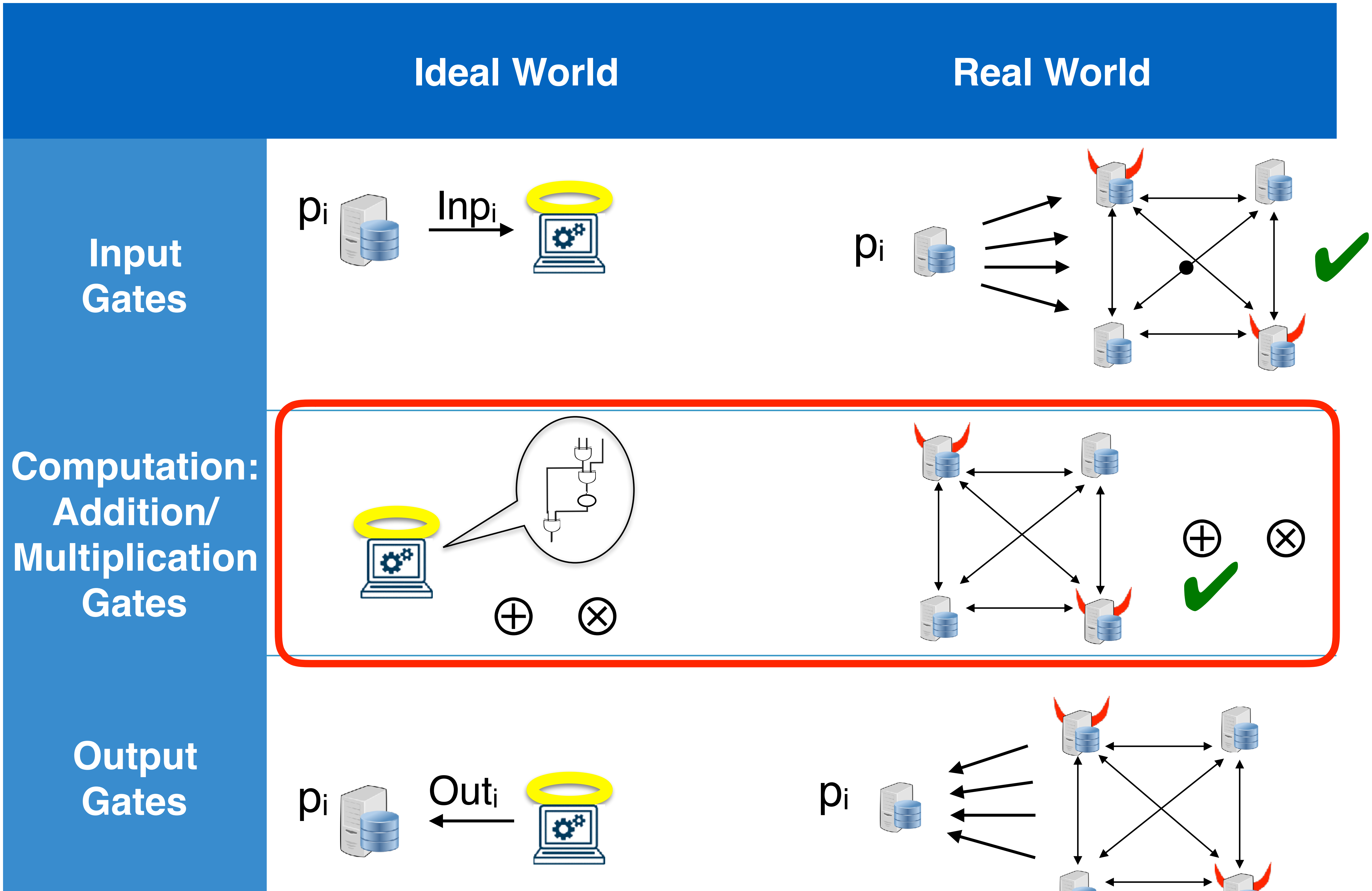
$$\begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & \ldots & 1 \\ 1 & -1 & -1 & \ldots & -1 \end{bmatrix} \begin{bmatrix} s \\ r_1 \\ \vdots \\ r_{n-1} \end{bmatrix}$$

Linear Secret Sharing

When s and s' are shared by a linear secret sharing then the parties can computer a sharing of s''= s + s' by locally adding their shares if s and s'

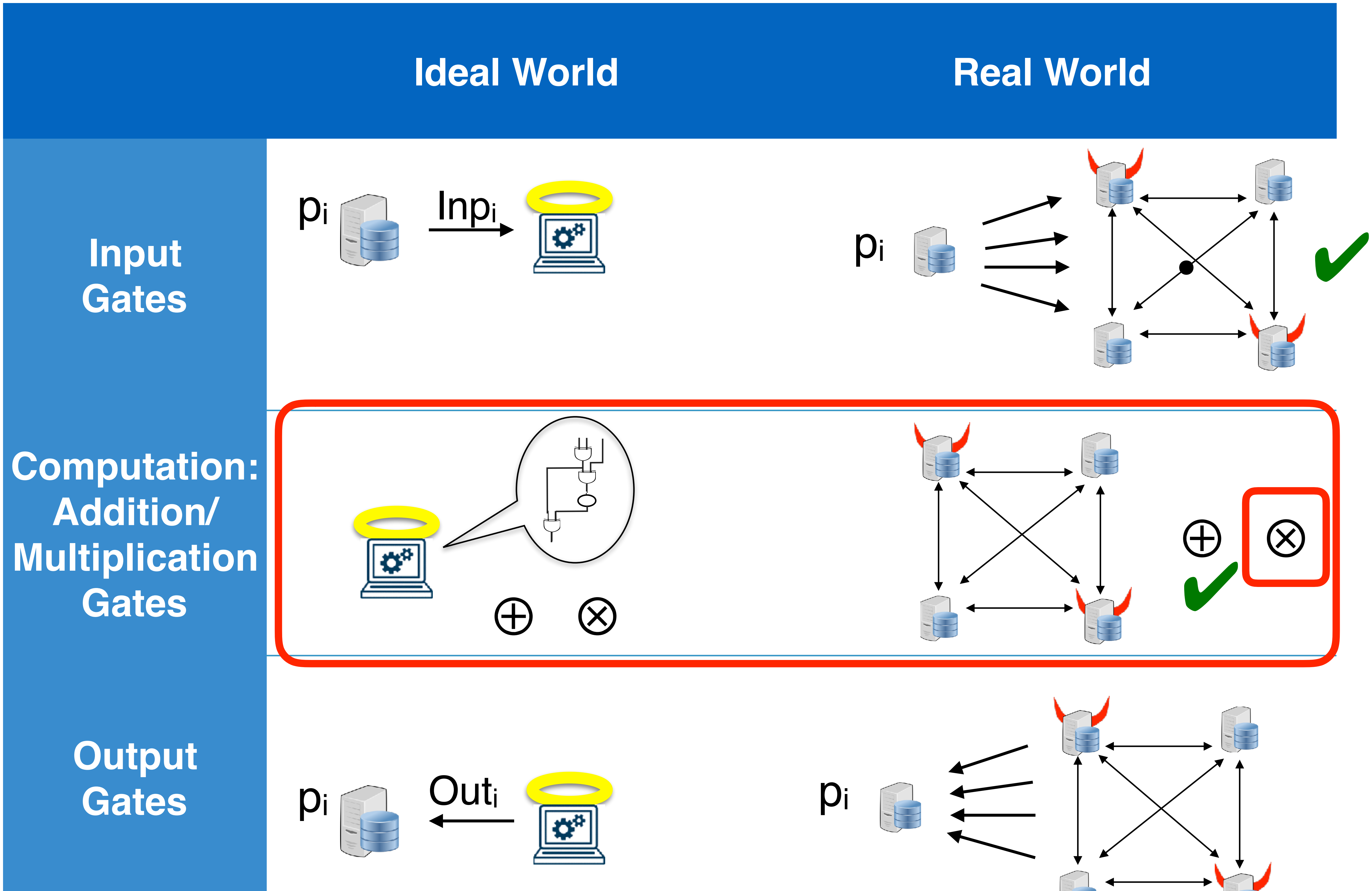
$$\begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} + \begin{bmatrix} s'_1 \\ \vdots \\ s'_n \end{bmatrix} = \begin{bmatrix} A_{10} & A_{11} & \ldots & A_{1m} \\ \vdots & & & \vdots \\ A_{n0} & A_{n1} & \ldots & A_{nm} \end{bmatrix} \left(\begin{bmatrix} s \\ r_1 \\ \vdots \\ r_m \end{bmatrix} + \begin{bmatrix} s' \\ r'_1 \\ \vdots \\ r'_m \end{bmatrix} \right) = \begin{bmatrix} s'' \\ r''_1 \\ \vdots \\ r''_{n-1} \end{bmatrix}$$

MPC Goal

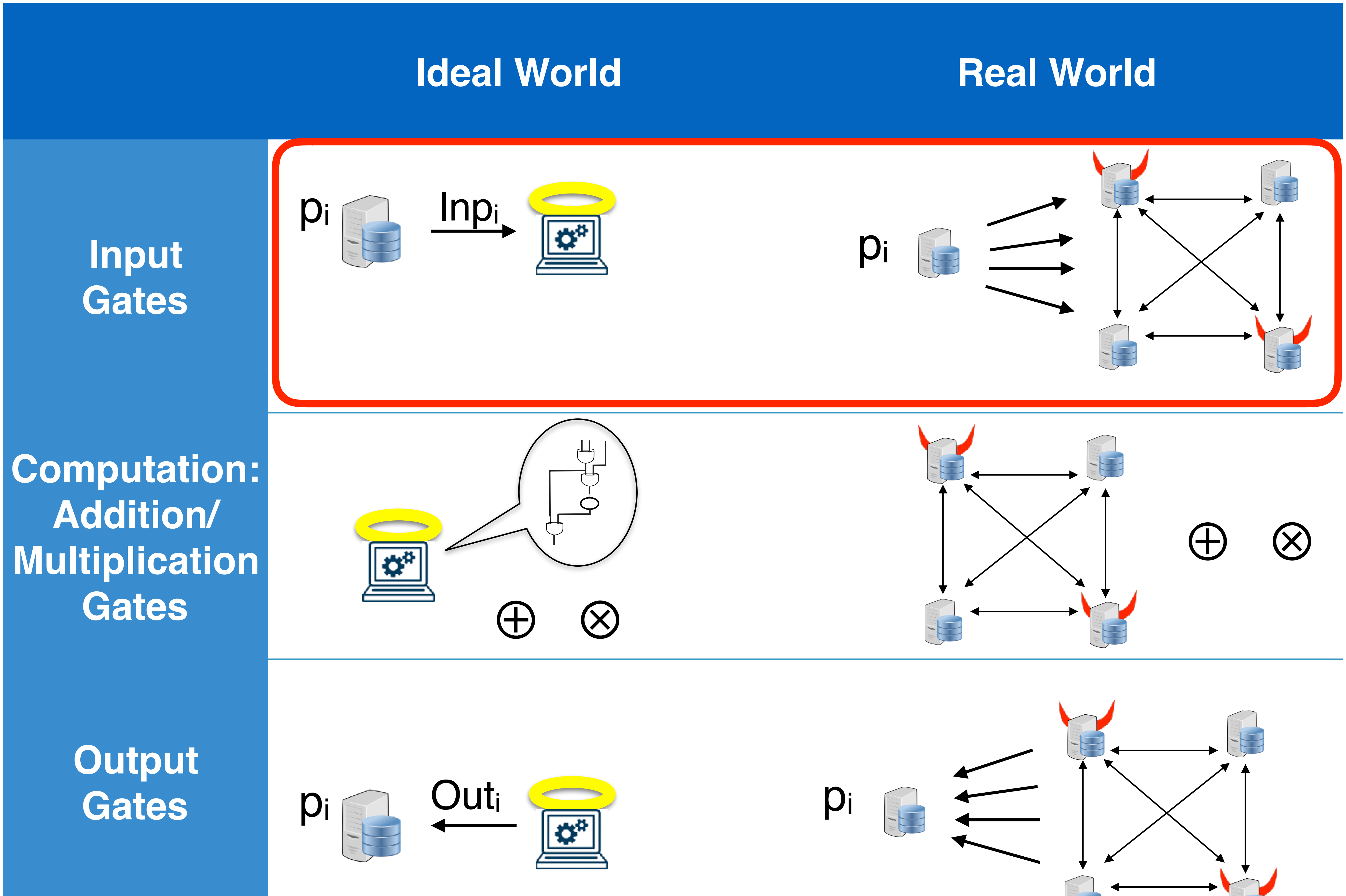


Ideal World
Real World
Input Gates
Computation: Addition/ Multiplication Gates
Output Gates
p_i
Inp_i
Out_i
⊕ ⊗
✔

MPC Goal


Ideal World
Real World
Input Gates
pi
Inpi
pi
Computation: Addition/ Multiplication Gates
⊕ ⊗
⊕ ⊗
Output Gates
pi
Outi
pi

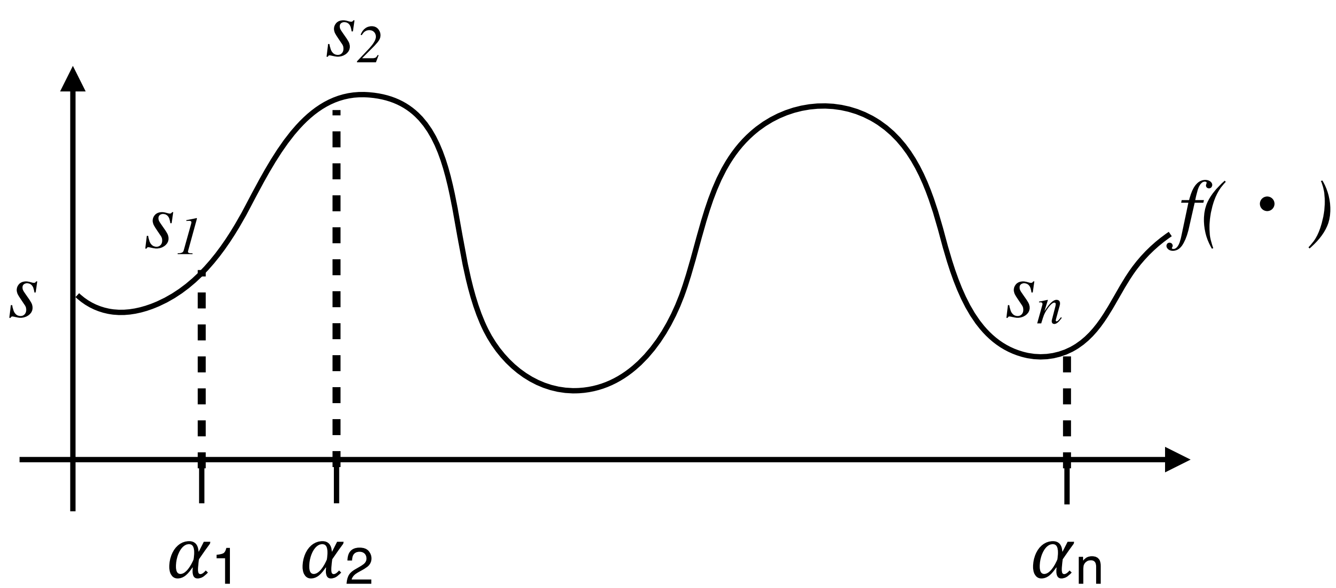
MPC Goal

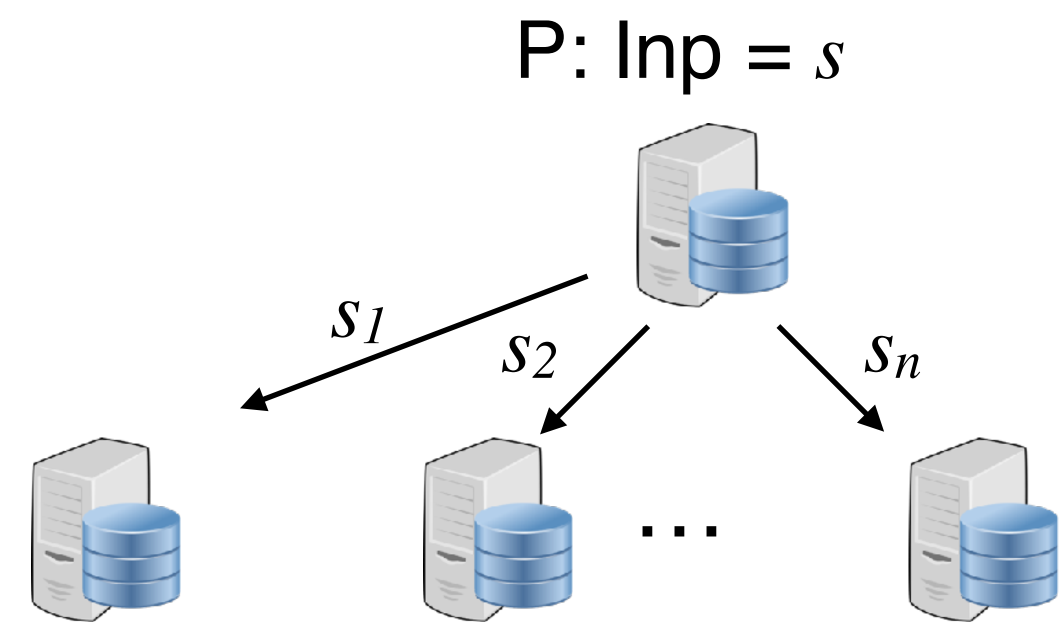


Ideal World
Real World
Input Gates
pi
Inpi
pi
Computation: Addition/ Multiplication Gates
⊕ ⊗
⊕ ⊗
Output Gates
pi
Outi
pi

Secret Sharing: (t+1)-out-of-n

Example: Polynomial (Shamir [Sha79]) Secret Sharing



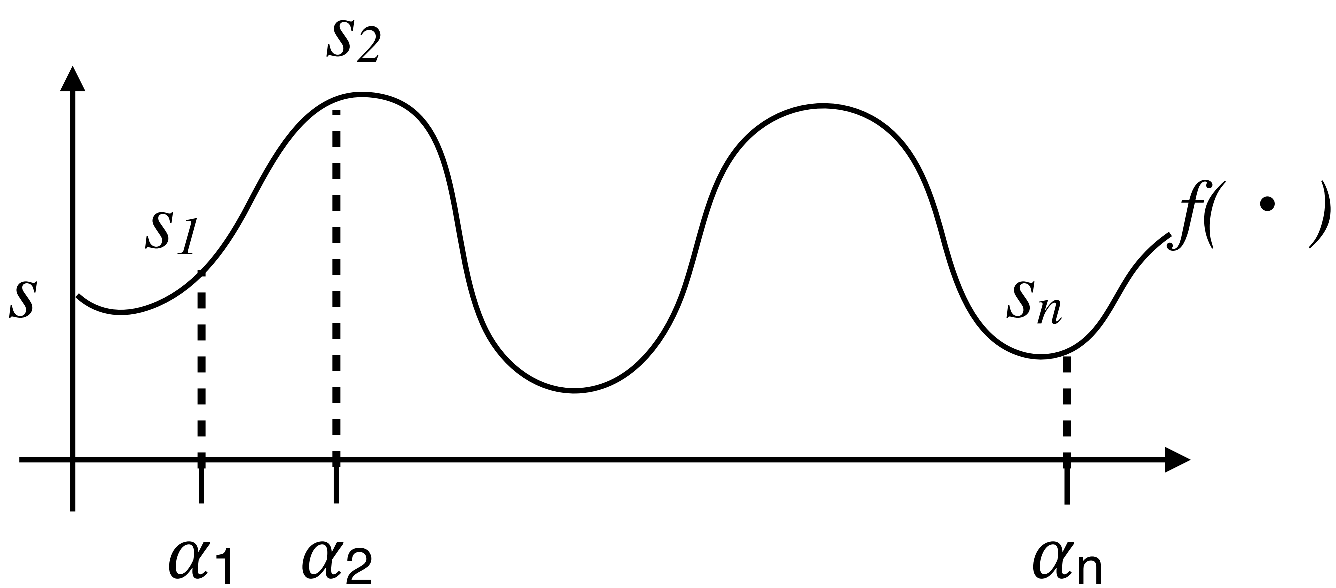


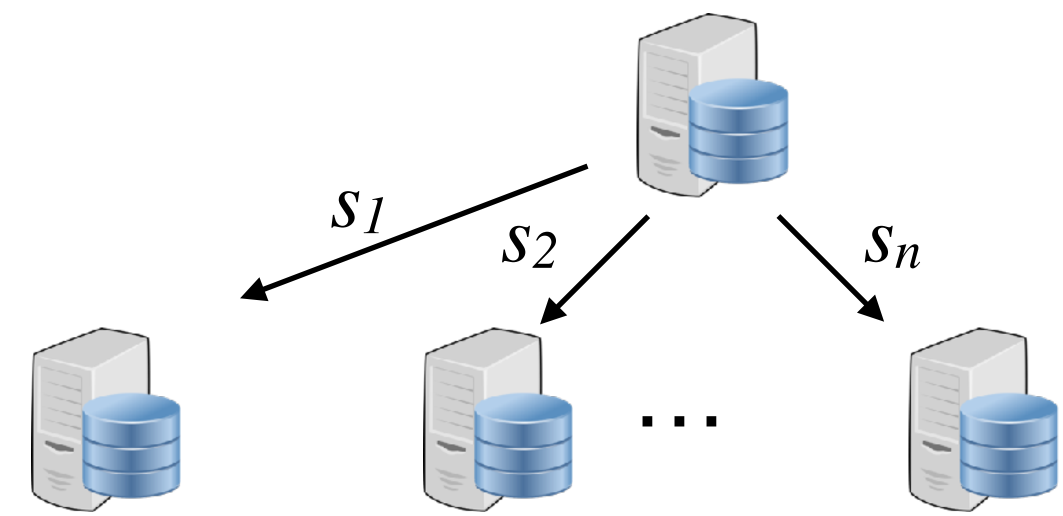
- Share: Dealer p sharing s:
  - Choose a random degree-t polynomial $f(\cdot)$ with $f(0)=s$
  - Give $s_i = f(\alpha_i)$ to player p_i
- Reconstruct:
  - Lagrange interpolation (for all $n > t\text{-}1$):

$$f(x) = \sum_{i=1}^{n} \ell_i(x) s_i \qquad \ell_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - \alpha_j}{\alpha_i - \alpha_j}$$

Secret Sharing: (t+1)-out-of-n

Example: Polynomial (Shamir [Sha79]) Secret Sharing





- Share: Dealer p sharing s:
  - Choose a random degree-t polynomial $f(\cdot)$ with $f(0)=s$
  - Give $s_i = f(\alpha_i)$ to player p_i

Choose random $a_1,\ldots,a_t$ and set

$$f(x) = s + a_1 x + \ldots + a_t x^t$$

- Reconstruct:
  - Lagrange interpolation (for all $n > t\text{-}1$):

$$f(x) = \sum_{i=1}^{n} \ell_i(x) s_i \qquad \ell_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - \alpha_j}{\alpha_i - \alpha_j}$$

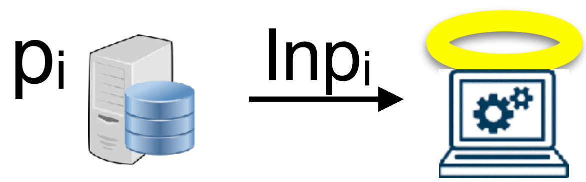
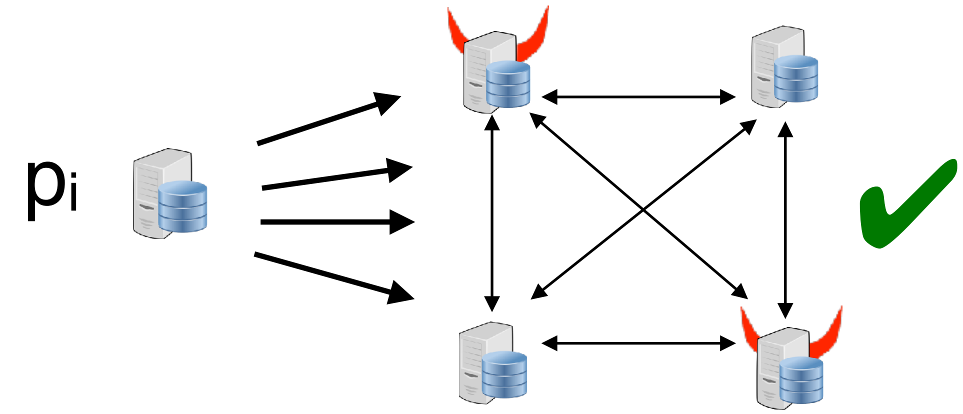
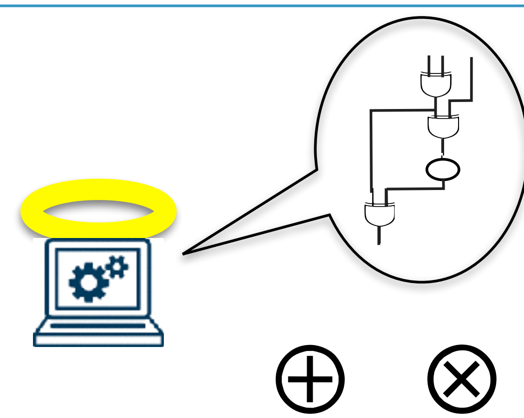
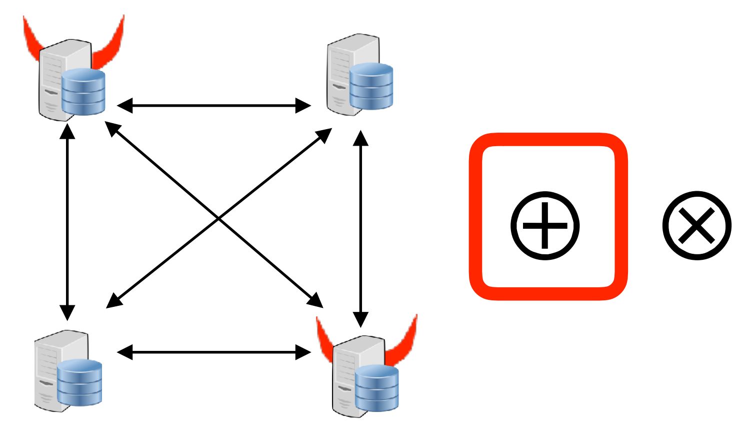
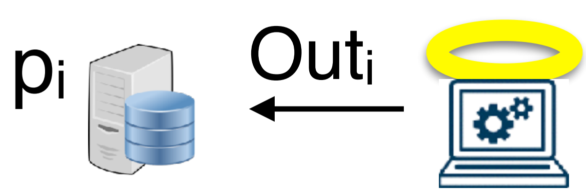
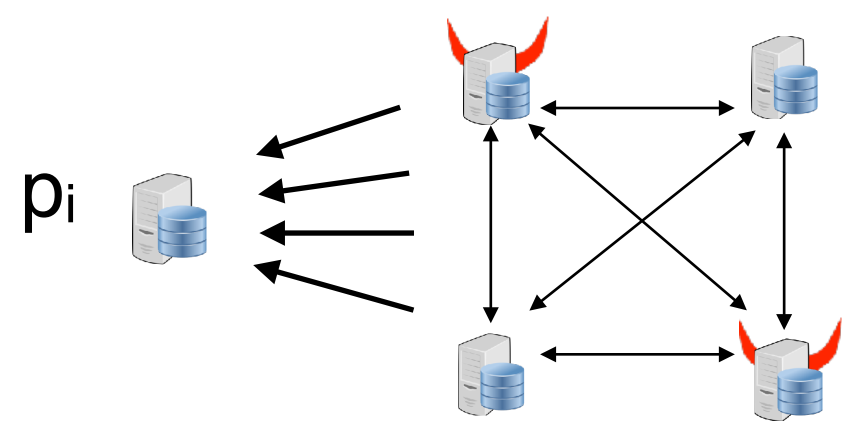
Shamir Secret Sharing is Linear

We say that a sharing $(s_1, \ldots, s_n)$ is **linear** if the shares are computed as a linear function of s and random values. That is if there exists a **constant** $n \times (m+1)$ matrix A such that for random values $r_1, \ldots, r_m$:

$$\begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} A_{10} & A_{11} & \cdots & A_{1m} \\ \vdots & & & \vdots \\ A_{n0} & A_{n1} & \cdots & A_{nm} \end{bmatrix} \begin{bmatrix} s \\ r_1 \\ \vdots \\ r_m \end{bmatrix}$$

$$\begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^t \\ 1 & \alpha_2 & \alpha_2^2 & \ldots & \alpha_2^t \\ \vdots & & & & \\ 1 & \alpha_n & \alpha_n^2 & \ldots & \alpha_n^t \end{bmatrix} \begin{bmatrix} s \\ a_1 \\ \vdots \\ a_t \end{bmatrix}$$

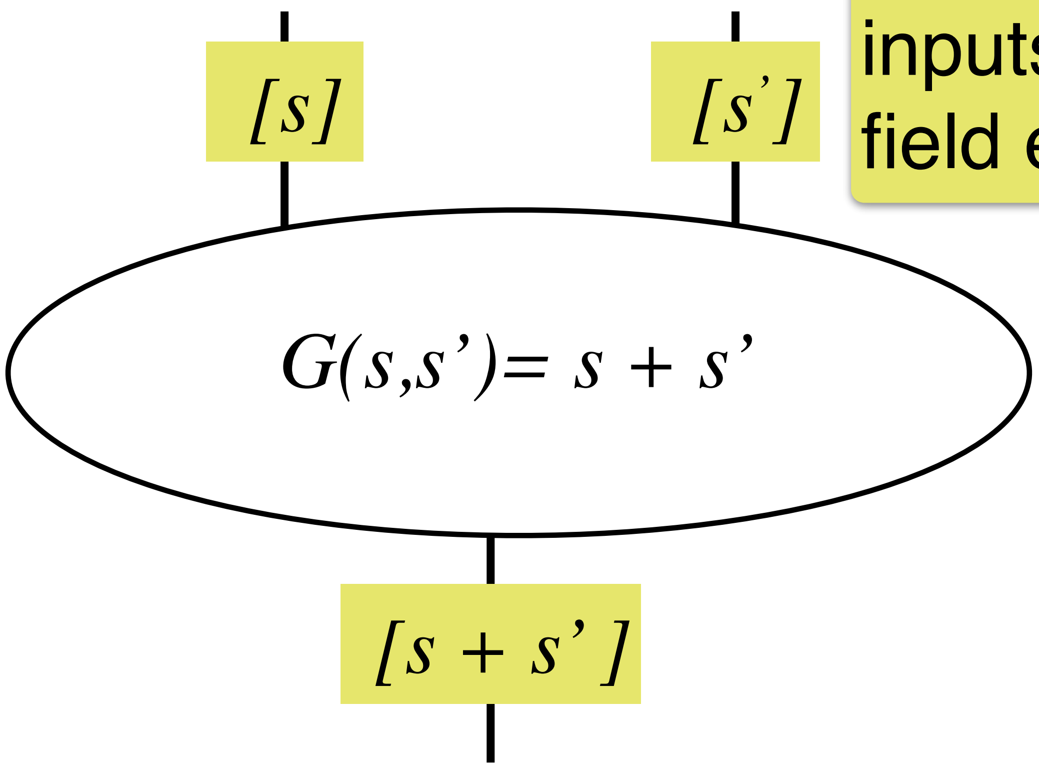
MPC Goal

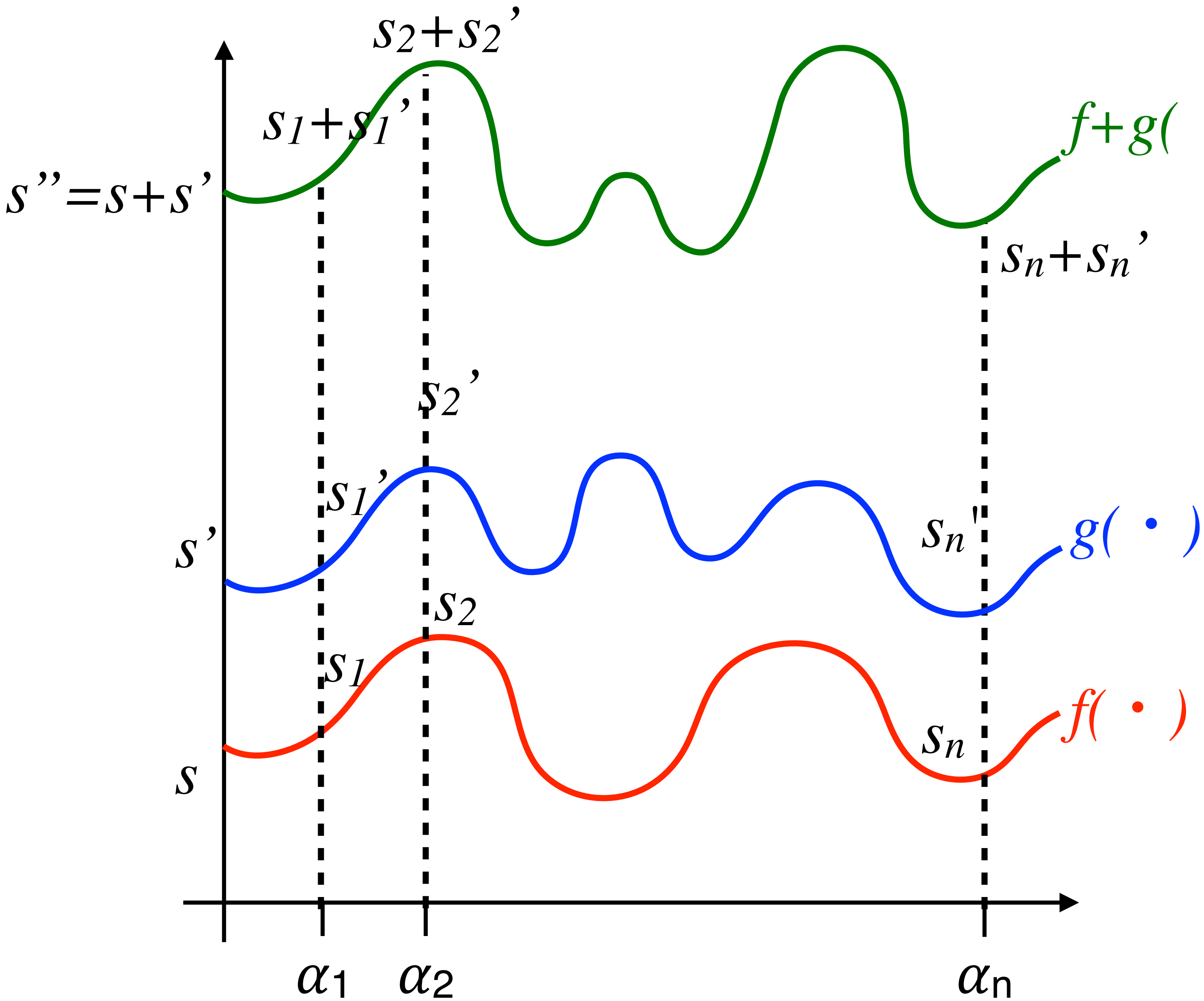
<table>
<tr><th></th><th>Ideal World</th><th>Real World</th></tr>
<tr><td>Input Gates</td><td>
</td><td>
</td></tr>
<tr><td>Computation: Addition/ Multiplication Gates</td><td>
</td><td>
</td></tr>
<tr><td>Output Gates</td><td>
</td><td>
</td></tr>
</table>

Addition Protocol

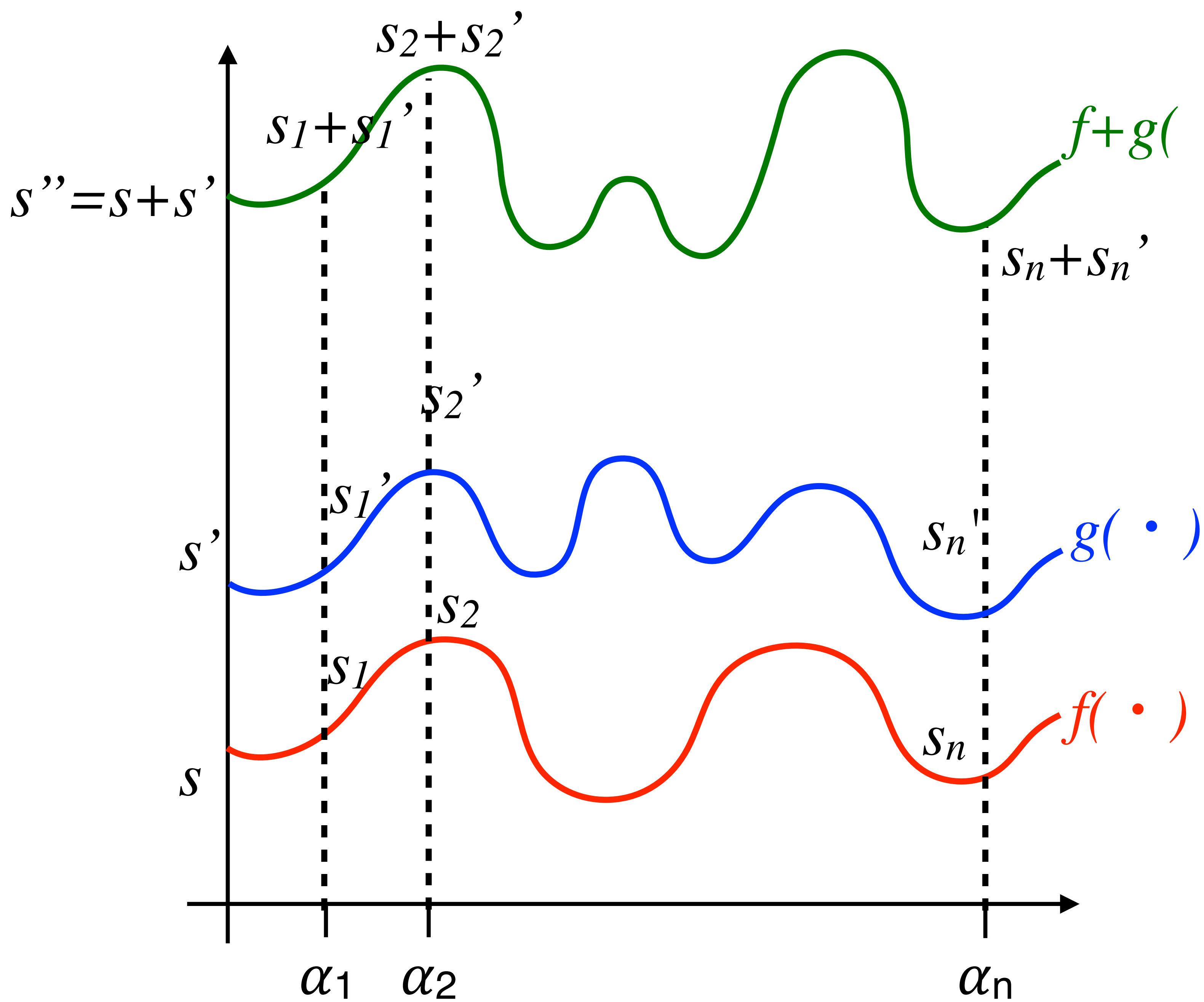
Goal: Addition Gadget

In this lecture:
"***gadget***" = protocol where inputs/outputs are *shares* or field elements





- Each party locally adds his share of s and s', i.e., p_i computes $s_i'' = s_i+s_i'$
- The result is a sharing of s'' by means of polynomial $f'' = f+g$



- Each party locally adds his share of s and s', i.e., p_i computes $s_i'' = s_i+s_i'$
- The result is a sharing of s'' by means of polynomial $f'' = f+g$

Security proof:

- **Correctness:** By Lagrange interpolation, the share sums lie on $f+g$
- **Privacy:** No information is exchanged (only local computation)

Linear Formulas Protocol

If I can compute sharing of s + s' from sharing of s and s' then I can compute any linear combination $a_1 s^{(1)} + a_2 s^{(2)} + \ldots + a_m s^{(m)}$ (for constants $a_1, \ldots, a_m$)

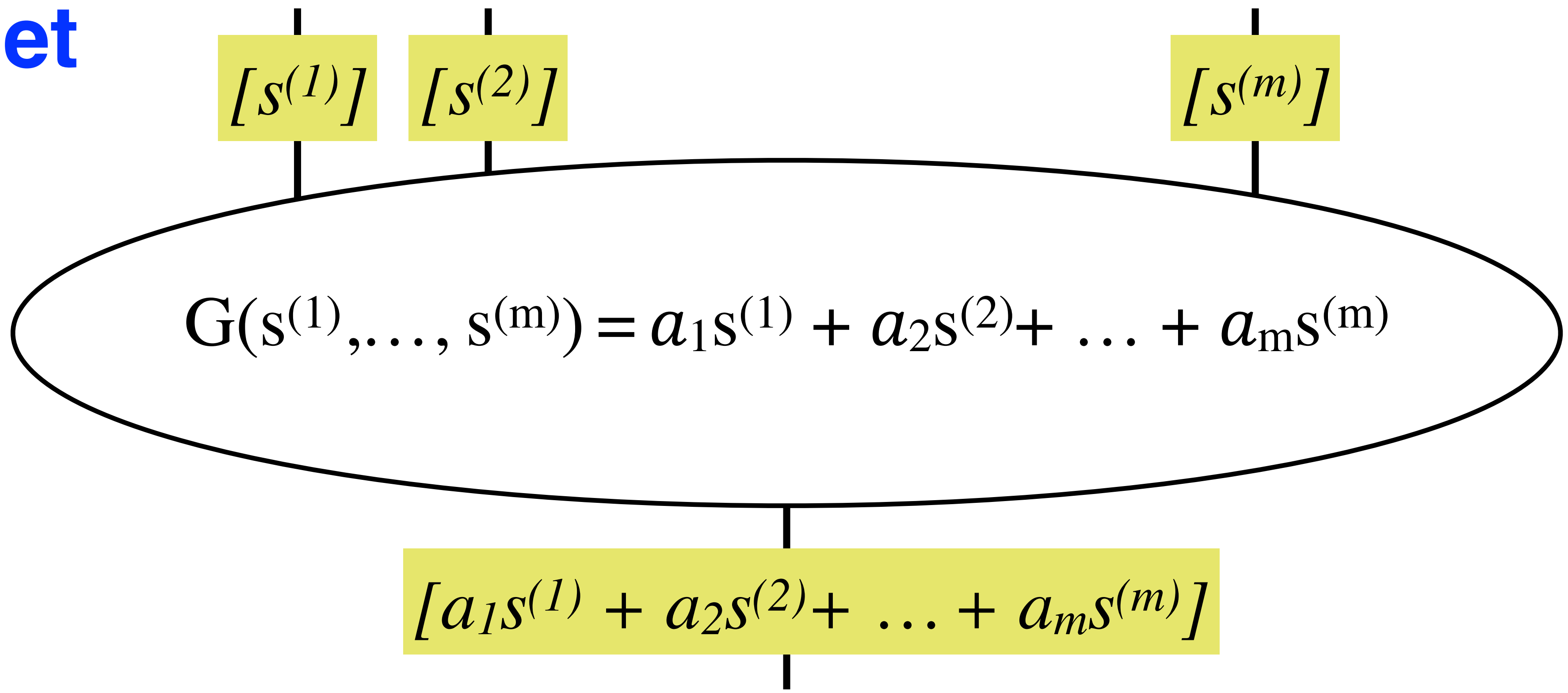
$$a_1 s^{(1)} + \ldots + a_m s^{(m)} = \underbrace{s^{(1)} + \ldots + s^{(1)}}_{a_1 \text{ times}} + \ldots + \underbrace{s^{(m)} + \ldots + s^{(m)}}_{a_m \text{ times}}$$

Linear Formulas Protocol

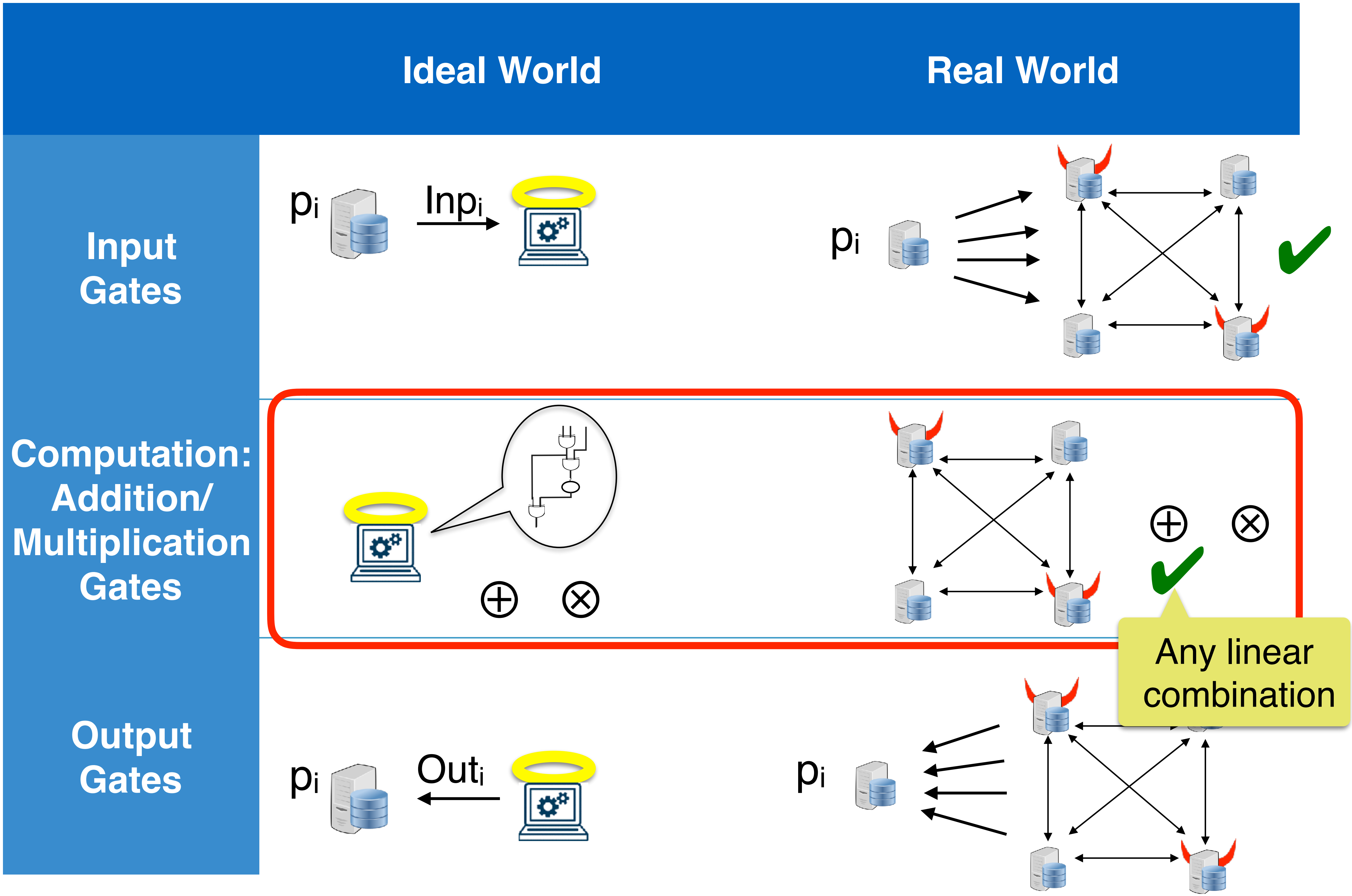
If I can compute sharing of s + s' from sharing of s and s' then I can compute any linear combination $a_1 s^{(1)} + a_2 s^{(2)} + \ldots + a_m s^{(m)}$ (for constants $a_1, \ldots, a_m$)

$$a_1 s^{(1)} + \ldots + a_m s^{(m)} = \underbrace{s^{(1)} + \ldots + s^{(1)}}_{a_1 \text{ times}} + \ldots + \underbrace{s^{(m)} + \ldots + s^{(m)}}_{a_m \text{ times}}$$

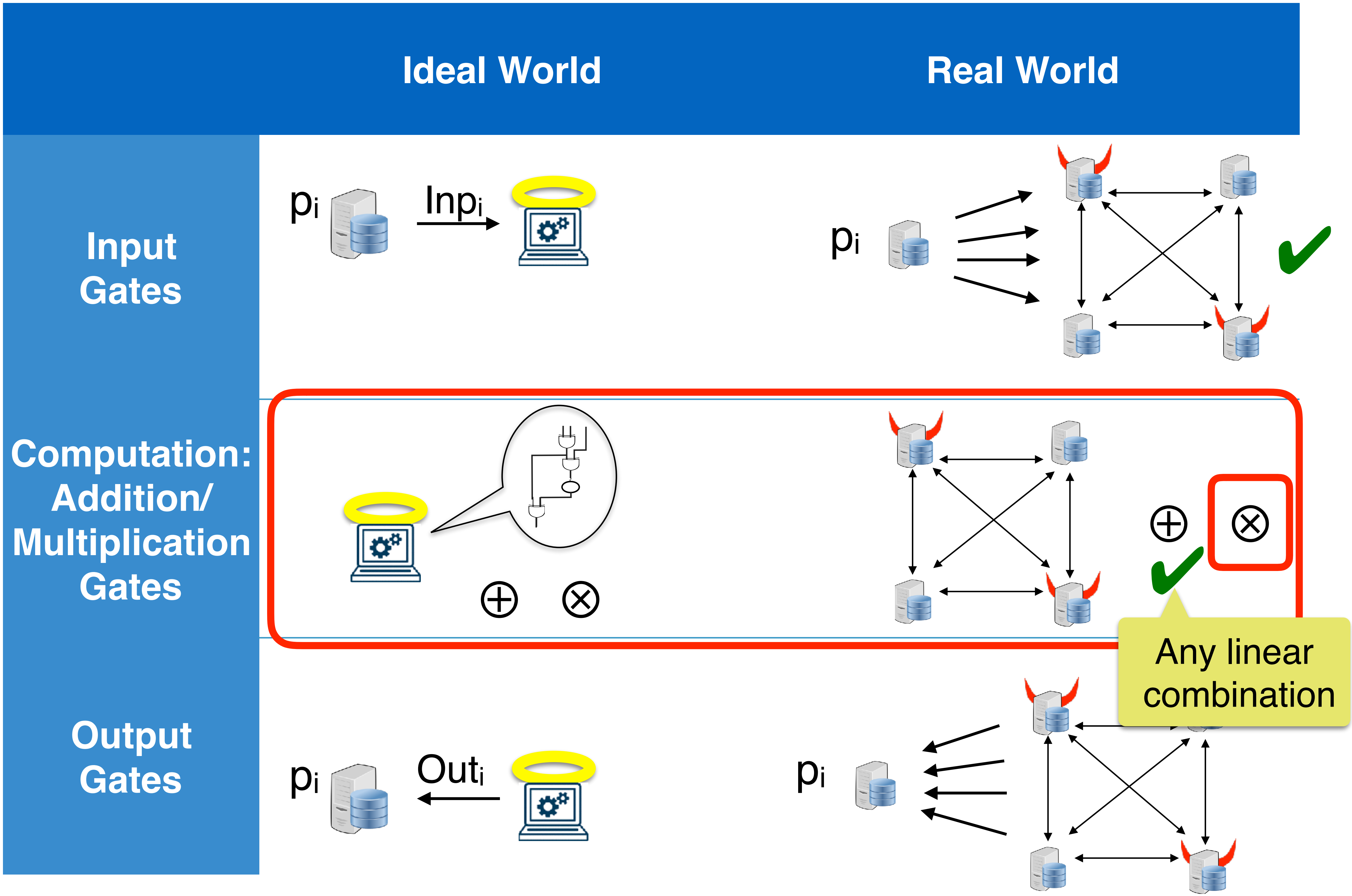
Linear Gadget



MPC Goal

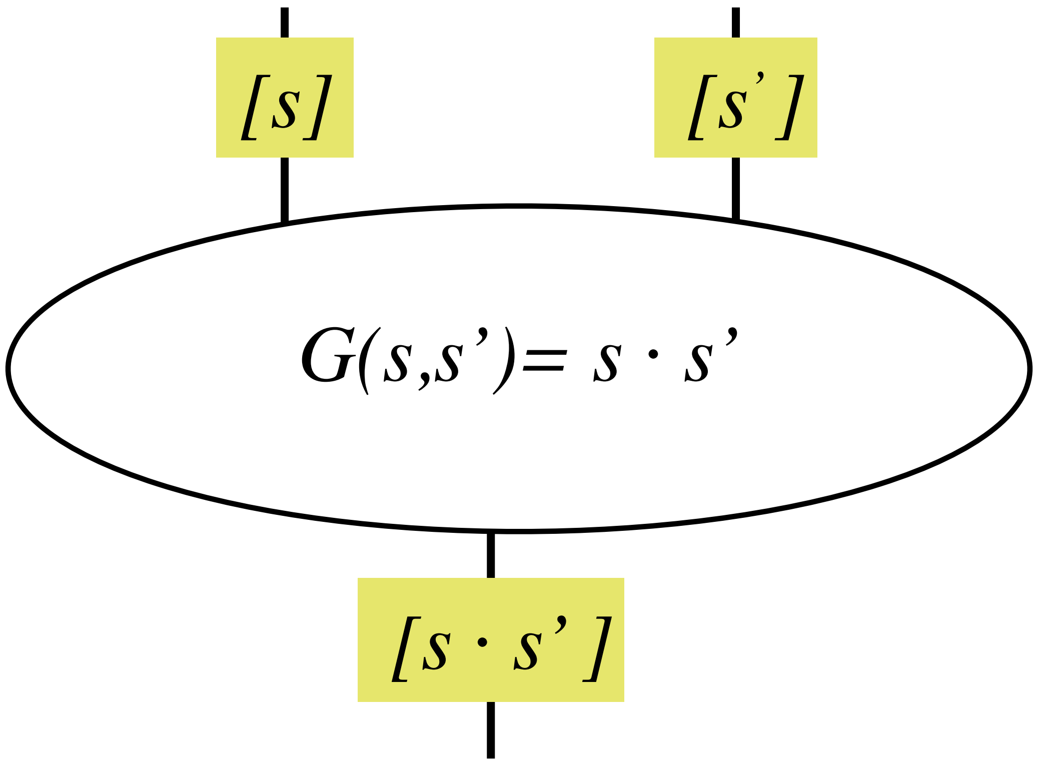

Ideal World
Real World
Input Gates
pi
Inpi
pi
Computation: Addition/ Multiplication Gates
Any linear combination
Output Gates
pi
Outi
pi

MPC Goal


Ideal World
Real World
Input Gates
Computation: Addition/ Multiplication Gates
Output Gates
p_i
Inp_i
Out_i
$\oplus$
$\otimes$
Any linear combination

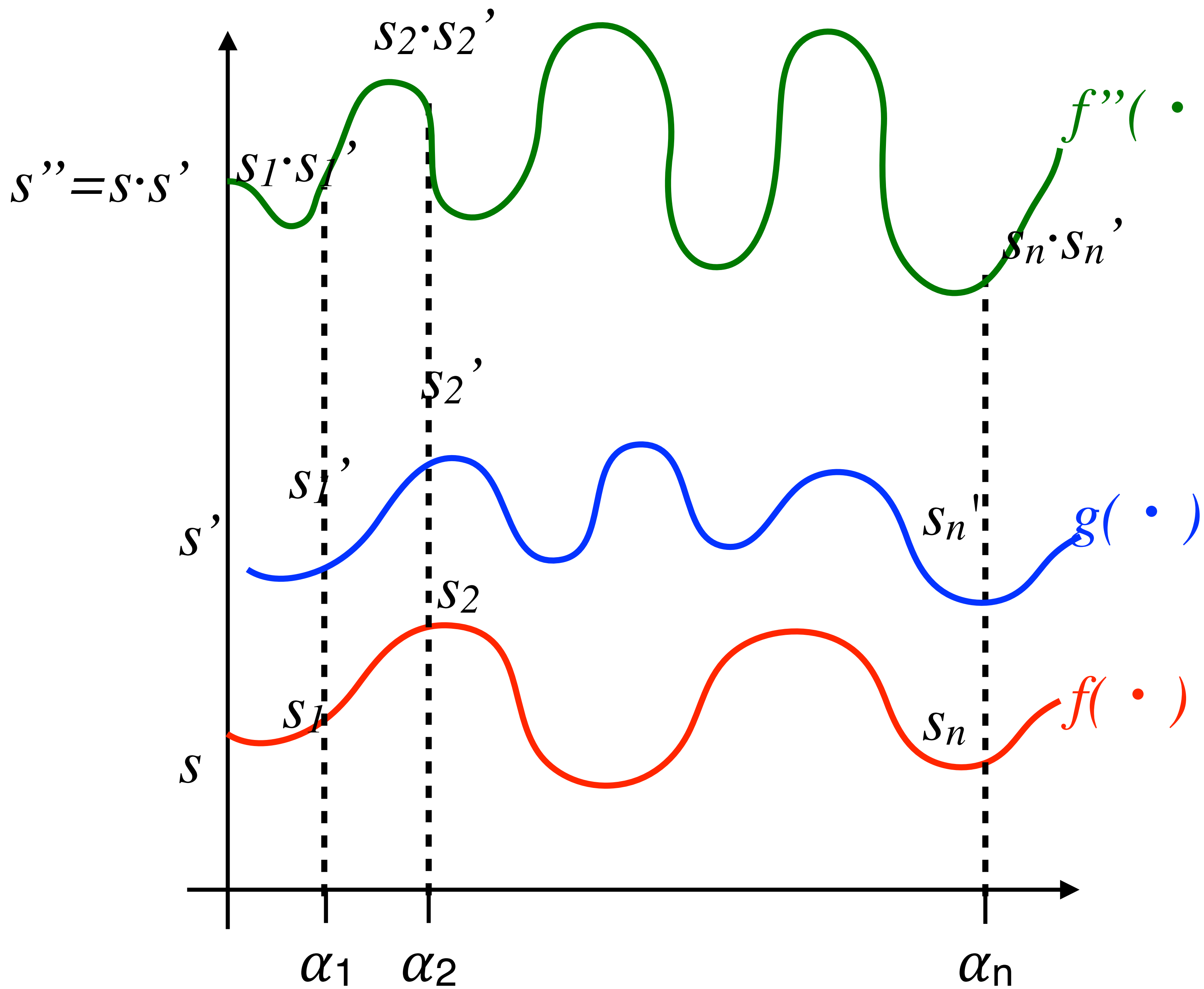
Multiplication Protocol

Goal: Multiplication Gadget



Multiplication Protocol

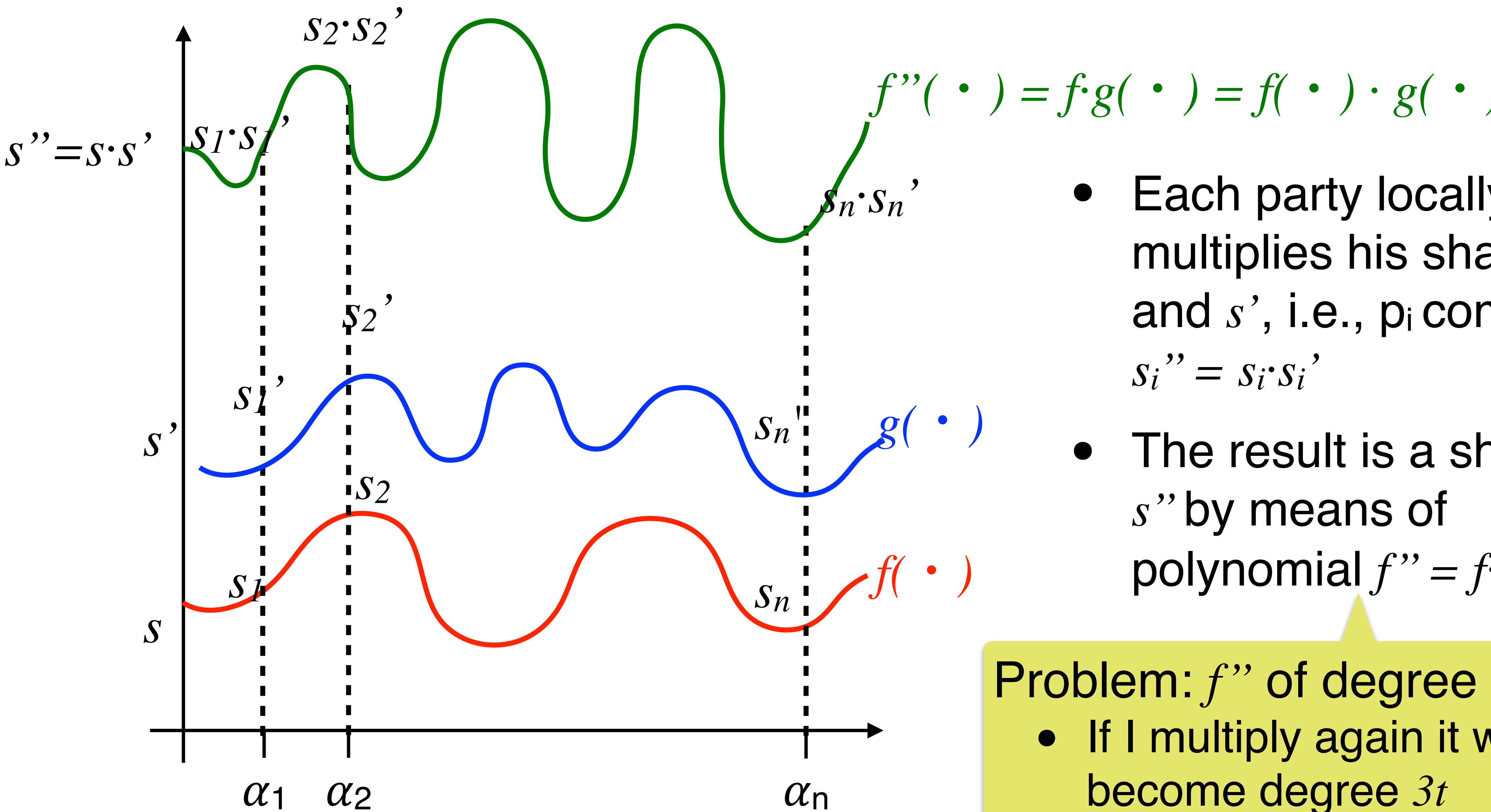
Attempt 1: Use the addition protocol idea …



- Each party locally multiplies his share of s and s', i.e., p_i computes $s_i'' = s_i \cdot s_i'$
- The result is a sharing of s'' by means of polynomial $f'' = f \cdot g$

Multiplication Protocol

Attempt 1: Use the addition protocol idea …



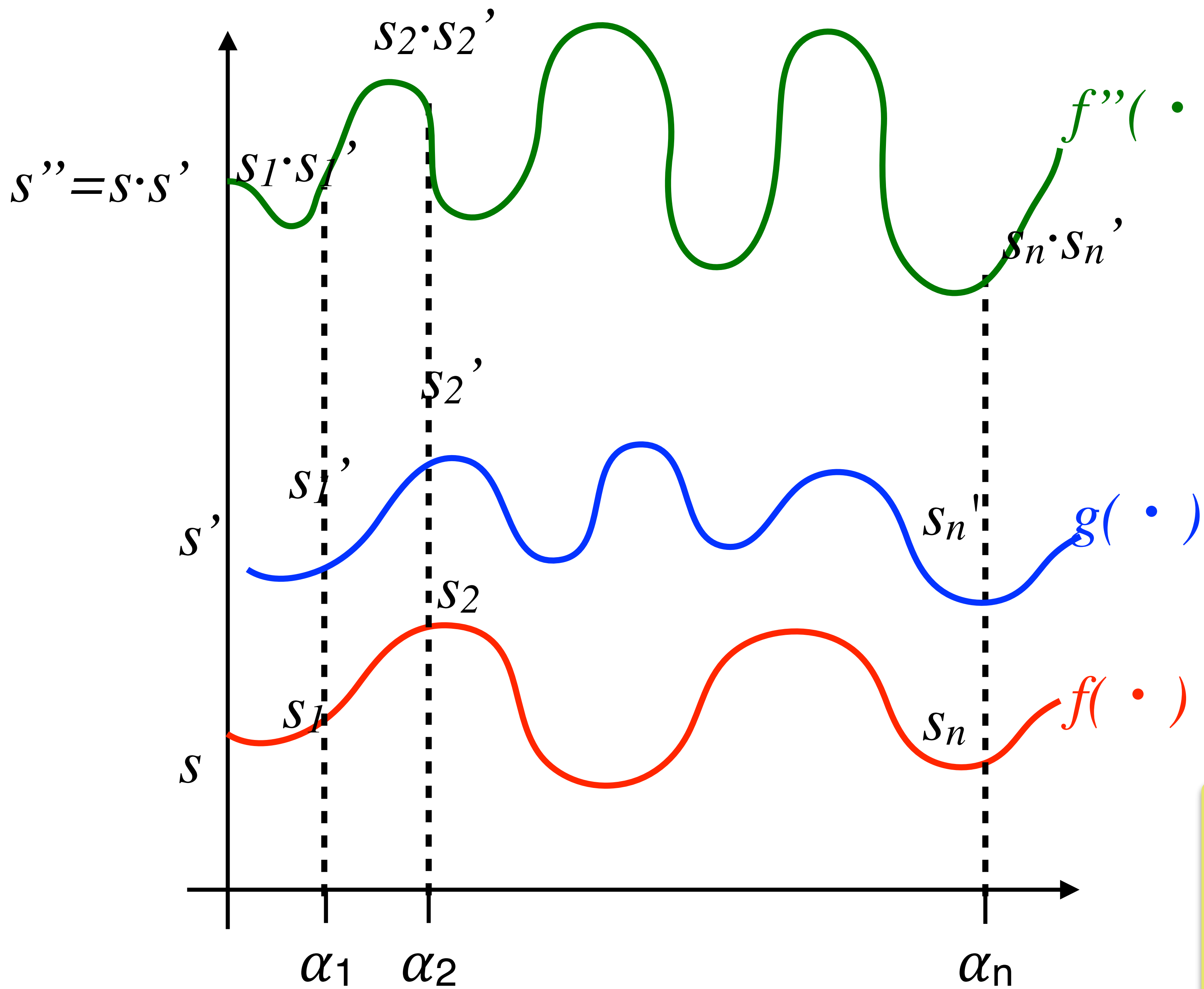
- Each party locally multiplies his share of s and s', i.e., p_i computes $s_i'' = s_i \cdot s_i'$
- The result is a sharing of s'' by means of polynomial $f'' = f \cdot g$

Problem: f'' of degree $2t$

- If I multiply again it will become degree $3t$
- $3t > n$ hence parties cannot reconstruct

Multiplication Protocol

~~Attempt 1: Use the addition protocol idea …~~



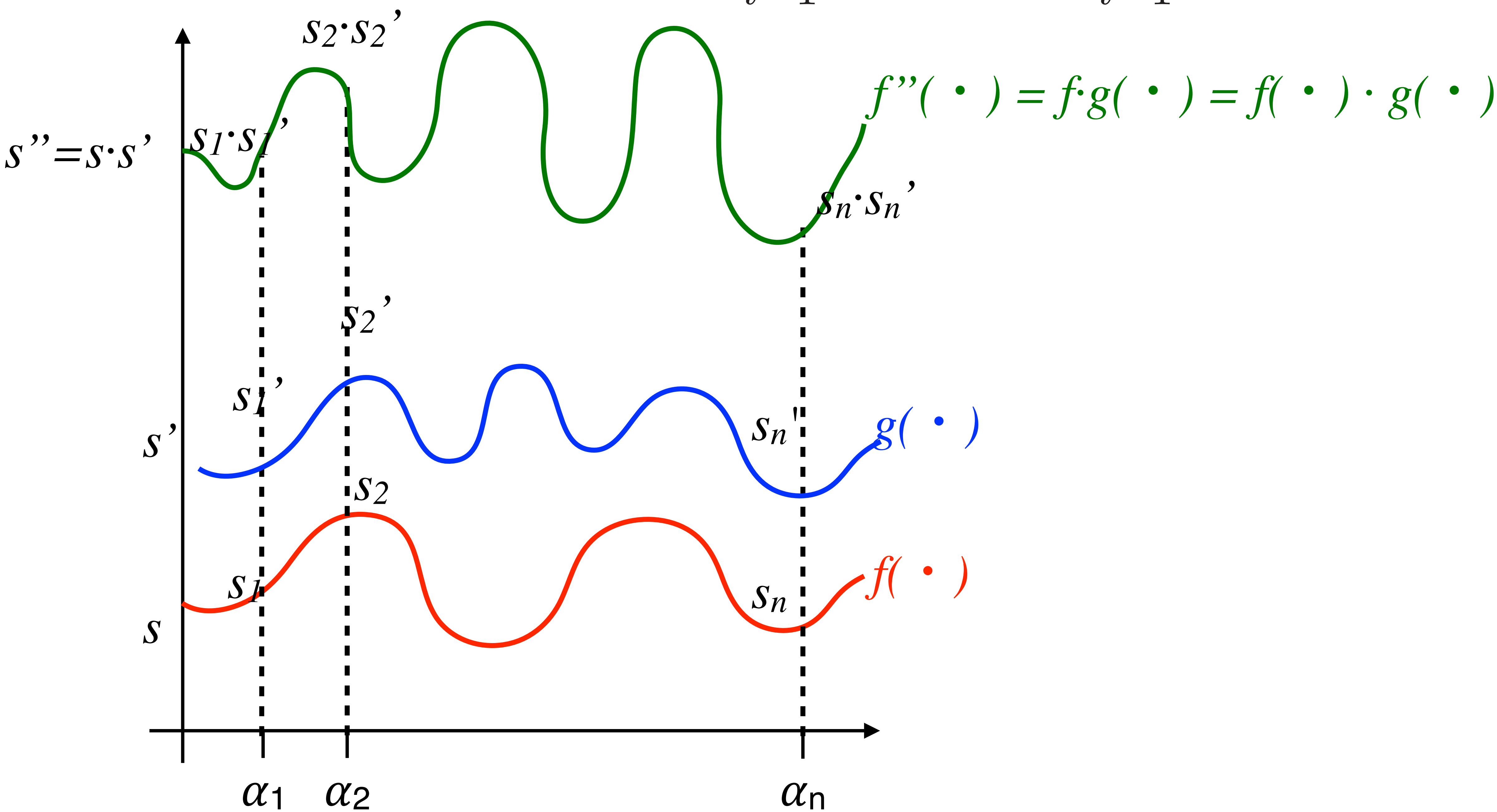
- Each party locally multiplies his share of s and s', i.e., p_i computes $s_i'' = s_i \cdot s_i'$
- The result is a sharing of s'' by means of polynomial $f'' = f \cdot g$

Problem: f'' of degree $2t$

- If I multiply again it will become degree $3t$
- $3t > n$ hence parties cannot reconstruct

Multiplication Protocol

Attempt 2: $s'' = f''(0) = \sum_{i=1}^{n} \ell_i(0) s''_i = \sum_{i=1}^{n} \ell_i(0)(s_i \cdot s'_i)$



Multiplication Protocol

Attempt 2: $s'' = f''(0) = \sum_{i=1}^{n} \ell_i(0) s''_i = \sum_{i=1}^{n} \ell_i(0)(s_i \cdot s'_i)$

Multiplication Protocol

Attempt 2: $s'' = f''(0) = \sum_{i=1}^{n} \ell_i(0) s''_i = \sum_{i=1}^{n} \ell_i(0)(s_i \cdot s'_i)$

degree *2t* hence there is enough parties to interpolate

$$\ell_i(0) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{0 - \alpha_j}{\alpha_i - \alpha_j} = \beta_0$$

Multiplication Protocol

Attempt 2: $s'' = f''(0) = \sum_{i=1}^{n} \ell_i(0) s_i'' = \sum_{i=1}^{n} \ell_i(0)(s_i \cdot s_i')$

degree $2t$ hence there is enough parties to interpolate

$$\ell_i(0) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{0 - \alpha_j}{\alpha_i - \alpha_j} = \beta_0$$

To compute a sharing of $s'' = s \cdot s'$ it suffices to compute a sharing of

$$\sum_{i=1}^{n} \beta_i (s_i \cdot s_i') = \sum_{i=1}^{n} \beta_i (s_i'') = \beta_1 s_1'' + \ldots \beta_n s_n''$$

Multiplication Protocol

Attempt 2: $s'' = f''(0) = \sum_{i=1}^{n} \ell_i(0) s''_i = \sum_{i=1}^{n} \ell_i(0)(s_i \cdot s'_i)$

degree $2t$ hence there is enough parties to interpolate

$$\ell_i(0) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{0 - \alpha_j}{\alpha_i - \alpha_j} = \beta_0$$

To compute a sharing of $s'' = s \cdot s'$ it suffices to compute a sharing of

$$\sum_{i=1}^{n} \beta_i(s_i \cdot s'_i) = \sum_{i=1}^{n} \beta_i(s''_i) = \beta_1 s''_1 + \ldots \beta_n s''_n$$

Multiplication (Gadget) Protocol

- Every p_i shares $s_i'' = s_i \cdot s_i'$
- Use the linear gadget to compute a sharing of s''

Multiplication Protocol

Attempt 2: $s'' = f''(0) = \sum_{i=1}^{n} \ell_i(0) s_i'' = \sum_{i=1}^{n} \ell_i(0)(s_i \cdot s_i')$

degree $2t$ hence there is enough parties to interpolate

$$\ell_i(0) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{0 - \alpha_j}{\alpha_i - \alpha_j} = \beta_0$$

To compute a sharing of $s'' = s \cdot s'$ it suffices to compute a sharing of

$$\sum_{i=1}^{n} \beta_i (s_i \cdot s_i') = \sum_{i=1}^{n} \beta_i (s_i'') = \beta_1 s_1'' + \ldots \beta_n s_n''$$

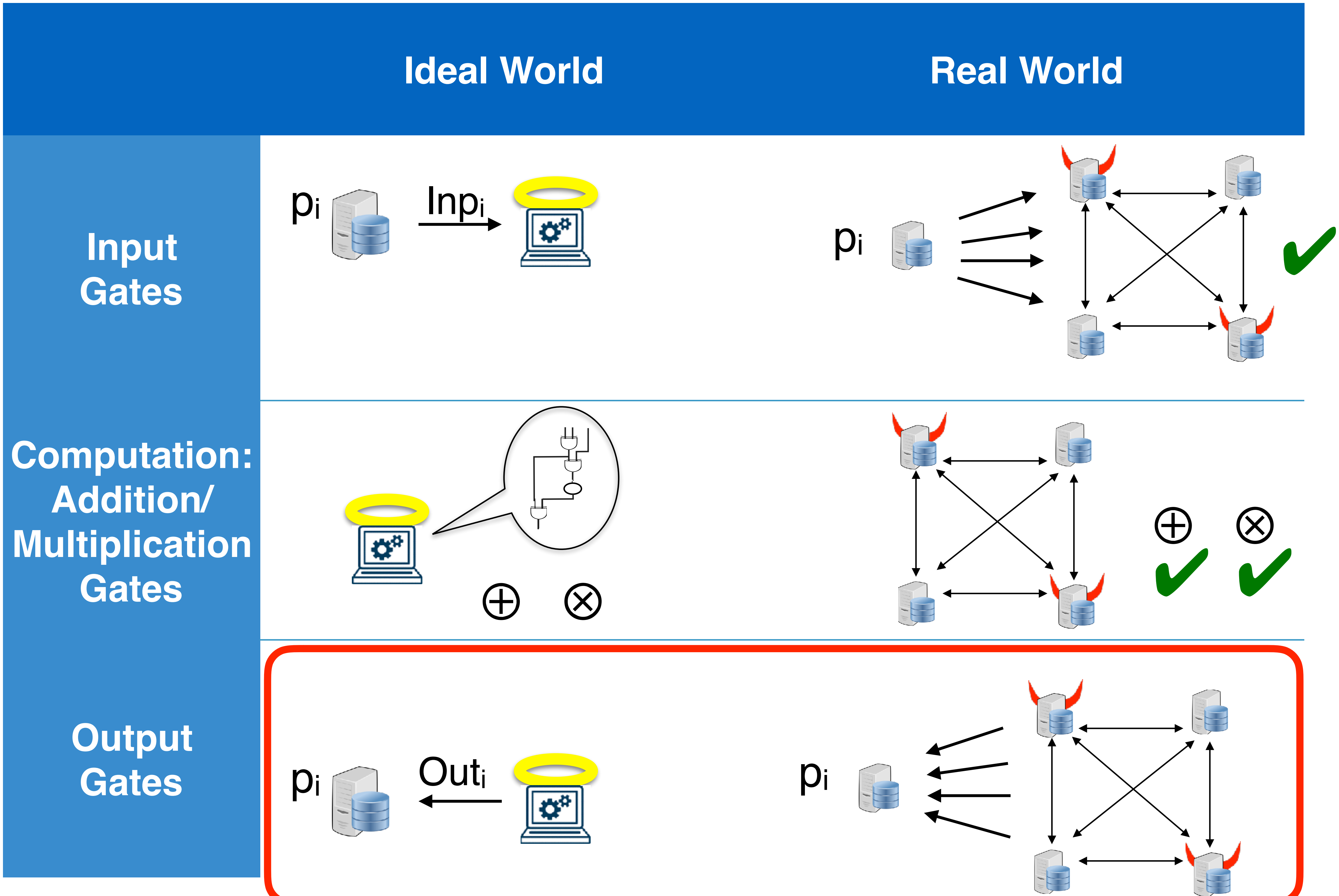
Multiplication (Gadget) Protocol

- Every p_i shares $s_i'' = s_i \cdot s_i'$
- Use the linear gadget to compute a sharing of s''

Security proof:

- **Correctness:** As shown above …
- **Privacy:** Follows from the privacy of the linear gadget and the SS

MPC Goal


Ideal World
Real World
Input Gates
p_i
Inp_i
p_i
Computation: Addition/ Multiplication Gates
⊕ ⊗
⊕ ⊗
Output Gates
p_i
Out_i
p_i

Known Feasibility Results

Adv. Type	Security	Corruption Bound	Requires
semi-honest (passive)	Information theoretic (IT)	t<n/2 [BGW88,CCD88]	Sec. channels ✔
	Computational	t<n [GMW87]	Sec. channels + OT
malicious (active)	information theoretic	t<n/3 [BGW88,CCD88]	Sec. channels
	computational (or IT w. negligible error)	t<n/2 [GMW87,RB89]	Broadcast
	computational without fairness	t<n [GMW87]	Broadcast + OT

Known Feasibility Results

Adv. Type	Security	Corruption Bound	Requires
semi-honest (passive)	Information theoretic (IT)	t<n/2 [BGW88,CCD88]	Sec. channels ✔
	Computational	t<n [GMW87]	Sec. channels + OT
malicious (active)	information theoretic	t<n/3 [BGW88,CCD88]	Sec. channels
	computational (or IT w. negligible error)	t<n/2 [GMW87,RB89]	Broadcast
	computational without fairness	t<n [GMW87]	Broadcast + OT

Malicious MPC with t<n/2 (GMW)

Tools 1/3 : Broadcast (Byzantine Agreement) [LSP82]

Inputs: A party p_i called *the sender* has input x

Outputs: Every p_j outputs y_j

- (consistency) There exists y s.t. $y_j = y$ for all j
- (validity) If p_i is honest then $y = x$

Malicious MPC with t<n/2 (GMW)

Tools 1/3 : Broadcast (Byzantine Agreement) [LSP82]

Inputs: A party p_i called *the sender* has input x

Outputs: Every p_j outputs y_j

- (consistency) There exists y s.t. $y_j = y$ for all j
- (validity) If p_i is honest then $y = x$

Theorem:

- Broadcast is possible (unconditionally) iff $t < n/3$ [LSP82 BGP89]
- Assuming digital signatures and a public-key infrastructure it is possible for any $t < n$ [DS83]

Malicious MPC with t<n/2 (GMW)

Tools 1/3 : Broadcast (Byzantine Agreement) [LSP82]

Inputs: A party p_i called *the sender* has input x

Outputs: Every p_j outputs y_j

- (consistency) There exists y s.t. $y_j = y$ for all j
- (validity) If p_i is honest then $y = x$

Theorem:

- Broadcast is possible (unconditionally) iff $t < n/3$ [LSP82 BGP89]
- Assuming digital signatures and a public-key infrastructure it is possible for any $t < n$ [DS83]

Broadcast + Encryption Setup (keys) = Secure channel

k_i: encryption key for p_i

p_1 $\xrightarrow{Enc(x, k_i) = c}$

$\xrightarrow{c}$ p_2 ??

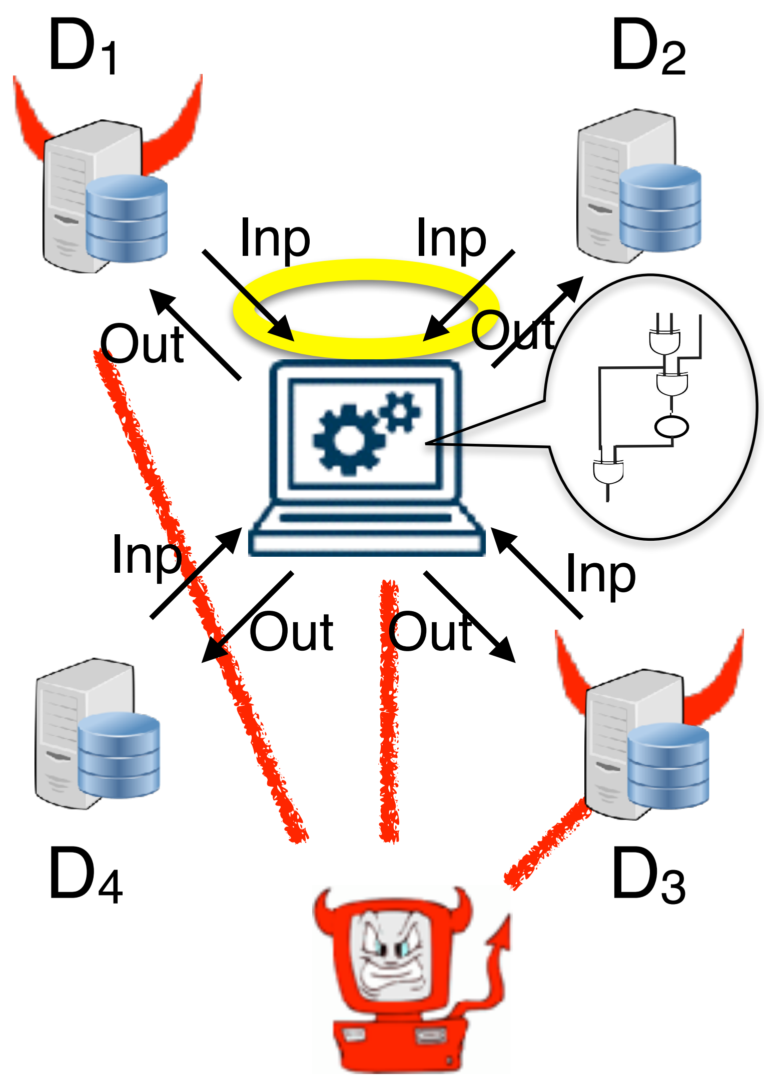
⋮

$\xrightarrow{c}$ p_i can decrypt and learn x

$\xrightarrow{c}$ p_n ??

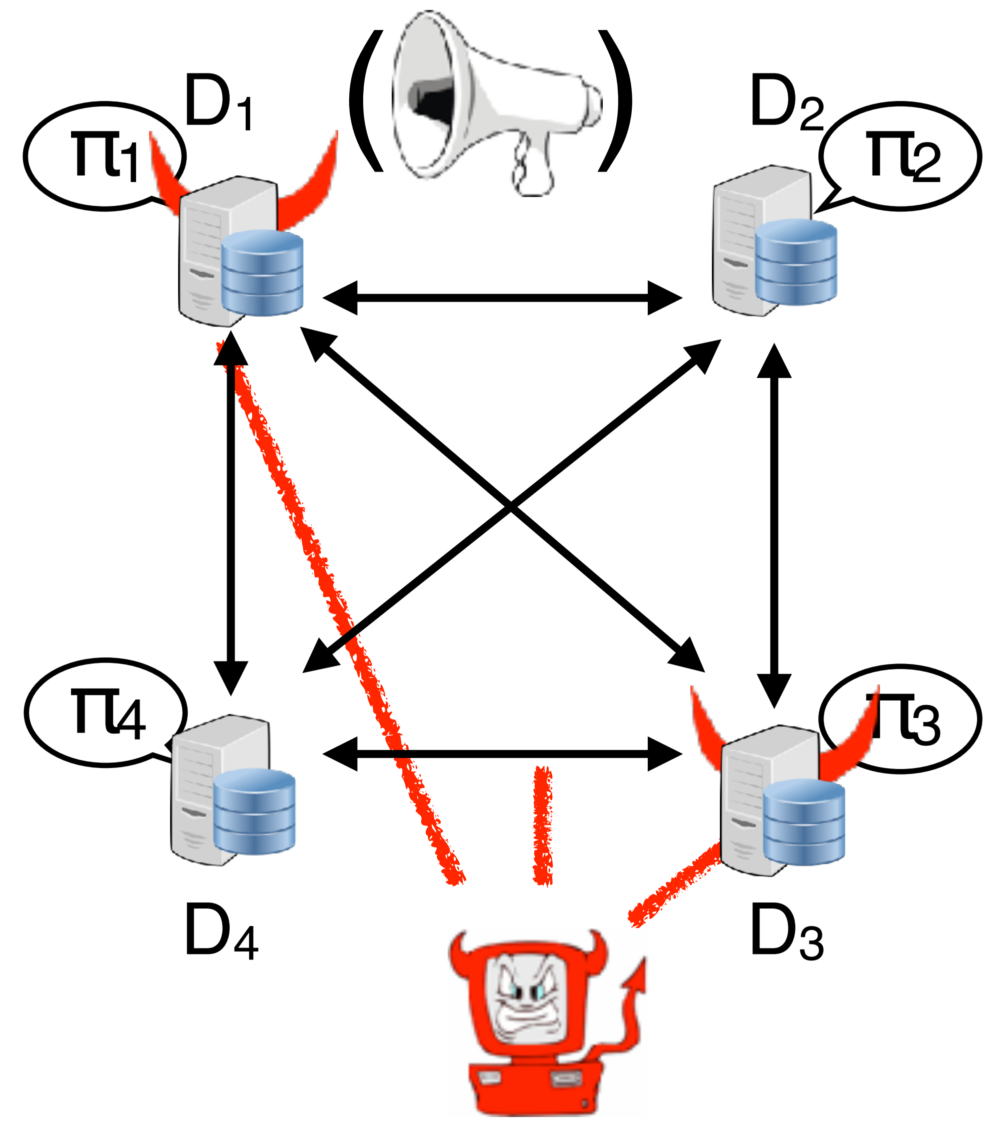
Back to MPC Security

Ideal World: Specification



$\approx$

Real World: Protocol

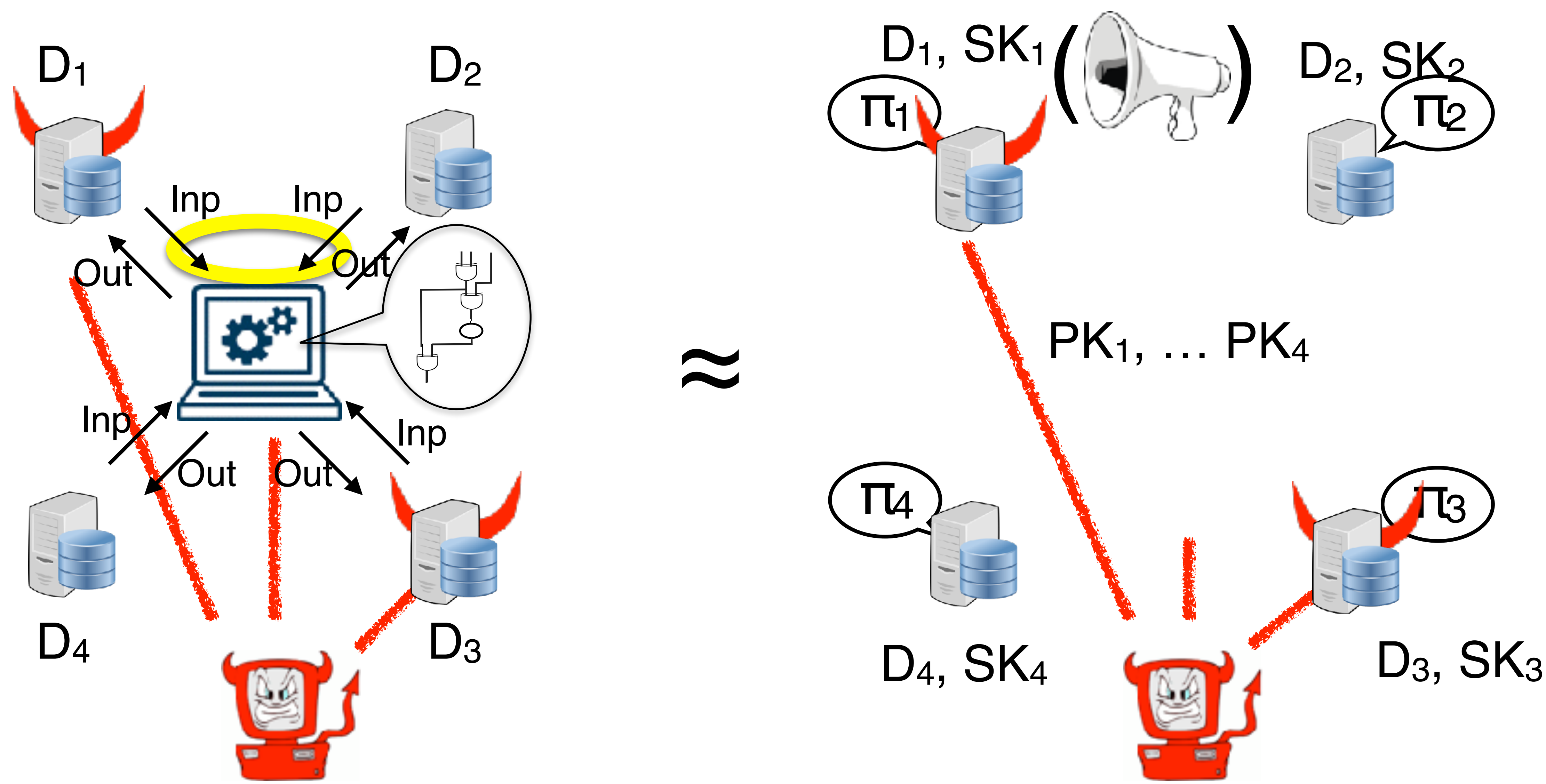


Model

- n players
- Computation over $(\mathbb{F}, \oplus, \otimes)$ — E.g. $(\mathbb{Z}_p, +, \cdot)$
- Communication: Point-to-point secure channels (and Broadcast)
- Synchrony: Messages sent in round i are delivered by round i+1

Back to MPC Security

Ideal World: Specification / Real World: Protocol



Model

- n players
- Computation over $(\mathbb{F}, \oplus, \otimes)$ — E.g. $(\mathbb{Z}_p, +, \cdot)$
- Communication: Broadcast + Public-key Infrastructure
- Synchrony: Messages sent in round i are delivered by round i+1

Malicious MPC with t<n/2 (GMW)

Tools 2/3 : (Non-interactive) Commitments

Committer P $\xrightarrow{\quad c \quad}$ **Verifier V**

Input x

Rand. r

Com $(x, r) = c$

Commit Phase

$\xrightarrow[\quad x\ ,\ r \quad]{}$

Ver $(c, x\ ,\ r) \in \{0,1\}$

Open Phase

Malicious MPC with t<n/2 (GMW)

Tools 2/3 : (Non-interactive) Commitments

Committer P $\xrightarrow{\quad c \quad}$ **Verifier V**

Input x

Rand. r

Com $(x, r) = c$

Commit Phase

$\xrightarrow[\quad x\ ,\ r \quad]{}$

Open Phase

Ver $(c, x\ ,\ r) \in \{0,1\}$

Security (informal)

- **Correctness:** If P follows the protocol, V always accepts (i.e., outputs 1).
- **Hiding:** From the Commit phase, V has no information about P's input x.
- **Binding:** After the Commit phase, there exists only one value x that will be accepted by V in the Open phase.

Malicious MPC with t<n/2 (GMW)

Tools 2/3 : (Non-interactive) Commitments

Committer P $\xrightarrow{\quad c \quad}$ **Verifier V**

Input x

Rand. r

Com $(x, r) = c$

Commit Phase

$\xrightarrow[\quad x\ ,\ r \quad]{}$

Open Phase

Ver $(c, x\ ,\ r) \in \{0,1\}$

Security (informal)

- **Correctness:** If P follows the protocol, V always accepts (i.e., outputs 1).
- **Hiding:** From the Commit phase, V has no information about P's input x.
- **Binding:** After the Commit phase, there exists only one value x that will be accepted by V in the Open phase.

- **Extra property:** Additive Homomorphism

Com $(x, r) = c$ Com $(x', r') = c'$ $\Rightarrow$ $c * c' = \mathrm{Com}(x + x'\ ,\ r + r')$

Malicious MPC with t<n/2 (GMW)

Tools 3/3 : Public Zero Knowledge Proofs of Knowledge

Inputs:

- All parties know a value y and a relation $R(\cdot, y) \in \{0,1\}$

Properties:

- *(completess)* Someone who knows a (witness) w such that $R(w, y)=1$ can convince everyone about his knowledge
- *(soundness)* If there exists no w such that $R(w, y)=1$, then no one can succeed in convincing the others about the opposite
- *(zero-knowledge)* The proof reveals no information about w

Malicious MPC with t<n/2 (GMW)

Tools 3/3 : Public Zero Knowledge Proofs of Knowledge

Inputs:

- All parties know a value y and a relation $R(\cdot, y) \in \{0,1\}$

Properties:

- *(completess)* Someone who knows a (witness) w such that $R(w, y)=1$ can convince everyone about his knowledge
- *(soundness)* If there exists no w such that $R(w, y)=1$, then no one can succeed in convincing the others about the opposite
- *(zero-knowledge)* The proof reveals no information about w

Example: Proving knowledge of a committed value without revealing anything about the value:

- y is a commitment c
- $R(w,y) = 1$ iff $w=(x,r)$ and $Ver(c,x,r)=1$

Malicious MPC with t<n/2 (GMW)

The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

Malicious MPC with t<n/2 (GMW)

The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

Round 0:
Every P_i commits to its input and randomness

Rounds $1 \ldots \varrho_\pi + 1$**:**
Execute π round-by-round *over Broadcast* so that in each round

- every party proves (in ZK) that he follows π
- if the ZKP of some p_i fails then invoke the ***Recovery process*** to publicly announce all p_i's shares.

Malicious MPC with t<n/2 (GMW)

The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

Round 0:
Every P_i commits to its input and randomness

Rounds $1 \ldots \varrho_\pi + 1$**:**
Execute π round-by-round *over Broadcast* so that in each round

- every party proves (in ZK) that he follows π
- if the ZKP of some p_i fails then invoke the ***Recovery process*** to publicly announce all p_i's shares.

Malicious MPC with t<n/2 (GMW)

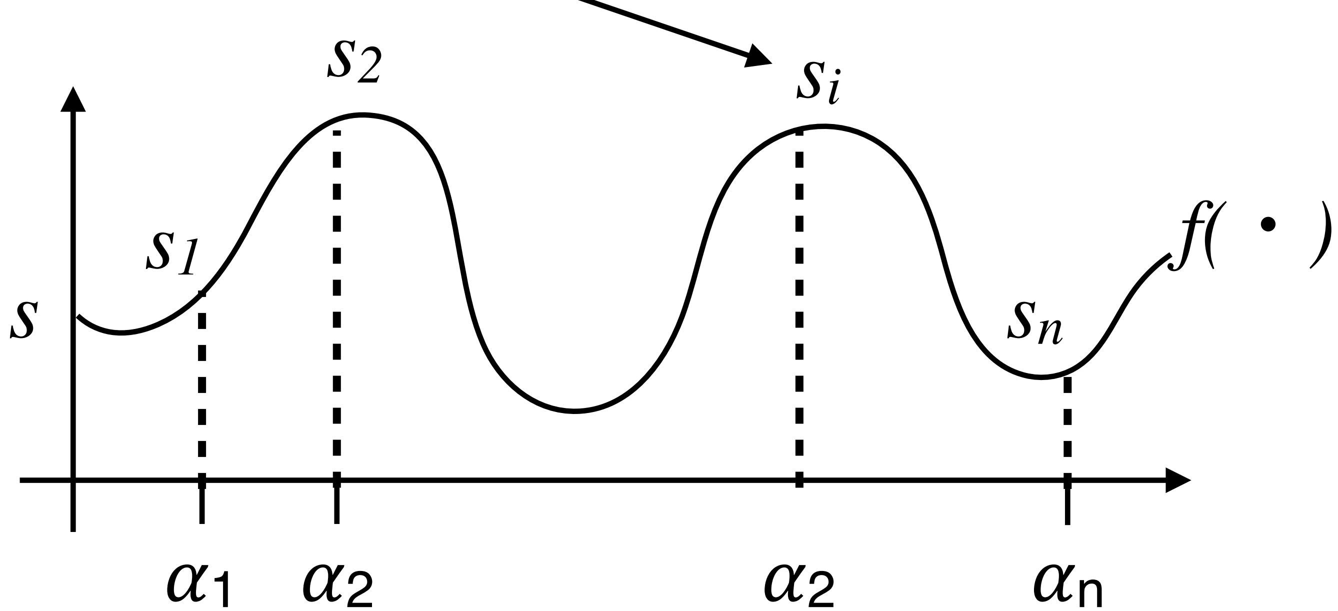
The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

Recovery gadget:

- When p_i fails then the remaining parties reconstruct all his shares
- For each share s_i of p_i the parties compute a sharing of s_i using the linearity gadget with ZK proofs and then reconstruct it.

$$s_i = f(\alpha_i) = \sum_{j \in [n] \setminus i} \ell_i(\alpha_i) s_j$$



Malicious MPC with t<n/2 (GMW)

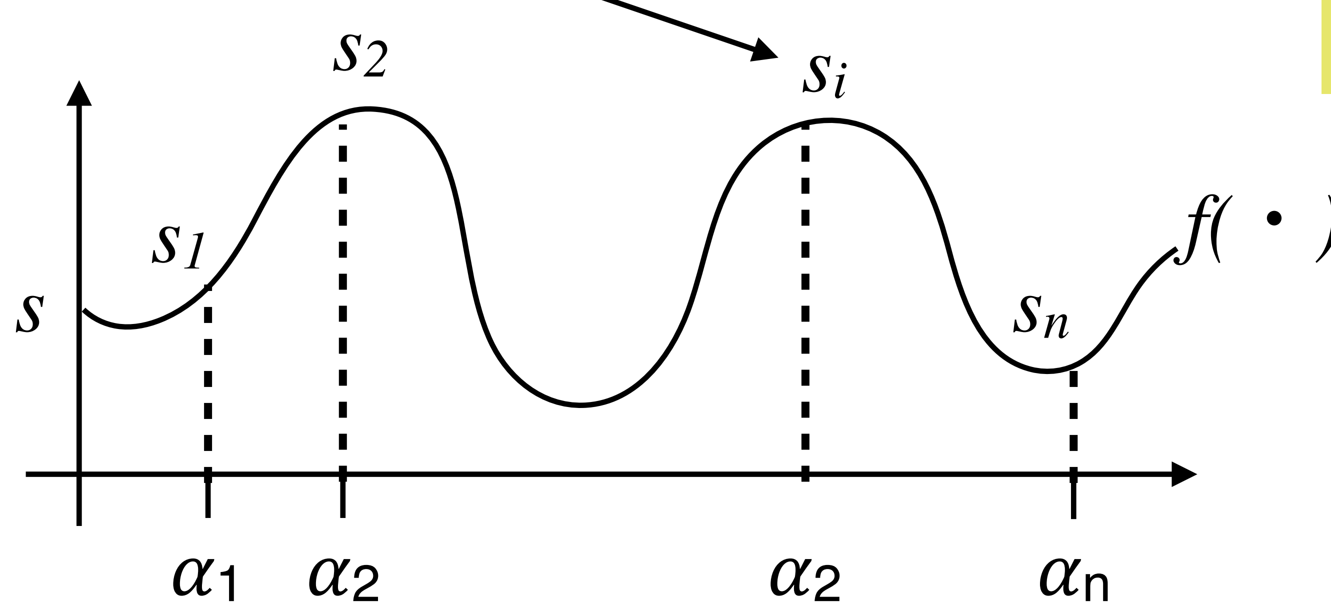
The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

Recovery gadget:

- When p_i fails then the remaining parties reconstruct all his shares
- For each share s_i of p_i the parties compute a sharing of s_i using the linearity gadget with ZK proofs and then reconstruct it.

$$s_i = f(\alpha_i) = \sum_{j \in [n] \setminus i} \ell_i(\alpha_i) s_j$$



Works because $t<n/2$, hence there are enough (i.e, $t+1$) parties to interpolate

Malicious MPC with t<n/2 (GMW)

The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

Round 0:
Every P_i commits to its input and randomness

Rounds $1 \ldots \varrho_\pi + 1$**:**
Execute π round-by-round *over Broadcast* so that in each round

- every party proves (in ZK) that he follows π
- if the ZKP of some p_i fails then invoke the ***Recovery process*** to publicly announce all p_i 's shares.

Malicious MPC with t<n/2 (GMW)

The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

Security (with abort)

- **Privacy:** The parties see the following:
  - Setup
  - Commitments
  - Messages from π
- **Correctness:**
  - If all ZKPs succeed this means that the parties follow their protocol
  - Only corrupted-prover ZKPs might fail ⇒ there will be $n - t > n/2$ to recover the missing values

Round 0:
Every P_i commits to its input and randomness

Rounds $1 \dots \varrho_\pi + 1$**:**
Execute π round-by-round *over Broadcast* so that in each round

- every party proves (in ZK) that he follows π
- if the ZKP of some p_i fails then invoke the ***Recovery process*** to publicly announce all p_i's shares.

Malicious MPC with t<n/2 (GMW)

The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

What if corrupted parties use bad randomness?

Round 0:
Every P_i commits to its input and randomness

Rounds $1 \ldots \varrho_\pi + 1$:
Execute π round-by-round *over Broadcast* so that in each round

- every party proves (in ZK) that he follows π
- if the ZKP of some p_i fails then invoke the ***Recovery process*** to publicly announce all p_i's shares.

Security (with abort)

- **Privacy:** The parties see the following:
  - Setup
  - Commitments
  - Messages from π
- **Correctness:**
  - If all ZKPs succeed this means that the parties follow their protocol
  - Only corrupted-prover ZKPs might fail ⇒ there will be $n - t > n/2$ to recover the missing values

Malicious MPC with t<n/2 (GMW)

The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

Coin-tossing protocol (idea):

Parties can make p_i committed to a random R_i

- Every p_j (including p_i) commits to a random R_{ij}, i.e., computes and broadcasts $c_{ij} = Com(R_{ij}, r_{ij})$
- Every p_j sends r_{ij} to p_i
- p_i computes $c_{i1} * \ldots * c_{in}$ which (using the homomorphic property) is a commitment to $R_i = R_{i1} + \ldots + R_{in}$ with opening-randomness $r_i = r_{i1} + \ldots + r_{in}$.

Malicious MPC with t<n/2 (GMW)

The GMW Compiler

Compile a semi-honest SFE protocol π into (malicious) secure

Round 0:
Every P_i commits to its input and randomness

coin-tossing

Rounds $1 \ldots \varrho_\pi + 1$**:**
Execute π round-by-round *over Broadcast* so that in each round

- every party proves (in ZK) that he follows π
- if the ZKP of some p_i fails then invoke the ***Recovery process*** to publicly announce all p_i's shares.

Security (with abort)

- **Privacy:** The parties see the following:
  - Setup
  - Commitments
  - Messages from π
- **Correctness:**
  - If all ZKPs succeed this means that the parties follow their protocol
  - Only corrupted-prover ZKPs might fail ⇒ there will be $n - t > n/2$ to recover the missing values

Known Bounds

Adv. Type	Security	Corruption Bound	Requires
semi-honest (passive)	Information theoretic (IT)	t<n/2 [BGW88,CCD88]	Sec. channels ✔
	Computational	t<n [GMW87]	Sec. channels + OT
malicious (active)	information theoretic	t<n/3 [BGW88,CCD88]	Sec. channels
	computational (or IT w. negligible error)	t<n/2 [GMW87,RB89]	Broadcast ✔
	computational without fairness	t<n [GMW87]	Broadcast + OT

Known Bounds

Adv. Type	Security	Corruption Bound	Requires
semi-honest (passive)	Information theoretic (IT)	t<n/2 [BGW88,CCD88]	Sec. channels ✔
	Computational	t<n [GMW87]	Sec. channels + OT
malicious (active)	information theoretic	t<n/3 [BGW88,CCD88]	Sec. channels
	computational (or IT w. negligible error)	t<n/2 [GMW87,RB89]	Broadcast ✔
	computational without fairness	t<n [GMW87]	Broadcast + OT

Known Bounds

Adv. Type	Security	Corruption Bound	Requires
semi-honest (passive)	Information theoretic (IT)	t<n/2 [BGW88,CCD88]	Sec. channels ✔
	Computational	t<n [GMW87]	Sec. channels + OT
malicious (active)	information theoretic	t<n/3 [BGW88,CCD88]	Sec. channels
	computational (or IT w. negligible error)	t<n/2 [GMW87,RB89]	Broadcast ??? ✔
	computational without fairness	t<n [GMW87]	Broadcast + OT

Broadcast for t<n/3

Consensus:(Inputs**:** $x_1, \ldots, x_n$, Outputs: $y_1, \ldots, y_n$)

- (consistency) There exists y s.t. $y_j = y$ for all p_j
- (validity) If all honest p_i has input $x_i = x$ then $y = x$

Broadcast for t<n/3

Consensus:(Inputs: $x_1, \ldots, x_n$, Outputs: $y_1, \ldots, y_n$)

- (consistency) There exists y s.t. $y_j = y$ for all p_j
- (validity) If all honest p_i has input $x_i = x$ then $y = x$

Theorem:

- Consensus is possible (unconditionally) iff $t < n/3$ [LSP82,BGP89]

Broadcast for t<n/3

Consensus:(Inputs: $x_1, \ldots, x_n$, Outputs: $y_1, \ldots, y_n$)

- (consistency) There exists y s.t. $y_j = y$ for all p_j
- (validity) If all honest p_i has input $x_i = x$ then $y = x$

Theorem:

- Consensus is possible (unconditionally) iff $t < n/3$ [LSP82,BGP89]

Consensus ⇒ Broadcast:

1. Sender sends his input to every p_i
2. The parties runs consensus on inputs the received values

Broadcast for t<n/3

Consensus:(Inputs: $x_1, \ldots, x_n$, Outputs: $y_1, \ldots, y_n$)

- (consistency) There exists y s.t. $y_j = y$ for all p_j
- (validity) If all honest p_i has input $x_i = x$ then $y = x$

Theorem:

- Consensus is possible (unconditionally) iff $t < n/3$ [LSP82,BGP89]

Consensus ⇒ Broadcast:

1. Sender sends his input to every p_i
2. The parties runs consensus on inputs the received values

Security proof of Consensus ⇒ Broadcast:

- (consistency) Follows from consistency of consensus
- (validity) If the sender is honest then consensus is executed with all honest p_i's having input the sender's input

Known Bounds

Adv. Type	Security	Corruption Bound	Requires
semi-honest (passive)	Information theoretic (IT)	t<n/2 [BGW88,CCD88]	Sec. channels ✔
	Computational	t<n [GMW87]	Sec. channels + OT
malicious (active)	information theoretic	t<n/3 [BGW88,CCD88]	Sec. channels
	computational (or IT w. negligible error)	t<n/2 [GMW87,RB89]	Broadcast ??? ✔
	computational without fairness	t<n [GMW87]	Broadcast + OT

Known Bounds

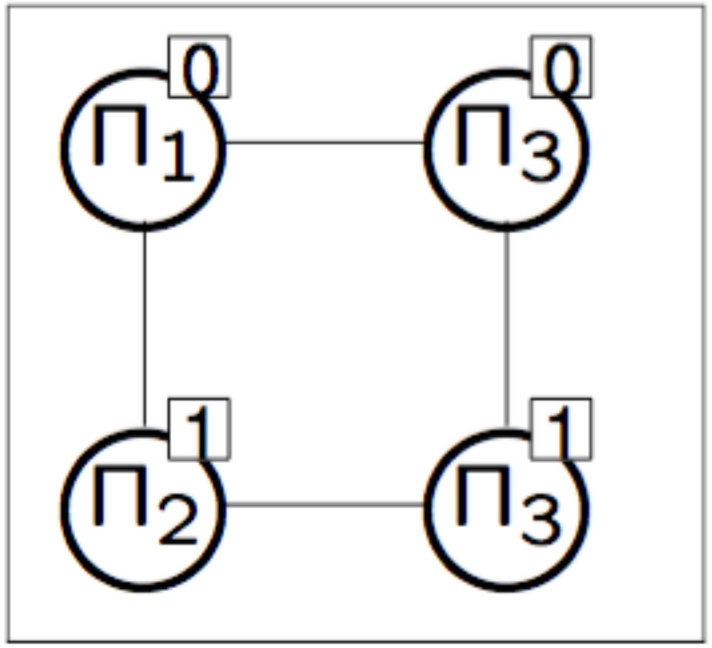
Adv. Type	Security	Corruption Bound	Requires
semi-honest (passive)	Information theoretic (IT)	t<n/2 [BGW88,CCD88]	Sec. channels ✔
	Computational	t<n [GMW87]	Sec. channels + OT
malicious (active)	information theoretic	t<n/3 [BGW88,CCD88]	Sec. channels
	computational (or IT w. negligible error)	t<n/2 [GMW87,RB89]	PKI + channels ✔
	computational without fairness	t<n [GMW87]	Broadcast + OT

Impossibility of Broadcast for n=3, t=1

Assume a protocol (Π_1, Π_2, Π_3) allowing p_3 to broadcast a bit.

Impossibility of Broadcast for n=3, t=1

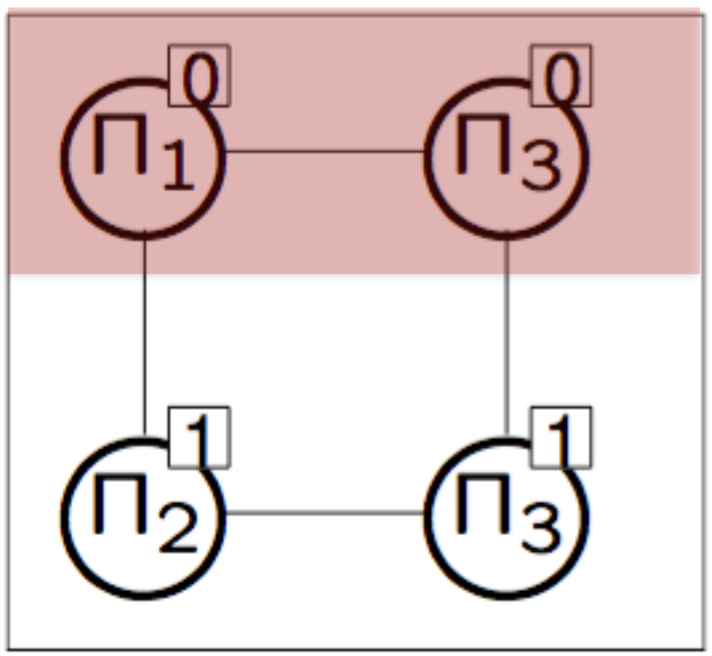
Assume a protocol (Π_1, Π_2, Π_3) allowing p_3 to broadcast a bit.



Impossibility of Broadcast for n=3, t=1

Assume a protocol (Π_1, Π_2, Π_3) allowing p_3 to broadcast a bit.

p_1 is corrupted
p_3 has input 1



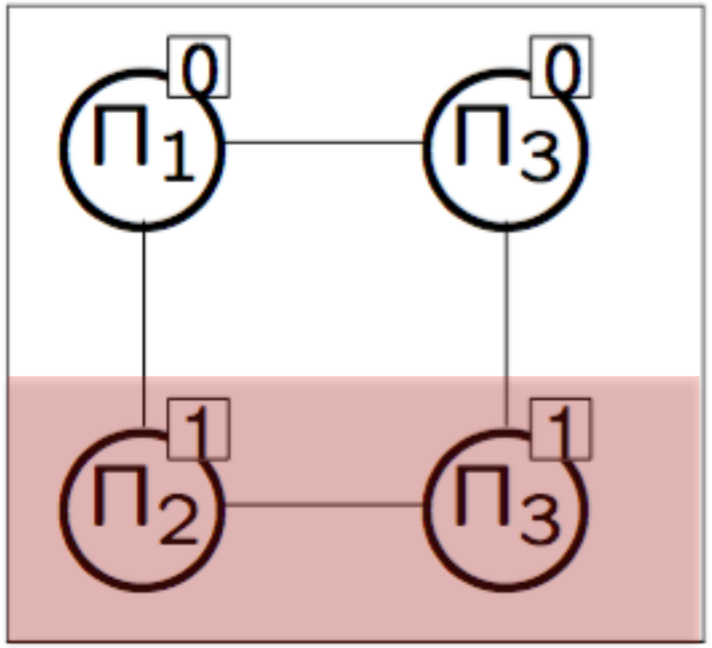
Correctness⇒

p_2 outputs 1

Impossibility of Broadcast for n=3, t=1

Assume a protocol (Π_1, Π_2, Π_3) allowing p_3 to broadcast a bit.

p_2 is corrupted
p_3 has input 0



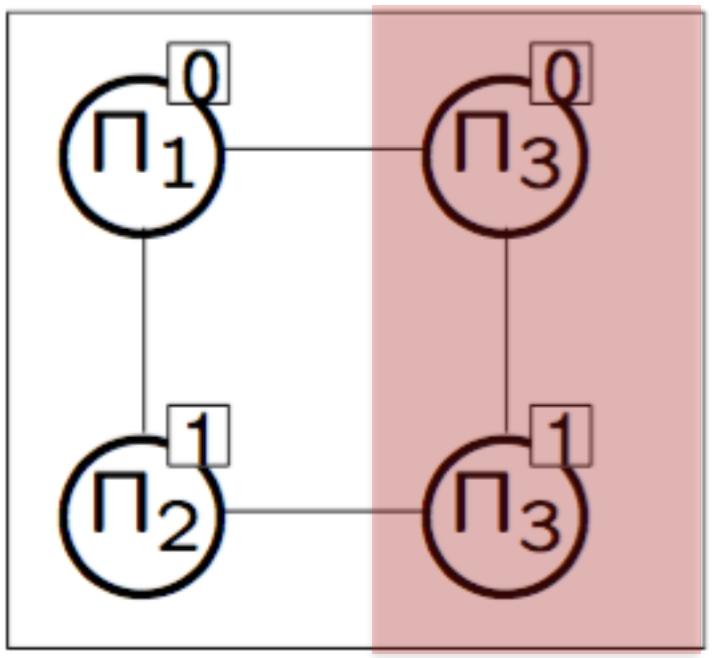
Correctness⇒

p_1 outputs 0

Impossibility of Broadcast for n=3, t=1

Assume a protocol (Π_1, Π_2, Π_3) allowing p_3 to broadcast a bit.

p_3 is corrupted

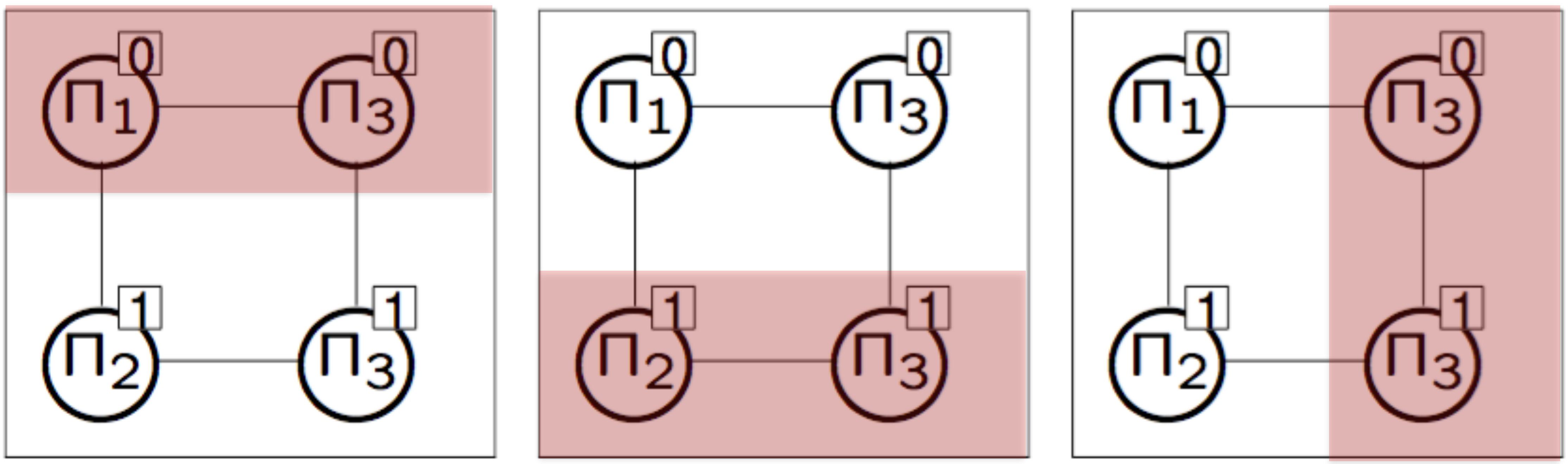


consistency⇒

p_1 outputs the same as p_2

Impossibility of Broadcast for n=3, t=1

Assume a protocol (Π_1, Π_2, Π_3) allowing p_3 to broadcast a bit.



Correctness⇒ p_2 outputs 1

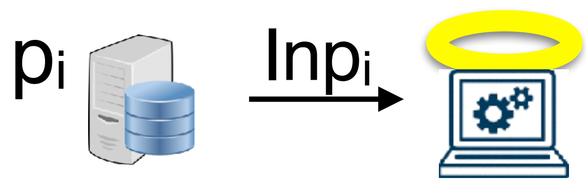
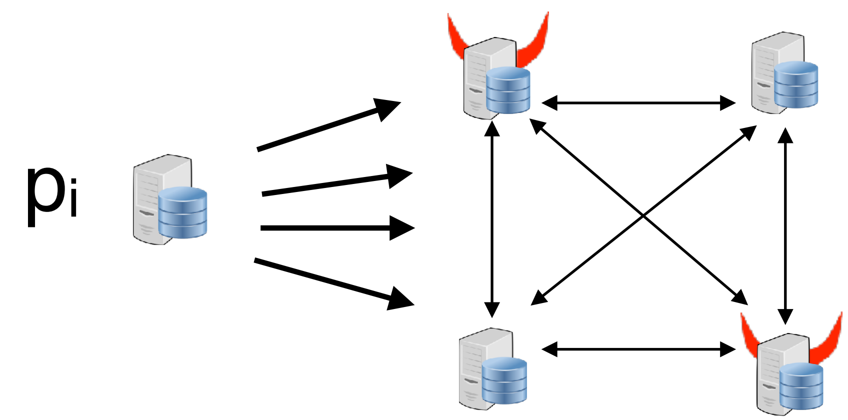
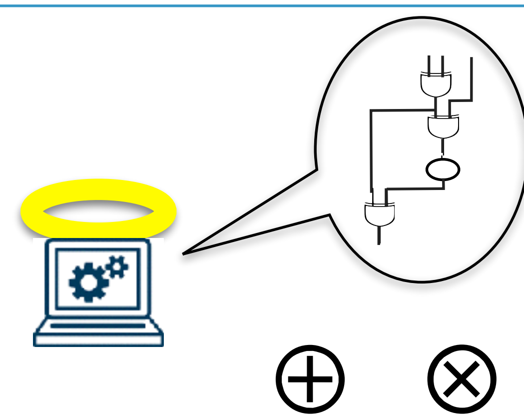
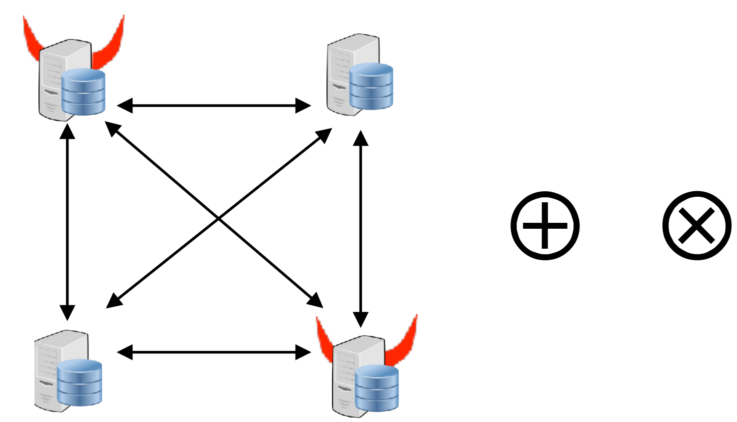
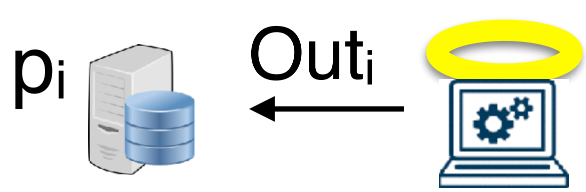
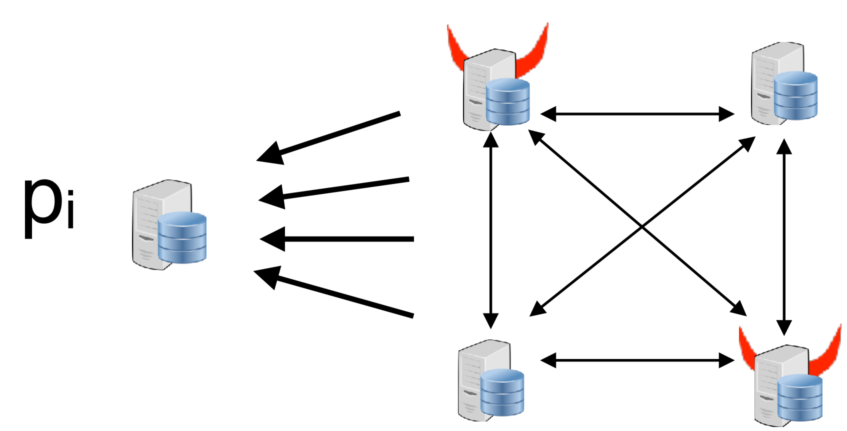
Correctness⇒ p_1 outputs 0

consistency⇒ p_1 outputs the same as p_2

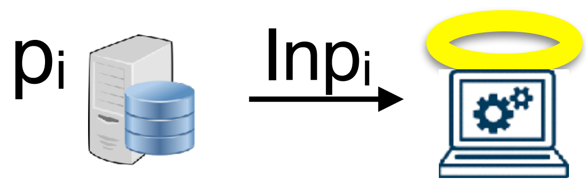
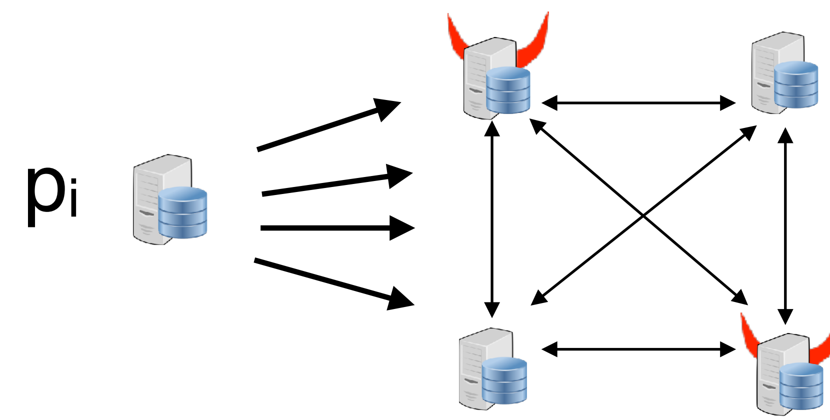
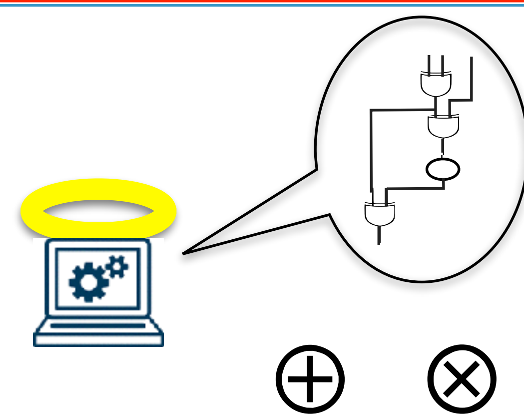
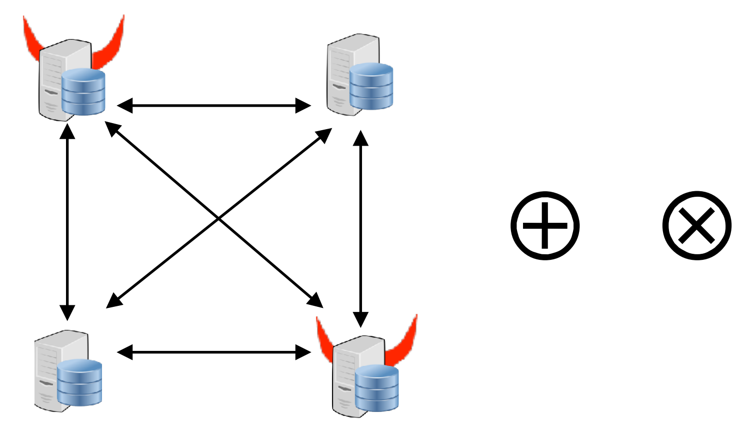
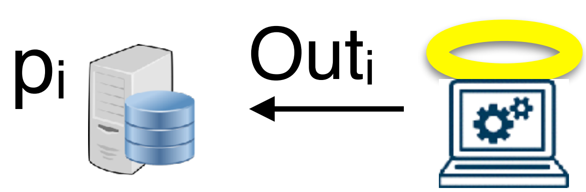
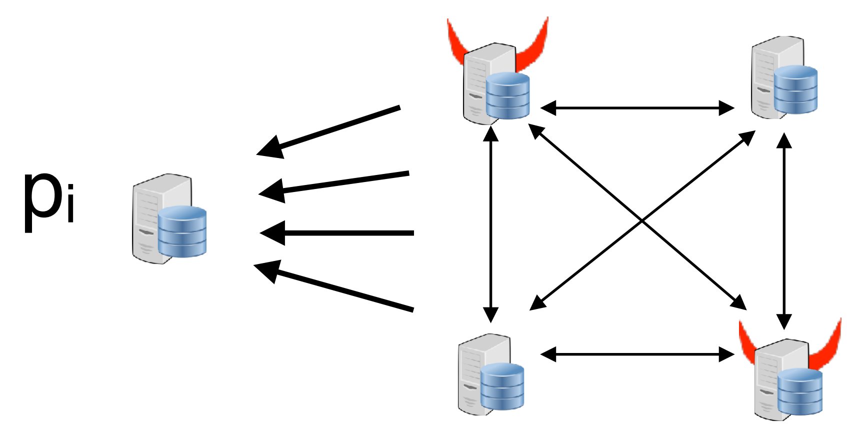
Known Bounds

Adv. Type	Security	Corruption Bound	Requires
semi-honest (passive)	Information theoretic (IT)	t<n/2 [BGW88,CCD88]	Sec. channels ✔
	Computational	t<n [GMW87]	Sec. channels + OT
malicious (active)	information theoretic	t<n/3 [BGW88,CCD88]	Sec. channels
	computational (or IT w. negligible error)	t<n/2 [GMW87,RB89]	PKI ✔
	computational without fairness	t<n [GMW87]	Broadcast + OT

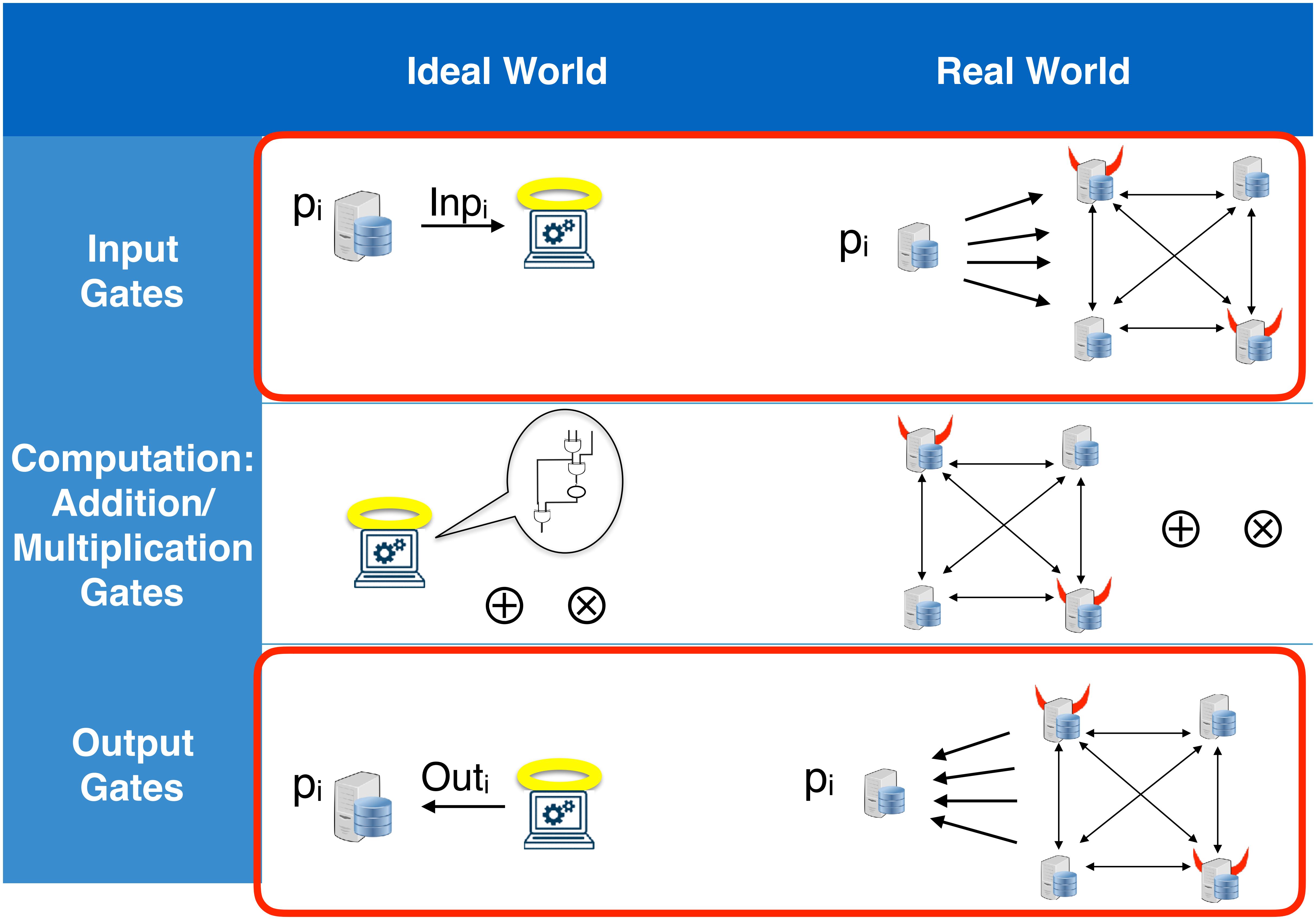
MPC Goal

	Ideal World	Real World
Input Gates		
Computation: Addition/ Multiplication Gates		
Output Gates		

MPC Goal

	Ideal World	Real World
Input Gates		
Computation: Addition/ Multiplication Gates		
Output Gates		

MPC Goal



Ideal World
Real World
Input Gates
Computation: Addition/ Multiplication Gates
Output Gates
p_i
Inp_i
Out_i
$\oplus$ $\otimes$

MPC with Malicious Adversary — t<n/3

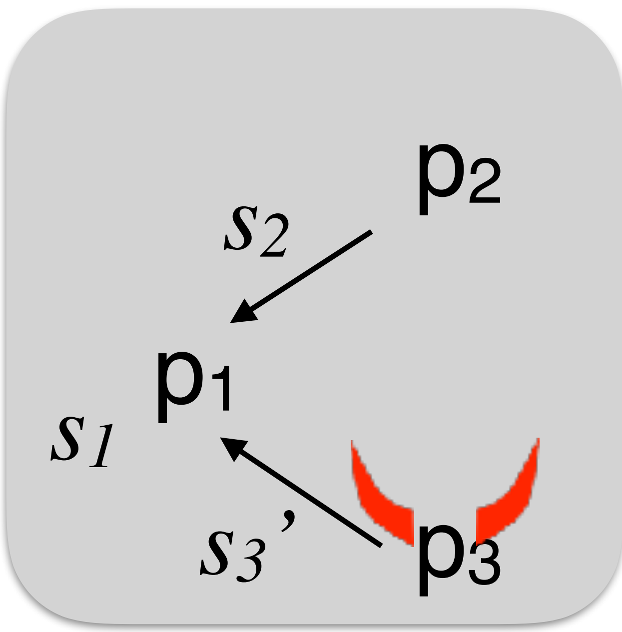
The t<n/2 solution does not even work given broadcast

- Let's look at 3 parties with 1 corruption
  - Secrets s shared as (s_1, s_2, s_3) , i.e., p_i holds s_i

MPC with Malicious Adversary — t<n/3

The t<n/2 solution does not even work given broadcast

- Let's look at 3 parties with 1 corruption
  - Secrets s shared as (s_1, s_2, s_3) , i.e., p_i holds s_i



correctness ⇒

$\forall\ s_3'\ Rec(s_1, s_2, s_3') = s$

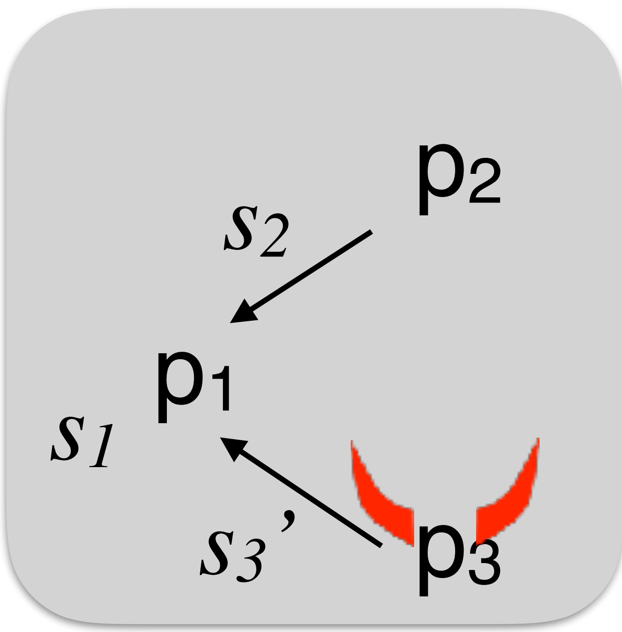
⇒ ∃ Rec_{12} s.t.

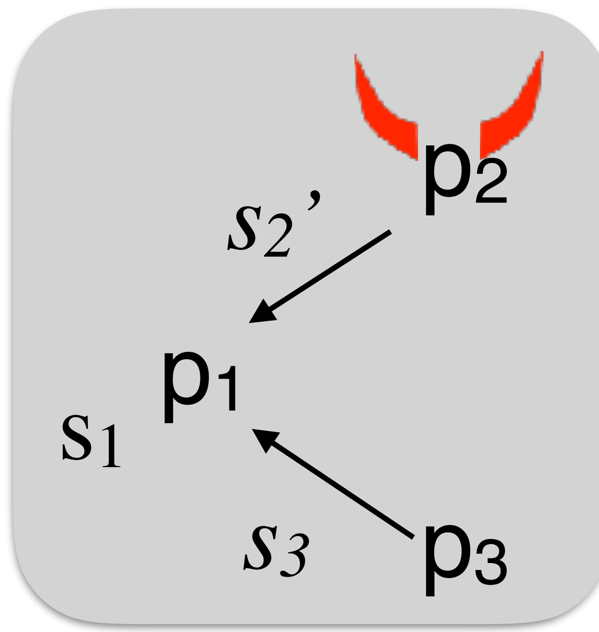
$Rec_{12}(s_1, s_2) = s$

MPC with Malicious Adversary — t<n/3

The t<n/2 solution does not even work given broadcast

- Let's look at 3 parties with 1 corruption
  - Secrets s shared as (s_1, s_2, s_3) , i.e., p_i holds s_i





correctness ⇒

$\forall\ s_3'\ Rec(s_1, s_2, s_3') = s$

$\Rightarrow \exists\ Rec_{12}$ s.t.

$Rec_{12}(s_1, s_2) = s$

correctness ⇒

$\forall\ s_2'\ Rec(s_1, s_2', s_3) = s$

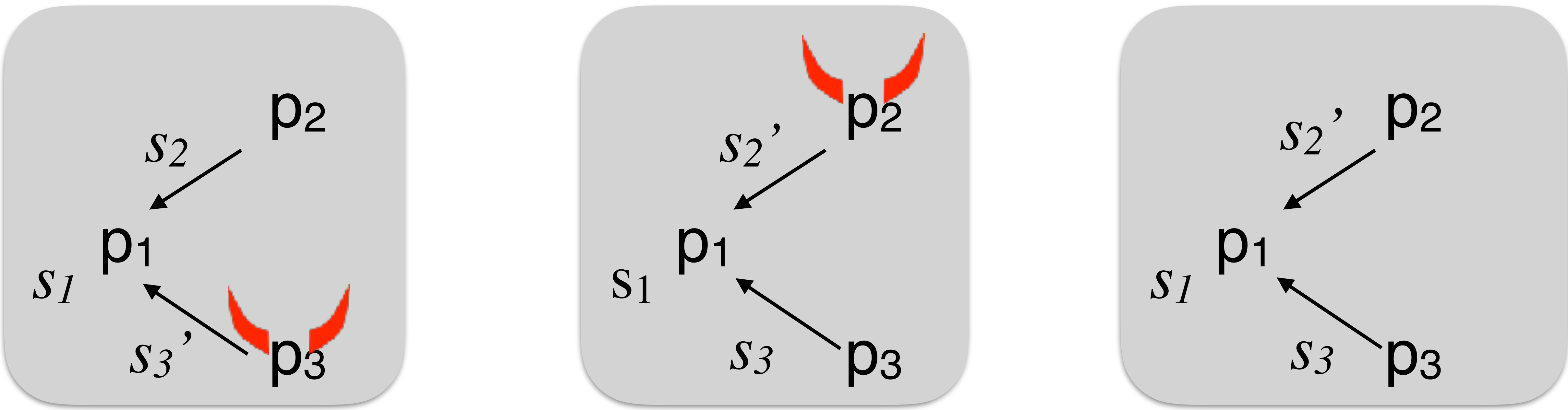
$\Rightarrow \exists\ Rec_{13}$ s.t.

$Rec_{13}(s_1, s_3) = s$

MPC with Malicious Adversary — t<n/3

The t<n/2 solution does not even work given broadcast

- Let's look at 3 parties with 1 corruption
  - Secrets s shared as (s_1, s_2, s_3) , i.e., p_i holds s_i



correctness ⇒

$\forall\, s_3'\ Rec(s_1, s_2, s_3') = s$

$\Rightarrow \exists\ Rec_{12}$ s.t.

$Rec_{12}(s_1, s_2) = s$

correctness ⇒

$\forall\, s_2'\ Rec(s_1, s_2', s_3) = s$

$\Rightarrow \exists\ Rec_{13}$ s.t.

$Rec_{13}(s_1, s_3) = s$

1-privacy ⇒

s_1 has no info about s

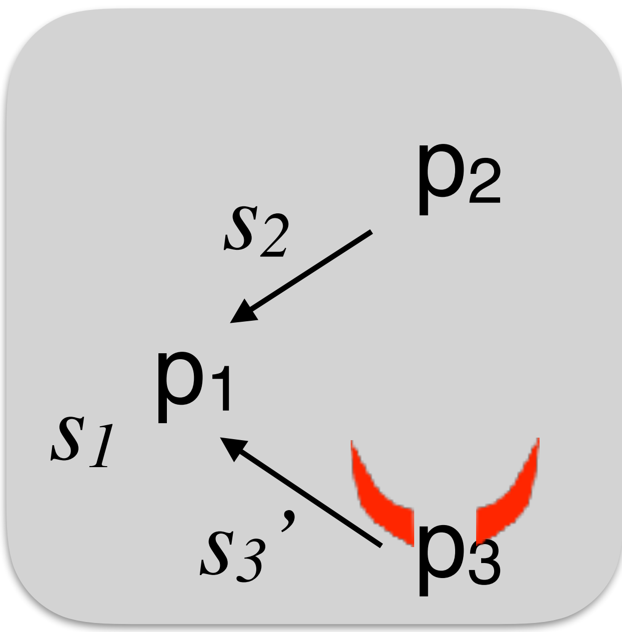
- $\forall\, s'\ \exists\, s_2'$ s.t.

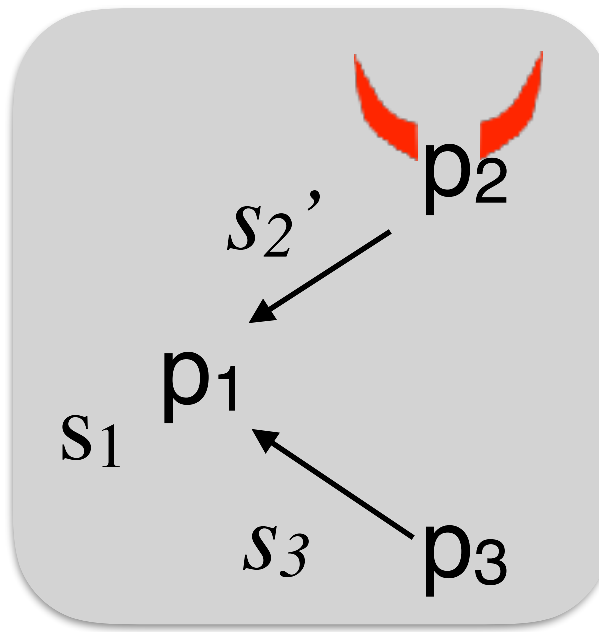
  $Rec_{12}(s_1, s_2') = s'$

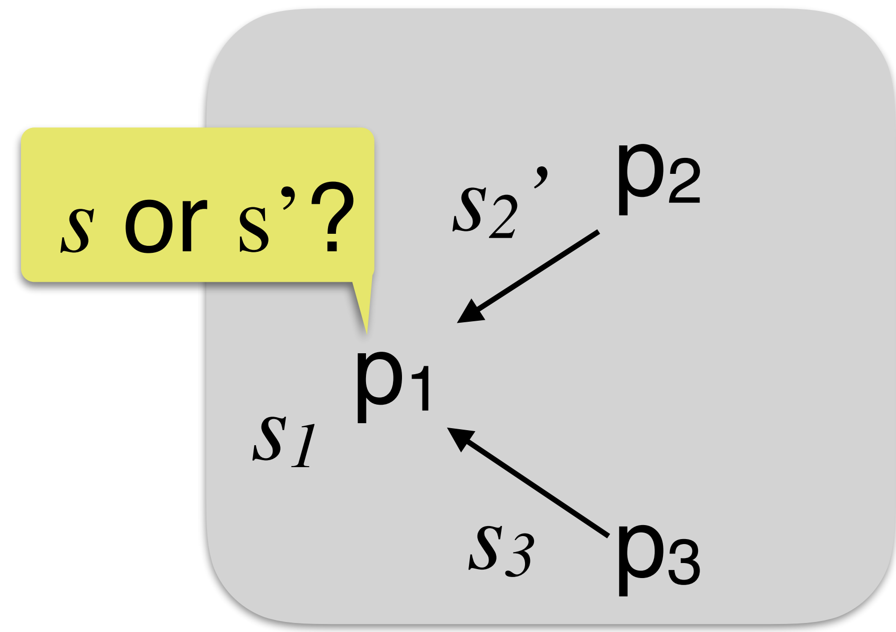
MPC with Malicious Adversary — t<n/3

The t<n/2 solution does not even work given broadcast

- Let's look at 3 parties with 1 corruption
  - Secrets s shared as (s_1, s_2, s_3) , i.e., p_i holds s_i







correctness ⇒

$\forall\ s_3'\ Rec(s_1, s_2, s_3') = s$

$\Rightarrow \exists\ Rec_{12}$ s.t.

$Rec_{12}(s_1, s_2) = s$

correctness ⇒

$\forall\ s_2'\ Rec(s_1, s_2', s_3) = s$

$\Rightarrow \exists\ Rec_{13}$ s.t.

$Rec_{13}(s_1, s_3) = s$

1-privacy ⇒

s_1 has no info about s

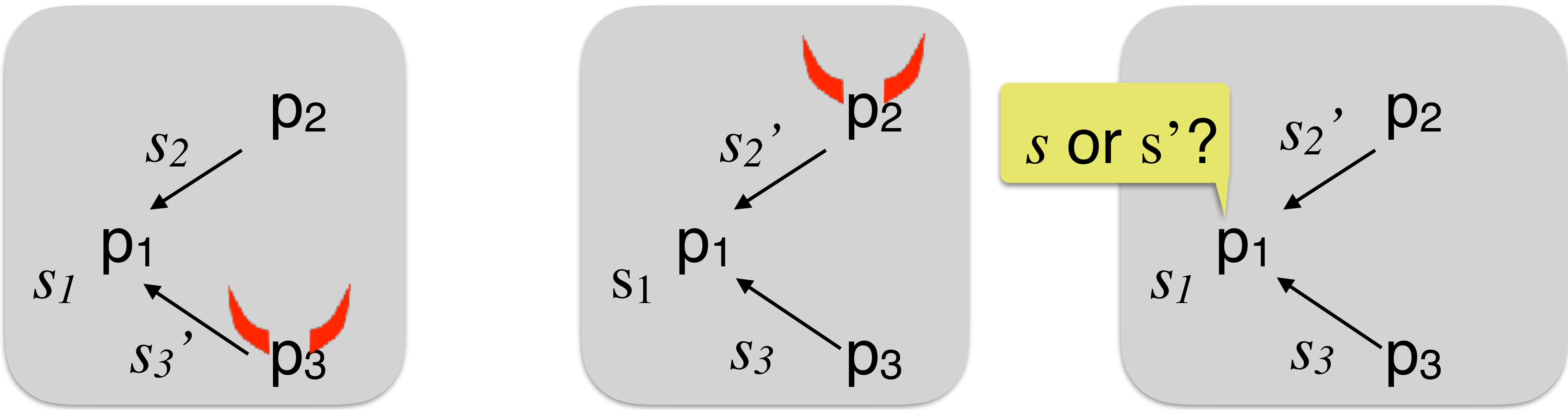
- $\forall\ s'\ \exists\ s_2'$ s.t.

  $Rec_{12}(s_1, s_2') = s'$

MPC with Malicious Adversary — t<n/3

The t<n/2 solution does not even work given broadcast

- Let's look at 3 parties with 1 corruption
  - Secrets s shared as (s_1, s_2, s_3) , i.e., p_i holds s_i



correctness ⇒

∀ s_3'

⇒ ∃

$Rec_{12}(s_1, s_2) = s$

correctness ⇒

$Rec_{13}(s_1, s_3) = s$

1-privacy ⇒

out s

$Rec_{12}(s_1, s_2') = s'$

We need a secret sharing scheme that ensures honest parties do not loose their shared state

Verifiable t-out-of-n Secret Sharing

Verifiable Secret Sharing: A t-out-of-n verifiable secret sharing (VSS) scheme is a t-out-of-n secret sharing scheme $(\mathrm{Share}, \mathrm{Reconstruct})$ with the following properties:

- **(correctness)** If the dealer is honest during Share, then given the shares of any t parties, $Reconstruct$ outputs the secret s.
- **(t-privacy)** The shares of any set of $t\text{-}1$ parties include not information about s.
- **(commitment)** At the end of Share there is a unique value s' such that if the parties invoke $\mathrm{Reconstruct}$ the output will be s'

Verifiable t-out-of-n Secret Sharing

Verifiable Secret Sharing: A t-out-of-n verifiable secret sharing (VSS) scheme is a t-out-of-n secret sharing scheme $(\mathrm{Share}, \mathrm{Reconstruct})$ with the following properties:

- **(correctness)** If the dealer is honest during Share, then given the shares of any t parties, *Reconstruct* outputs the secret s.
- **(t-privacy)** The shares of any set of $t\text{-}1$ parties include not information about s.
- **(commitment)** At the end of Share there is a unique value s' such that if the parties invoke $\mathrm{Reconstruct}$ the output will be s'

(correctness) $\Rightarrow$ $s' = s$ when Dealer is honest in Share

Verifiable t-out-of-n Secret Sharing

Verifiable Secret Sharing: A t-out-of-n verifiable secret sharing (VSS) scheme is a t-out-of-n secret sharing scheme $(\mathrm{Share}, \mathrm{Reconstruct})$ with the following properties:

- **(correctness)** If the dealer is honest during Share, then given the shares of any t parties, $Reconstruct$ outputs the secret s.
- **(t-privacy)** The shares of any set of $t\text{-}1$ parties include not information about s.
- **(commitment)** At the end of Share there is a unique value $\mathrm{s'}$ such that if the parties invoke $\mathrm{Reconstruct}$ the output will be $\mathrm{s'}$

(correctness) $\Rightarrow$ $s' = s$ when Dealer is honest in Share

In a VSS the adversary cannot make the parties loose a shared value

Verifiable t-out-of-n Secret Sharing

Verifiable Secret Sharing: A t-out-of-n verifiable secret sharing (VSS) scheme is a t-out-of-n secret sharing scheme (Share, Reconstruct) with the following properties:

- **(correctness)** If the dealer is honest during Share, then given the shares of any t parties, *Reconstruct* outputs the secret s.
- **(t-privacy)** The shares of any set of $t\text{-}1$ parties include not information about s.
- **(commitment)** At the end of Share there is a unique value s' such that if the parties invoke Reconstruct the output will be s'

(correctness) $\Rightarrow$ $s' = s$ when Dealer is honest in Share

In a VSS the adversary cannot make the parties loose a shared value

Previous argument shows that VSS (without signatures) exists only if $t<n/3$

(t+1)-out-of-n VSS (t<n/3)

(t+1)-out-of-n VSS (t<n/3)

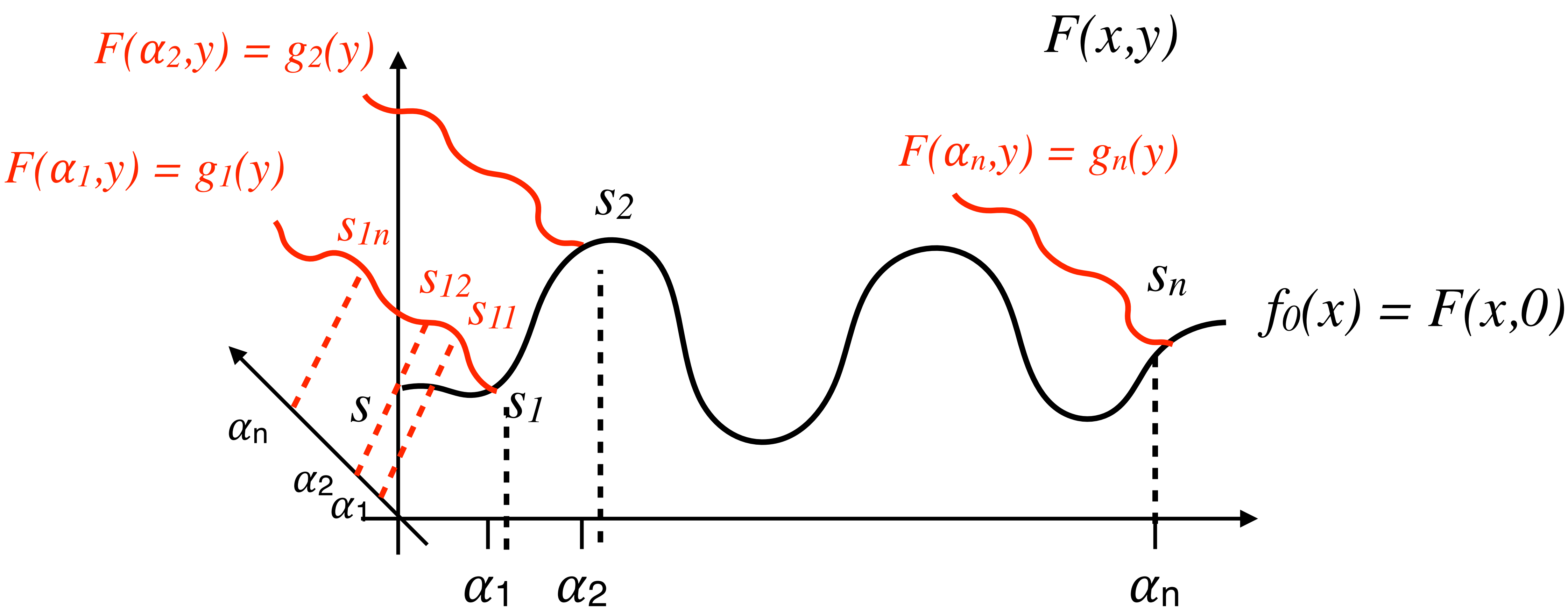
Share:

1. D chooses a random bivariate polynomial $F(x,y)$ of degree t in each variable, such that $f(0,0)=s$. Denote: $f_i(x) = F(x, \alpha_i)$, $g_j(y) = F(\alpha_j, y)$
2. Each party p_i receives $f_i(x)$ and $g_i(y)$

(t+1)-out-of-n VSS (t<n/3)

Share:

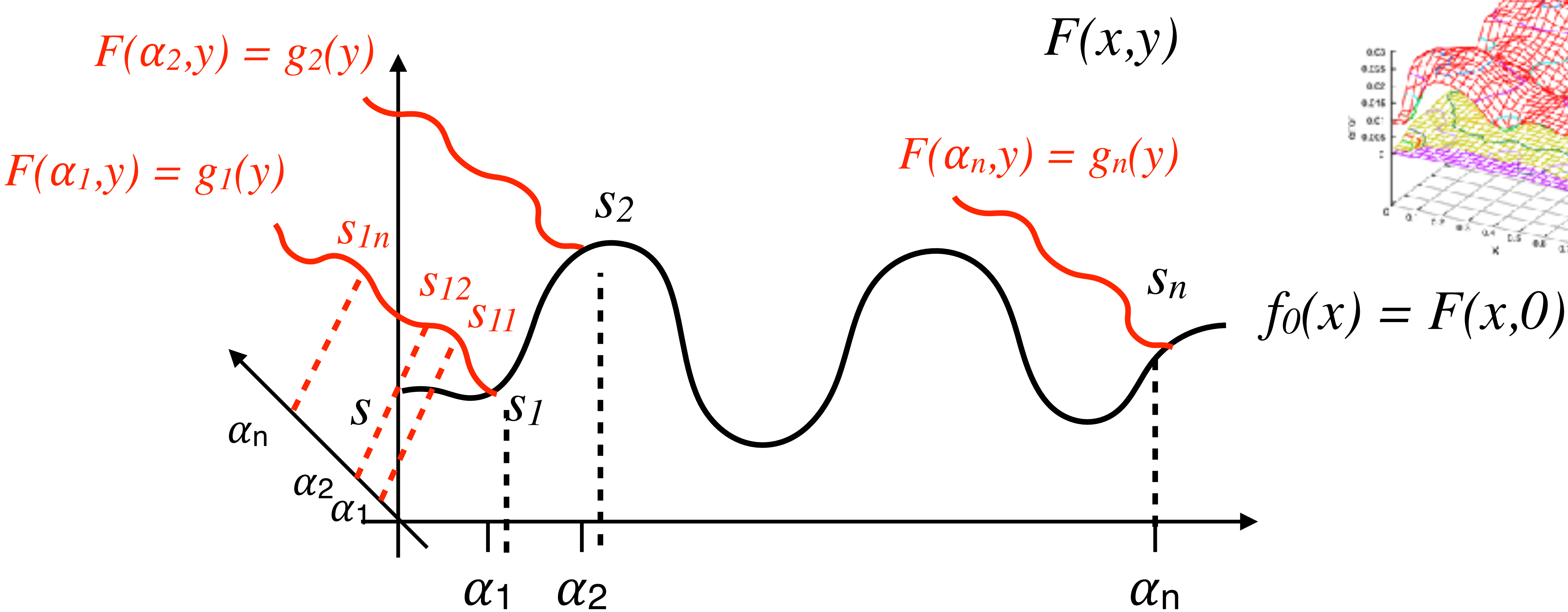
1. D chooses a random bivariate polynomial $F(x,y)$ of degree t in each variable, such that $f(0,0)=s$. Denote: $f_i(x) = F(x, \alpha_i)$, $g_j(y) = F(\alpha_j, y)$
2. Each party p_i receives $f_i(x)$ and $g_i(y)$

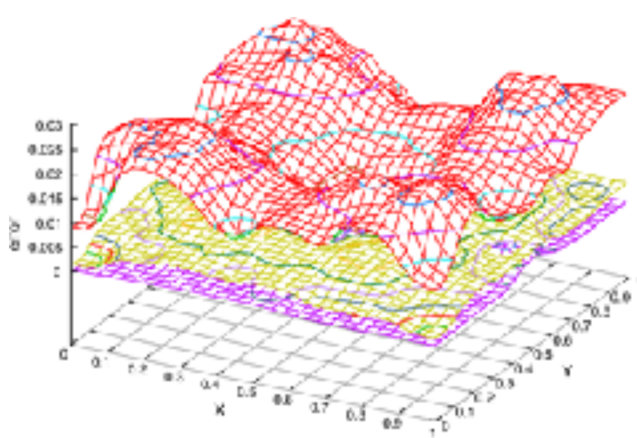


(t+1)-out-of-n VSS (t<n/3)

Share:

1. D chooses a random bivariate polynomial $F(x,y)$ of degree t in each variable, such that $f(0,0)=s$. Denote: $f_i(x) = F(x, \alpha_i)$, $g_j(y) = F(\alpha_j, y)$
2. Each party p_i receives $f_i(x)$ and $g_i(y)$

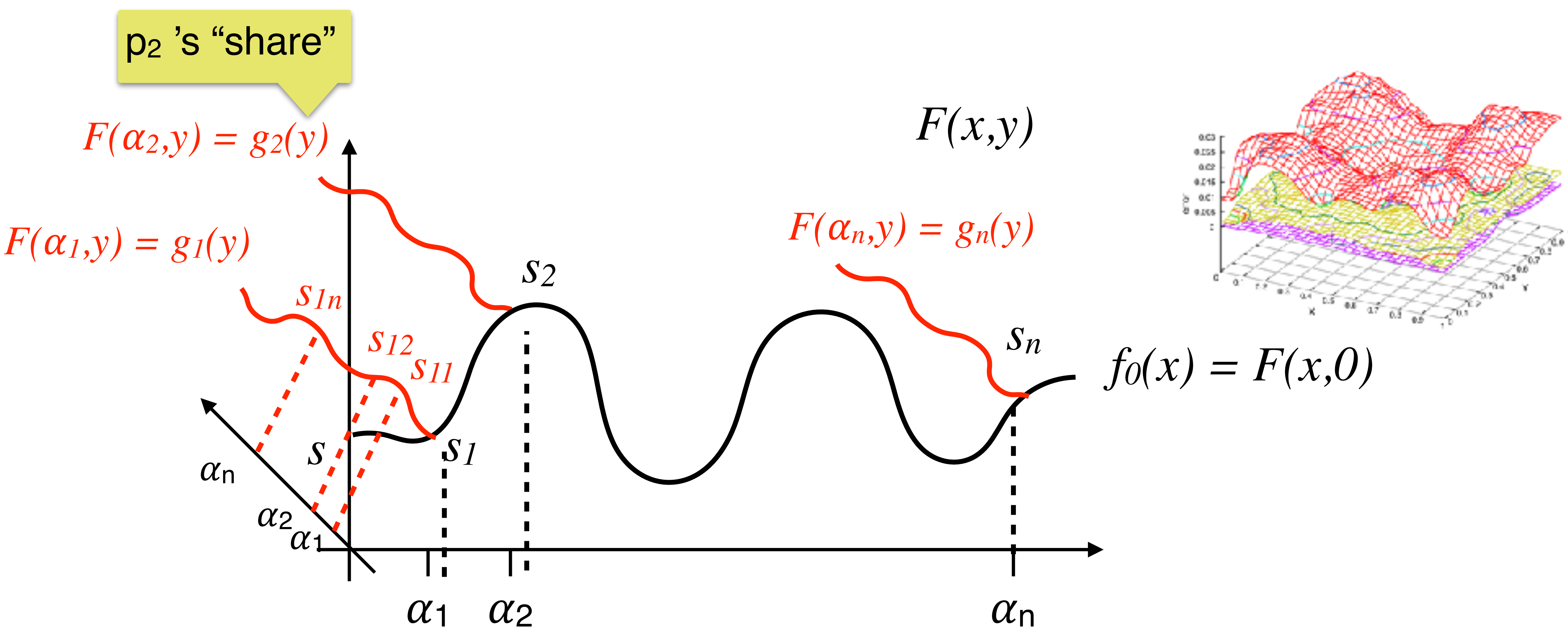




(t+1)-out-of-n VSS (t<n/3)

Share:

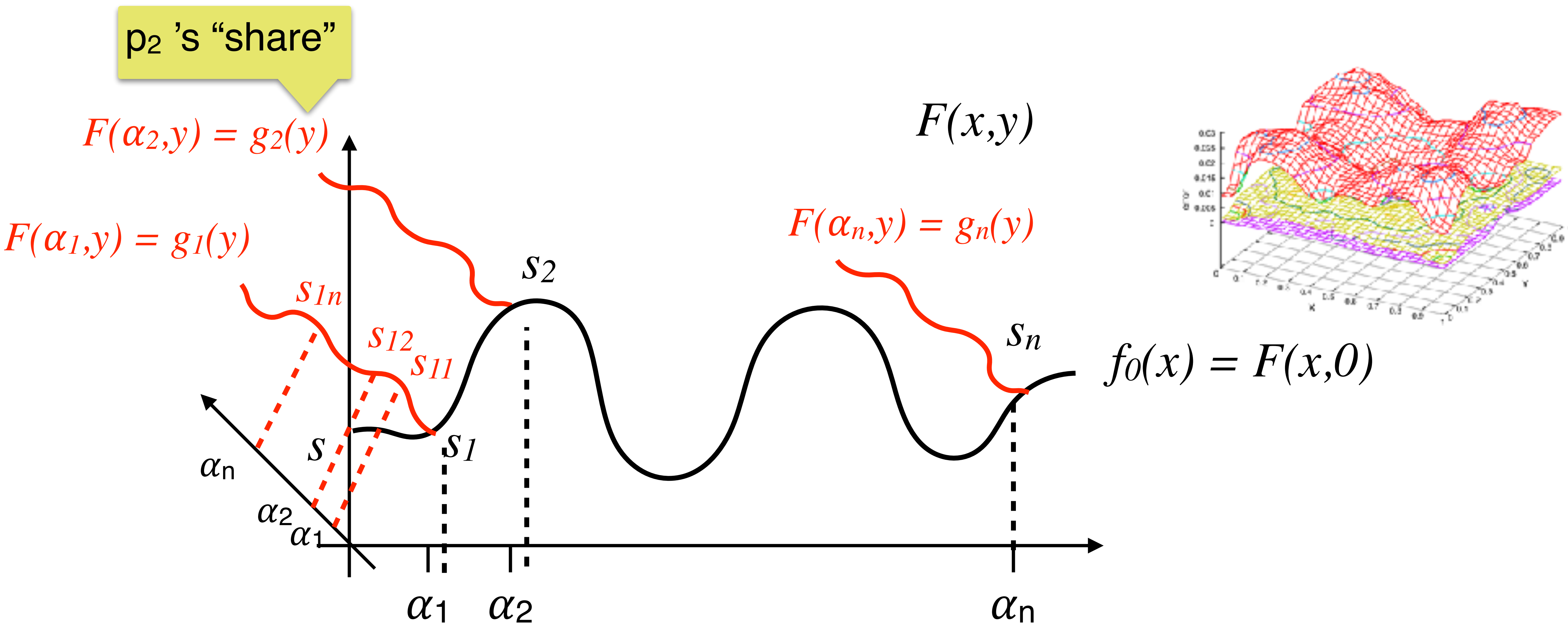
1. D chooses a random bivariate polynomial $F(x,y)$ of degree t in each variable, such that $f(0,0)=s$. Denote: $f_i(x) = F(x, \alpha_i)$, $g_j(y) = F(\alpha_j, y)$
2. Each party p_i receives $f_i(x)$ and $g_i(y)$



(t+1)-out-of-n VSS (t<n/3)

Share:

1. D chooses a random bivariate polynomial $F(x,y)$ of degree t in each variable, such that $f(0,0)=s$. Denote: $f_i(x) = F(x, \alpha_i)$, $g_j(y) = F(\alpha_j, y)$
2. Each party p_i receives $f_i(x)$ and $g_i(y)$
3. Each pair (p_i, p_j) confirms that $s_{ij} = f_i(\alpha_j) = g_j(\alpha_i)$ and $s_{ji}=f_j(\alpha_i) = g_i(\alpha_j)$.
4. Resolve conflict by public accusations answered by the dealer.



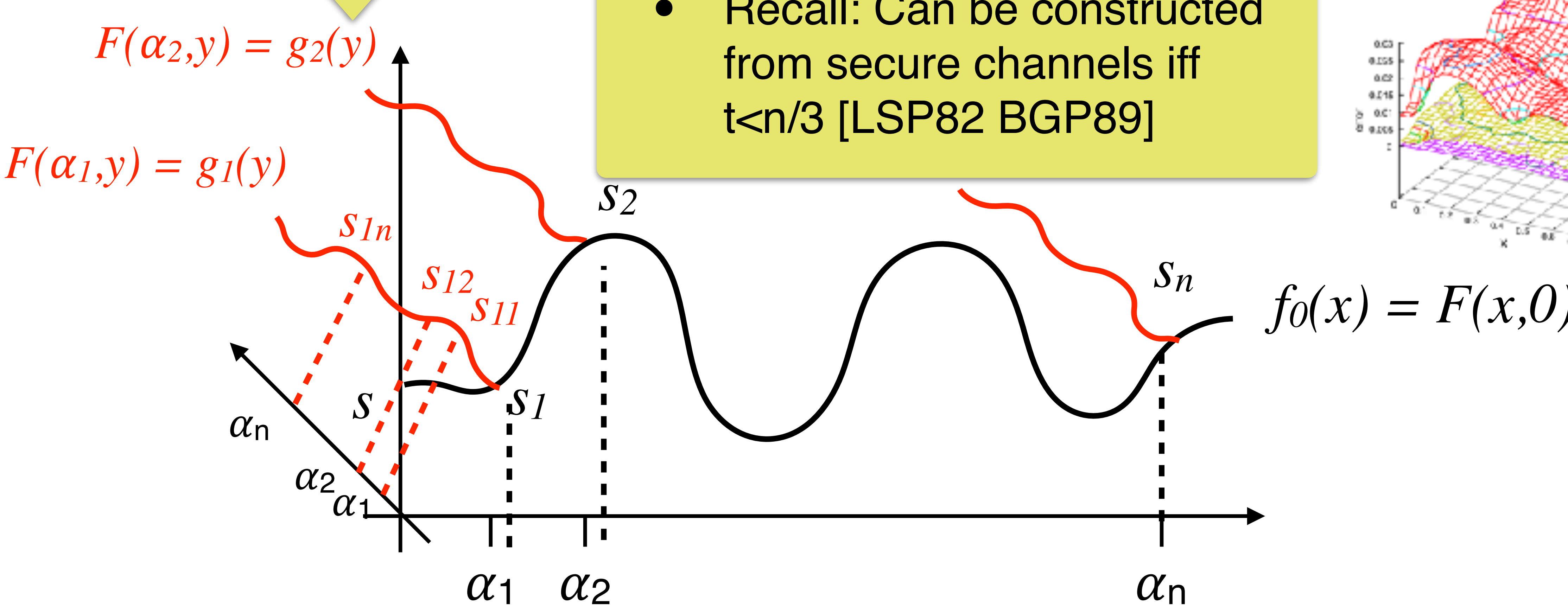
(t+1)-out-of-n VSS (t<n/3)

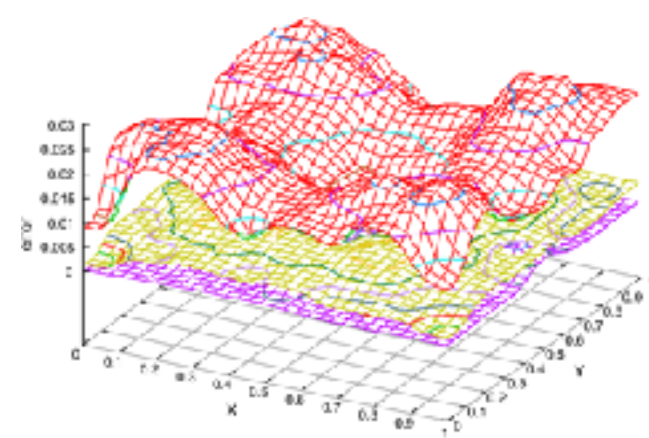
Share:

1. D chooses a random bivariate polynomial $F(x,y)$ of degree t in each variable, such that $f(0,0)=s$. Denote: $f_i(x) = F(x, \alpha_i)$, $g_j(y) = F(\alpha_j, y)$
2. Each party p_i receives $f_i(x)$ and $g_i(y)$
3. Each pair (p_i, p_j) confirms that $s_{ij} = f_i(\alpha_j) = g_j(\alpha_i)$ and $s_{ji}=f_j(\alpha_i) = g_i(\alpha_j)$.
4. Resolve conflict by public accusations answered by the dealer.

p_2 's "share"

Requires Broadcast
- Recall: Can be constructed from secure channels iff t<n/3 [LSP82 BGP89]

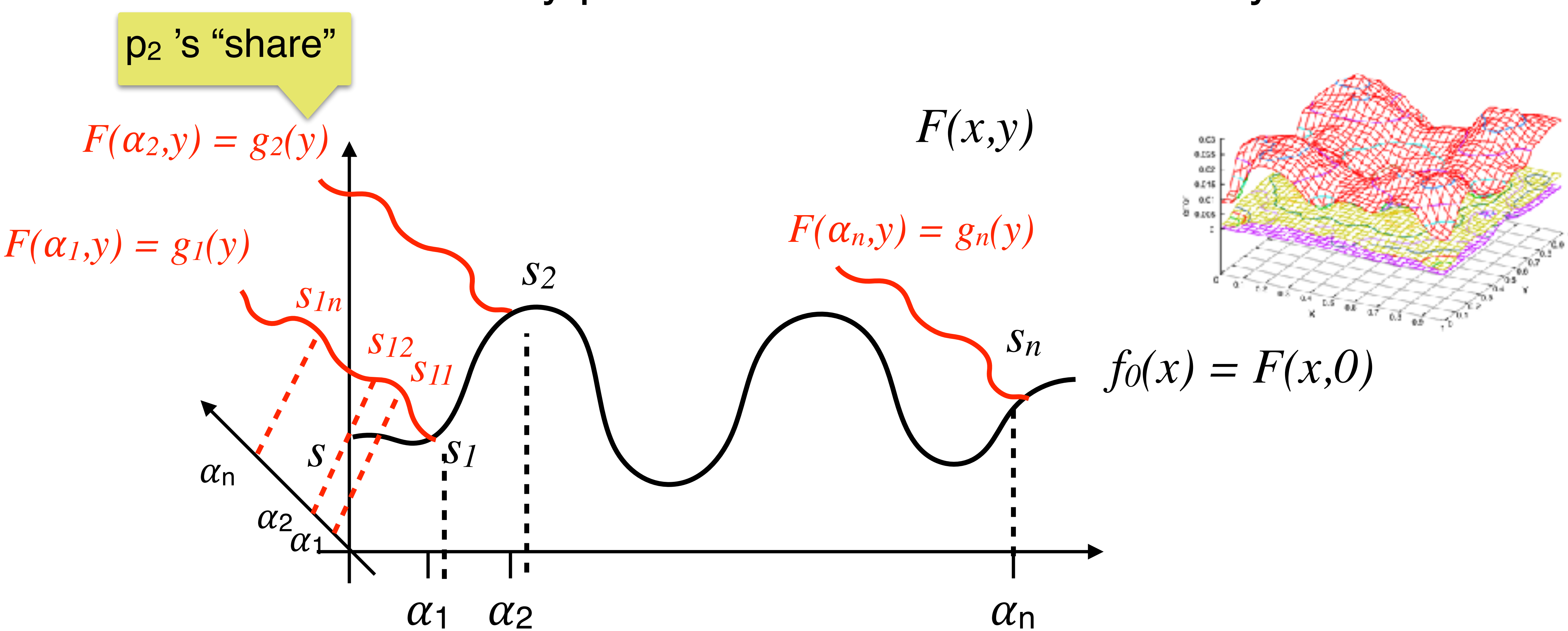




(t+1)-out-of-n VSS (t<n/3)

Share:

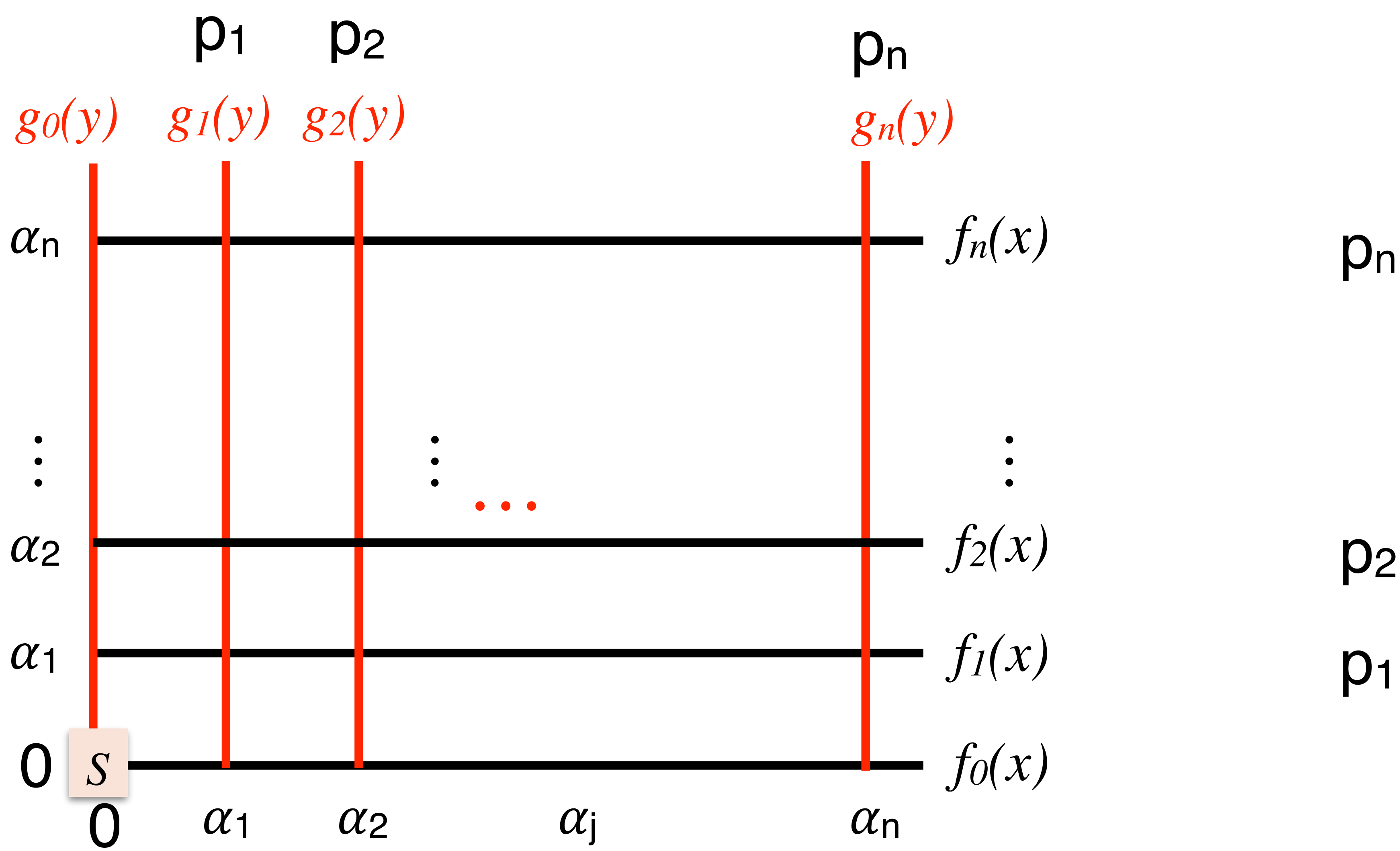
1. D chooses a random bivariate polynomial $F(x,y)$ of degree t in each variable, such that $f(0,0)=s$. Denote: $f_i(x) = F(x, \alpha_i)$, $g_j(y) = F(\alpha_j, y)$
2. Each party p_i receives $f_i(x)$ and $g_i(y)$
3. Each pair (p_i, p_j) confirms that $s_{ij} = f_i(\alpha_j) = g_j(\alpha_i)$ and $s_{ji}=f_j(\alpha_i) = g_i(\alpha_j)$.
4. Resolve conflict by public accusations answered by the dealer.



(t+1)-out-of-n VSS (t<n/3)

Share:

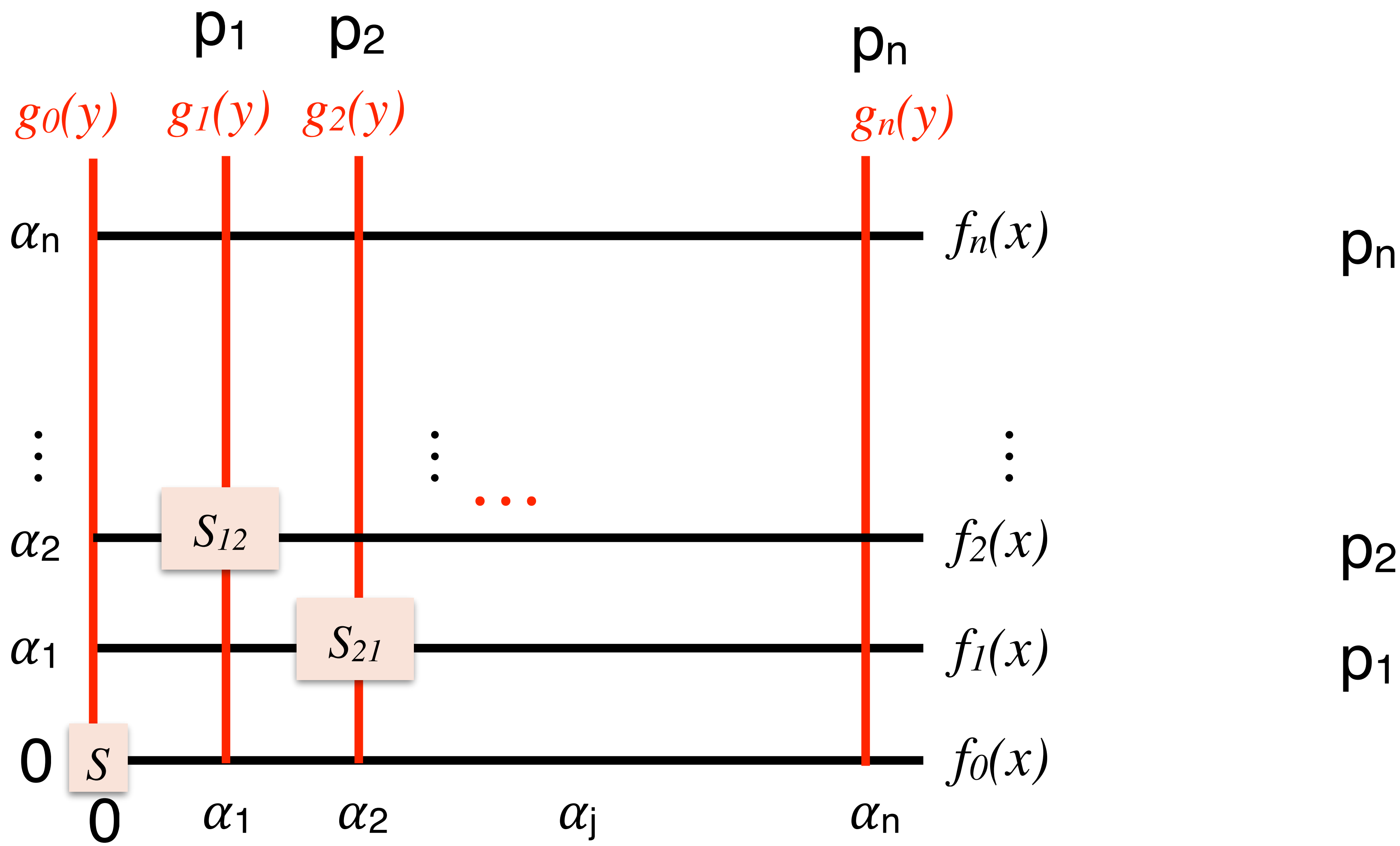
1. D chooses a random bivariate polynomial $F(x,y)$ of degree t in each variable, such that $f(0,0)=s$. Denote: $f_i(x) = F(x, \alpha_i)$, $g_j(y) = F(\alpha_j, y)$
2. Each party p_i receives $f_i(x)$ and $g_i(y)$
3. Each pair (p_i, p_j) confirms that $s_{ij} = f_i(\alpha_j) = g_j(\alpha_i)$ and $s_{ji}=f_j(\alpha_i) = g_i(\alpha_j)$.
4. Resolve conflict by public accusations answered by the dealer.



(t+1)-out-of-n VSS (t<n/3)

Share:

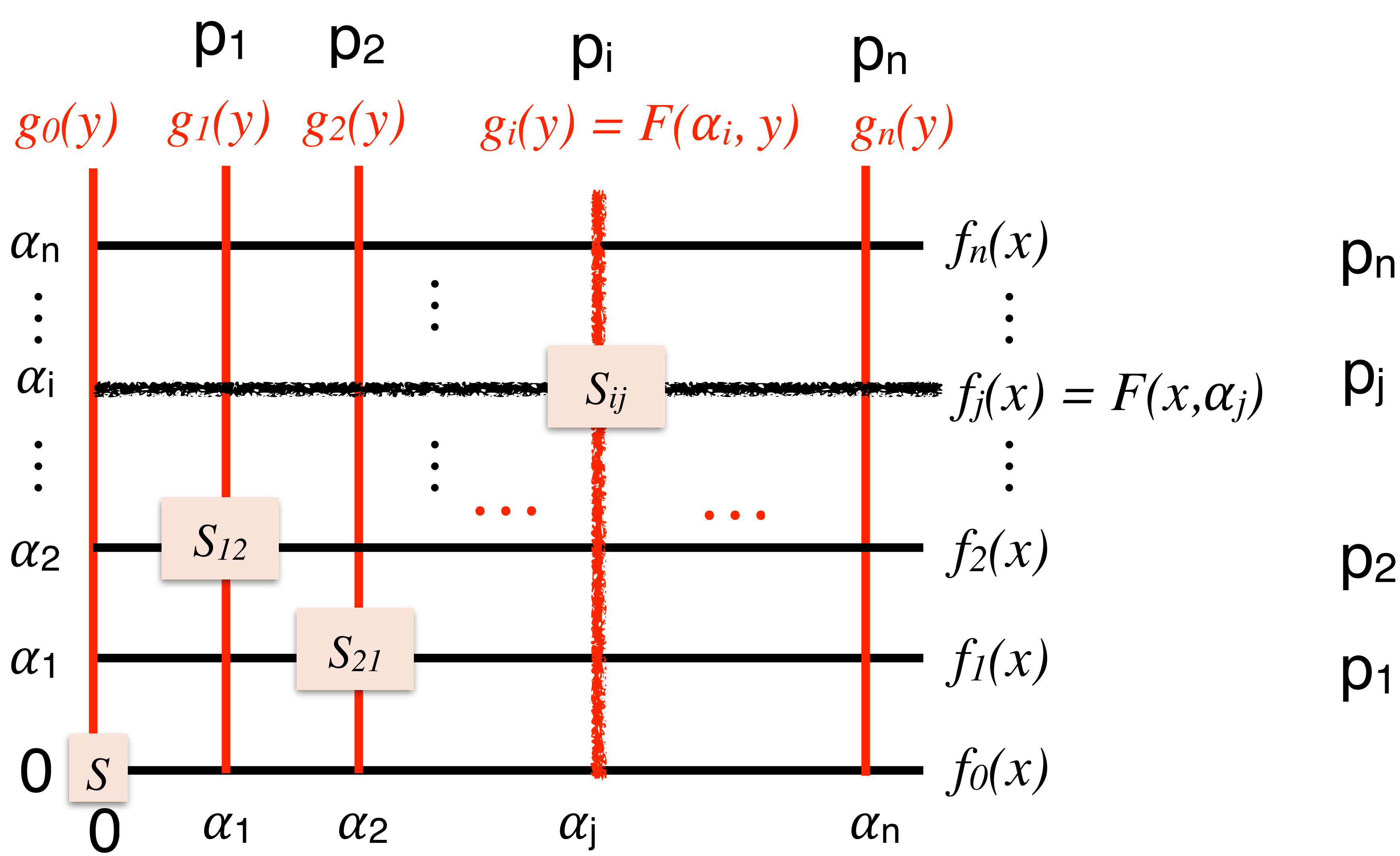
1. D chooses a random bivariate polynomial $F(x,y)$ of degree t in each variable, such that $f(0,0)=s$. Denote: $f_i(x) = F(x, \alpha_i)$, $g_j(y) = F(\alpha_j, y)$
2. Each party p_i receives $f_i(x)$ and $g_i(y)$
3. Each pair (p_i, p_j) confirms that $s_{ij} = f_i(\alpha_j) = g_j(\alpha_i)$ and $s_{ji}=f_j(\alpha_i) = g_i(\alpha_j)$.
4. Resolve conflict by public accusations answered by the dealer.



(t+1)-out-of-n VSS (t<n/3)

Share:

1. D chooses a random bivariate polynomial $F(x,y)$ of degree t in each variable, such that $f(0,0)=s$. Denote: $f_i(x) = F(x, \alpha_i)$, $g_j(y) = F(\alpha_j, y)$
2. Each party p_i receives $f_i(x)$ and $g_i(y)$
3. Each pair (p_i, p_j) confirms that $s_{ij} = f_i(\alpha_j) = g_j(\alpha_i)$ and $s_{ji}=f_j(\alpha_i) = g_i(\alpha_j)$.
4. Resolve conflict by public accusations answered by the dealer.



t-out-of-n VSS (t<n/3)

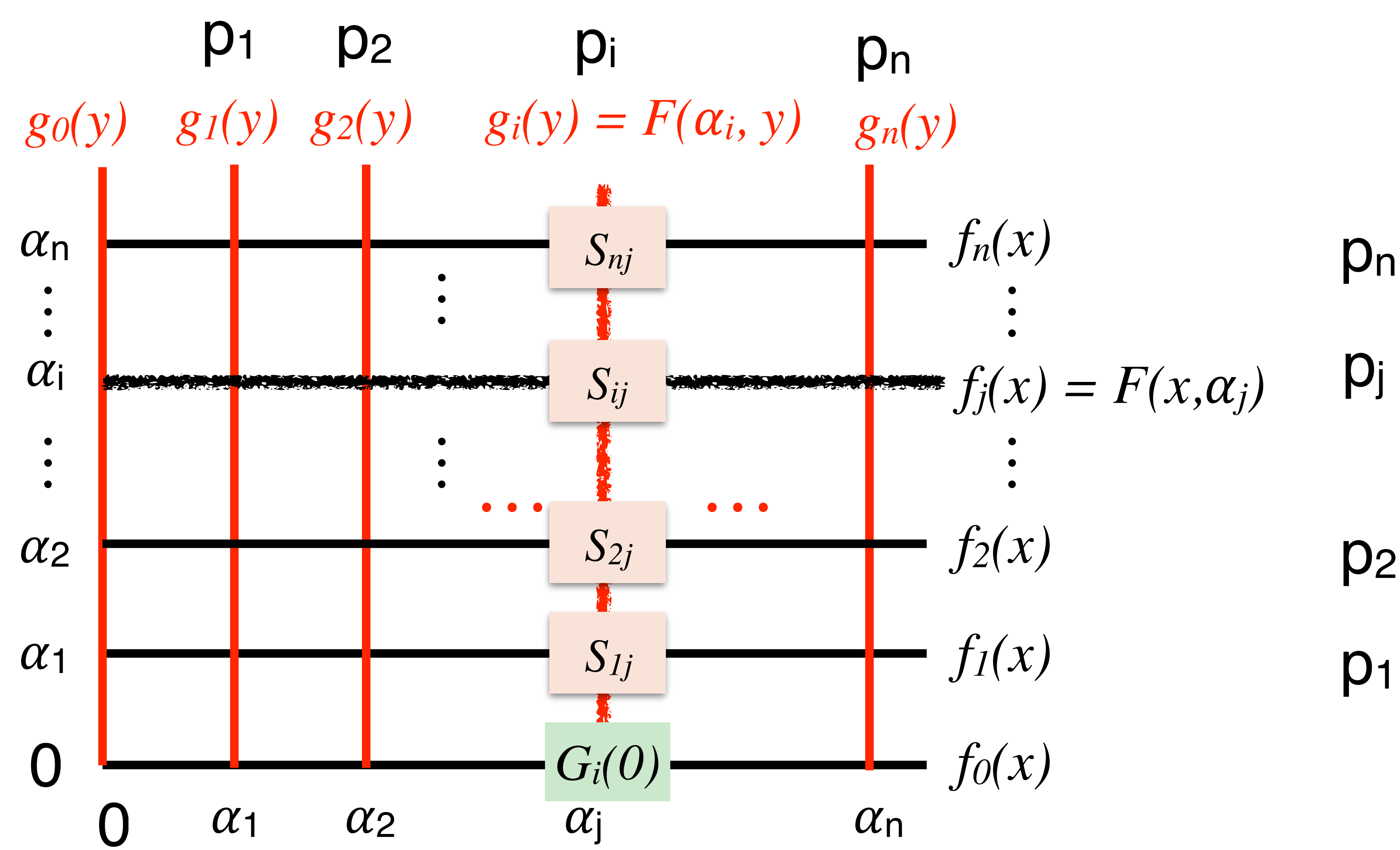
Reconstruct:

1. For each $g_j(y)$:
   1. p_j announces s_{ij}
   2. Find the degree-t polynomial $G_j(y)$ which passes through at least $2t+1$ points from the announces $s_{1j}, \ldots, s_{nj}$
   3. Use $G_1(0), \ldots, G_n(0)$ to interpolate $f_0(x)$ and compute $s=f_0(0)$

t-out-of-n VSS (t<n/3)

Reconstruct:

1. For each $g_j(y)$:
   1. p_j announces s_{ij}
   2. Find the degree-t polynomial $G_j(y)$ which passes through at least $2t+1$ points from the announces $s_{1j}, \ldots, s_{nj}$
   3. Use $G_1(0), \ldots, G_n(0)$ to interpolate $f_0(x)$ and compute $s=f_0(0)$



t-out-of-n VSS (t<n/3)

Reconstruct:

1. For each $g_j(y)$:
   1. p_j announces s_{ij}
   2. Find the degree-t polynomial $G_j(y)$ which passes through at least $2t+1$ points from the announces $s_{1j}, \ldots, s_{nj}$
   3. Use $G_1(0), \ldots, G_n(0)$ to interpolate $f_0(x)$ and compute $s=f_0(0)$

Claim: $G_j(y) = g_j(y)$

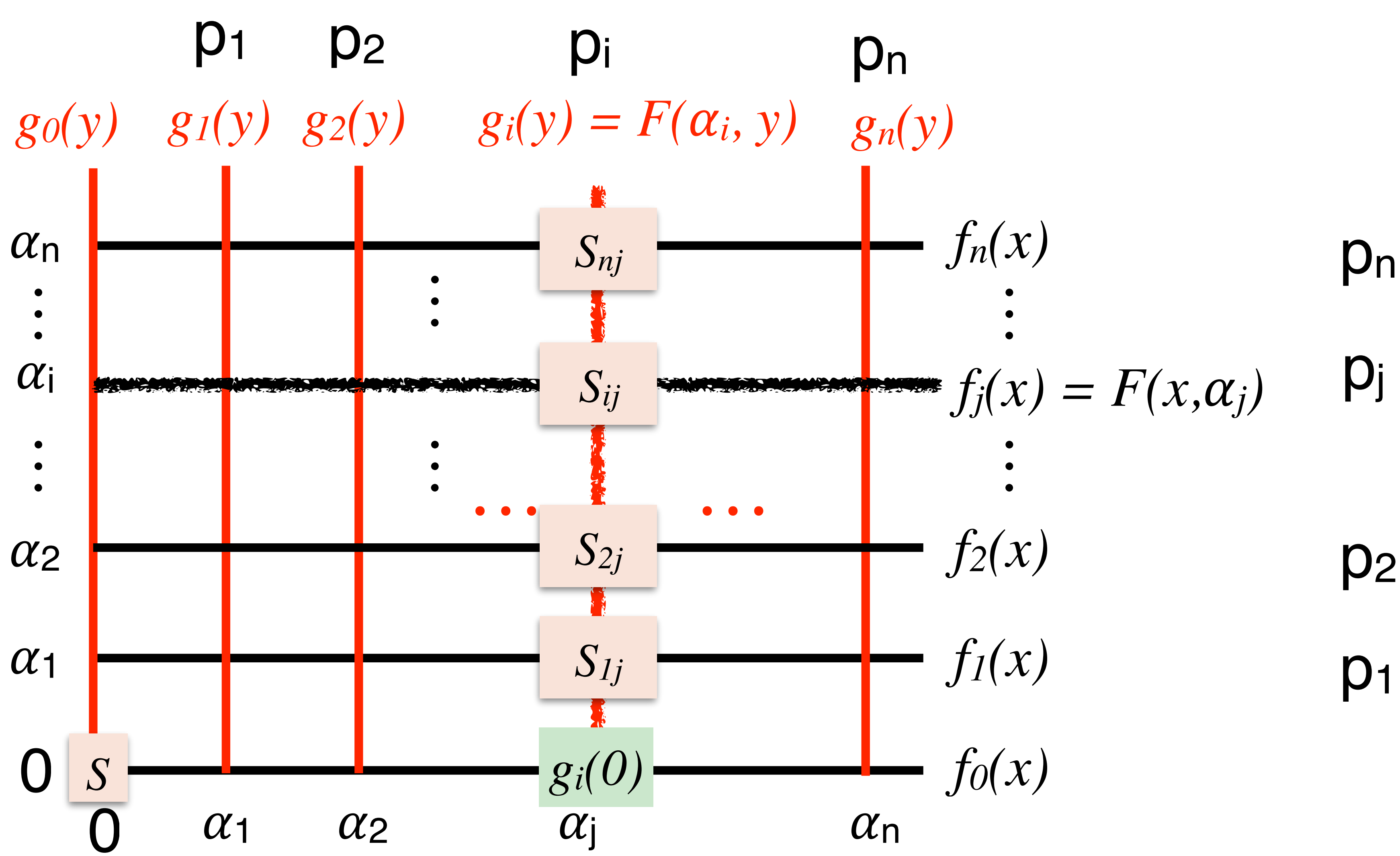
Proof:

- $G_j(y)$ passes through the $t+1$ values from the honest parties which all lie on g_j.
- By the Lagrange interpolation, there exists no other degree-t polynomial with this property, hence this is the only polynomial that might be reconstructed.

t-out-of-n VSS (t<n/3)

Reconstruct:

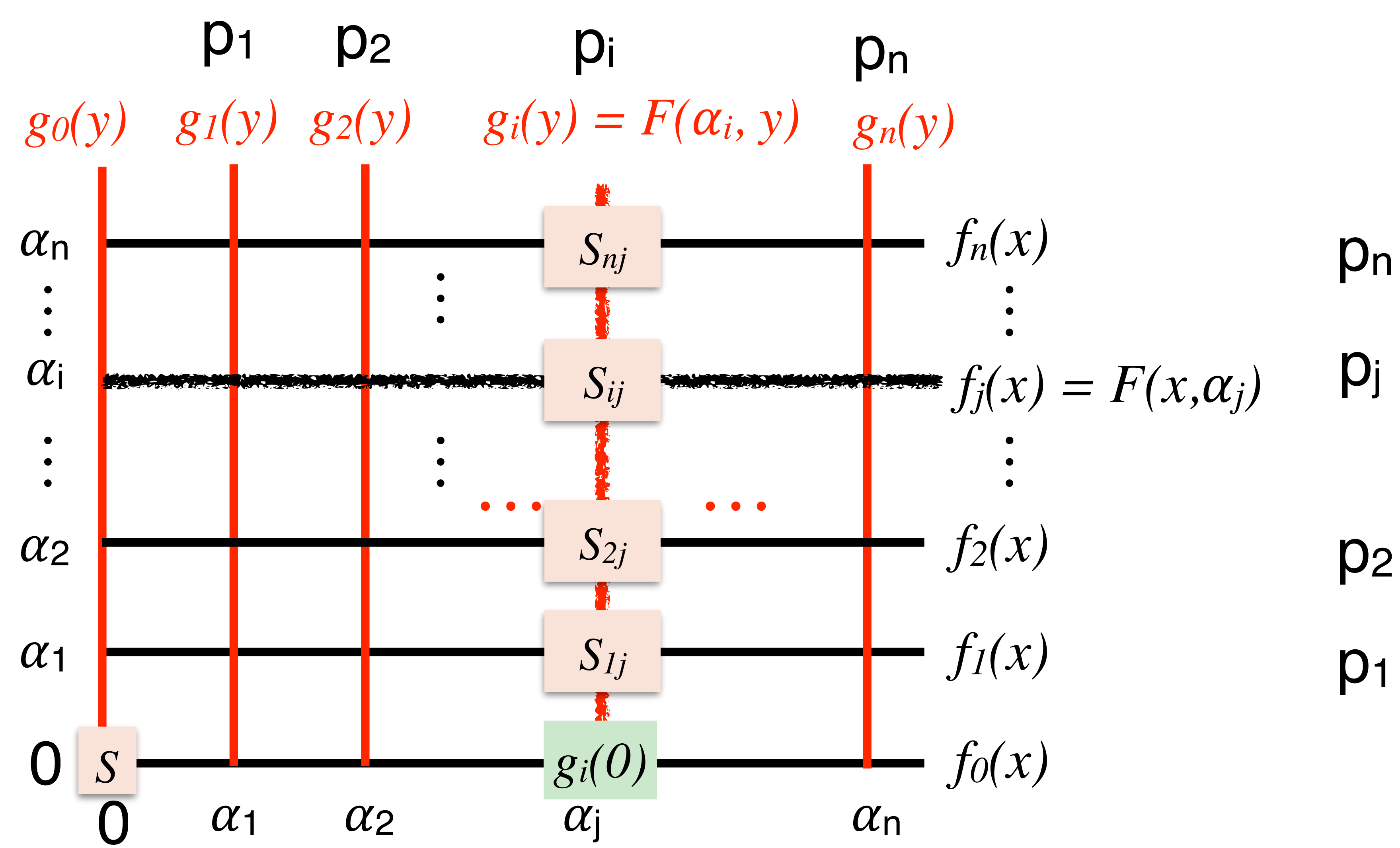
1. For each $g_j(y)$:
   1. p_j announces s_{ij}
   2. Find the degree-t polynomial $G_j(y)$ which passes through at least $2t+1$ points from the announces $s_{1j}, \ldots, s_{nj}$
   3. Use $G_1(0), \ldots, G_n(0)$ to interpolate $f_0(x)$ and compute $s=f_0(0)$



t-out-of-n VSS (t<n/3)

Reconstruct:

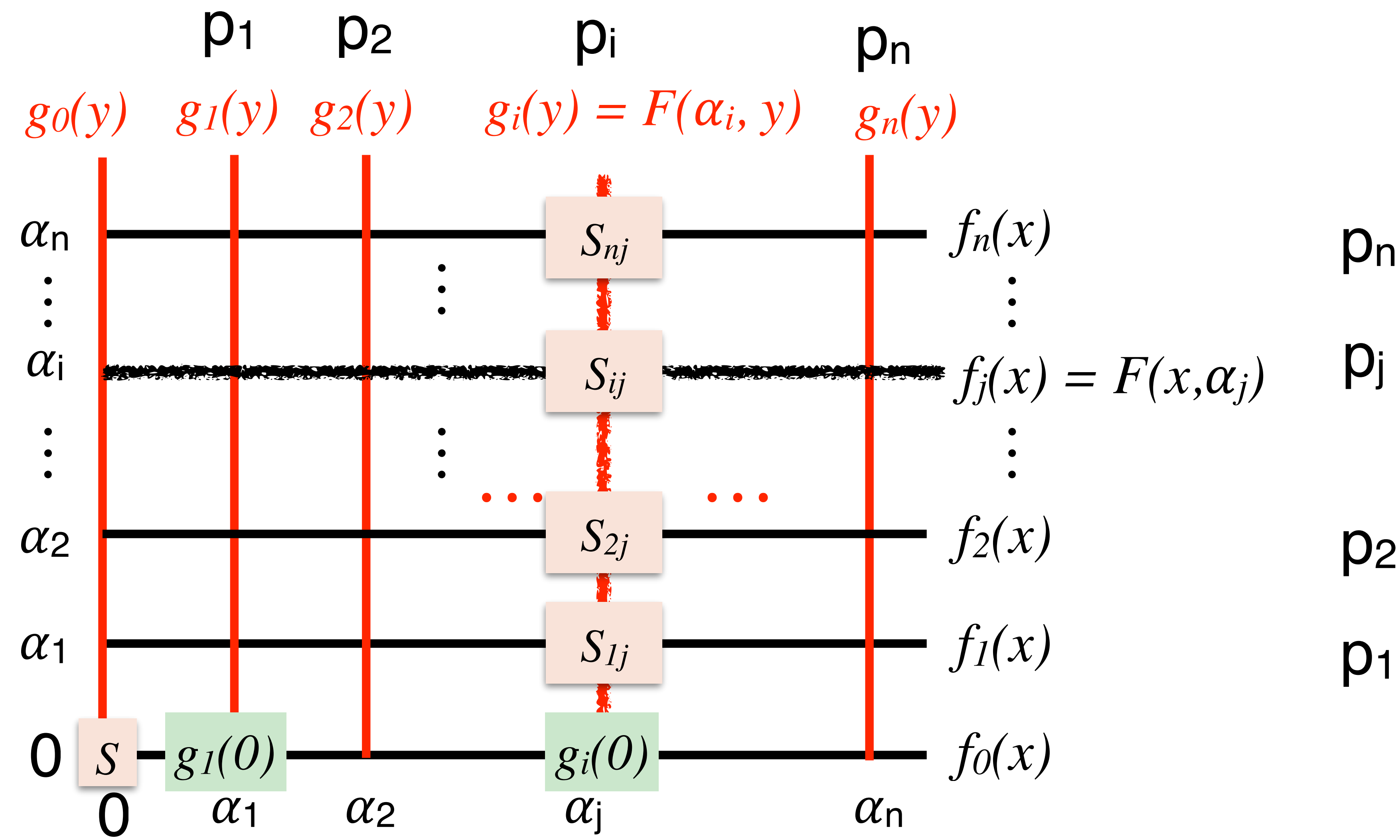
1. For each $g_j(y)$:
   1. p_j announces s_{ij}
   2. Find the degree-t polynomial $G_j(y)$ which passes through at least $2t+1$ points from the announces $s_{1j}, \ldots, s_{nj}$
   3. Use $G_1(0), \ldots, G_n(0)$ to interpolate $f_0(x)$ and compute $s=f_0(0)$



t-out-of-n VSS (t<n/3)

Reconstruct:

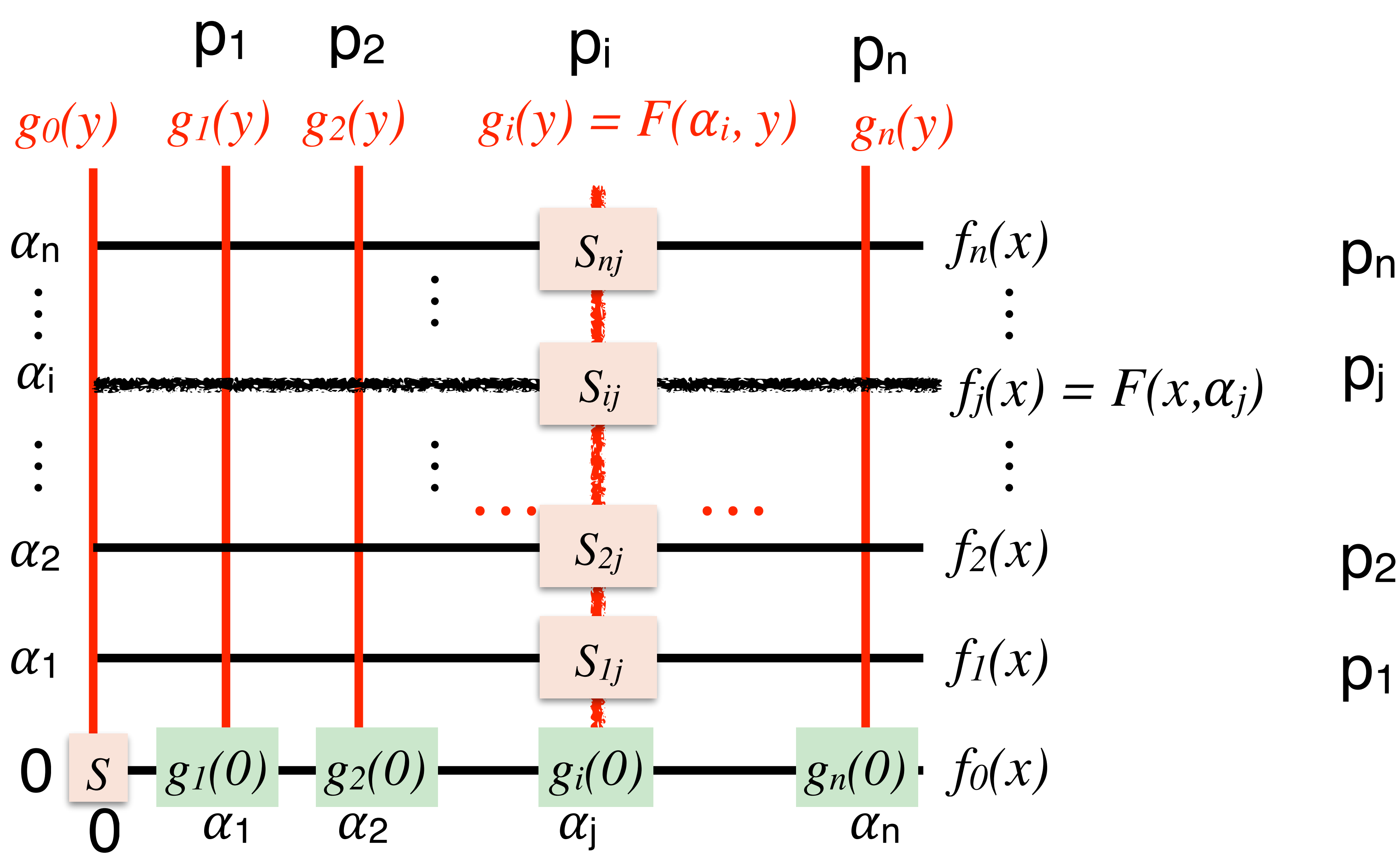
1. For each $g_j(y)$:
   1. p_j announces s_{ij}
   2. Find the degree-t polynomial $G_j(y)$ which passes through at least $2t+1$ points from the announces $s_{1j}, \ldots, s_{nj}$
   3. Use $G_1(0), \ldots, G_n(0)$ to interpolate $f_0(x)$ and compute $s=f_0(0)$



t-out-of-n VSS (t<n/3)

Reconstruct:

1. For each $g_j(y)$:
   1. p_j announces s_{ij}
   2. Find the degree-t polynomial $G_j(y)$ which passes through at least $2t+1$ points from the announces $s_{1j}, \ldots, s_{nj}$
   3. Use $G_1(0), \ldots, G_n(0)$ to interpolate $f_0(x)$ and compute $s=f_0(0)$

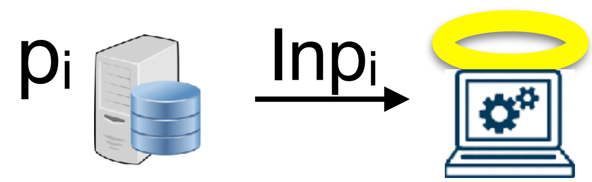
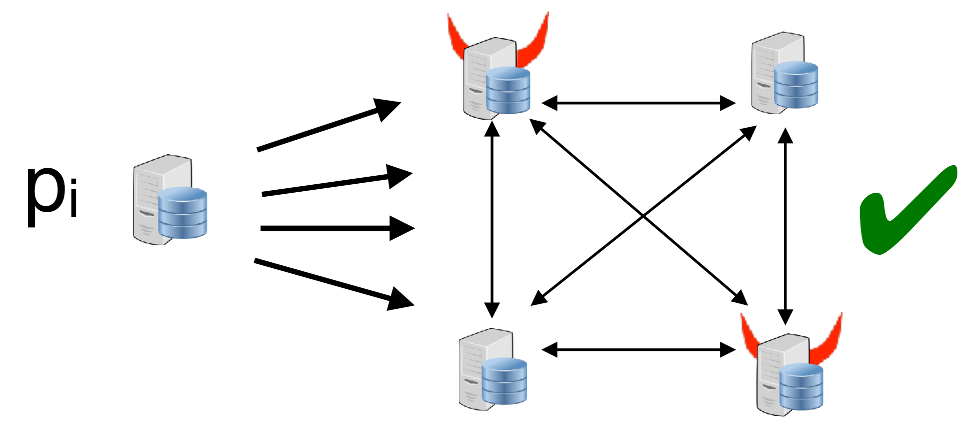
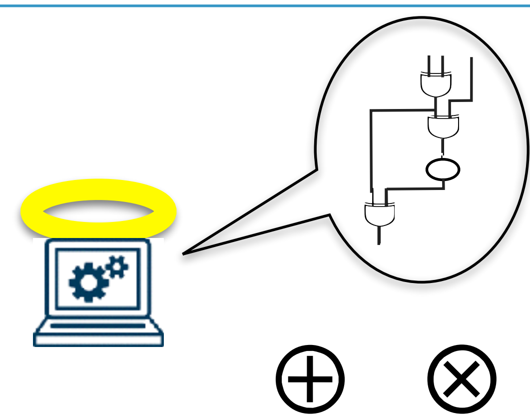
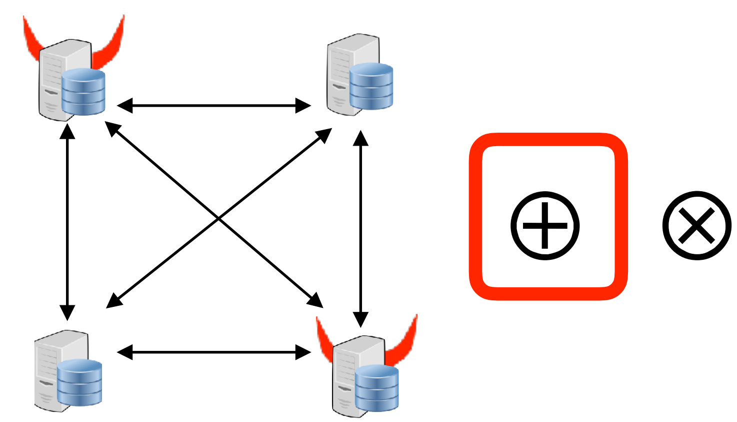
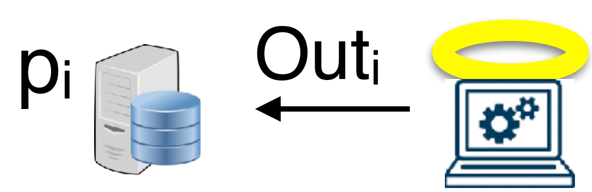
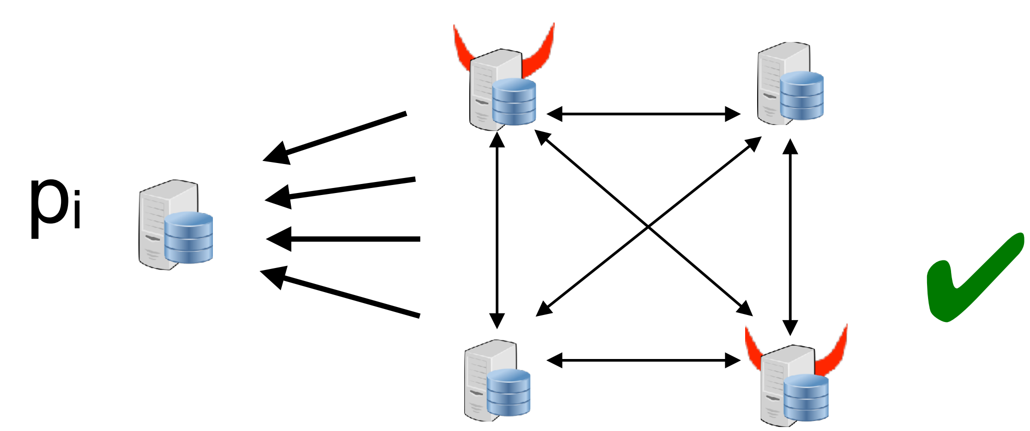


t-out-of-n VSS (t<n/3)

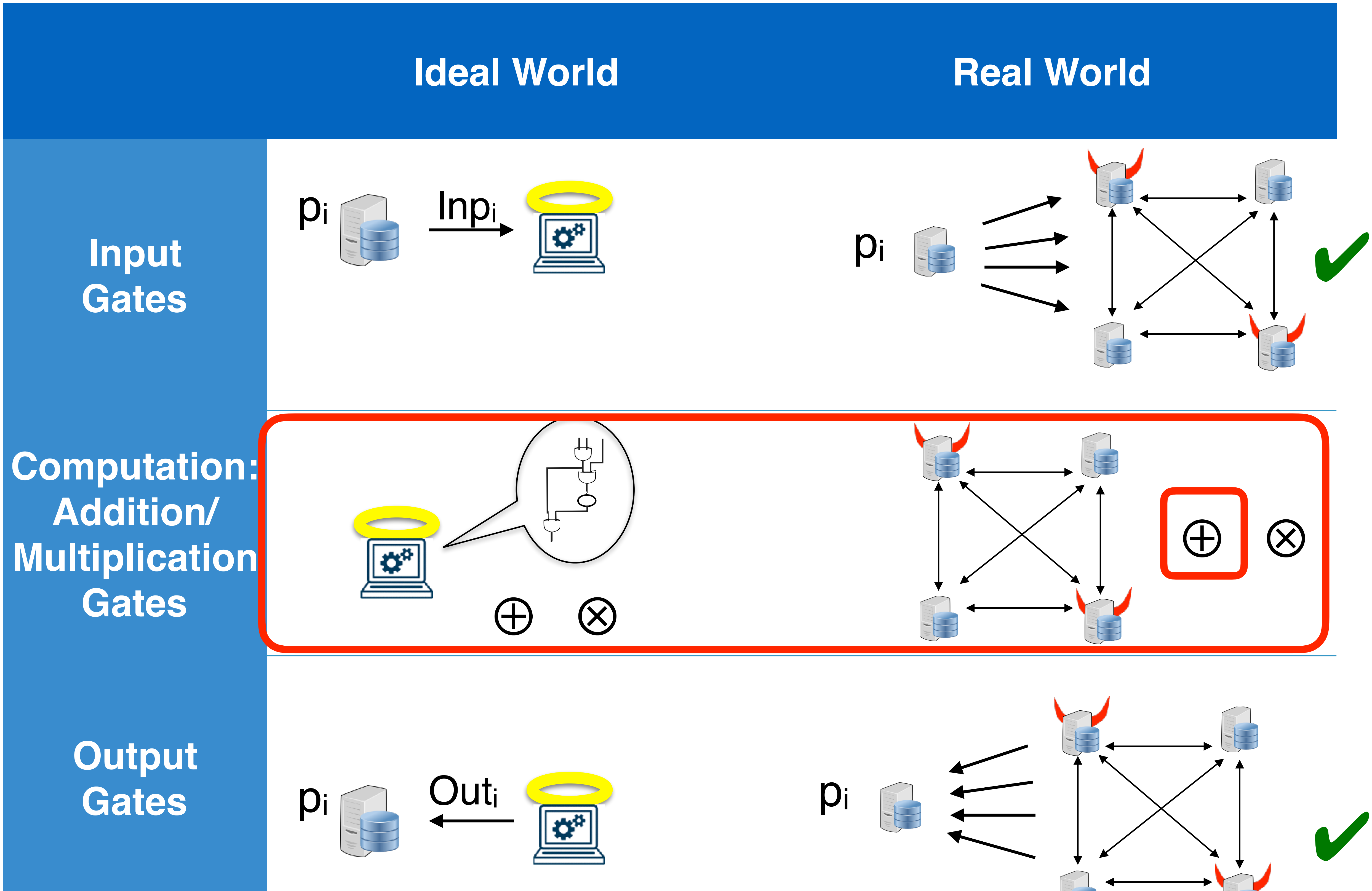
Properties:

- At the end of the sharing phase
  - t parties have no information ⇒ VSS privacy
  - The dealer is committed to the shared secret ⇒ VSS commitment
  - If the dealer is honest then the sharing is of s ⇒ VSS correctness
  - *Every party (even malicious) is committed to his share (i.e., polynomial $g_i(y)$): the honest parties can reconstruct it*

MPC Goal

	Ideal World	Real World
Input Gates		
Computation: Addition/ Multiplication Gates		
Output Gates		

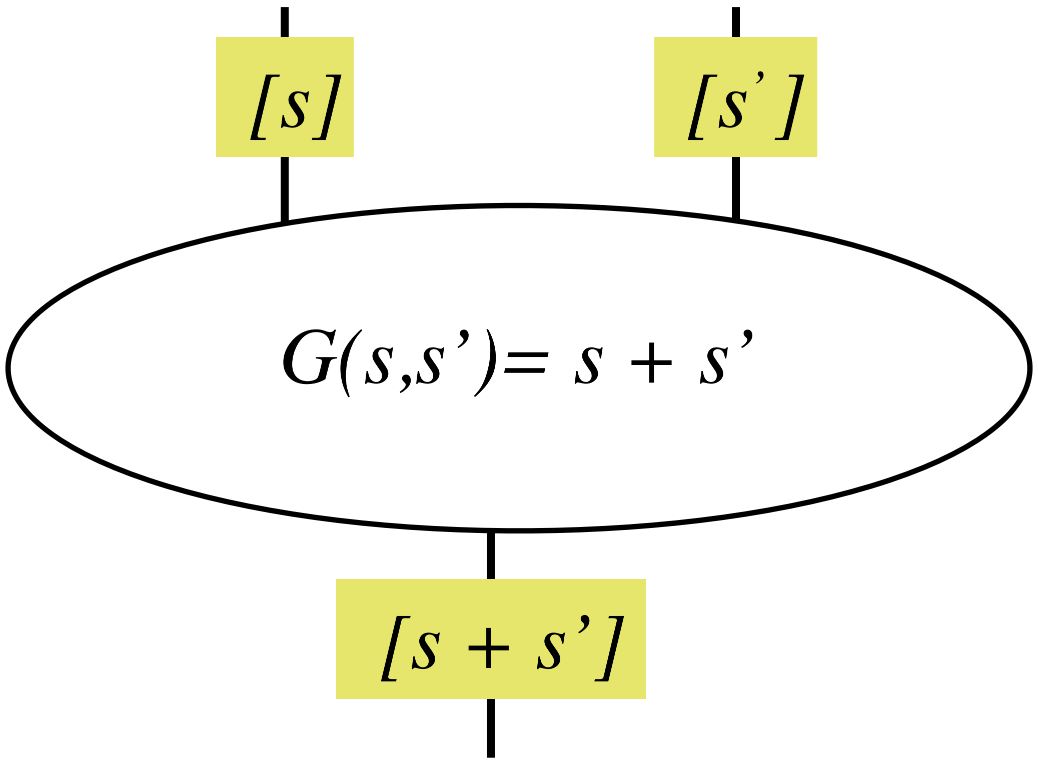
MPC Goal



Ideal World
Real World
Input Gates
pi
Inpi
pi
Computation: Addition/ Multiplication Gates
⊕ ⊗
⊕ ⊗
Output Gates
pi
Outi
pi

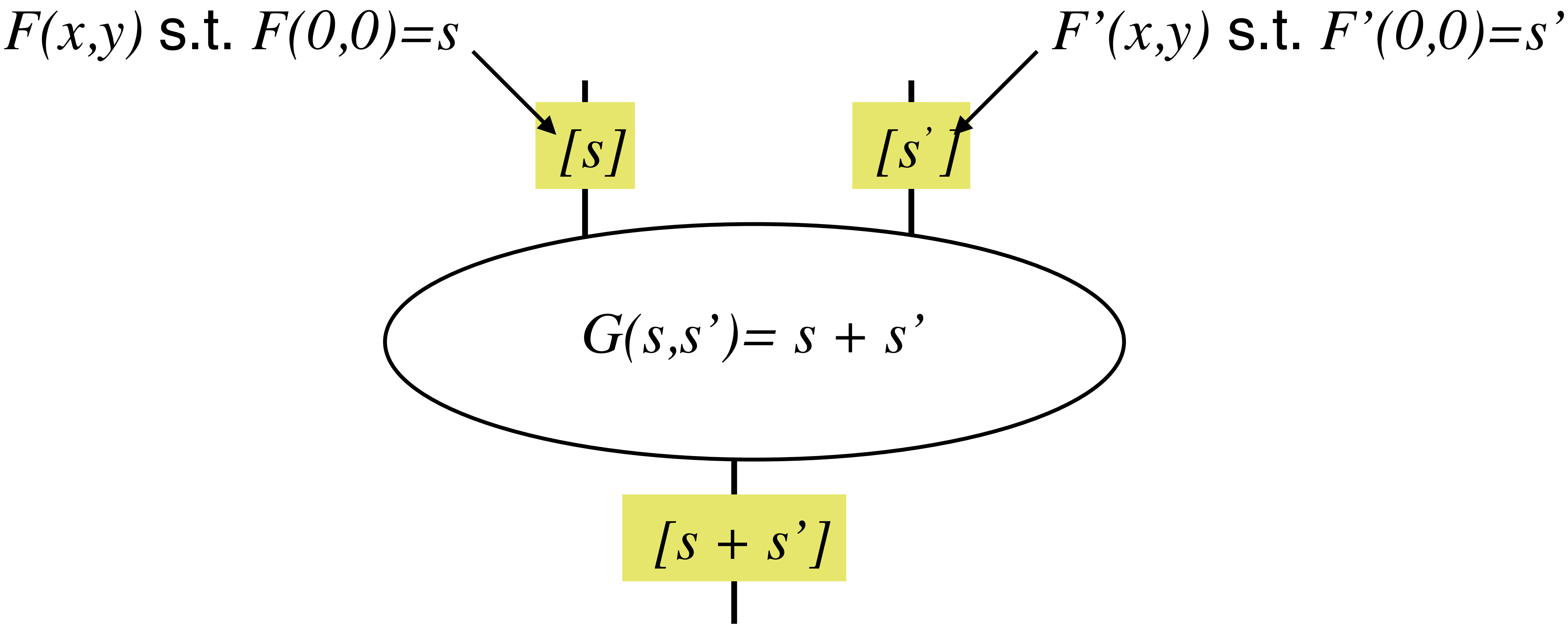
Malicious MPC: Addition

Goal: Addition Gadget



Malicious MPC: Addition

Goal: Addition Gadget

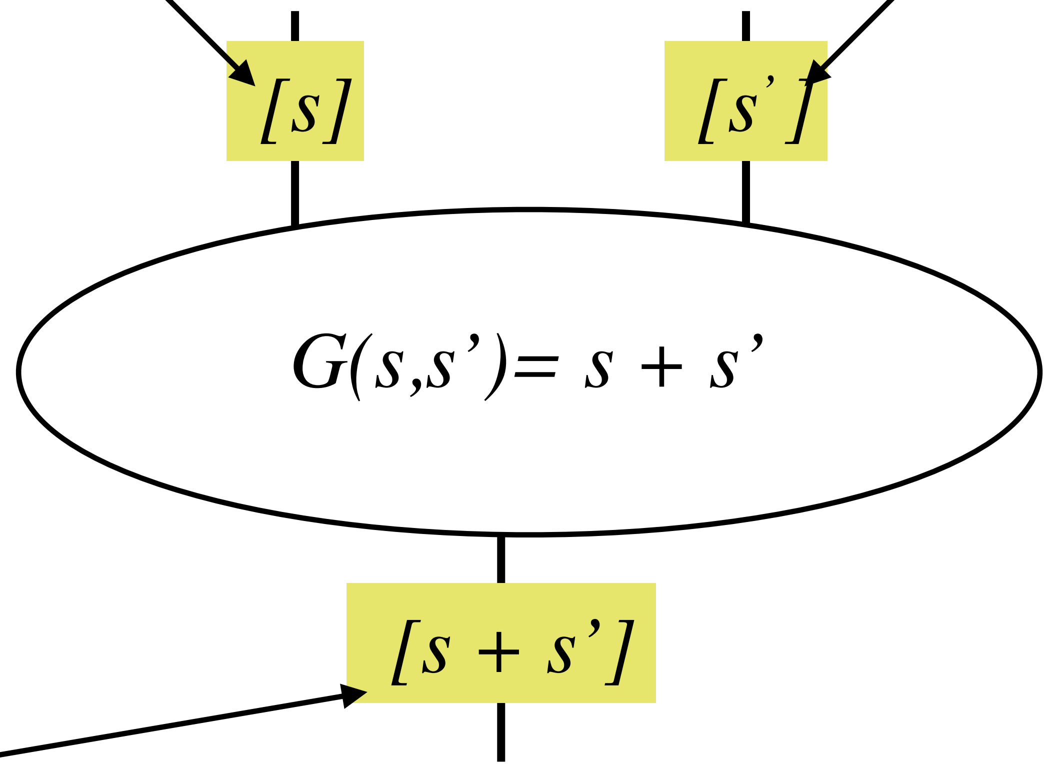


Malicious MPC: Addition

Goal: Addition Gadget

$F(x,y)$ s.t. $F(0,0)=s$

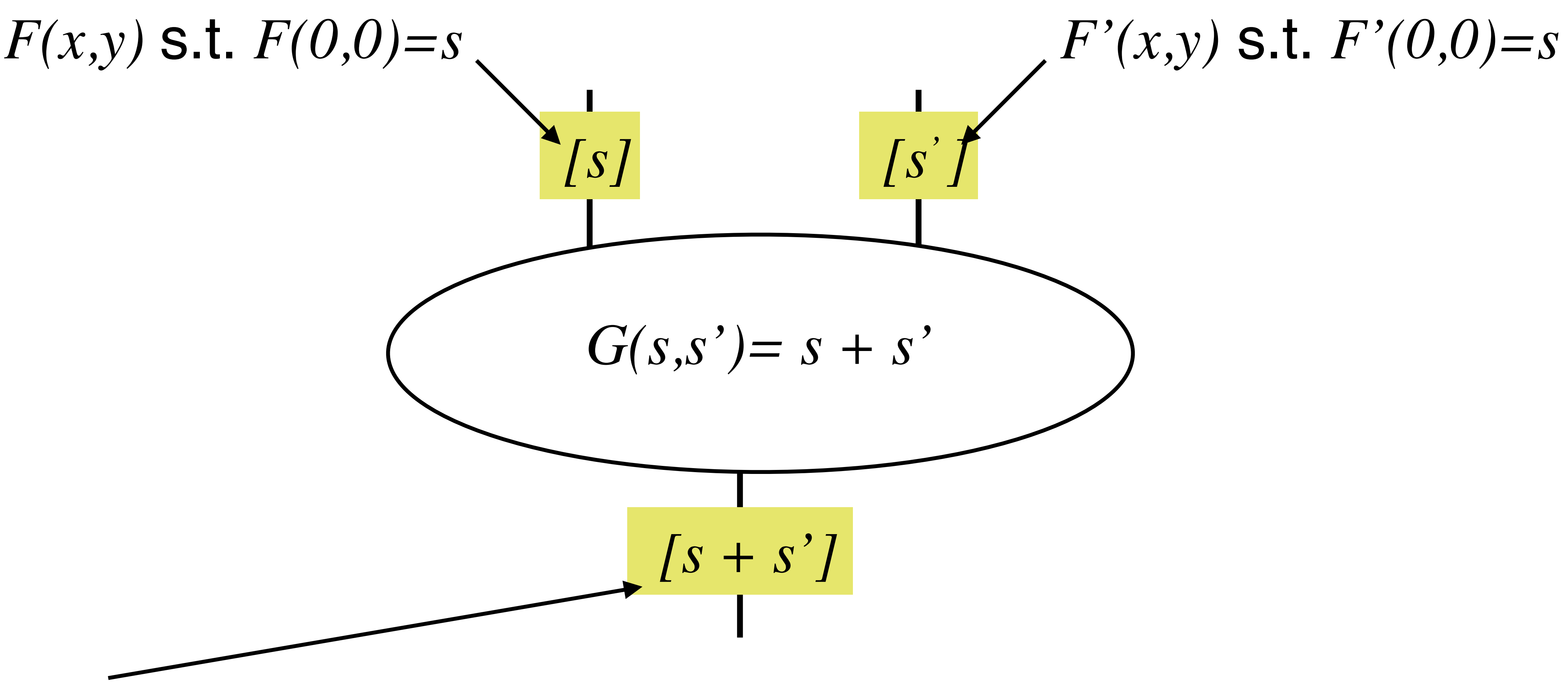
$F'(x,y)$ s.t. $F'(0,0)=s'$



Define $F''(x,y)=F(x,y) + F'(x,y) \Rightarrow F''(0,0) = F(0,0) + F'(0,0) = s' + s'$

Malicious MPC: Addition

Goal: Addition Gadget

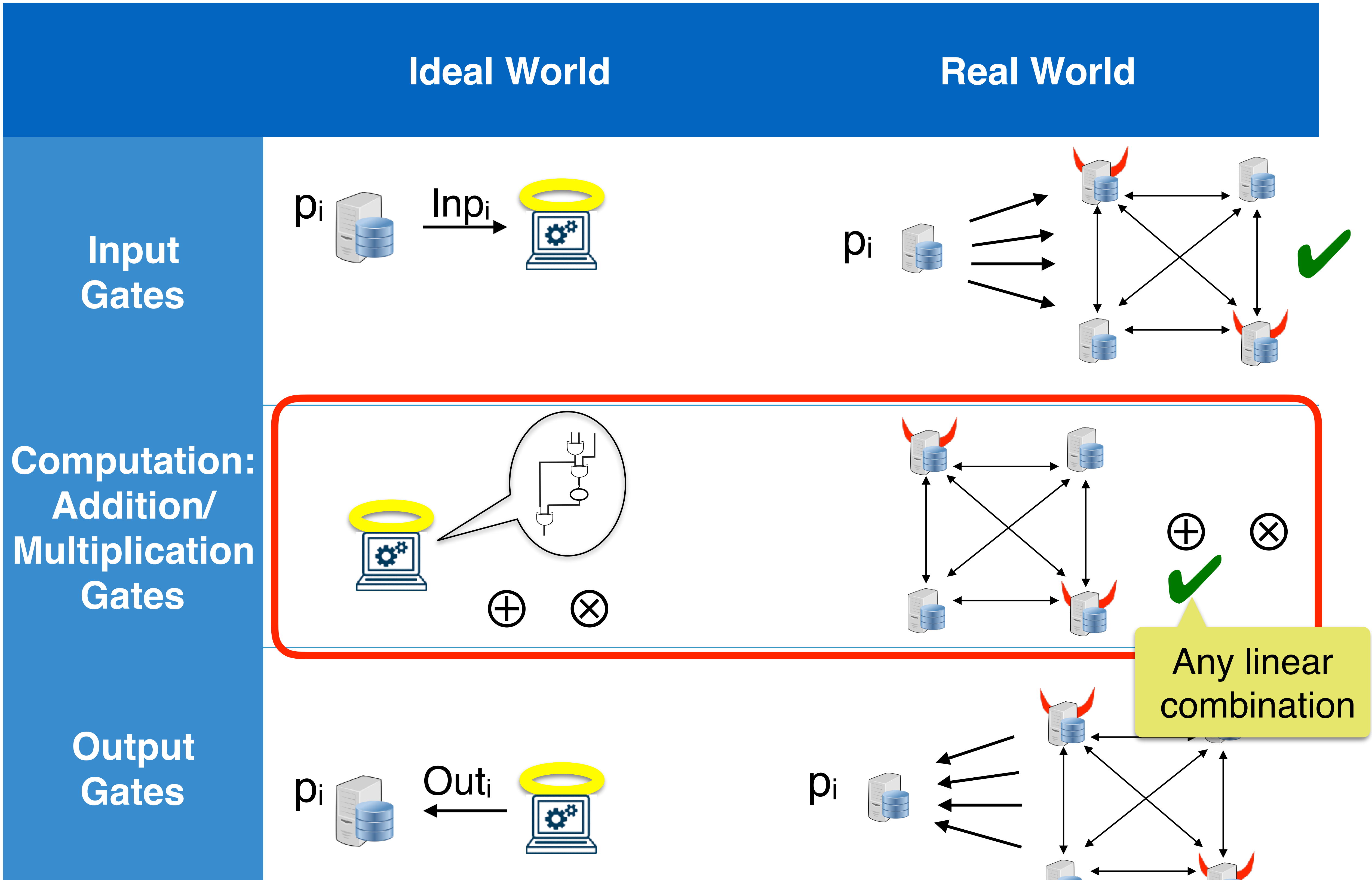


Define $F''(x,y)=F(x,y) + F'(x,y) \Rightarrow F''(0,0) = F(0,0) + F'(0,0) = s' + s'$

Addition protocol

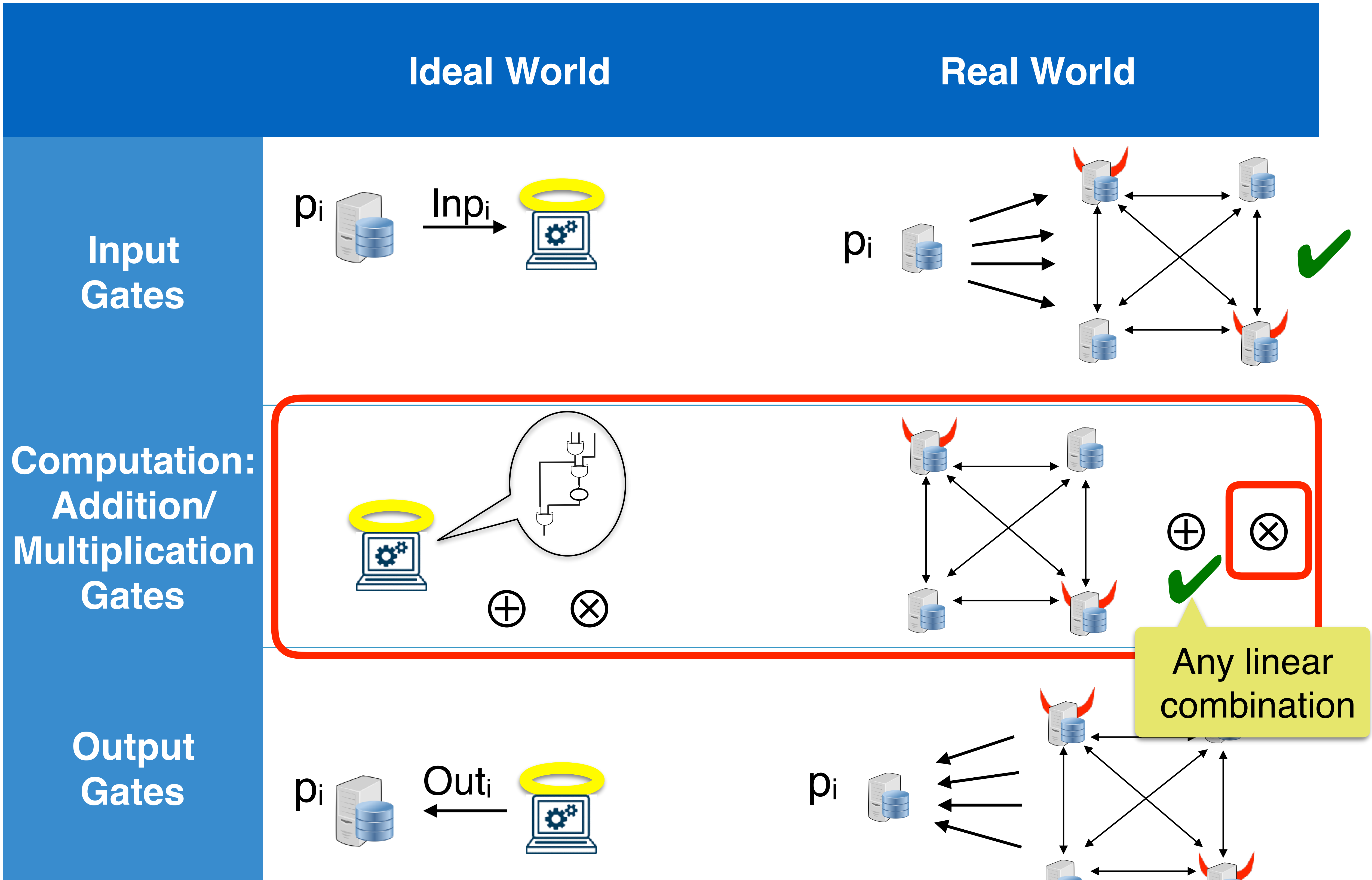
- Each party locally adds his share-shares of s and s', i.e., p_i computes $s_{ij}'' = s_{ij}+s_{ij}'$ and $s_{ji}'' = s_{ji}+s_{ji}'$
- The result is a sharing of s'' by means of polynomial $F'' = F + F'$

MPC Goal



Ideal World
Real World
Input Gates
Computation: Addition/ Multiplication Gates
Output Gates
pi
Inpi
Outi
⊕ ⊗
Any linear combination

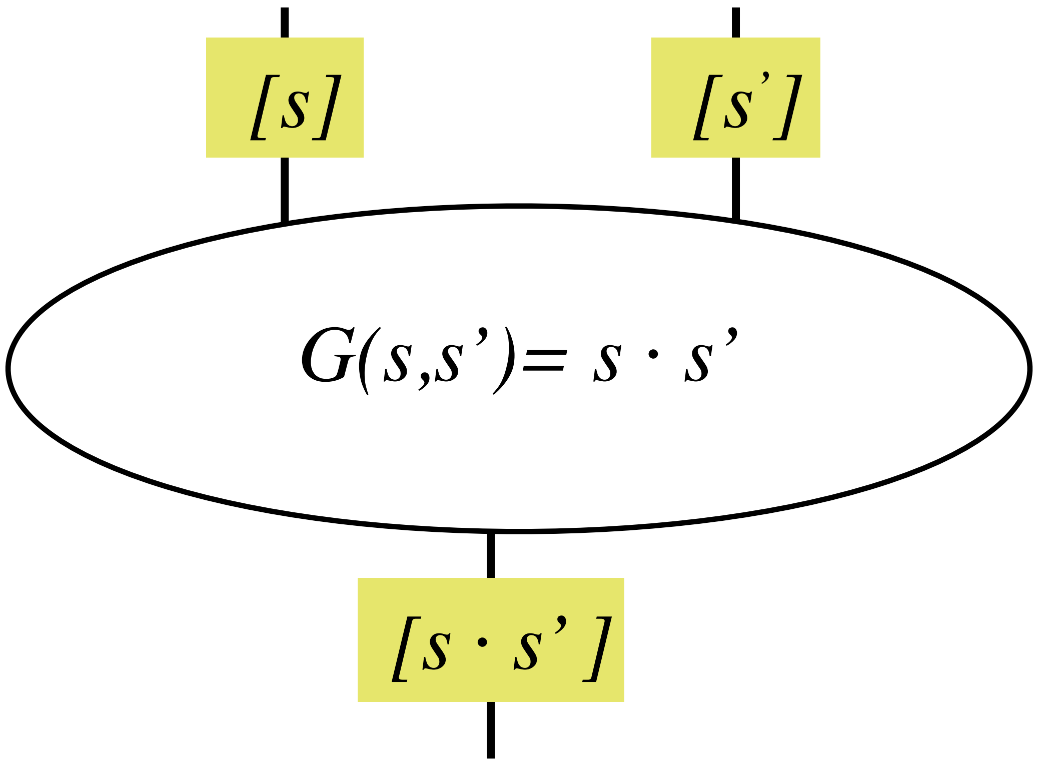
MPC Goal



Ideal World
Real World
Input Gates
Computation: Addition/ Multiplication Gates
Output Gates
p_i
Inp_i
Out_i
⊕
⊗
Any linear combination

Multiplication Protocol: Malicious

Goal: Multiplication Gadget



t-out-of-n VSS

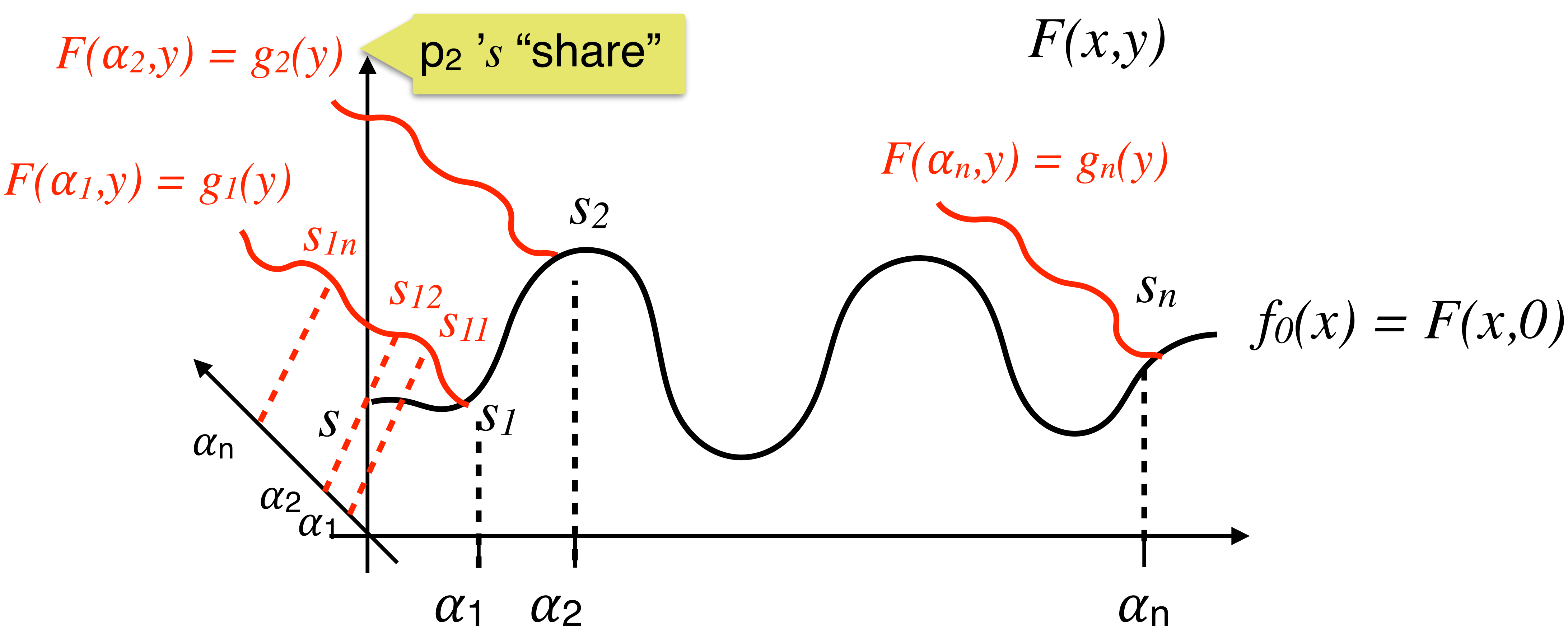
Properties (recall):

- At the end of the sharing phase
  - t-1 parties have no information ⇒ VSS privacy
  - The dealer is committed to the shared secret ⇒ VSS commitment
  - If the dealer is honest then the sharing is of s ⇒ VSS correctness
  - *Every party (even malicious) is committed to his share (i.e., polynomial $g_i(y)$): the honest parties can reconstruct it*

t-out-of-n VSS

Properties (recall):

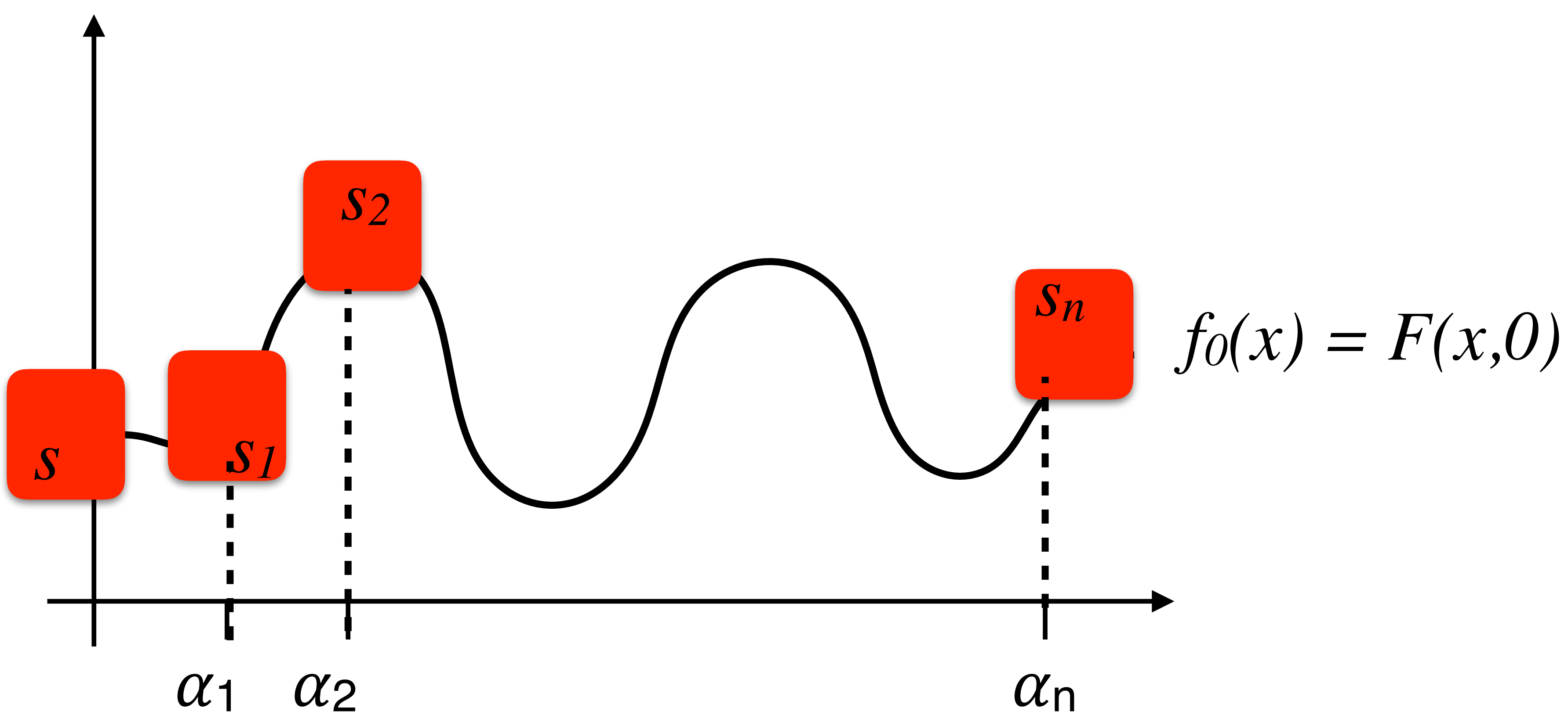
- At the end of the sharing phase
  - t-1 parties have no information ⇒ VSS privacy
  - The dealer is committed to the shared secret ⇒ VSS commitment
  - If the dealer is honest then the sharing is of s ⇒ VSS correctness
  - *Every party (even malicious) is committed to his share (i.e., polynomial $g_i(y)$): the honest parties can reconstruct it*



t-out-of-n VSS

Properties (recall):

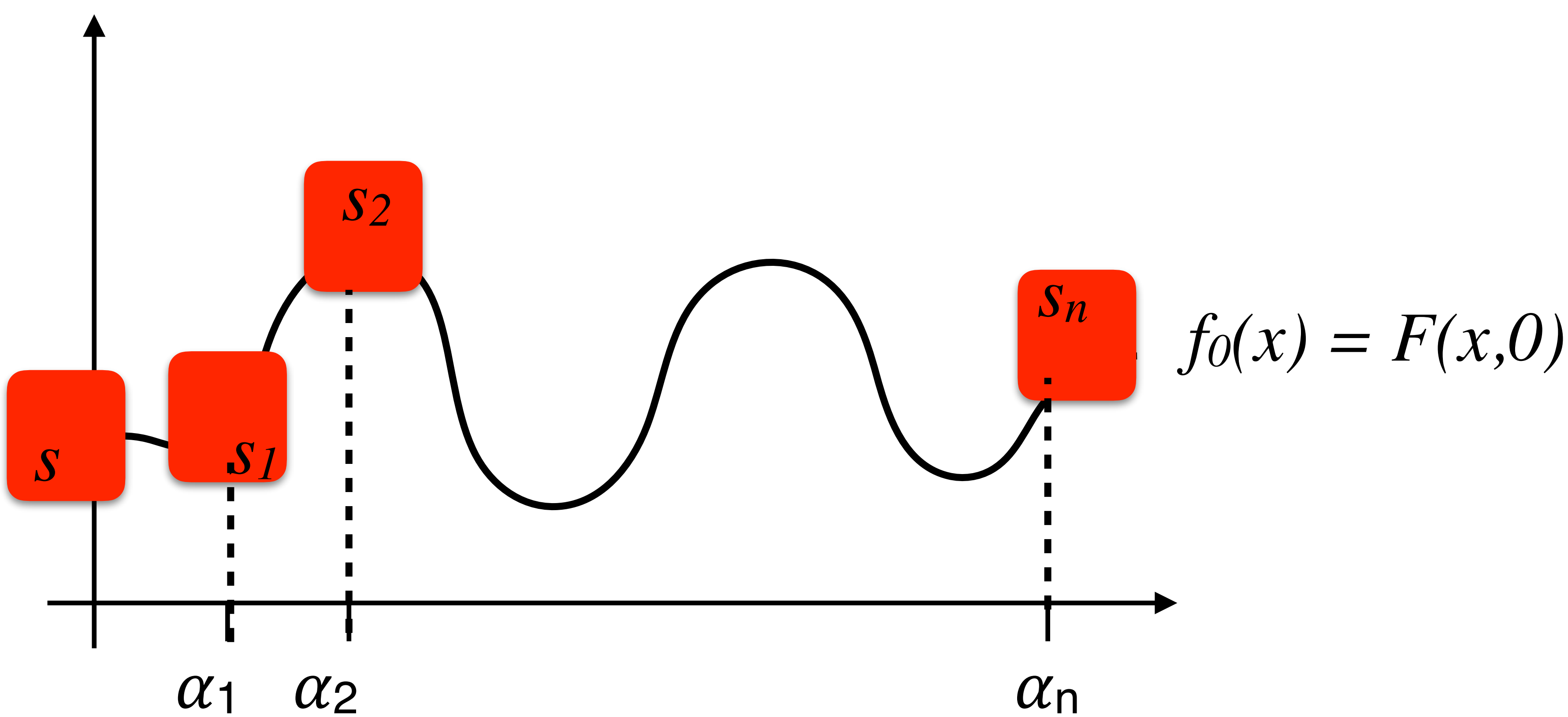
- At the end of the sharing phase
  - t-1 parties have no information ⇒ VSS privacy
  - The dealer is committed to the shared secret ⇒ VSS commitment
  - If the dealer is honest then the sharing is of s ⇒ VSS correctness
  - *Every party (even malicious) is committed to his share (i.e., polynomial $g_i(y)$): the honest parties can reconstruct it*



Multiplication Protocol: Malicious

$\boxed{s_i}$: commitment to s_i held by p_i

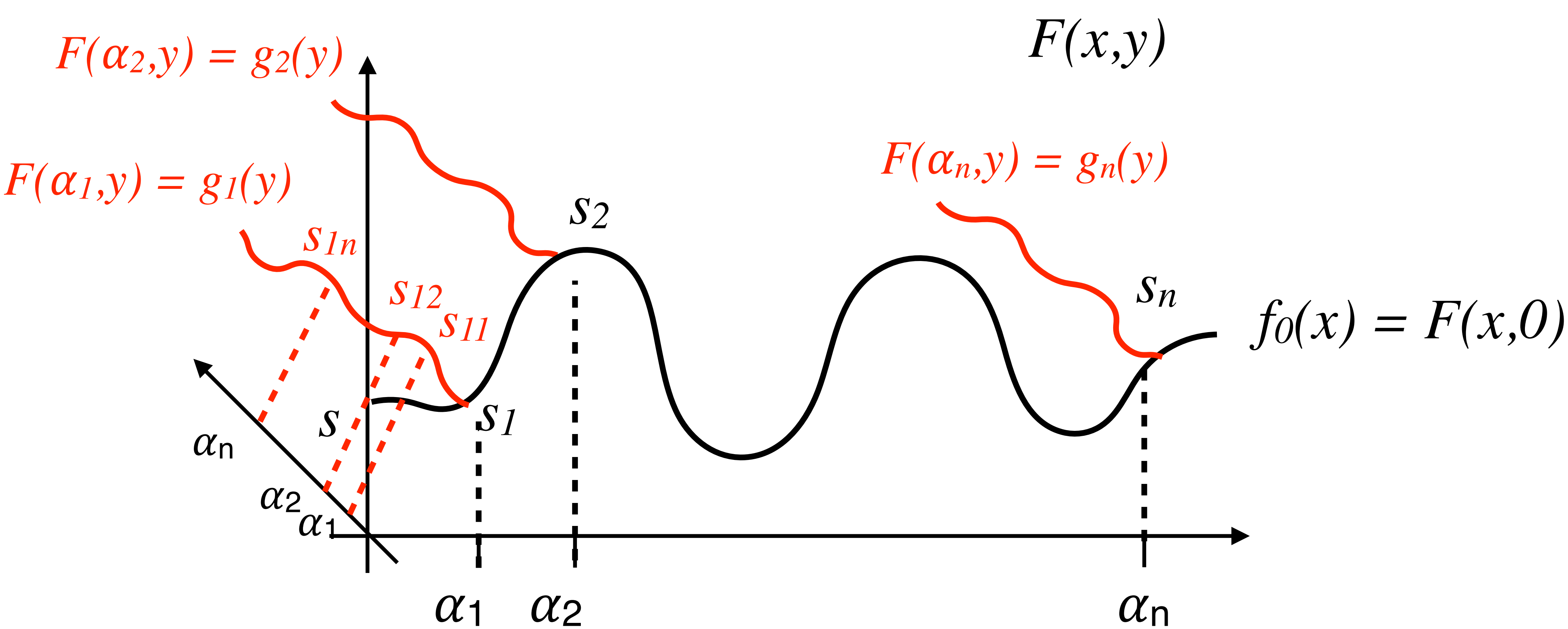
Linearity: $\boxed{s_i} + \boxed{s_i} = \boxed{s_i + s_i'}$



Multiplication Protocol: Malicious

$\boxed{s_i}$: commitment to s_i held by p_i

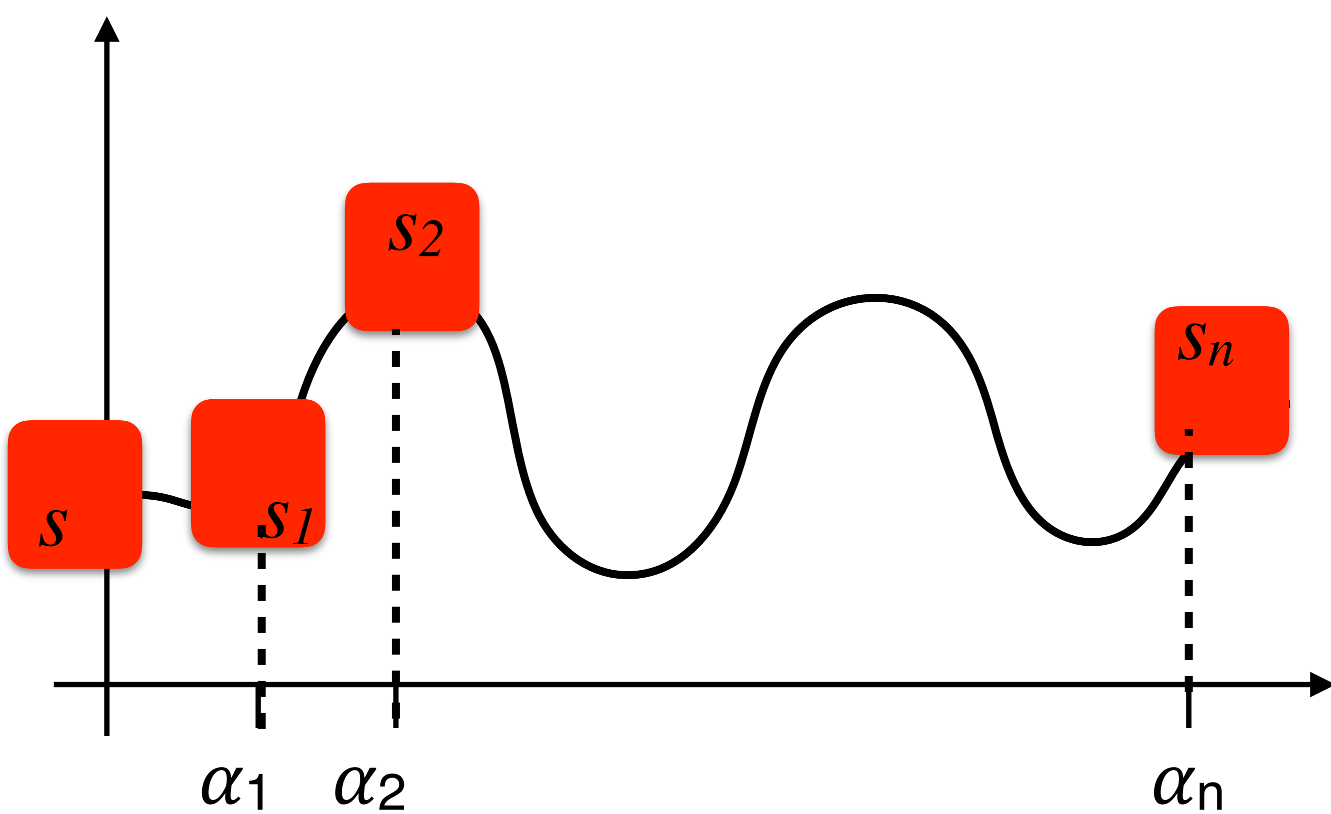
Linearity: $\boxed{s_i} + \boxed{s_i} = \boxed{s_i + s_i'}$



Multiplication Protocol: Malicious

$\boxed{s_i}$: commitment to s_i held by p_i

Linearity: $\boxed{s_i} + \boxed{s_i} = \boxed{s_i + s_i'}$



$f_0(x) = F(x,0)$

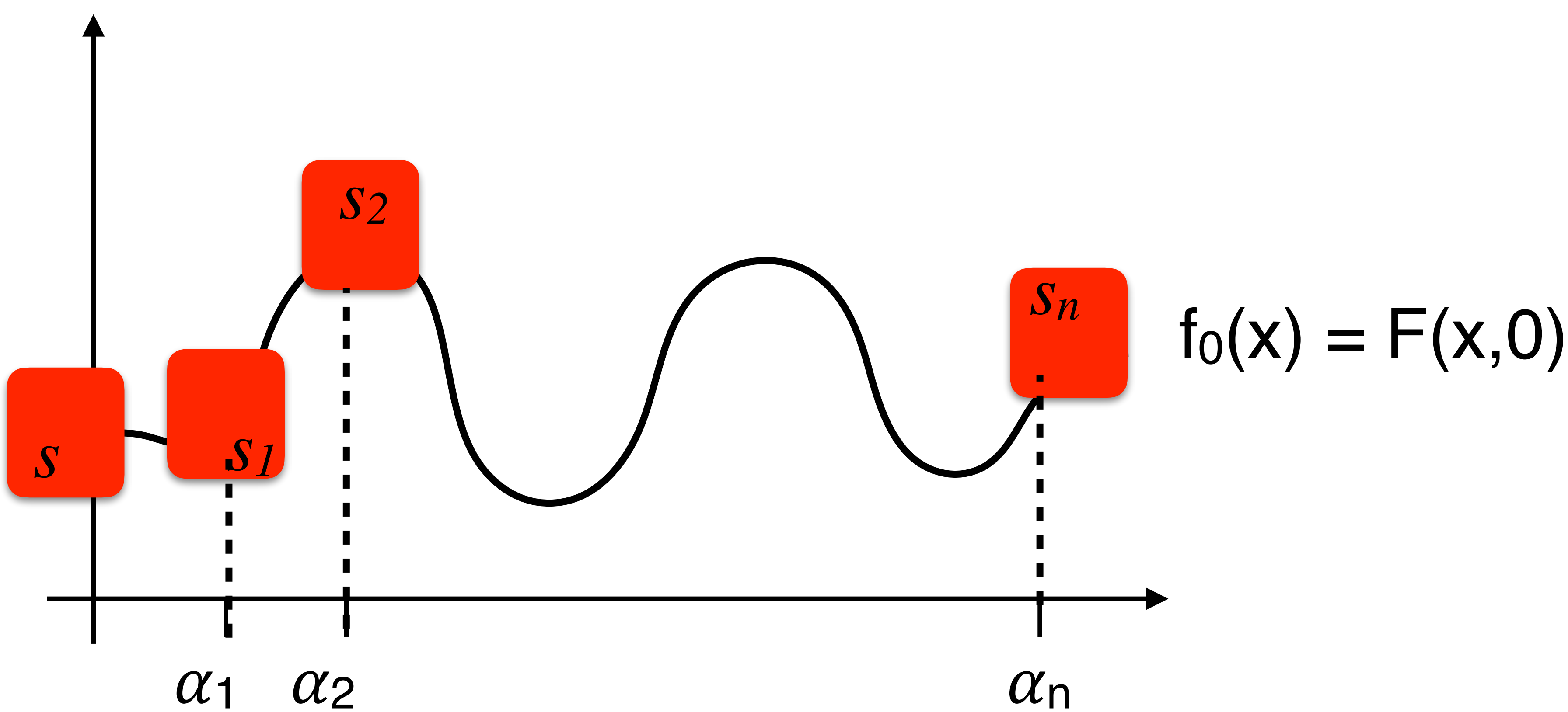
Multiplication Protocol: Malicious

$\boxed{s_i}$: commitment to s_i held by p_i

Linearity: $\boxed{s_i} + \boxed{s_i} = \boxed{s_i + s_i'}$

As in the semi honest setting to multiply shared s and s'

- Every p_i computes $\boxed{s_i''} = \boxed{s_i \cdot s_i'}$
- Use the linearity to compute a VSS of s''



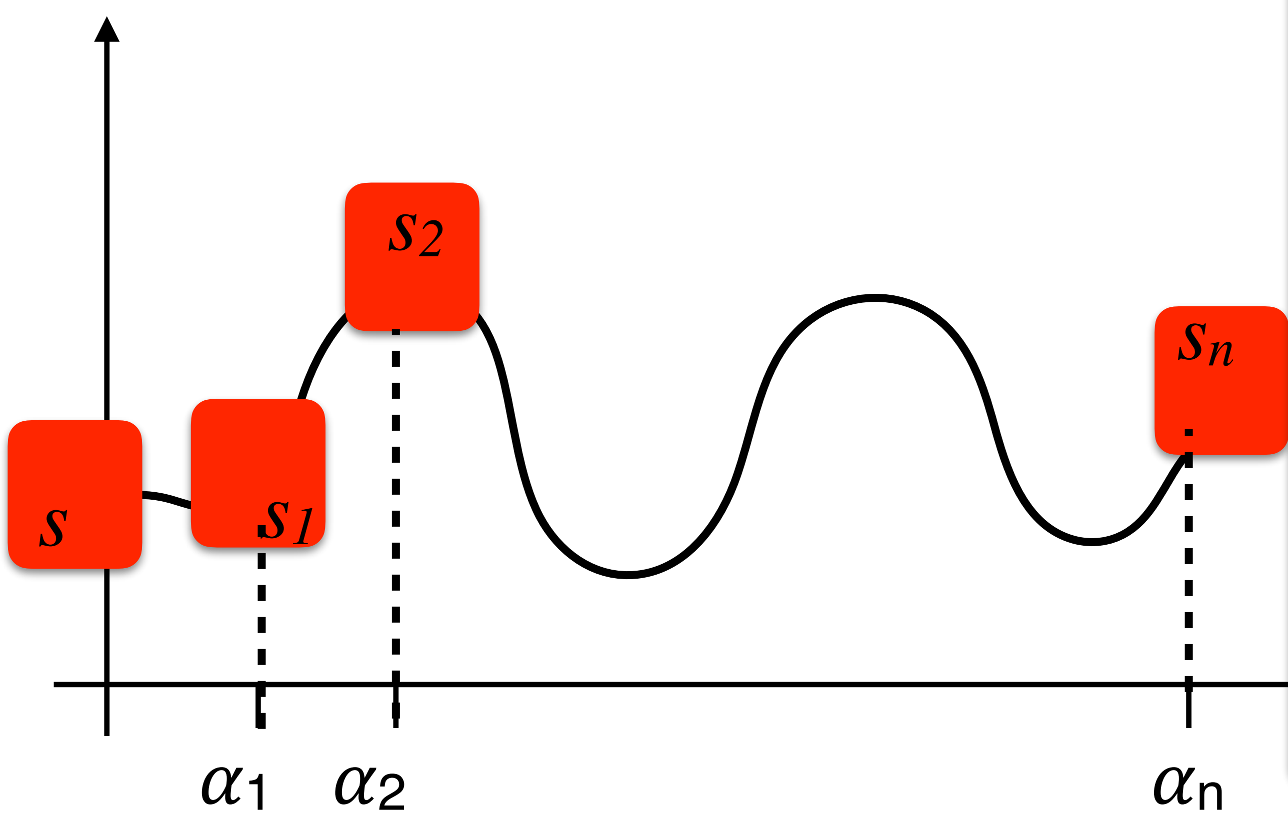
Multiplication Protocol: Malicious

$\boxed{s_i}$: commitment to s_i held by p_i

Linearity: $\boxed{s_i} + \boxed{s_i} = \boxed{s_i + s_i'}$

As in the semi honest setting to multiply shared s and s'

- Every p_i computes $\boxed{s_i''} = \boxed{s_i \cdot s_i'}$
- Use the linearity to compute a VSS of s''



we need a commitment multiplication protocol

- Similar idea to the semi honest protocol: Have every party commit to its share product and use linearity to combine them.
- + a check that the commitment is correct

References

- [Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.
- [LSP82] L. Lamport, R. Shostak, and M. Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (July 1982), 382-401. DOI=http://dx.doi.org/10.1145/357172.357176
- [DS83] D. Dolev and H. Strong. Authenticated algorithms for Byzantine agreement. SIAM J. Computing, 12(4):656–666, 1983.
- [BCR86] :G. Brassard, C. Crepeau, and J.-M. Robert. 1986. Information theoretic reductions among disclosure problems. FOCS '86. IEEE Computer Society, Washington, DC, USA, 168-173.
- [GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symposium on the Theory of Computing (STOC)*, pages 218–229, 1987.

References

- [BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant dis- tributed computation. In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.
- [CCD88] D. Chaum, C. Cre´peau, and I. Damga˚rd. Multi- party unconditionally secure protocols (extended abstract). In *Proc. 20th ACM Symposium on the Theory of Computing (STOC)*, pages 11–19, 1988.
- [BGP89] P. Berman, J. A. Garay, and K. J. Perry. 1989. Towards optimal distributed consensus. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science* (SFCS '89). IEEE Computer Society, Washington, DC, USA, 410-415. DOI=http://dx.doi.org/10.1109/SFCS.1989.63511
- [RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st ACM Symposium on the Theory of Computing (STOC)*, pages 73–85, 1989.